Dependency Pairs for Equational Rewriting*

Jürgen Giesl[1] and Deepak Kapur[2]

[1] LuFG Informatik II, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany,
giesl@informatik.rwth-aachen.de
[2] Computer Science Dept., University of New Mexico, Albuquerque, NM 87131, USA
kapur@cs.unm.edu

Abstract. The dependency pair technique of Arts and Giesl [1–3] for termination proofs of term rewrite systems (TRSs) is extended to rewriting modulo equations. Up to now, such an extension was only known in the special case of AC-rewriting [16, 18]. In contrast to that, the proposed technique works for arbitrary non-collapsing equations (satisfying a certain linearity condition). With the proposed approach, it is now possible to perform automated termination proofs for many systems where this was not possible before. In other words, the power of dependency pairs can now also be used for rewriting modulo equations.

1 Introduction

Termination of ordinary term rewriting has been extensively studied (e.g., in classical approaches based on *simplification orderings* [9, 23] and new powerful techniques like *dependency pairs* [1–3]). There has also been significant progress for termination of equational term rewrite systems whose equations only contain associativity and commutativity axioms (e.g., [8, 14, 15, 21, 22]). In particular, the dependency pair approach has also been extended to the AC-case [16, 18].

For equations other than AC-axioms, however, there are not many techniques available to prove termination. In an early paper [17], sufficient conditions are given for reducing termination of equational rewriting to termination of its underlying rewrite system. Another early paper [6] describes how to apply polynomial interpretations for AC-termination proofs (and this approach can also be used for equations other than AC-axioms). In newer papers, dummy elimination [11] and the semantic labelling method [19] are extended to rewriting modulo equations. However, dummy elimination is only applicable for certain subclasses of TRSs and semantic labelling is not amenable to automation.

This paper presents an extension of the dependency pair approach to rewriting modulo equations. In the special case of AC-axioms, our technique corresponds to the methods of [16, 18], but in contrast to these methods, our technique can also be used if the equations are not AC-axioms. This allows much

* Extended version of a paper which appeared in the *Proceedings of the 12th International Conference on Rewriting Techniques and Applications, RTA-2001*, Utrecht, The Netherlands, Lecture Notes in Computer Science, Springer-Verlag. Supported by the Deutsche Forschungsgemeinschaft Grant GI 274/4-1 and the National Science Foundation Grants nos. CCR-9996150, CDA-9503064, and CCR-9712396.

more automated termination proofs for equational rewrite systems than those possible with directly applying simplification orderings for equational rewriting (like equational polynomial orderings or AC-versions of path orderings).

We first review the dependency pair approach for ordinary term rewriting in Sect. 2. In Sect. 3, we show why a straightforward extension of dependency pairs to rewriting modulo equations is not possible. As observed in [16], the reason is that there can be *minimal non-terminating terms* (i.e., terms without proper non-terminating subterms) whose infinite reductions only involve reduction steps below the root level. Therefore, we follow an idea similar to the one of [18] for the special case of AC-axioms: We consider a restricted form of rewriting modulo equations, which is more suitable for termination proofs with dependency pairs.

In Sect. 4, we show how to ensure that termination of this restricted equational rewrite relation is equivalent to termination of full rewriting modulo equations. Under certain conditions on the equations $\mathcal{E}$, we give a method for computing an extended rewrite system $Ext_{\mathcal{E}}(\mathcal{R})$ from the given TRS $\mathcal{R}$ such that the restricted rewrite relation of $Ext_{\mathcal{E}}(\mathcal{R})$ modulo $\mathcal{E}$ is terminating iff $\mathcal{R}$ is terminating modulo $\mathcal{E}$. This is proved for (almost) arbitrary $\mathcal{E}$-rewriting, thus generalizing a related result for AC-rewriting. This general result may be of independent interest, and may also be found useful in investigating other properties of $\mathcal{E}$-rewriting.

Then in Sect. 5, we extend the dependency pair approach to rewriting modulo equations. The notion of defined symbols is modified by taking into consideration the function symbols appearing as the outermost symbols in equations in $\mathcal{E}$ as well. It is shown how for every non-terminating term, it is possible to build a reduction using the restricted form of rewriting induced by $Ext_{\mathcal{E}}(\mathcal{R})$ where only terminating or minimal non-terminating subterms are reduced. In order to ensure that an infinite reduction from a minimal non-terminating term can be achieved by applying only instantiations of rules where all variables are instantiated with terminating terms, it also becomes necessary to consider finitely many instantiations of the rules in $Ext_{\mathcal{E}}(\mathcal{R})$. The main result is then proved, generalizing the dependency pair method for showing termination of rewrite systems $\mathcal{R}$ [1–3] to rewrite systems $\mathcal{R}$ modulo sets $\mathcal{E}$ of non-collapsing equations with identical unique variables. This result can serve as the basis of an automatic method for showing termination of rewrite systems modulo equations. Finally, in Sect. 6 we show how the refinement of dependency graphs [1–3] can also be applied for rewriting modulo equations. Appendix A contains a collection of examples to demonstrate the power and the usefulness of our technique.

2 Dependency Pairs for Ordinary Rewriting

The dependency pair approach allows the use of standard methods like simplification orderings [9, 23] for automated termination proofs where they were not applicable before. In this section we briefly summarize the basic concepts of this approach. All results in this section are due to Arts and Giesl and we refer to [1–3] for further details, refinements, and explanations.

In contrast to the standard techniques for termination proofs, which compare left and right-hand sides of rules, in this approach one concentrates on the subterms in the right-hand sides that have a defined[1] root symbol, because these are the only terms responsible for starting new reductions.

More precisely, for every rule $f(s_1,\ldots,s_n) \to C[g(t_1,\ldots,t_m)]$ (where f and g are defined symbols), we compare the argument tuples $s_1,\ldots,s_n$ and $t_1,\ldots,t_m$. To avoid the handling of tuples, for every defined symbol f, we introduce a fresh *tuple* symbol F. To ease readability, we assume that the original signature consists of lower case function symbols only, whereas the tuple symbols are denoted by the corresponding upper case symbols. Now instead of the tuples $s_1,\ldots,s_n$ and $t_1,\ldots,t_m$ we compare the *terms* $F(s_1,\ldots,s_n)$ and $G(t_1,\ldots,t_m)$.

Definition 1 (Dependency Pair [1–3]). *If $f(s_1,\ldots,s_n) \to C[g(t_1,\ldots,t_m)]$ is a rule of a TRS $\mathcal{R}$ and g is a defined symbol, then $\langle F(s_1,\ldots,s_n), G(t_1,\ldots,t_m)\rangle$ is a* dependency pair *of $\mathcal{R}$.*

Example 2. As an example, consider the TRS $\{\mathsf{a}+\mathsf{b} \to \mathsf{a}+(\mathsf{b}+\mathsf{c})\}$, cf. [18]. Termination of this system cannot be shown by simplification orderings, since the left-hand side of the rule is embedded in the right-hand side. In this system, the defined symbol is $+$ and thus, we obtain the dependency pairs $\langle \mathsf{P}(\mathsf{a},\mathsf{b}), \mathsf{P}(\mathsf{a},\mathsf{b}+\mathsf{c})\rangle$ and $\langle \mathsf{P}(\mathsf{a},\mathsf{b}), \mathsf{P}(\mathsf{b},\mathsf{c})\rangle$ (where P is the tuple symbol for the plus-function "+").

Arts and Giesl developed the following new termination criterion. As usual, a quasi-ordering $\succsim$ is a reflexive and transitive relation, and we say that an ordering $>$ is *compatible* with $\succsim$ if we have $> \circ \succsim \subseteq >$ or $\succsim \circ > \subseteq >$.

Theorem 3 (Termination with Dependency Pairs [1–3]). *A TRS $\mathcal{R}$ is terminating iff there exists a weakly monotonic quasi-ordering $\succsim$ and a well-founded ordering $>$ compatible with $\succsim$, where both $\succsim$ and $>$ are closed under substitution, such that*

(1) $s > t$ for all dependency pairs $\langle s,t\rangle$ of $\mathcal{R}$ and
(2) $l \succsim r$ for all rules $l \to r$ of $\mathcal{R}$.

Consider the TRS from Ex. 2 again. In order to prove its termination according to Thm. 3, we have to find a suitable quasi-ordering $\succsim$ and ordering $>$ such that $\mathsf{P}(\mathsf{a},\mathsf{b}) > \mathsf{P}(\mathsf{a},\mathsf{b}+\mathsf{c})$, $\mathsf{P}(\mathsf{a},\mathsf{b}) > \mathsf{P}(\mathsf{b},\mathsf{c})$, and $\mathsf{a}+\mathsf{b} \succsim \mathsf{a}+(\mathsf{b}+\mathsf{c})$.

Most standard orderings amenable to automation are *strongly* monotonic (cf. e.g. [9, 23]), whereas here we only need *weak* monotonicity. Hence, before synthesizing a suitable ordering, some of the arguments of function symbols may be eliminated, cf. [3]. For example, in our inequalities, one may eliminate the first argument of $+$. Then every term $s+t$ in the inequalities is replaced by $+'(t)$ (where $+'$ is a new unary function symbol). By comparing the terms resulting from this replacement instead of the original terms, we can take advantage of the fact that $+$ does not have to be strongly monotonic in its first argument.

[1] Root symbols of left-hand sides are *defined* and all other functions are *constructors*.

Note that there are only finitely many possibilities to eliminate arguments of function symbols. Therefore all these possibilities can be checked automatically.

In this way, we obtain the inequalities $\mathsf{P}(\mathsf{a},\mathsf{b}) > \mathsf{P}(\mathsf{a},+'(\mathsf{c}))$, $\mathsf{P}(\mathsf{a},\mathsf{b}) > \mathsf{P}(\mathsf{b},\mathsf{c})$, and $+'(\mathsf{b}) \gtrsim +'(+'(\mathsf{c}))$. These inequalities are satisfied by the recursive path ordering (rpo) [9] with the precedence $\mathsf{a} \sqsupset \mathsf{b} \sqsupset \mathsf{c} \sqsupset +'$ (i.e., we choose $\gtrsim$ to be $\gtrsim_{rpo}$ and $>$ to be $\succ_{rpo}$). So termination of this TRS can now be proved automatically.

Apart from eliminating arguments of function symbols, another possibility is to replace functions by one of their arguments. So instead of deleting the first argument of $+$ one could also replace all terms $s+t$ by $+$'s second argument t. Then the resulting inequalities are again satisfied by the rpo. For implementations of the dependency pair approach see [4, 7].

3 Rewriting Modulo Equations

For a set $\mathcal{E}$ of equations between terms, we write $s \to_{\mathcal{E}} t$ if there exist an equation $l \approx r$ in $\mathcal{E}$, a substitution σ, and a context C such that $s = C[l\sigma]$ and $t = C[r\sigma]$. The symmetric closure of $\to_{\mathcal{E}}$ is denoted by $\vdash\!\dashv_{\mathcal{E}}$ and the transitive reflexive closure of $\vdash\!\dashv_{\mathcal{E}}$ is denoted by $\sim_{\mathcal{E}}$. In the following, we restrict ourselves to equations $\mathcal{E}$ where $\sim_{\mathcal{E}}$ is decidable.

Definition 4 (Rewriting Modulo Equations). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ be a set of equations. A term s rewrites to a term t modulo $\mathcal{E}$, denoted $s \to_{\mathcal{R}/\mathcal{E}} t$, iff there exist terms s' and t' such that $s \sim_{\mathcal{E}} s' \to_{\mathcal{R}} t' \sim_{\mathcal{E}} t$. The TRS $\mathcal{R}$ is called terminating modulo $\mathcal{E}$ iff there does not exist an infinite $\to_{\mathcal{R}/\mathcal{E}}$ reduction.*

Example 5. An interesting special case are equations $\mathcal{E}$ which state that certain function symbols are associative and commutative (AC). As an example, consider the TRS $\mathcal{R} = \{\mathsf{a}+\mathsf{b} \to \mathsf{a}+(\mathsf{b}+\mathsf{c})\}$ again and let $\mathcal{E}$ consist of the associativity and commutativity axioms for $+$, i.e., $\mathcal{E} = \{x_1 + x_2 \approx x_2 + x_1, x_1 + (x_2 + x_3) \approx (x_1 + x_2) + x_3\}$, cf. [18]. $\mathcal{R}$ is not terminating modulo $\mathcal{E}$, since we have

$$\mathsf{a}+\mathsf{b} \to_{\mathcal{R}} \mathsf{a}+(\mathsf{b}+\mathsf{c}) \sim_{\mathcal{E}} (\mathsf{a}+\mathsf{b})+\mathsf{c} \to_{\mathcal{R}} (\mathsf{a}+(\mathsf{b}+\mathsf{c}))+\mathsf{c} \sim_{\mathcal{E}} ((\mathsf{a}+\mathsf{b})+\mathsf{c})+\mathsf{c} \to_{\mathcal{R}} \ldots$$

There are, however, many other sets of equations $\mathcal{E}$ apart from associativity and commutativity, which are also important in practice. Hence, our aim is to extend dependency pairs to rewriting modulo (almost) arbitrary equations.

The soundness of dependency pairs for ordinary rewriting relies on the fact that whenever a term starts an infinite reduction, then one can also construct an infinite reduction where only *terminating or minimal non-terminating subterms* are reduced (i.e., one only applies rules to redexes without proper non-terminating subterms). The contexts of minimal non-terminating redexes can be completely disregarded. If a rule is applied at the root position of a minimal non-terminating subterm s (i.e., $s \to^{\epsilon}_{\mathcal{R}} t$ where ϵ denotes the root position), then s and each minimal non-terminating subterm t' of t correspond to a dependency pair. Hence, Thm. 3 (1) implies $s > t'$. If a rule is applied at a non-root

position of a minimal non-terminating subterm s (i.e., $s \to_{\mathcal{R}}^{\geq\epsilon} t$), then we have $s \gtrsim t$ by Thm. 3 (2). However, due to the minimality of s, after finitely many such non-root rewrite steps, a rule must be applied at the root position of the minimal non-terminating term. Thus, every infinite reduction of minimal non-terminating subterms corresponds to an infinite $>$-sequence. This contradicts the well-foundedness of $>$.

So for ordinary rewriting, any infinite reduction from a minimal non-terminating subterm involves an $\mathcal{R}$-reduction at the root position. But when extending the dependency pair approach to rewriting modulo equations, this is no longer true, cf. [16]. For an illustration, consider Ex. 5 again, where $\mathsf{a} + (\mathsf{b} + \mathsf{c})$ is a minimal non-terminating term. However, in its infinite $\mathcal{R}/\mathcal{E}$-reduction no $\mathcal{R}$-step is ever applicable at the root position. (Instead one applies an $\mathcal{E}$-step at the root position and further $\mathcal{R}$- and $\mathcal{E}$-steps below the root.)

In the rest of the paper, from a rewrite system $\mathcal{R}$, we generate a new rewrite system $\mathcal{R}'$ with the following three properties: (i) the termination of a weaker form of rewriting by $\mathcal{R}'$ modulo $\mathcal{E}$ is equivalent to the termination of $\mathcal{R}$ modulo $\mathcal{E}$, (ii) every infinite reduction of a minimal non-terminating term in this weaker form of rewriting by $\mathcal{R}'$ modulo $\mathcal{E}$ involves a reduction step at the root level, and (iii) every such minimal non-terminating term has an infinite reduction where the variables of the $\mathcal{R}'$-rules are instantiated with terminating terms only.

4 $\mathcal{E}$-Extended Rewriting

We showed why the dependency pair approach cannot be extended to rewriting modulo equations directly. As a solution for this problem, we propose to consider a restricted form of rewriting modulo equations, i.e., the so-called *$\mathcal{E}$-extended $\mathcal{R}$-rewrite relation* $\to_{\mathcal{E}\setminus\mathcal{R}}$. (This approach was already taken in [18] for rewriting modulo AC.) The relation $\to_{\mathcal{E}\setminus\mathcal{R}}$ was originally introduced in [20] in order to circumvent the problems with infinite or impractically large $\mathcal{E}$-equivalence classes.[2]

Definition 6 ($\mathcal{E}$-extended $\mathcal{R}$-rewriting [20]). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ be a set of equations. The $\mathcal{E}$-extended $\mathcal{R}$-rewrite relation is defined as $s \to_{\mathcal{E}\setminus\mathcal{R}}^{\pi} t$ iff $s|_\pi \sim_{\mathcal{E}} l\sigma$ and $t = s[r\sigma]_\pi$ for some rule $l \to r$ in $\mathcal{R}$, some position π of s, and some substitution σ. We also write $\to_{\mathcal{E}\setminus\mathcal{R}}$ instead of $\to_{\mathcal{E}\setminus\mathcal{R}}^{\pi}$.*

To demonstrate the difference between $\to_{\mathcal{R}/\mathcal{E}}$ and $\to_{\mathcal{E}\setminus\mathcal{R}}$, consider Ex. 5 again. We have already seen that $\to_{\mathcal{R}/\mathcal{E}}$ is not terminating, since $\mathsf{a} + \mathsf{b} \to_{\mathcal{R}/\mathcal{E}} (\mathsf{a} + \mathsf{b}) + \mathsf{c} \to_{\mathcal{R}/\mathcal{E}} ((\mathsf{a} + \mathsf{b}) + \mathsf{c}) + \mathsf{c} \to_{\mathcal{R}/\mathcal{E}} \ldots$ But $\to_{\mathcal{E}\setminus\mathcal{R}}$ is terminating, because $\mathsf{a} + \mathsf{b} \to_{\mathcal{E}\setminus\mathcal{R}} \mathsf{a} + (\mathsf{b} + \mathsf{c})$, which is a normal form w.r.t. $\to_{\mathcal{E}\setminus\mathcal{R}}$.

The above example also demonstrates that in general, termination of $\to_{\mathcal{E}\setminus\mathcal{R}}$ is not sufficient for termination of $\to_{\mathcal{R}/\mathcal{E}}$. In this section we will show how termination of $\to_{\mathcal{R}/\mathcal{E}}$ can nevertheless be ensured by only regarding an $\mathcal{E}$-extended rewrite relation induced by a larger $\mathcal{R}' \supseteq \mathcal{R}$.

[2] In [12], the relation $\to_{\mathcal{E}\setminus\mathcal{R}}$ is denoted "$\to_{\mathcal{R},\mathcal{E}}$".

For the special case of AC-rewriting, this problem can be solved by extending $\mathcal{R}$ as follows: Let $\mathcal{G}$ be the set of all AC-symbols and

$$Ext_{AC(\mathcal{G})} = \mathcal{R} \cup \{f(l,y) \to f(r,y) \mid l \to r \in \mathcal{R},\ root(l) = f \in \mathcal{G}\},$$

where y is a new variable not occurring in the respective rule $l \to r$. A similar extension has also been used in previous work on extending dependency pairs to AC-rewriting [18]. The reason is that for AC-equations $\mathcal{E}$, the termination of $\to_{\mathcal{R}/\mathcal{E}}$ is in fact equivalent to the termination of $\to_{\mathcal{E}\backslash Ext_{AC(\mathcal{G})}(\mathcal{R})}$. In fact, it is even possible to reduce the set $Ext_{AC(\mathcal{G})}$ a bit, since rules of the form $f(l,y) \to f(r,y)$ do not have to be included in $Ext_{AC(\mathcal{G})}$ if $l \sim_{\mathcal{E}} f(l',z)$ and $r \sim_{\mathcal{E}} f(r',z)$ holds for some terms l', r' and a variable z which does not occur in l' or r', cf. [20].

For Ex. 5, we obtain $Ext_{AC(\mathcal{G})}(\mathcal{R}) = \{\mathsf{a} + \mathsf{b} \to \mathsf{a} + (\mathsf{b} + \mathsf{c}), (\mathsf{a} + \mathsf{b}) + y \to (\mathsf{a}+(\mathsf{b}+\mathsf{c}))+y\}$. Thus, in order to prove termination of $\to_{\mathcal{R}/\mathcal{E}}$, it is now sufficient to verify termination of $\to_{\mathcal{E}\backslash Ext_{AC(\mathcal{G})}(\mathcal{R})}$.

The above extension of [20] only works for AC-axioms $\mathcal{E}$. A later paper [12] treats arbitrary equations, but it does not contain any definition for extensions $Ext_{\mathcal{E}}(\mathcal{R})$, and termination of $\to_{\mathcal{R}/\mathcal{E}}$ is always a prerequisite in [12]. The reason is that [12] and also subsequent work on symmetrization and coherence were devoted to the development of completion algorithms (i.e., here the goal was to generate a convergent rewrite system and not to investigate the termination behavior of possibly non-terminating TRSs). Thus, these papers did not compare the termination behavior of full rewriting modulo equations with the termination of restricted versions of rewriting modulo equations. In fact, [12] focuses on the notion of coherence, which is not suitable for our purpose since coherence of $\mathcal{E}\backslash\mathcal{R}$ modulo $\mathcal{E}$ does not imply that termination of $\to_{\mathcal{R}/\mathcal{E}}$ is equivalent to termination of $\to_{\mathcal{E}\backslash\mathcal{R}}$.[3]

To extend dependency pairs to rewriting modulo non-AC-equations $\mathcal{E}$, we have to compute extensions $Ext_{\mathcal{E}}(\mathcal{R})$ such that termination of $\to_{\mathcal{R}/\mathcal{E}}$ is equivalent to termination of $\to_{\mathcal{E}\backslash Ext_{\mathcal{E}}(\mathcal{R})}$. The only restriction we will impose on the equations in $\mathcal{E}$ is that they must have *identical unique variables*. As usual, a term t is called *linear* if no variable occurs more than once in t.

Definition 7 (Equations with Identical Unique Variables [20]). *An equation $u \approx v$ is said to have* identical unique variables *if u and v are both linear and the variables in u are the same as the variables in v.*

While this requirement may seem restrictive at first sight, it turns out that most practical examples where $\mathcal{R}/\mathcal{E}$ is terminating satisfy this restriction: The restriction that the set of variables must be the same in both terms of an equation

[3] In [12], $\mathcal{E}\backslash\mathcal{R}$ is coherent modulo $\mathcal{E}$ iff for all terms s,t,u, we have that $s \sim_{\mathcal{E}} t \to^+_{\mathcal{E}\backslash\mathcal{R}} u$ implies $s \to^+_{\mathcal{E}\backslash\mathcal{R}} v \sim_{\mathcal{E}} w \leftarrow^*_{\mathcal{E}\backslash\mathcal{R}} u$ for some v, w. Consider $\mathcal{R} = \{\mathsf{a} + \mathsf{b} \to \mathsf{a} + (\mathsf{b} + \mathsf{c}),\ \ x + y \to \mathsf{d}\}$ with $\mathcal{E}$ being the AC-axioms for $+$. The above system is coherent, since $s \sim_{\mathcal{E}} t \to^+_{\mathcal{E}\backslash\mathcal{R}} u$ implies $s \to^+_{\mathcal{R}} \mathsf{d} \leftarrow^*_{\mathcal{R}} u$. However, $\to_{\mathcal{E}\backslash\mathcal{R}}$ is terminating but $\to_{\mathcal{R}/\mathcal{E}}$ is not terminating.

is not severe, because otherwise rewriting modulo such an equation would not terminate (as long as there exists a function symbol f with arity ≥ 2 and $\mathcal{R} \neq \emptyset$).[4] The reason is that if x occurs in u but not in v and if $l \to r$ is a rewrite rule, then we obtain

$$\begin{aligned} v &\sim_{\mathcal{E}} u[x/f(v,l,\ldots)] \\ &\to_{\mathcal{R}} u[x/f(v,r,\ldots)] \\ &\sim_{\mathcal{E}} u[x/f(u[x/f(v,l,\ldots)],r,\ldots)] \\ &\to_{\mathcal{R}} u[x/f(u[x/f(v,r,\ldots)],r,\ldots)] \\ &\sim_{\mathcal{E}} \ldots \end{aligned}$$

Thus, rewriting modulo equations such as $x \cdot 0 \approx 0$ or $x \cdot x^{-1} \approx 1$ never terminates if $\mathcal{R} \neq \emptyset$.

Moreover, as already pointed out in [10], the linearity condition is also not too restrictive, since if u is a non-linear term $f(\ldots x \ldots x \ldots)$, then at least if v is the single variable x, the relation $\to_{\mathcal{R}/\mathcal{E}}$ would again be non-terminating if $\mathcal{R} \neq \emptyset$. (In fact, collapsing equations will be forbidden anyway in Sect. 5 in order to make the dependency pair approach sound.) The reason is that if $l \to r \in \mathcal{R}$, then we would have

$$\begin{aligned} l &\sim_{\mathcal{E}} f(\ldots l \ldots l \ldots) \\ &\to_{\mathcal{R}} f(\ldots r \ldots l \ldots) \\ &\sim_{\mathcal{E}} f(\ldots r \ldots f(\ldots l \ldots l \ldots) \ldots) \\ &\to_{\mathcal{R}} f(\ldots r \ldots f(\ldots r \ldots l \ldots) \ldots) \\ &\sim_{\mathcal{E}} \ldots \end{aligned}$$

This means that rewriting modulo an equation like $x \cdot x \approx x$ is always non-terminating if $\mathcal{R} \neq \emptyset$.

Let $uni_{\mathcal{E}}(s,t)$ denote a complete set of $\mathcal{E}$-unifiers of two terms s and t. As usual, δ is an *$\mathcal{E}$-unifier* of s and t iff $s\delta \sim_{\mathcal{E}} t\delta$ and a set $uni_{\mathcal{E}}(s,t)$ of $\mathcal{E}$-unifiers is *complete* iff for every $\mathcal{E}$-unifier δ there exists a $\sigma \in uni_{\mathcal{E}}(s,t)$ and a substitution ρ such that $\delta \sim_{\mathcal{E}} \sigma\rho$, cf. [5]. ("$\sigma\rho$" is the composition of σ and ρ where σ is applied first and "$\delta \sim_{\mathcal{E}} \sigma\rho$" means that for all variables x we have $x\delta \sim_{\mathcal{E}} x\sigma\rho$.)

To construct $Ext_{\mathcal{E}}(\mathcal{R})$, we consider all overlaps between equations $u \approx v$ or $v \approx u$ from $\mathcal{E}$ and rules $l \to r$ from $\mathcal{R}$. More precisely, we check whether a non-variable subterm $v|_\pi$ of v $\mathcal{E}$-unifies with l (where we always assume that rules in $\mathcal{R}$ are variable disjoint from equations in $\mathcal{E}$). In this case one adds the rules $(v[l]_\pi)\sigma \to (v[r]_\pi)\sigma$ for all $\sigma \in uni_{\mathcal{E}}(v|_\pi, l)$.[5] In Ex. 5, the subterm $x_1 + x_2$ of the right-hand side of $x_1 + (x_2 + x_3) \approx (x_1 + x_2) + x_3$ unifies with the left-hand

[4] A similar observation was already mentioned in [10], but here the requirement of function symbols with arity ≥ 2 was neglected. Note however that without this condition $\to_{\mathcal{R}/\mathcal{E}}$ may still be terminating. As an example consider $\mathcal{E} = \{\mathsf{f}(x) \approx \mathsf{a}\}$ and $\mathcal{R} = \{\mathsf{a} \to \mathsf{b}\}$.

[5] Obviously, $uni_{\mathcal{E}}(v|_\pi, l)$ always exists, but it can be infinite in general. So when automating our approach for equational termination proofs, we have to restrict our-

side of the only rule $\mathsf{a}+\mathsf{b} \to \mathsf{a}+(\mathsf{b}+\mathsf{c})$. Thus, in the extension of $\mathcal{R}$, we obtain the rule $(\mathsf{a}+\mathsf{b})+y \to (\mathsf{a}+(\mathsf{b}+\mathsf{c}))+y$.

$Ext_{\mathcal{E}}(\mathcal{R})$ is built via a kind of fixpoint construction, i.e., we also have to consider overlaps between equations of $\mathcal{E}$ and the newly constructed rules of $Ext_{\mathcal{E}}(\mathcal{R})$. For example, the subterm x_1+x_2 also unifies with the left-hand side of the new rule $(\mathsf{a}+\mathsf{b})+y \to (\mathsf{a}+(\mathsf{b}+\mathsf{c}))+y$. Thus, one would now construct a new rule $((\mathsf{a}+\mathsf{b})+y)+z \to ((\mathsf{a}+(\mathsf{b}+\mathsf{c}))+y)+z$.

Obviously, in this way one obtains an infinite number of rules by subsequently overlapping equations with the newly constructed rules. However, in order to use $Ext_{\mathcal{E}}(\mathcal{R})$ for automated termination proofs, our aim is to restrict ourselves to finitely many rules. It turns out that we do not have to include new rules $(v[l]_\pi)\sigma \to (v[r]_\pi)\sigma$ in $Ext_{\mathcal{E}}(\mathcal{R})$ if $u\sigma \to^{\pi'}_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} q \sim_{\mathcal{E}} (v[r]_\pi)\sigma$ already holds for some position π' of u and some term q (using just the old rules of $Ext_{\mathcal{E}}(\mathcal{R})$).

When constructing the rule $((\mathsf{a}+\mathsf{b})+y)+z \to ((\mathsf{a}+(\mathsf{b}+\mathsf{c}))+y)+z$ above, the equation $u \approx v$ used was $x_1+(x_2+x_3) \approx (x_1+x_2)+x_3$ and the unifier σ replaced x_1 by $(\mathsf{a}+\mathsf{b})$ and x_2 by y. Hence, here $u\sigma$ is the term $(\mathsf{a}+\mathsf{b})+(y+x_3)$. But this term reduces with $\to^1_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$ to $(\mathsf{a}+(\mathsf{b}+\mathsf{c}))+(y+x_3)$ which is indeed $\sim_{\mathcal{E}}$-equivalent to $(v[r]_\pi)\sigma$, i.e., to $((\mathsf{a}+(\mathsf{b}+\mathsf{c}))+y)+x_3$. Thus, we do not have to include the rule $((\mathsf{a}+\mathsf{b})+y)+z \to ((\mathsf{a}+(\mathsf{b}+\mathsf{c}))+y)+z$ in $Ext_{\mathcal{E}}(\mathcal{R})$.

The following definition shows how suitable extensions can be computed for arbitrary equations with identical unique variables. It will turn out that with these extensions one can indeed simulate $\to_{\mathcal{R}/\mathcal{E}}$ by $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$, i.e., $s \to_{\mathcal{R}/\mathcal{E}} t$ implies $s \to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} t'$ for some $t' \sim_{\mathcal{E}} t$. This constitutes a crucial contribution of the paper, since it is the main requirement needed in order to extend dependency pairs to rewriting modulo equations.

Definition 8 (Extending $\mathcal{R}$ for Arbitrary Equations). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ be a set of equations. Let $\mathcal{R}'$ be a set containing only rules of the form $C[l\sigma] \to C[r\sigma]$ (where C is a context, σ is a substitution, and $l \to r \in \mathcal{R}$). $\mathcal{R}'$ is an* extension of $\mathcal{R}$ for the equations $\mathcal{E}$ *iff*

(a) $\mathcal{R} \subseteq \mathcal{R}'$ and
(b) for all $l \to r \in \mathcal{R}'$, $u \approx v \in \mathcal{E}$ and $v \approx u \in \mathcal{E}$, all positions π of v and $\sigma \in uni_{\mathcal{E}}(v|_\pi, l)$, there is a position π' in u and a $q \sim_{\mathcal{E}} (v[r]_\pi)\sigma$ with $u\sigma \to^{\pi'}_{\mathcal{E}\setminus\mathcal{R}'} q$.

In the following, let $Ext_{\mathcal{E}}(\mathcal{R})$ always denote an arbitrary extension of $\mathcal{R}$ for $\mathcal{E}$.

In order to satisfy Condition (b) of Def. 8, it is always sufficient to add the rule $(v[l]_\pi)\sigma \to (v[r]_\pi)\sigma$ to $\mathcal{R}'$. The reason is that then we have $u\sigma \to^{\epsilon}_{\mathcal{E}\setminus\mathcal{R}'} (v[r]_\pi)\sigma$. But if $u\sigma \to^{\pi'}_{\mathcal{E}\setminus\mathcal{R}'} q \sim_{\mathcal{E}} (v[r]_\pi)\sigma$ already holds with the other rules of $\mathcal{R}'$, then the rule $(v[l]_\pi)\sigma \to (v[r]_\pi)\sigma$ does not have to be added to $\mathcal{R}'$.

selves to equations $\mathcal{E}$ where $uni_{\mathcal{E}}(v|_\pi, l)$ can be chosen to be finite for all subterms $v|_\pi$ of equations and left-hand sides of rules l. This includes all sets $\mathcal{E}$ of finitary unification type, but our restriction is weaker, since we only need finiteness for certain terms $v|_\pi$ and l.

Condition (b) of Def. 8 also makes sure that as long as the equations have identical unique variables, we do not have to consider overlaps at variable positions.[6] The reason is that if $v|_\pi$ is a variable $x \in \mathcal{V}$, then we have $u\sigma = u[x\sigma]_{\pi'} \sim_\mathcal{E} u[l\sigma]_{\pi'} \to_\mathcal{R} u[r\sigma]_{\pi'} \sim_\mathcal{E} v[r\sigma]_\pi = (v[r]_\pi)\sigma$, where π' is the position of x in u. Hence, such rules $(v[l]_\pi)\sigma \to (v[r]_\pi)\sigma$ do not have to be included in $\mathcal{R}'$.

Overlaps at root positions do not have to be considered either. To see this, assume that π is the top position ϵ of v, i.e., that $v\sigma \sim_\mathcal{E} l\sigma$. In this case we have $u\sigma \sim_\mathcal{E} v\sigma \sim_\mathcal{E} l\sigma \to_\mathcal{R} r\sigma$ and thus, $u\sigma \to^\epsilon_{\mathcal{E}\setminus\mathcal{R}} r\sigma = (v[r]_\pi)\sigma$. So again, such rules $(v[l]_\pi) \to (v[r]_\pi)\sigma$ do not have to be included in $\mathcal{R}'$.

The following procedure is used to compute extensions. Here, we assume both $\mathcal{R}$ and $\mathcal{E}$ to be finite, where the equations $\mathcal{E}$ must have identical unique variables.

1. $\mathcal{R}' := \mathcal{R}$
2. For all $l \to r \in \mathcal{R}'$,
   all $u \approx v$ or $v \approx u$ from $\mathcal{E}$,
   and all positions π of v where $\pi \neq \epsilon$ and $v|_\pi \notin \mathcal{V}$ do:
   2.1. Let $\Sigma := uni_\mathcal{E}(v|_\pi, l)$.
   2.2. For all $\sigma \in \Sigma$ do:
      2.2.1. Let $T := \{q \mid u\sigma \to^{\pi'}_{\mathcal{E}\setminus\mathcal{R}'} q$ for a position π' of $u\}$.
      2.2.2. If there exists a $q \in T$ with $(v[r]_\pi)\sigma \sim_\mathcal{E} q$, then $\Sigma := \Sigma \setminus \{\sigma\}$.
   2.3. $\mathcal{R}' := \mathcal{R}' \cup \{(v[l]_\pi)\sigma \to (v[r]_\pi)\sigma \mid \sigma \in \Sigma\}$.

This algorithm has the following properties:

(a) If in Step 2.1, $uni_\mathcal{E}(v|_\pi, l)$ is finite and computable, then every step in the algorithm is computable.
(b) If the algorithm terminates, then the final value of $\mathcal{R}'$ is an extension of $\mathcal{R}$ for the equations $\mathcal{E}$.

Note that the condition for stopping further computations of new rules in Steps 2.2.1. and 2.2.2. can indeed be checked automatically, since a term $u\sigma$ can only $\to_{\mathcal{E}\setminus\mathcal{R}'}$-reduce to finitely many terms q.

With the TRS of Ex. 5, $Ext_\mathcal{E}(\mathcal{R}) = \{\mathsf{a}+\mathsf{b} \to \mathsf{a}+(\mathsf{b}+\mathsf{c}), (\mathsf{a}+\mathsf{b})+y \to (\mathsf{a}+(\mathsf{b}+\mathsf{c}))+y\}$. In general, if $\mathcal{E}$ only consists of AC-axioms for some function symbols $\mathcal{G}$, then Def. 8 "coincides" with the well-known extension for AC-axioms.[7]

Lemma 9 (Coincidence of $Ext_{AC(\mathcal{G})}$ and Def. 8 for AC-axioms). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ consist of the associativity and commutativity axioms for all function symbols from a subset $\mathcal{G}$ of the signature. Then $\mathcal{R} \cup \{f(l,y) \to f(r,y) \mid l \to r \in \mathcal{R}, root(l) = f \in \mathcal{G}\}$ is an extension of $\mathcal{R}$ for the equations $\mathcal{E}$ (as defined in Def. 8).*

[6] Note that considering overlaps at variable positions as well would still not allow us to treat equations with non-linear terms. As an example regard $\mathcal{E} = \{\mathsf{f}(x) \approx \mathsf{g}(x,x)\}$ and $\mathcal{R} = \{\mathsf{g}(\mathsf{a},\mathsf{b}) \to \mathsf{f}(\mathsf{a}), \mathsf{a} \to \mathsf{b}\}$. Here, $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$ is well founded although $\mathcal{R}$ is not terminating modulo $\mathcal{E}$.

[7] This statement also holds for the reduced version of $Ext_{AC(\mathcal{G})}$, where rules of the form $f(l,y) \to f(r,y)$ are deleted if $l \sim_\mathcal{E} f(l',z)$, $r \sim_\mathcal{E} f(r',z)$ and the variable z does not occur in l' or r'.

Proof. Let $\mathcal{R}' = \mathcal{R} \cup \{f(l,y) \to f(r,y) \mid l \to r \in \mathcal{R}, root(l) = f \in \mathcal{G}\}$. We have to show that this set satisfies the conditions (a) and (b) of Def. 8. Condition (a) is obvious since $\mathcal{R} \subseteq \mathcal{R}'$. Hence, it remains to show that Condition (b) does not enforce the addition of other rules.

As illustrated in the discussion after Def. 8, if π is the top position ϵ or if $v|_\pi \in \mathcal{V}$, then Condition (b) is always fulfilled. We now regard the case where $v = f(f(x_1,x_2),x_3)$ and $v|_\pi = f(x_1,x_2)$. The case where $v = f(x_1,f(x_2,x_3))$ and $v|_\pi = f(x_2,x_3)$ works analogously.

If $l \to r \in \mathcal{R}$, then we obtain $u\sigma \sim_\mathcal{E} v\sigma \sim_\mathcal{E} f(l,x_3)\sigma \to^\epsilon_{\mathcal{R}'} f(r,x_3)\sigma = (v[r]_\pi)\sigma$ with the rule $f(l,y) \to f(r,y)$ from $\mathcal{R}'$. Otherwise, if $l = f(l',y)$ and $r = f(r',y)$ for some rule $l' \to r' \in \mathcal{R}$, we have $u\sigma \sim_\mathcal{E} v\sigma \sim_\mathcal{E} f(f(l',y),x_3)\sigma \sim_\mathcal{E} f(l',f(y,x_3))\sigma \to^\epsilon_{\mathcal{R}'} f(r',f(y,x_3))\sigma \sim_\mathcal{E} f(f(r',y),x_3)\sigma = (v[r]_\pi)\sigma$ with the rule $f(l',y) \to f(r',y)$ from $\mathcal{R}'$. □

So in case of AC-equations, our approach indeed corresponds to the approaches of [16, 18]. However, Def. 8 can also be used for other forms of equations.

Example 10. As a simple example where the equations are no associativity and commutativity axioms, consider $\mathcal{E} = \{\mathsf{f}(\mathsf{f}(x)) \approx \mathsf{f}(x)\}$ (i.e., $\mathcal{E}$ states that f is idempotent) and $\mathcal{R} = \{\mathsf{f}(\mathsf{s}(y)) \to \mathsf{f}(y)\}$. There is only one non-variable proper subterm of a term in $\mathcal{E}$ which unifies with the left-hand side $\mathsf{f}(\mathsf{s}(y))$ of the rule (viz. $\mathsf{f}(x)$). The (minimal) complete set of $\mathcal{E}$-unifiers consists of $\{x/\mathsf{s}(y)\}$ and $\{x/\mathsf{f}(\mathsf{s}(y))\}$ (all other unifiers $\{x/\mathsf{f}^n(\mathsf{s}(y))\}$ for $n \geq 2$ are subsumed by the second $\mathcal{E}$-unifier). This would yield the new rules $\mathsf{f}(\mathsf{f}(\mathsf{s}(y))) \to \mathsf{f}(\mathsf{f}(y))$ and $\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{s}(y)))) \to \mathsf{f}(\mathsf{f}(\mathsf{f}(y)))$. However, the first rule does not have to be included in $Ext_\mathcal{E}(\mathcal{R})$, because the corresponding other term of the equation, $\mathsf{f}(\mathsf{s}(y))$, reduces at the top position to $\mathsf{f}(y)$ which is $\mathcal{E}$-equivalent to $\mathsf{f}(\mathsf{f}(y))$ and the second rule is not included either for a similar reason. Thus, we may choose $Ext_\mathcal{E}(\mathcal{R}) = \mathcal{R}$.

Example 11. As another example, consider the following system from [19].

$$\mathcal{R} = \{ \begin{array}{l} x - \mathsf{0} \to x, \\ \mathsf{s}(x) - \mathsf{s}(y) \to x - y, \\ \mathsf{0} \div \mathsf{s}(y) \to \mathsf{0}, \\ \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x-y) \div \mathsf{s}(y)) \} \end{array} \qquad \mathcal{E} = \{(u \div v) \div w \approx (u \div w) \div v\}$$

By overlapping the subterm $u \div w$ in the right-hand side of the equation with the left-hand sides of the last two rules we obtain

$$\begin{array}{rl} Ext_\mathcal{E}(\mathcal{R}) = \mathcal{R} \cup \{ & (\mathsf{0} \div \mathsf{s}(y)) \div z \to \mathsf{0} \div z, \\ & (\mathsf{s}(x) \div \mathsf{s}(y)) \div z \to \mathsf{s}((x-y) \div \mathsf{s}(y)) \div z \ \}. \end{array}$$

Note that these are indeed all the rules of $Ext_\mathcal{E}(\mathcal{R})$. Overlapping the subterm $u \div v$ of the equation's left-hand side with the third rule would result in $(\mathsf{0} \div \mathsf{s}(y)) \div z' \to \mathsf{0} \div z'$. But this new rule does not have to be included in $Ext_\mathcal{E}(\mathcal{R})$, since the corresponding other term of the equation, $(\mathsf{0} \div z') \div \mathsf{s}(y)$, would $\to^\epsilon_{\mathcal{E} \setminus Ext_\mathcal{E}(\mathcal{R})}$-reduce with the rule $(\mathsf{0} \div \mathsf{s}(y)) \div z \to \mathsf{0} \div z$ to $\mathsf{0} \div z'$. Overlapping $u \div v$ with the left-hand side of the fourth rule is also superfluous.

Similarly, overlaps with the new rules $(0 \div \mathsf{s}(y)) \div z \to 0 \div z$ or $(\mathsf{s}(x) \div \mathsf{s}(y)) \div z \to \mathsf{s}((x - y) \div \mathsf{s}(y)) \div z$ also do not give rise to additional rules in $Ext_{\mathcal{E}}(\mathcal{R})$. To see this, overlap the subterm $u \div w$ in the right-hand side of the equation with the left-hand side of $(0 \div \mathsf{s}(y)) \div z \to 0 \div z$. This gives the rule $((0 \div \mathsf{s}(y)) \div z) \div z' \to (0 \div z) \div z'$. However, the corresponding other term of the equation is $((0 \div \mathsf{s}(y)) \div z') \div z$. This reduces at position 1 (or position 11) to $(0 \div z') \div z$, which is $\mathcal{E}$-equivalent to $(0 \div z) \div z'$. Overlaps with the other new rule $(\mathsf{s}(x) \div \mathsf{s}(y)) \div z \to \mathsf{s}((x - y) \div \mathsf{s}(y)) \div z$ are not needed either.

Nevertheless, the above algorithm for computing extensions does not always terminate. For example, for $\mathcal{R} = \{\mathsf{a}(x) \to \mathsf{c}(x)\}$, $\mathcal{E} = \{\mathsf{a}(\mathsf{b}(\mathsf{a}(x))) \approx \mathsf{b}(\mathsf{a}(\mathsf{b}(x)))\}$, it can be shown that all extensions $Ext_{\mathcal{E}}(\mathcal{R})$ are infinite.

We prove below that $Ext_{\mathcal{E}}(\mathcal{R})$ (according to Def. 8) has the desired property needed to reduce rewriting modulo equations to $\mathcal{E}$-extended rewriting. The following important lemma states that whenever s rewrites to t with $\to_{\mathcal{R}/\mathcal{E}}$ modulo $\mathcal{E}$, then s also rewrites with $\to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})}$ to a term which is $\mathcal{E}$-equivalent to t.[8]

Lemma 12 (Connection between $\to_{\mathcal{R}/\mathcal{E}}$ and $\to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})}$). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ be a set of equations with identical unique variables. If $s \to_{\mathcal{R}/\mathcal{E}} t$, then there exists a term $t' \sim_{\mathcal{E}} t$ such that $s \to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})} t'$.*

Proof. Let $s \to_{\mathcal{R}/\mathcal{E}} t$, i.e., there exist terms $s_0, \ldots, s_n, p$ with $n \geq 0$ such that $s = s_n \vdash_{\mathcal{E}} s_{n-1} \vdash_{\mathcal{E}} \ldots \vdash_{\mathcal{E}} s_0 \to_{\mathcal{R}} p \sim_{\mathcal{E}} t$. For the lemma, it suffices to show that there is a $t' \sim_{\mathcal{E}} p$ such that $s \to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})} t'$, since $t' \sim_{\mathcal{E}} p$ implies $t' \sim_{\mathcal{E}} t$.

We perform induction on n. If $n = 0$, we have $s = s_n = s_0 \to_{\mathcal{R}} p$. This implies $s \to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})} p$ since $\mathcal{R} \subseteq Ext_{\mathcal{E}}(\mathcal{R})$. So with $t' = p$ the claim is proved.

If $n > 0$, the induction hypothesis implies $s = s_n \vdash_{\mathcal{E}} s_{n-1} \to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})} t'$ such that $t' \sim_{\mathcal{E}} p$. So there exists an equation $u \approx v$ or $v \approx u$ from $\mathcal{E}$ and a rule $l \to r$ from $Ext_{\mathcal{E}}(\mathcal{R})$ such that $s|_\tau = u\delta$, $s_{n-1} = s[v\delta]_\tau$, $s_{n-1}|_\xi \sim_{\mathcal{E}} l\delta$, and $t' = s_{n-1}[r\delta]_\xi$ for positions τ and ξ and a substitution δ. We can use the same substitution δ for instantiating the equation $u \approx v$ (or $v \approx u$) and the rule $l \to r$, since equations and rules are assumed variable disjoint. We now perform a case analysis depending on the relationship of the positions τ and ξ.

Case 1: $\tau = \xi\pi$ for some π. In this case, we have $s|_\xi = s|_\xi[u\delta]_\pi \vdash_{\mathcal{E}} s|_\xi[v\delta]_\pi = s_{n-1}|_\xi \sim_{\mathcal{E}} l\delta$. This implies $s \to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})} s[r\delta]_\xi = s_{n-1}[r\delta]_\xi = t'$, as desired.

Case 2: $\tau \perp \xi$. Now we have $s|_\xi = s_{n-1}|_\xi \sim_{\mathcal{E}} l\delta$ and thus, $s \to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})} s[r\delta]_\xi = s[r\delta]_\xi[u\delta]_\tau \vdash_{\mathcal{E}} s[r\delta]_\xi[v\delta]_\tau = s[v\delta]_\tau[r\delta]_\xi = s_{n-1}[r\delta]_\xi = t'$.

Case 3: $\xi = \tau\pi$ for some π. Thus, $(v\delta)|_\pi \sim_{\mathcal{E}} l\delta$. We distinguish two sub-cases.

Case 3.1: $u\delta \to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})} q \sim_{\mathcal{E}} (v[r]_\pi)\delta$ for some term q. This implies $s = s[u\delta]_\tau \to_{\mathcal{E} \setminus Ext_{\mathcal{E}}(\mathcal{R})} s[q]_\tau \sim_{\mathcal{E}} s[v[r]_\pi\delta]_\tau = (s[v\delta]_\tau)[r\delta]_\xi = s_{n-1}[r\delta]_\xi = t'$.

[8] Our extension $Ext_{\mathcal{E}}$ has some similarities to the construction of contexts in [24]. However, in contrast to [24] we also consider the rules of $\mathcal{R}'$ in Condition (b) of Def. 8 in order to reduce the number of rules in $Ext_{\mathcal{E}}$. Moreover, in [24] equations may also be non-linear (and thus, Lemma 12 does not hold there).

Case 3.2: Otherwise. First assume that $\pi = \pi_1\pi_2$ where $v|_{\pi_1}$ is a variable x. Hence, $(v\delta)|_\pi = \delta(x)|_{\pi_2}$. Let $\delta'(y) = \delta(y)$ for $y \neq x$ and let $\delta'(x) = \delta(x)[r\delta]_{\pi_2}$. Since $u \approx v$ (or $v \approx u$) is an equation with identical unique variables, x also occurs in u at some position π'. This implies $u\delta|_{\pi'\pi_2} = \delta(x)|_{\pi_2} \sim_\mathcal{E} l\delta \to_{Ext_\mathcal{E}(\mathcal{R})} r\delta$. Hence, we obtain $u\delta \to^{\pi'\pi_2}_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} u\delta[r\delta]_{\pi'\pi_2} = u\delta' \sim_\mathcal{E} v\delta' = (v[r]_\pi)\delta$ in contradiction to the condition of Case 3.2.

Hence, π is a position of v and $v|_\pi$ is not a variable. Thus, $(v\delta)|_\pi = v|_\pi\delta \sim_\mathcal{E} l\delta$. Since rules and equations are assumed variable disjoint, the subterm $v|_\pi$ $\mathcal{E}$-unifies with l. Thus, there exists a $\sigma \in uni_\mathcal{E}(v|_\pi, l)$ such that $\delta \sim_\mathcal{E} \sigma\rho$.

Due to the Condition (b) of Def. 8, there is a term q' such that $u\sigma \to^{\pi'}_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} q' \sim_\mathcal{E} (v[r]_\pi)\sigma$. Since π' is a position in u, we have $u|_{\pi'}\sigma \sim_\mathcal{E} \circ \to_{Ext_\mathcal{E}(\mathcal{R})} q''$, where $q' = u\sigma[q'']_{\pi'}$. This also implies $u|_{\pi'}\delta \sim_\mathcal{E} u|_{\pi'}\sigma\rho \sim_\mathcal{E} \circ \to_{Ext_\mathcal{E}(\mathcal{R})} q''\rho$, and thus $u\delta \to^{\pi'}_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} u\delta[q''\rho]_{\pi'} \sim_\mathcal{E} u\sigma[q'']_{\pi'}\rho = q'\rho \sim_\mathcal{E} (v[r]_\pi)\sigma\rho \sim_\mathcal{E} (v[r]_\pi)\delta$. This is a contradiction to the condition of Case 3.2. □

The following theorem shows that $Ext_\mathcal{E}$ indeed has the desired property.

Theorem 13 (Termination of $\mathcal{R}/\mathcal{E}$ by $\mathcal{E}$-Extended Rewriting). *Let $\mathcal{R}$ be a TRS, let $\mathcal{E}$ be a set of equations with identical unique variables, and let t be a term. Then t does not start an infinite $\to_{\mathcal{R}/\mathcal{E}}$-reduction iff t does not start an infinite $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$-reduction. So in particular, $\mathcal{R}$ is terminating modulo $\mathcal{E}$ (i.e., $\to_{\mathcal{R}/\mathcal{E}}$ is well founded) iff $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$ is well founded.*

Proof. The "only if" direction is straightforward because $\to_{Ext_\mathcal{E}(\mathcal{R})} = \to_\mathcal{R}$ and therefore, $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} \subseteq \to_{Ext_\mathcal{E}(\mathcal{R})/\mathcal{E}} = \to_{\mathcal{R}/\mathcal{E}}$.
For the "if" direction, assume that t starts an infinite $\to_{\mathcal{R}/\mathcal{E}}$-reduction

$$t = t_0 \to_{\mathcal{R}/\mathcal{E}} t_1 \to_{\mathcal{R}/\mathcal{E}} t_2 \to_{\mathcal{R}/\mathcal{E}} \dots$$

For every $i \in \mathbb{N}$, let f_{i+1} be a function from terms to terms such that for every $t'_i \sim_\mathcal{E} t_i$, $f_{i+1}(t'_i)$ is a term $\mathcal{E}$-equivalent to t_{i+1} such that $t'_i \to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} f_{i+1}(t'_i)$. These functions f_{i+1} must exist due to Lemma 12, since $t'_i \sim_\mathcal{E} t_i$ and $t_i \to_{\mathcal{R}/\mathcal{E}} t_{i+1}$ implies $t'_i \to_{\mathcal{R}/\mathcal{E}} t_{i+1}$. Hence, t starts an infinite $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$-reduction:

$$t \to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} f_1(t) \to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} f_2(f_1(t)) \to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} f_3(f_2(f_1(t))) \to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} \dots \quad \square$$

5 Dependency Pairs for Rewriting Modulo Equations

In this section we finally extend the dependency pair approach to rewriting modulo equations: To show that $\mathcal{R}$ modulo $\mathcal{E}$ terminates, one first constructs the extension $Ext_\mathcal{E}(\mathcal{R})$ of $\mathcal{R}$. Subsequently, dependency pairs can be used to prove well-foundedness of $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$ (which is equivalent to termination of $\mathcal{R}$ modulo $\mathcal{E}$). The idea for the extension of the dependency pair approach is simply to modify Thm. 3 as follows.

1. The equations should be satisfied by the equivalence $\sim$ corresponding to the quasi-ordering $\succsim$, i.e., we demand $u \sim v$ for all equations $u \approx v$ in $\mathcal{E}$.

2. A similar requirement is needed for equations $u \approx v$ when the root symbols of u and v are replaced by the corresponding tuple symbols. We denote tuples of terms $s_1, \ldots, s_n$ by $\boldsymbol{s}$ and for any term $t = f(\boldsymbol{s})$ with a defined root symbol f, let $t^\sharp$ be the term $F(\boldsymbol{s})$. Hence, we also have to demand $u^\sharp \sim v^\sharp$.
3. The notion of "defined symbols" must be changed accordingly. As before, all root symbols of left-hand sides of rules are regarded as being defined, but if there is an equation $f(\boldsymbol{u}) = g(\boldsymbol{v})$ in $\mathcal{E}$ and f is defined, then g must be considered defined as well, as otherwise we would not be able to trace the redex in a reduction by only regarding subterms with defined root symbols.

Definition 14 (Defined Symbols for Rewriting Modulo Equations). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ be a set of equations. Then the set of* defined *symbols $\mathcal{D}$ of $\mathcal{R}/\mathcal{E}$ is the smallest set such that $\mathcal{D} = \{root(l) \mid l \to r \in \mathcal{R}\} \cup \{root(v) \mid u \approx v \in \mathcal{E}$ or $v \approx u \in \mathcal{E}$, $root(u) \in \mathcal{D}\}$.*

The constraints of the dependency pair approach as sketched above are not yet sufficient for termination of $\to_{\mathcal{E}\setminus\mathcal{R}}$ as the following example illustrates.

Example 15. Consider $\mathcal{R} = \{\mathsf{f}(x) \to \mathsf{s}(x)\}$ and $\mathcal{E} = \{\mathsf{f}(\mathsf{a}) \approx \mathsf{a}\}$. There is no dependency pair in this example and thus, the only constraints would be $\mathsf{f}(x) \gtrsim \mathsf{s}(x)$, $\mathsf{f}(\mathsf{a}) \sim \mathsf{a}$, and $\mathsf{F}(\mathsf{a}) \sim \mathsf{A}$. Obviously, these constraints are satisfiable (by using an equivalence relation $\sim$ where all terms are equal). However, $\to_{\mathcal{E}\setminus\mathcal{R}}$ is not terminating since we have $\mathsf{a} \vdash_{\mathcal{E}} \mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{s}(\mathsf{a}) \vdash_{\mathcal{E}} \mathsf{s}(\mathsf{f}(\mathsf{a})) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(\mathsf{a})) \vdash_{\mathcal{E}} \ldots$

The soundness of the dependency pair approach for ordinary rewriting (Thm. 3) relies on the fact that an infinite reduction from a minimal non-terminating term can be achieved by applying only normalized instantiations of $\mathcal{R}$-rules. But for $\mathcal{E}$-extended rewriting (or full rewriting modulo equations), this is not true any more. For instance, the minimal non-terminating subterm a in Ex. 15 is first modified by applying an $\mathcal{E}$-equation (resulting in $\mathsf{f}(\mathsf{a})$) and then an $\mathcal{R}$-rule is applied whose variable is instantiated with the non-terminating term a. Hence, the problem is that the new minimal non-terminating subterm a which results from application of the $\mathcal{R}$-rule does not correspond to the right-hand side of a dependency pair, because this minimal non-terminating subterm is completely inside the instantiation of a *variable* of the $\mathcal{R}$-rule. With ordinary rewriting, this situation can never occur.

In Ex. 15, the problem can be avoided by adding a suitable *instance* of the rule $\mathsf{f}(x) \to \mathsf{s}(x)$ (viz. $\mathsf{f}(\mathsf{a}) \to \mathsf{s}(\mathsf{a})$) to $\mathcal{R}$, since this instance is used in the infinite reduction. Now there would be a dependency pair $\langle \mathsf{F}(\mathsf{a}), \mathsf{A} \rangle$ and with the additional constraint $\mathsf{F}(\mathsf{a}) > \mathsf{A}$ the resulting inequalities are no longer satisfiable.

Note that since in practice, we are interested in proving termination of $\to_{\mathcal{R}/\mathcal{E}}$, we would of course first build the extension $Ext_{\mathcal{E}}(\mathcal{R})$ and then we would try to prove well-foundedness of $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$ instead of well-foundedness of $\to_{\mathcal{E}\setminus\mathcal{R}}$. Note that for the system of Ex. 15 we obtain $Ext_{\mathcal{E}}(\mathcal{R}) = \{\mathsf{f}(x) \to \mathsf{s}(x), \mathsf{f}(\mathsf{f}(\mathsf{a})) \to \mathsf{f}(\mathsf{s}(\mathsf{a}))\}$, because the subterm a of $\mathsf{f}(\mathsf{a})$ $\mathcal{E}$-unifies with $\mathsf{f}(x)$ (using the $\mathcal{E}$-unifier $\sigma = \{x/\mathsf{a}\}$). The (minimal) complete set of $\mathcal{E}$-unifiers only consists of σ, since all other $\mathcal{E}$-unifiers $\delta = \{x/\mathsf{f}^n(\mathsf{a})\}$ satisfy $\delta(x) \sim_{\mathcal{E}} \sigma(x)$. Thus, a system like the

one of Ex. 15 would never be tested for $\to_{\mathcal{E}\setminus\mathcal{R}}$-termination in practice, since one would always extend $\mathcal{R}$ first and for $Ext_{\mathcal{E}}(\mathcal{R})$, the dependency pair approach would no longer falsely conclude termination.

Thus, one might be tempted to assume that the problem of Ex. 15 never occurs in practice, because after extending $\mathcal{R}$ to $Ext_{\mathcal{E}}(\mathcal{R})$ the systems would always have a form where the "right instantiation" of the rules is already included. In other words, one might hope that although the dependency pair approach cannot be used to prove termination of $\to_{\mathcal{E}\setminus\mathcal{R}}$ in general, it would still be sound for TRSs $\mathcal{R}$ which result from the extension process of Sect. 4. Unfortunately, this is not true, as the following example shows.

Example 16. Let us modify the rule from Ex. 15 to $\mathsf{f}(x) \to x$, i.e., regard $\mathcal{R} = \{\mathsf{f}(x) \to x\}$ and $\mathcal{E} = \{\mathsf{f}(\mathsf{a}) \approx \mathsf{a}\}$. Now we may choose $Ext_{\mathcal{E}}(\mathcal{R}) = \mathcal{R}$: The subterm a of $v = \mathsf{f}(\mathsf{a})$ unifies with $\mathsf{f}(x)$ using the $\mathcal{E}$-unifier $\sigma = \{x/\mathsf{a}\}$ (where again $uni_{\mathcal{E}}(\mathsf{a}, \mathsf{f}(x)) = \{\sigma\}$ is a minimal complete set of $\mathcal{E}$-unifiers). But the rule $\mathsf{f}(\mathsf{f}(\mathsf{a})) \to \mathsf{f}(\mathsf{a})$ does not have to be added, since we have $u\sigma = \mathsf{a} \sim_{\mathcal{E}} \mathsf{f}(\mathsf{a}) \to^{\epsilon}_{\mathcal{R}} \mathsf{a} \sim_{\mathcal{E}} \mathsf{f}(\mathsf{a}) = (v[r]_{\pi})\sigma$. Obviously, $\to_{\mathcal{E}\setminus\mathcal{R}}$ is still non-terminating and there is still no dependency pair in this example.

The following definition shows how to add the right instantiations of the rules in $\mathcal{R}$ in order to allow a sound application of dependency pairs. As usual, a substitution ν is called a *variable renaming* iff the range of ν only contains variables and if $\nu(x) \neq \nu(y)$ for $x \neq y$.

Definition 17 (Adding Instantiations). *Given a TRS $\mathcal{R}$, a set $\mathcal{E}$ of equations, let $\mathcal{R}'$ be a set containing only rules of the form $l\sigma \to r\sigma$ (where σ is a substitution and $l \to r \in \mathcal{R}$). $\mathcal{R}'$ is an* instantiation of $\mathcal{R}$ for the equations $\mathcal{E}$ *iff*

(a) $\mathcal{R} \subseteq \mathcal{R}'$,
(b) for all $l \to r \in \mathcal{R}$, all $u \approx v \in \mathcal{E}$ and $v \approx u \in \mathcal{E}$, and all $\sigma \in uni_{\mathcal{E}}(v, l)$, there exists a rule $l' \to r' \in \mathcal{R}'$ and a variable renaming ν such that $l\sigma \sim_{\mathcal{E}} l'\nu$ and $r\sigma \sim_{\mathcal{E}} r'\nu$.

In the following, let $Ins_{\mathcal{E}}(\mathcal{R})$ always denote an instantiation of $\mathcal{R}$ for $\mathcal{E}$.

Unlike extensions $Ext_{\mathcal{E}}(\mathcal{R})$, instantiations $Ins_{\mathcal{E}}(\mathcal{R})$ are never infinite if $\mathcal{R}$ and $\mathcal{E}$ are finite and if $uni_{\mathcal{E}}(v, l)$ is always finite (i.e., they are not defined via a fixpoint construction). In fact, one might even demand that for all $l \to r \in \mathcal{R}$, all equations, and all σ from the corresponding complete set of $\mathcal{E}$-unifiers, $Ins_{\mathcal{E}}(\mathcal{R})$ should contain $l\sigma \to r\sigma$. The condition that it is enough if some $\mathcal{E}$-equivalent variable-renamed rule is already contained in $Ins_{\mathcal{E}}(\mathcal{R})$ is only added for efficiency considerations in order to reduce the number of rules in $Ins_{\mathcal{E}}(\mathcal{R})$. Even without this condition, $Ins_{\mathcal{E}}(\mathcal{R})$ would still be finite and all the following theorems would hold as well.

However, the above instantiation technique only serves its purpose if there are no collapsing equations (i.e., no equations $u \approx v$ or $v \approx u$ with $v \in \mathcal{V}$).

Example 18. Consider $\mathcal{R} = \{\mathsf{f}(x) \to x\}$ and $\mathcal{E} = \{\mathsf{f}(x) \approx x\}$. Note that $Ins_{\mathcal{E}}(\mathcal{R}) = \mathcal{R}$. Although $\to_{\mathcal{E}\setminus\mathcal{R}}$ is clearly not terminating, the dependency pair approach would falsely prove termination of $\to_{\mathcal{E}\setminus\mathcal{R}}$, since there is no dependency pair.

For non-collapsing equations, the construction used to build $Ins_{\mathcal{E}}(\mathcal{R})$ leads to the desired property: Whenever we have a terminating or a minimal non-terminating term which is $\mathcal{E}$-equivalent to an instantiated left-hand side $l\delta$ of a rule $l \to r \in \mathcal{R}$, there exists a corresponding rule $l' \to r'$ in $Ins_{\mathcal{E}}(\mathcal{R})$, such that $l'\rho \sim_{\mathcal{E}} l\delta$ and ρ only instantiates the variables of l' with terminating terms.
Now we can present the main result of the paper.

Theorem 19 (Termination of Equational Rewriting using Dependency Pairs). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ be a set of non-collapsing equations with identical unique variables. $\mathcal{R}$ is terminating modulo $\mathcal{E}$ (i.e., $\to_{\mathcal{R}/\mathcal{E}}$ is well founded) if there exists a weakly monotonic quasi-ordering $\succsim$ and a well-founded ordering $>$ compatible with $\succsim$ where both $\succsim$ and $>$ are closed under substitution, such that*

(1) $s > t$ for all dependency pairs $\langle s, t\rangle$ of $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R}))$,
(2) $l \succsim r$ for all rules $l \to r$ of $\mathcal{R}$,
(3) $u \sim v$ for all equations $u \approx v$ of $\mathcal{E}$, and
(4) $u^\sharp \sim v^\sharp$ for all equations $u \approx v$ of $\mathcal{E}$ where $root(u)$ and $root(v)$ are defined.

Proof. Suppose that there is a term t_0 with an infinite $\to_{\mathcal{R}/\mathcal{E}}$-reduction. Thm. 13 implies that t_0 also has an infinite $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-reduction. By a minimality argument, $t_0 = C[t_0']$, where t_0' is an *minimal non-terminating* term (i.e., t_0' is non-terminating, but all its subterms only have finite $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-reductions). We will show that there exists a term t_1 with $t_0 \to^+_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} t_1$, t_1 contains a minimal non-terminating subterm t_1', and $t_0'^\sharp \succsim \circ > t_1'^\sharp$. By repeated application of this construction we obtain an infinite sequence $t_0 \to^+_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} t_1 \to^+_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} t_2 \to^+_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} \ldots$ such that $t_0'^\sharp \succsim \circ > t_1'^\sharp \succsim \circ > t_2'^\sharp \succsim \circ > \ldots$. This, however, is a contradiction to the well-foundedness of $>$.

Let t_0' have the form $f(\boldsymbol{u})$. In the infinite $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-reduction of $f(\boldsymbol{u})$, first some $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-steps may be applied to $\boldsymbol{u}$ which yields new terms $\boldsymbol{v}$. Note that due to the definition of $\mathcal{E}$-extended rewriting, in these reductions, no $\mathcal{E}$-steps can be applied outside of $\boldsymbol{u}$. Due to the termination of $\boldsymbol{u}$, after a finite number of those steps, an $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-step must be applied on the root position of $f(\boldsymbol{v})$.

Thus, there exists a rule $l \to r \in Ext_{\mathcal{E}}(\mathcal{R})$ such that $f(\boldsymbol{v}) \sim_{\mathcal{E}} l\alpha$ and hence, the reduction yields $r\alpha$. Now the infinite $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-reduction continues with $r\alpha$, i.e., the term $r\alpha$ starts an infinite $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-reduction, too. So up to now the reduction has the following form (where $\to_{Ext_{\mathcal{E}}(\mathcal{R})}$ equals $\to_{\mathcal{R}}$):

$$t_0 = C[f(\boldsymbol{u})] \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} C[f(\boldsymbol{v})] \sim_{\mathcal{E}} C[l\alpha] \to_{Ext_{\mathcal{E}}(\mathcal{R})} C[r\alpha].$$

We perform a case analysis depending on the positions of $\mathcal{E}$-steps in $f(\boldsymbol{v}) \sim_{\mathcal{E}} l\alpha$.

First consider the case where all $\mathcal{E}$-steps in $f(\boldsymbol{v}) \sim_{\mathcal{E}} l\alpha$ take place below the root. Then we have $l = f(\boldsymbol{w})$ and $\boldsymbol{v} \sim_{\mathcal{E}} \boldsymbol{w}\alpha$. Let $t_1 := C[r\alpha]$. Note that $\boldsymbol{v}$ do not start infinite $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-reductions and by Thm. 13, they do not start infinite $\to_{\mathcal{R}/\mathcal{E}}$-reductions either. But then $\boldsymbol{w}\alpha$ also cannot start infinite $\to_{\mathcal{R}/\mathcal{E}}$-reductions and therefore they also do not start infinite $\to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})}$-reductions. This implies that for all variables x occurring in $f(\boldsymbol{w})$ the terms $\alpha(x)$ are terminating. Thus,

since $r\alpha$ starts an infinite reduction, there occurs a non-variable subterm s in r, such that $t_1' := s\alpha$ is a minimal non-terminating term. Since $\langle l^\sharp, s^\sharp \rangle$ is a dependency pair, we obtain $t_0'^\sharp = F(\boldsymbol{u}) \gtrsim F(\boldsymbol{v}) \sim l^\sharp\alpha > s^\sharp\alpha = t_1'^\sharp$. Here, $F(\boldsymbol{u}) \gtrsim F(\boldsymbol{v})$ holds since $\boldsymbol{u} \to^*_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} \boldsymbol{v}$ and since $l \gtrsim r$ for every rule $l \to r \in Ext_\mathcal{E}(\mathcal{R})$.

Now we consider the case where there are $\mathcal{E}$-steps in $f(\boldsymbol{v}) \sim_\mathcal{E} l\alpha$ at the root position. Thus we have $f(\boldsymbol{v}) \sim_\mathcal{E} f(\boldsymbol{q}) \vdash_\mathcal{E} p \sim_\mathcal{E} l\alpha$, where $f(\boldsymbol{q}) \vdash_\mathcal{E} p$ is the first $\mathcal{E}$-step at the root position. In other words, there is an equation $u \approx v$ or $v \approx u$ in $\mathcal{E}$ such that $f(\boldsymbol{q})$ is an instantiation of v.

Note that since $\boldsymbol{v} \sim_\mathcal{E} \boldsymbol{q}$, the terms $\boldsymbol{q}$ only have finite $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$-reductions (the argumentation is similar as in the first case). Let δ be the substitution which operates like α on the variables of l and which yields $v\delta = f(\boldsymbol{q})$. Thus, δ is an $\mathcal{E}$-unifier of l and v. Since l is $\mathcal{E}$-unifiable with v, there also exists a corresponding complete $\mathcal{E}$-unifier σ from $uni_\mathcal{E}(l, v)$. Thus, there is also a substitution ρ such that $\delta \sim_\mathcal{E} \sigma\rho$. As l is a left-hand side of a rule from $Ext_\mathcal{E}(\mathcal{R})$, there is a rule $l' \to r'$ in $Ins_\mathcal{E}(Ext_\mathcal{E}(\mathcal{R}))$ and a variable renaming ν such that $l\sigma \sim_\mathcal{E} l'\nu$ and $r\sigma \sim_\mathcal{E} r'\nu$.

Hence, $v\sigma\rho \sim_\mathcal{E} v\delta = f(\boldsymbol{q})$, $l'\nu\rho \sim_\mathcal{E} l\sigma\rho \sim_\mathcal{E} l\delta = l\alpha$, and $r'\nu\rho \sim_\mathcal{E} r\sigma\rho \sim_\mathcal{E} r\delta = r\alpha$. So instead we now consider the following reduction (where $\to_{Ins_\mathcal{E}(Ext_\mathcal{E}(\mathcal{R}))}$ equals $\to_\mathcal{R}$):

$$t_0 = C[f(\boldsymbol{u})] \to^*_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})} C[f(\boldsymbol{v})] \sim_\mathcal{E} C[l'\nu\rho] \to_{Ins_\mathcal{E}(Ext_\mathcal{E}(\mathcal{R}))} C[r'\nu\rho] = t_1.$$

Since all proper subterms of $v\delta$ only have finite $\to_{\mathcal{R}/\mathcal{E}}$-reductions, for all variables x of $l'\nu$, the term $x\rho$ only has finite $\to_{\mathcal{R}/\mathcal{E}}$-reductions and hence, also only finite $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$-reductions. To see this, note that since all equations have identical unique variables, $v\sigma \sim_\mathcal{E} l\sigma \sim_\mathcal{E} l'\nu$ implies that all variables of $l'\nu$ also occur in $v\sigma$. Thus, if x is a variable from $l'\nu$, then there exists a variable y in v such that x occurs in $y\sigma$. Since $\mathcal{E}$ does not contain collapsing equations, y is a proper subterm of v and thus, $y\delta$ is a proper subterm of $v\delta$. As all proper subterms of $v\delta$ only have finite $\to_{\mathcal{R}/\mathcal{E}}$-reductions, this implies that $y\delta$ only has finite $\to_{\mathcal{R}/\mathcal{E}}$-reductions, too. But then, since $y\delta \sim_\mathcal{E} y\sigma\rho$, the term $y\sigma\rho$ only has finite $\to_{\mathcal{R}/\mathcal{E}}$-reductions, too. Then this also holds for all subterms of $y\sigma\rho$, i.e., all $\to_{\mathcal{R}/\mathcal{E}}$-reductions of $x\rho$ are also finite.

So for all variables x of l', $x\nu\rho$ only has finite $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$-reductions. (Note that this only holds because ν is just a variable renaming.) Since $r\alpha$ starts an infinite $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$-reduction, $r'\nu\rho \sim_\mathcal{E} r\alpha$ must start an infinite $\to_{\mathcal{R}/\mathcal{E}}$-reduction (and hence, an infinite $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$-reduction) as well. As for all variables x of r', $x\nu\rho$ is $\to_{\mathcal{E}\setminus Ext_\mathcal{E}(\mathcal{R})}$-terminating, there must be a non-variable subterm s of r', such that $t_1' := s\nu\rho$ is a minimal non-terminating term. As $\langle l'^\sharp, s^\sharp \rangle$ is a dependency pair, we obtain $t_0'^\sharp = F(\boldsymbol{u}) \gtrsim F(\boldsymbol{v}) \sim l'^\sharp\nu\rho > s^\sharp\nu\rho = t_1'^\sharp$. Here, $F(\boldsymbol{v}) \sim_\mathcal{E} l'^\sharp\nu\rho$ is a consequence of Condition (4). □

To summarize, our procedure to prove termination of $\mathcal{R}$ modulo $\mathcal{E}$ is described by the following corollary.

Corollary 20 (Proving Termination of Rewriting Modulo Equations). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ be a set of non-collapsing equations with identical unique variables. Then the following algorithm is sound:*

1.Let C be the set of the following constraints:
- *$s > t$ for all dependency pairs $\langle s, t\rangle$ of $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R}))$,*
- *$l \succsim r$ for all rules $l \to r$ of $\mathcal{R}$,*
- *$u \sim v$ for all equations $u \approx v$ of $\mathcal{E}$, and*
- *$u^\sharp \sim v^\sharp$ for all equations $u \approx v$ of $\mathcal{E}$ where $root(u)$ and $root(v)$ are defined*

Here, $Ext_{\mathcal{E}}$ is computed by the algorithm on p. 8.

2.Eliminate arguments of function symbols
(i.e., normalize the terms in C by rules of the form
$f(x_1, \ldots, x_n) \to f'(x_{i_1}, \ldots, x_{i_m})$ where $1 \le i_1 < \ldots < i_m \le n$ or
$f(x_1, \ldots, x_n) \to x_i$ where $1 \le i \le n$)

3.Use standard techniques to find suitable relations satisfying C.
In case of success, $\mathcal{R}$ is terminating modulo $\mathcal{E}$.

Example 21. Regard the system from Ex. 10 again. Here, we had $Ext_{\mathcal{E}}(\mathcal{R}) = \mathcal{R}$ and we also have $Ins_{\mathcal{E}}(\mathcal{R}) = \mathcal{R}$. By Thm. 19 or Cor. 20, we obtain the following constraints for the dependency pair approach:

$$\begin{aligned} \mathsf{F}(\mathsf{s}(y)) &> \mathsf{F}(y) \\ \mathsf{f}(\mathsf{s}(y)) &\succsim \mathsf{f}(y) \\ \mathsf{f}(\mathsf{f}(x)) &\sim \mathsf{f}(x) \\ \mathsf{F}(\mathsf{f}(x)) &\sim \mathsf{F}(x) \end{aligned}$$

As explained in Sect. 2, one may first eliminate arguments of function symbols or replace function symbols by one of their arguments before searching for suitable orderings. By replacing f by its argument (i.e., by normalizing the terms w.r.t. the rule $\mathsf{f}(x) \to x$), these constraints are transformed into

$$\begin{aligned} \mathsf{F}(\mathsf{s}(y)) &> \mathsf{F}(y) \\ \mathsf{s}(y) &\succsim y \\ x &\sim x \\ \mathsf{F}(x) &\sim \mathsf{F}(x). \end{aligned}$$

These inequalities are satisfied by the recursive path ordering. □

Example 22. Similarly, termination of the division-system (Ex. 11) can also be proved by dependency pairs. In [19], Ohsaki, Middeldorp, and Giesl developed a new extension of the semantic labelling technique [25] to rewriting modulo equations. This example was used to demonstrate the power of their method, because with their new definition of equational semantic labelling one can prove termination of this system, whereas the original definition of Zantema [25] fails

here. However, semantic labelling is a technique designed for *manual* termination proofs (and it is also useful as a proof technique for correctness proofs of other methods). In contrast, with the method of the present paper, one can now verify termination of this example *automatically* for the first time.

Here we have $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R})) = Ext_{\mathcal{E}}(\mathcal{R})$ and thus, the resulting constraints are

$$\begin{aligned}
\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{M}(x,y),\\
\mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{M}(x,y),\\
\mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{Q}(x-y,\mathsf{s}(y))\\
\mathsf{Q}(\mathsf{0}\div\mathsf{s}(y),z) &> \mathsf{Q}(\mathsf{0},z)\\
\mathsf{Q}(\mathsf{s}(x)\div\mathsf{s}(y),z) &> \mathsf{M}(x,y)\\
\mathsf{Q}(\mathsf{s}(x)\div\mathsf{s}(y),z) &> \mathsf{Q}(x-y,\mathsf{s}(y))\\
\mathsf{Q}(\mathsf{s}(x)\div\mathsf{s}(y),z) &> \mathsf{Q}(\mathsf{s}((x-y)\div\mathsf{s}(y)),z)
\end{aligned}$$

as well as $l \gtrsim r$ for all rules $l \to r$, $(u \div v) \div w \sim (u \div w) \div v$, and $\mathsf{Q}(u \div v, w) \sim \mathsf{Q}(u \div w, v)$. (Here, M and Q are the tuple symbols for the minus-symbol "$-$" and the quot-symbol "$\div$".) In this example we will eliminate the second arguments of $-$, $\div$, M, and Q (i.e., every term $s-t$ is replaced by $-'(s)$, etc.). Then the resulting inequalities are satisfied by the rpo with the precedence $\div' \sqsupset \mathsf{s} \sqsupset -'$, $\mathsf{Q}' \sqsupset \mathsf{M}'$. Ex. 21 and Ex. 22 also demonstrate that by using dependency pairs, termination of equational rewriting can sometimes even be shown by *ordinary* base orderings (e.g., the ordinary rpo which on its own cannot be used for rewriting modulo equations).

6 Dependency Graphs for Rewriting Modulo Equations

In [1–3], Arts and Giesl presented a refinement of the dependency pair approach based on the observation that instead of considering all dependency pairs at the same time, it is advantageous to treat groups of dependency pairs separately. These groups correspond to *clusters* in the *dependency graph* of $\mathcal{R}$. One should remark that this refinement is only possible for finite TRSs $\mathcal{R}$. In this section we show how this refinement can also be used for rewriting modulo equations. This extension of dependency graphs to the equational setting is quite straightforward and similar to the extensions of the dependency graph refinement to the AC-case in [16, 18].

The nodes of the dependency graph are the dependency pairs and there is an arrow from node $\langle v^\sharp, w^\sharp\rangle$ to $\langle l^\sharp, t^\sharp\rangle$ if there exist substitutions σ_1 and σ_2 such that $w^\sharp\sigma_1 \to^*_{\mathcal{R}/\mathcal{E}} \circ \sim_{\mathcal{E}^\sharp} l^\sharp\sigma_2$. By renaming variables in different occurrences of dependency pairs we may assume that $\sigma_1 = \sigma_2$. Here, $\mathcal{E}^\sharp$ also contains the equations $u^\sharp \approx v^\sharp$ where $u \approx v$ is an equation from $\mathcal{E}$ whose roots are defined, i.e.

$$\mathcal{E}^\sharp = \mathcal{E} \cup \{u^\sharp \approx v^\sharp \mid u \approx v \in \mathcal{E},\ root(u) \text{ and } root(v) \text{ are defined }\}.$$

For the division-system (Ex. 11) we had seven dependency pairs.

$$\langle \mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)), \mathsf{M}(x,y)\rangle \quad (1)$$
$$\langle \mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)), \mathsf{M}(x,y)\rangle \quad (2)$$
$$\langle \mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)), \mathsf{Q}(x-y,\mathsf{s}(y))\rangle \quad (3)$$
$$\langle \mathsf{Q}(0 \div \mathsf{s}(y), z), \mathsf{Q}(0,z)\rangle \quad (4)$$
$$\langle \mathsf{Q}(\mathsf{s}(x) \div \mathsf{s}(y), z), \mathsf{M}(x,y)\rangle \quad (5)$$
$$\langle \mathsf{Q}(\mathsf{s}(x) \div \mathsf{s}(y), z), \mathsf{Q}(x-y,\mathsf{s}(y))\rangle \quad (6)$$
$$\langle \mathsf{Q}(\mathsf{s}(x) \div \mathsf{s}(y), z), \mathsf{Q}(\mathsf{s}((x-y) \div \mathsf{s}(y)), z)\rangle \quad (7)$$

The dependency graph contains an arrow from (1) to itself and arrows from (2) and (5) to (1). Moreover, from each of the pairs (3), (6), and (7) there are arrows to all pairs (2) - (7). The reason is that a term $\mathsf{Q}(\ldots)$ can only reduce w.r.t. $\to^*_{\mathcal{R}/\mathcal{E}} \circ \sim_{\mathcal{E}^\sharp}$ to terms with the root symbol Q and a term with the root symbol M can also just reduce to other terms built with M. Moreover, an instantiation of the right-hand side $\mathsf{Q}(0,z)$ of (4) can never reduce to any left-hand side of a dependency pair.

We call a non-empty subset $\mathcal{C}$ of dependency pairs a *cluster* if for every two (not necessarily distinct) pairs $\langle v^\sharp, w^\sharp\rangle$ and $\langle l^\sharp, t^\sharp\rangle$ in $\mathcal{C}$ there exists a non-empty path in $\mathcal{C}$ from $\langle v^\sharp, w^\sharp\rangle$ to $\langle l^\sharp, t^\sharp\rangle$. So the clusters in the division-example are $\{(1)\}$ and all non-empty subsets of $\{(3),(6),(7)\}$.

From the proof of Thm. 19 it is straightforward that for termination of $\mathcal{R}$ modulo $\mathcal{E}$ one can consider the clusters of dependency pairs separately and that for each cluster it is sufficient to find one dependency pair which is *strictly* decreasing (w.r.t. $>$), whereas the others only have to be weakly decreasing (w.r.t. $\gtrsim$). For the division-example this implies that we may use different orderings for the M- and the Q-clusters.

While in general the dependency graph cannot be computed automatically (since it is undecidable whether $w^\sharp\sigma \to^*_{\mathcal{R}/\mathcal{E}} \circ \sim_{\mathcal{E}^\sharp} l^\sharp\sigma$ holds for some σ), one can nevertheless approximate this graph automatically. The estimation is based on comparing the constructors of the terms in the dependency pairs. For any term w, let $\text{CAP}(w)$ result from replacing all *proper*[9] subterms of w that have a defined root symbol by different fresh variables and let $\text{REN}(w)$ result from replacing all occurrences of variables in w by different fresh variables. Then, to determine whether $w^\sharp\sigma \to^*_{\mathcal{R}/\mathcal{E}} \circ \sim_{\mathcal{E}^\sharp} l^\sharp\sigma$ holds for some σ, we check whether $\text{REN}(\text{CAP}(w))$ $\mathcal{E}$-unifies with l. In this case, we draw an arrow from $\langle v^\sharp, w^\sharp\rangle$ to $\langle l^\sharp, t^\sharp\rangle$ in the *estimated* dependency graph. Here, the function REN is needed to rename multiple occurrences of the same variable x, because for certain substitutions σ, two occurrences of $x\sigma$ could reduce to different terms.

For example, $\text{CAP}((x-y) \div \mathsf{s}(y)) = z \div \mathsf{s}(y)$, since the proper subterm $x-y$ has a defined root symbol. Moreover, $\text{REN}(z \div \mathsf{s}(y)) = z' \div \mathsf{s}(y')$. Since $z' \div \mathsf{s}(y')$

[9] We defined CAP slightly different from [1–3], because we only replace *proper* subterms with defined root symbols. The advantage is that in this way, CAP can be applied to terms without tuple symbols and hence, the estimated dependency graph can be computed by using $\mathcal{E}$-unification instead of $\mathcal{E}^\sharp$-unification.

$\mathcal{E}$-unifies with $\mathsf{s}(x) \div \mathsf{s}(y)$, we have to draw an arrow from (3) to (2) and to (3) itself. It turns out that in the division-example, the estimated dependency graph is identical to the real dependency graph, i.e., by the approximation above we can compute the dependency graph automatically.

In general, we obtain the following refined termination criterion, which can be checked mechanically.

Theorem 23 (Termination of Equational Rewriting with Dependency Graph). *Let $\mathcal{R}$ be a TRS and let $\mathcal{E}$ be a set of non-collapsing equations with identical unique variables. $\mathcal{R}$ is terminating modulo $\mathcal{E}$ if for every cluster in the estimated dependency graph of $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R}))$, there exists a weakly monotonic quasi-ordering $\succsim$ and a well-founded ordering $>$ compatible with $\succsim$ where both $\succsim$ and $>$ are closed under substitution, such that*

(1) $s > t$ for at least one dependency pair $\langle s, t\rangle$ of the cluster,
(1′) $s \succsim t$ for all other dependency pairs $\langle s, t\rangle$ of the cluster,
(2) $l \succsim r$ for all rules $l \to r$ of $\mathcal{R}$,
(3) $u \sim v$ for all equations $u \approx v$ of $\mathcal{E}$, and
(4) $u^\sharp \sim v^\sharp$ for all equations $u \approx v$ of $\mathcal{E}$ where $root(u)$ and $root(v)$ are defined.

Proof. We first show that the estimation of the dependency graph is correct, i.e., $w^\sharp\sigma \to^*_{\mathcal{R}/\mathcal{E}} \circ \sim_{\mathcal{E}^\sharp} l^\sharp\sigma$ implies that $\text{REN}(\text{CAP}(w))$ and l are $\mathcal{E}$-unifiable. By Lemma 12, $w^\sharp\sigma \to^*_{\mathcal{R}/\mathcal{E}} \circ \sim_{\mathcal{E}^\sharp} l^\sharp\sigma$ implies $w^\sharp\sigma \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} \circ \sim_{\mathcal{E}^\sharp} l^\sharp\sigma$.

Note that for any two terms t_1, t_2 with defined roots, ${t_1}^\sharp \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} {t_2}^\sharp$ implies that $\text{REN}(\text{CAP}(t_1))$ matches $\text{REN}(\text{CAP}(t_2))$. This can be proved by induction on the length of the reduction ${t_1}^\sharp \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} {t_2}^\sharp$. If ${t_1}^\sharp = {t_2}^\sharp$, we have $t_1 = t_2$ and thus, the claim is obvious. Otherwise, we have ${t_1}^\sharp \to_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} {t'_1}^\sharp \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} {t_2}^\sharp$. Thus, there exists a position π with ${t_1}^\sharp|_\pi = l\sigma$ and ${t'_1}^\sharp = {t_1}^\sharp[r\sigma]_\pi$. This implies $\pi \neq \epsilon$, $t_1|_\pi = l\sigma$, and $t'_1 = t_1[r\sigma]_\pi$ for some rule $l \to r \in Ext_{\mathcal{E}}(\mathcal{R})$. Since π is the position of a defined symbol below the root (which may of course be below another defined symbol), $\text{REN}(\text{CAP}(t_1))$ matches $\text{REN}(\text{CAP}(t'_1))$. Since $\text{REN}(\text{CAP}(t'_1))$ matches $\text{REN}(\text{CAP}(t_2))$ by the induction hypothesis, $\text{REN}(\text{CAP}(t_1))$ also matches $\text{REN}(\text{CAP}(t_2))$.

So $w^\sharp\sigma \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} q^\sharp$ implies that $\text{REN}(\text{CAP}(w\sigma))$ matches $\text{REN}(\text{CAP}(q))$. Since $\text{REN}(\text{CAP}(w))$ matches $\text{REN}(\text{CAP}(w\sigma))$ and since $\text{REN}(\text{CAP}(q))$ matches q, therefore $\text{REN}(\text{CAP}(w))$ matches q, i.e., we have $\text{REN}(\text{CAP}(w))\mu = q$ for some substituton μ. Hence, if $w^\sharp\sigma \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} q^\sharp \sim_{\mathcal{E}^\sharp} l^\sharp\sigma$, we also have $q \sim_{\mathcal{E}} l\sigma$ and thus, $\text{REN}(\text{CAP}(w))\mu \sim_{\mathcal{E}} l\sigma$. Since the variables of $\text{REN}(\text{CAP}(w)$ and l are disjoint, this means that $\text{REN}(\text{CAP}(w))$ and l are $\mathcal{E}$-unifiable.

So the estimated dependency graph is a supergraph of the real dependency graph, i.e., every cluster in the dependency graph is also a cluster in the estimated dependency graph. As in the proof of Thm. 19, for every non-terminating term t_0 we construct the infinite sequence

$$t_0 \to^+_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} t_1 \to^+_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} t_2 \to^+_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} \cdots$$

such that all t_i contain the minimal non-terminating subterm t'_i.

From the proof of Thm. 19 we see that for all $i \geq 0$ there is a rule $l_i \to r_i \in Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R}))$ and a substitution σ_i such that

- $t_i \to^{>\epsilon *}_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} \circ \sim_{\mathcal{E}} l_i\sigma_i$ and thus, ${t_i}^\sharp \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} \circ \sim_{\mathcal{E}^\sharp} {l_i}^\sharp\sigma_i$,
- $t_{i+1} = s_i\sigma_i$ and thus ${t_{i+1}}^\sharp = {s_i}^\sharp\sigma_i$ for some non-variable subterm s_i of r_i,
- $\langle {l_i}^\sharp, {s_i}^\sharp\rangle$ is a dependency pair of $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R}))$.

This means that for all $i \geq 1$ we have

$$ {s_{i-1}}^\sharp\sigma_{i-1} \to^*_{\mathcal{E}\setminus Ext_{\mathcal{E}}(\mathcal{R})} \circ \sim_{\mathcal{E}^\sharp} {l_i}^\sharp\sigma_i. $$

Thus, there are arcs from $\langle {l_{i-1}}^\sharp, {s_{i-1}}^\sharp\rangle$ to $\langle {l_i}^\sharp, {s_i}^\sharp\rangle$ in the dependency graph.

Since there are only finitely many dependency pairs, the pairs of at least one cluster occur infinitely many times in the sequence

$$ \langle {l_0}^\sharp, {s_0}^\sharp\rangle,\ \langle {l_1}^\sharp, {s_1}^\sharp\rangle,\ \langle {l_2}^\sharp, {s_2}^\sharp\rangle,\ \ldots $$

For at least one dependency pair in this cluster we demanded that the left-hand side is strictly greater than its right-hand side. Thus, ${t'_0}^\sharp, {t'_1}^\sharp, \ldots$ is a decreasing sequence of terms with ${t'_i}^\sharp \gtrsim {t'_{i+1}}^\sharp$ or ${t'_i}^\sharp \gtrsim \circ > {t'_{i+1}}^\sharp$ where the strict inequality ${t'_i}^\sharp \gtrsim \circ > {t'_{i+1}}^\sharp$ holds infinitely many times. This is a contradiction to the well-foundedness of $>$. □

Example 24. For the division-example (Ex. 11) we now obtain different groups of constraints corresponding to the different clusters. The dependency pairs (2), (4), and (5) are not in any cluster and hence, they can be completely disregarded for the termination proof. For example, it would be sufficient to search for orderings $>_1, \gtrsim_1, >_2, \gtrsim_2$ where $>_1$ is compatible with $\gtrsim_1$ and $>_2$ is compatible with $\gtrsim_2$ such that $\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) >_1 \mathsf{M}(x, y)$, $\mathsf{Q}(\ldots) >_2 \mathsf{Q}(\ldots)$ for the dependency pairs (3), (6), (7), and for both $i \in \{1, 2\}$ we need $l \gtrsim_i r$ for all rules $l \to r$, $(u \div v) \div w \sim_i (u \div w) \div v$, and $\mathsf{Q}(u \div v, w) \sim_i \mathsf{Q}(u \div w, v)$. Obviously, here we can choose $>_1 = >_2 = >$ and $\gtrsim_1 = \gtrsim_2 = \gtrsim$ for the orderings $>$ and $\gtrsim$ from Ex. 22. But in general, the modular decomposition of termination proofs by dependency graphs allows many (automated) termination proofs which would not be possible otherwise, cf. [1–3] and the examples A.5, A.7, A.9, and A.10 in the appendix.

7 Conclusion

We have extended the dependency pair approach to equational rewriting. The equations allowed are much more general than just AC-axioms: Any non-collapsing equation is allowed if it satisfies the identical unique variables property (i.e., every variable appears uniquely in each side of the equation, and the same set of variables appear on the two sides of the equation). From a given rewrite system $\mathcal{R}$ and an equation set $\mathcal{E}$, an extended rewrite system $Ext_{\mathcal{E}}(\mathcal{R})$ is computed using a complete unification algorithm for $\mathcal{E}$, so that the termination of its associated weak rewrite relation is sufficient for termination of $\mathcal{R}$ modulo $\mathcal{E}$. The

associated weak rewrite relation of $Ext_{\mathcal{E}}(\mathcal{R})$ is more suited for the dependency pair approach since one never has to consider any applications of $\mathcal{E}$-equations *above* the redex being reduced next. To solve the problem that minimal non-terminating subterms might only occur in variable positions of $\mathcal{R}$-rules, it also becomes necessary to include finitely many instances of $Ext_{\mathcal{E}}(\mathcal{R})$ in the final rewrite system $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R}))$ from which the dependency pairs are computed. Finally, the refinement of dependency graphs carries over to the equational case in a straightforward way (by using $\mathcal{E}$-unification to compute an estimation of these graphs).

In the special case of AC-axioms, our method is similar to the ones previously presented in [16, 18]. In fact, as long as the equations only consist of AC-axioms, one can show that using the instances $Ins_{\mathcal{E}}$ in Thm. 19 is not necessary.[10] (Hence, such a concept cannot be found in [18]). However, even then the only additional inequalities resulting from $Ins_{\mathcal{E}}$ are instantiations of other inequalities already present and inequalities which are special cases of an AC-deletion property (which is satisfied by all known AC-orderings and similar to the one required in [16]). This indicates that in practical examples with AC-axioms, our technique is at least as powerful as the ones of [16, 18] (actually, we conjecture that for AC-examples, these three techniques are virtually equally powerful). But compared to the approaches of [16, 18], our technique has a more elegant treatment of tuple symbols. (For example, if the TRS contains a rule $\mathsf{f}(t_1, t_2) \to \mathsf{g}(\mathsf{f}(s_1, s_2), s_3)$ were f and g are defined AC-symbols, then we do not have to extend the TRS by rules with tuple symbols like $\mathsf{f}(t_1, t_2) \to \mathsf{G}(\mathsf{f}(s_1, s_2), s_2)$ in [18]. Moreover, we do not need dependency pairs where tuple symbols occur outside the root position such as $\langle \mathsf{F}(\mathsf{F}(t_1, t_2), y), \ldots \rangle$ in [18] and [16] and $\langle \mathsf{F}(t_1, t_2), \mathsf{G}(\mathsf{F}(s_1, s_2), s_3) \rangle$ in [16]. Finally, we also do not need the "AC-marked condition" $\mathsf{F}(\mathsf{f}(x, y), z) \sim \mathsf{F}(\mathsf{F}(x, y), z)$ of [16].) But most significantly, unlike [16, 18] our technique works for *arbitrary* non-collapsing equations $\mathcal{E}$ with identical unique variables where $\mathcal{E}$-unification is finitary (for subterms of equations and left-hand sides of rules). Obviously, an implementation of our technique also requires $\mathcal{E}$-unification algorithms [5] for the concrete sets of equations $\mathcal{E}$ under consideration.

Acknowledgements. We thank Aart Middeldorp, Thomas Arts, and the referees for many helpful comments.

A Examples

In this appendix we present a collection of examples to demonstrate the power and the applicability of our approach. In all these examples, the set of equations is different from just AC-axioms. Thus, the previous extensions of the dependency approach [16, 18] are not applicable here. Up to now, the only automatic standard technique for such examples was the direct use of polynomial orderings (as in

[10] Then in the proof of Thm. 19, instead of a minimal non-terminating term t' one regards a term t' which is non-terminating and minimal up to some extra f-occurrences on the top (where f is an AC-symbol).

[6]). However, in the examples A.1 - A.5, termination cannot be proved in this way (because $\mathcal{R}$ alone is already non-simply terminating). Similarly, Examples A.9 and A.10 can easily be transformed into systems that are clearly non-simply terminating, too.

The examples A.6 - A.8 can be proved by polynomial orderings directly, but here our approach has the advantage that it allows the application of the standard rpo or lpo, which is much easier to automate than the search for polynomial orderings.

Examples A.8 - A.10 are TRSs which implement algorithms on non-free data structures like integers or multisets. Equational rewriting is particularly well suited to model non-free data structures (and of course, this requires equations different from AC-axioms). Hence, here the technique of the present paper is very useful. Moreover, even for data structures like the naturals, using $+$ as an additional constructor can be helpful, since it allows an easy definition of many algorithms. The use of our approach for such algorithms is illustrated in Ex. A.4.

The examples A.1, A.2 and A.5 - A.10 demonstrate that with our approach, termination of equational rewriting can often be proved using ordinary base orderings like standard rpo or lpo (i.e., base orderings which themselves are not usable for termination of equational TRSs).

A.1 Division 1

This is the running example from the text (Ex. 11).

$$\begin{array}{lll} \mathcal{R}: & x - \mathsf{0} \to x & \mathcal{E}: \quad (u \div v) \div w \approx (u \div w) \div v \\ & \mathsf{s}(x) - \mathsf{s}(y) \to x - y & \\ & \mathsf{0} \div \mathsf{s}(y) \to \mathsf{0} & \\ & \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y)) & \end{array}$$

Here we obtain

$$\begin{array}{rl} Ext_{\mathcal{E}}(\mathcal{R}) = \mathcal{R} \cup \{ & (\mathsf{0} \div \mathsf{s}(y)) \div z \to \mathsf{0} \div z, \\ & (\mathsf{s}(x) \div \mathsf{s}(y)) \div z \to \mathsf{s}((x - y) \div \mathsf{s}(y)) \div z \ \} \end{array}$$

and $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R})) = Ext_{\mathcal{E}}(\mathcal{R})$. If we use the same ordering for all clusters of the dependency graph we obtain the following constraints:

$$\begin{array}{rll} \mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) > \mathsf{M}(x, y) & & (u \div v) \div w \approx (u \div w) \div v \\ \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) > \mathsf{Q}(x - y, \mathsf{s}(y)) & & \mathsf{Q}(u \div v, w) \approx \mathsf{Q}(u \div w, v) \\ \mathsf{Q}(\mathsf{s}(x) \div \mathsf{s}(y), z) > \mathsf{Q}(x - y, \mathsf{s}(y)) & & \\ \mathsf{Q}(\mathsf{s}(x) \div \mathsf{s}(y), z) > \mathsf{Q}(\mathsf{s}((x - y) \div \mathsf{s}(y)), z) & & \\ x - \mathsf{0} \gtrsim x & & \\ \mathsf{s}(x) - \mathsf{s}(y) \gtrsim x - y & & \\ \mathsf{0} \div \mathsf{s}(y) \gtrsim \mathsf{0} & & \\ \mathsf{s}(x) \div \mathsf{s}(y) \gtrsim \mathsf{s}((x - y) \div \mathsf{s}(y)) & & \end{array}$$

As mentioned in Ex. 22, we eliminate the second arguments of $-$, $\div$, M, and Q. Then the resulting inequalities are satisfied by the rpo with the precedence $\div \sqsupset \mathsf{s} \sqsupset -$.

A.2 Division 2

Apart from the equation $(u \div v) \div w \approx (u \div w) \div v$ one can also add additional equations to the equational theory, e.g., a similar equation for $-$. This yields

$$\begin{array}{lll} \mathcal{R}: & x - \mathsf{0} \to x & \mathcal{E}: \quad (u \div v) \div w \approx (u \div w) \div v \\ & \mathsf{s}(x) - \mathsf{s}(y) \to x - y & \qquad\quad (u - v) - w \approx (u - w) - v \\ & \mathsf{0} \div \mathsf{s}(y) \to \mathsf{0} & \\ & \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y)) & \end{array}$$

In addition to the rules mentioned in the previous example, now $Ext_{\mathcal{E}}(\mathcal{R})$ also contains the rule

$$(\mathsf{s}(x) - \mathsf{s}(y)) - z \;\to\; (x - y) - z.$$

Again we have $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R})) = Ext_{\mathcal{E}}(\mathcal{R})$ and thus, the resulting constraints are the same as in the previous example plus the additional constraints

$$\begin{array}{ll} \mathsf{M}(\mathsf{s}(x) - \mathsf{s}(y), z) > \mathsf{M}(x, y) & (u - v) - w \sim (u - w) - v \\ \mathsf{M}(\mathsf{s}(x) - \mathsf{s}(y), z) > \mathsf{M}(x - y, z) & \mathsf{M}(u - v, w) \sim \mathsf{M}(u - w, v). \end{array}$$

With the same argument elimination as in Ex. A.1, the same rpo satisfies these constraints.

A.3 Division 3

The next example shows that instead of the equation $(u \div v) \div w \approx (u \div w) \div v$ one can also use other equations relating nested $\div$-applications such as $u \div (v \div w) \approx w \div (v \div u)$.

$$\begin{array}{lll} \mathcal{R}: & x - \mathsf{0} \to x & \mathcal{E}: \quad u \div (v \div w) \approx w \div (v \div u) \\ & \mathsf{s}(x) - \mathsf{s}(y) \to x - y & \\ & \mathsf{0} \div \mathsf{s}(y) \to \mathsf{0} & \\ & \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y)) & \end{array}$$

Now we obtain

$$\begin{array}{l} Ext_{\mathcal{E}}(\mathcal{R}) = \mathcal{R} \cup \{ \quad z \div (\mathsf{0} \div \mathsf{s}(y)) \to z \div \mathsf{0}, \\ \qquad\qquad\qquad z \div (\mathsf{s}(x) \div \mathsf{s}(y)) \to z \div \mathsf{s}((x - y) \div \mathsf{s}(y)) \; \}. \end{array}$$

Overlaps with these new rules do not have to be included in $Ext_{\mathcal{E}}(\mathcal{R})$: If the subterm $v \div w$ is overlapped with $z \div (\mathsf{0} \div \mathsf{s}(y))$, we would obtain the new rule $z' \div (z \div (\mathsf{0} \div \mathsf{s}(y))) \to z' \div (z \div \mathsf{0})$, but the other side of the instantiated equation, $(\mathsf{0} \div \mathsf{s}(y)) \div (z \div z')$ already $\mathcal{R}$-reduces to $\mathsf{0} \div (z \div z')$ which is $\mathcal{E}$-equivalent to $z' \div (z \div \mathsf{0})$. A similar statement holds for overlaps with the second new rule.

Again we have $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R})) = Ext_{\mathcal{E}}(\mathcal{R})$. Thus, we obtain the following constraints, if we use the same ordering for all clusters of the dependency graph:

$$
\begin{aligned}
\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{M}(x,y)\\
\mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{Q}(x-y,\mathsf{s}(y))\\
\mathsf{Q}(z,\mathsf{s}(x)\div\mathsf{s}(y)) &> \mathsf{Q}(x-y,\mathsf{s}(y))\\
\mathsf{Q}(z,\mathsf{s}(x)\div\mathsf{s}(y)) &> \mathsf{Q}(z,\mathsf{s}((x-y)\div\mathsf{s}(y)))\\
x-\mathsf{0} &\succsim x\\
\mathsf{s}(x)-\mathsf{s}(y) &\succsim x-y\\
\mathsf{0}\div\mathsf{s}(y) &\succsim \mathsf{0}\\
\mathsf{s}(x)\div\mathsf{s}(y) &\succsim \mathsf{s}((x-y)\div\mathsf{s}(y))
\end{aligned}
\qquad
\begin{aligned}
u\div(v\div w)\div w &\approx w\div(v\div u)\\
\mathsf{Q}(u,v\div w) &\approx \mathsf{Q}(w,v\div u)
\end{aligned}
$$

The constraints are satisfied by the following polynomial interpretation:

$$
\begin{aligned}
\mathsf{0} &\Rightarrow 1\\
\mathsf{s}(x) &\Rightarrow x+1\\
x-y &\Rightarrow x\\
x\div y &\Rightarrow x\cdot y\\
\mathsf{M}(x,y) &\Rightarrow x\\
\mathsf{Q}(x,y) &\Rightarrow x\cdot y
\end{aligned}
$$

A.4 Equational Theory with Addition

It is also possible to use $+$ as an (associative and commutative) constructor. Then the set of constructors is $\mathsf{0}$, s and $+$, i.e., we have a set of non-free constructors. Now subtraction can be defined very easily by just the rule $(x+y)-y\to x$. In this way, the division system looks as follows:

$$
\mathcal{R}:\quad
\begin{aligned}
(x+y)-y &\to x\\
\mathsf{0}\div\mathsf{s}(y) &\to \mathsf{0}\\
\mathsf{s}(x)\div\mathsf{s}(y) &\to \mathsf{s}((x-y)\div\mathsf{s}(y))
\end{aligned}
\qquad
\mathcal{E}:\quad
\begin{aligned}
\mathsf{0}+v &\approx v\\
\mathsf{s}(u)+v &\approx \mathsf{s}(u+v)\\
u+v &\approx v+u\\
u+(v+w) &\approx (u+v)+w
\end{aligned}
$$

Obviously, we have $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R}))=\mathcal{R}$ (this is always the case if the equations contain no defined symbols). When using the same ordering for all clusters we obtain

$$
\begin{aligned}
\mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{Q}(x-y,\mathsf{s}(y))\\
(x+y)-y &\succsim x\\
\mathsf{0}\div\mathsf{s}(y) &\succsim \mathsf{0}\\
\mathsf{s}(x)\div\mathsf{s}(y) &\succsim \mathsf{s}((x-y)\div\mathsf{s}(y))
\end{aligned}
\qquad
\begin{aligned}
\mathsf{0}+v &\approx v\\
\mathsf{s}(u)+v &\approx \mathsf{s}(u+v)\\
u+v &\approx v+u\\
u+(v+w) &\approx (u+v)+w.
\end{aligned}
$$

The constraints are satisfied by the following polynomial ordering:

$$
\begin{aligned}
\mathsf{0} &\Rightarrow 0\\
\mathsf{s}(x) &\Rightarrow x+1\\
x+y &\Rightarrow x+y\\
x-y &\Rightarrow x\\
x\div y &\Rightarrow x\\
\mathsf{Q}(x,y) &\Rightarrow x
\end{aligned}
$$

Alternatively, it would also be possible to use the equational theory

$$\begin{aligned} 1 + 0 &\approx 1 \\ 0 + 1 &\approx 1 \\ 0 + 0 &\approx 0 \\ u + (v + w) &\approx (u + v) + w \end{aligned}$$

and to reformulate the second division rule as follows.

$$(x + 1) \div (y + 1) \;\to\; ((x - y) \div (y + 1)) + 1$$

When interpreting 1 as the number 1, the same polynomial interpretation as above can be used to prove termination of this modified system, too.

A.5 Conversion into Bitstrings

The next example is a TRS used to convert numbers into a representation w.r.t. an arbitrary base. (Thus, if one uses base 2, the following TRS can be used to convert numbers into bitstrings.) Here, we use the equation $\mathsf{cons}(n, \mathsf{cons}(0, \mathsf{nil})) \approx \mathsf{cons}(n, \mathsf{nil})$, since 0's in the most significant digits do not matter.

$$\begin{aligned} \mathcal{R}: \quad \mathsf{convert}(0, \mathsf{s}(\mathsf{s}(b))) &\to \mathsf{cons}(0, \mathsf{nil}) \\ \mathsf{convert}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(b))) &\to \mathsf{cons}(\mathsf{s}(0), \mathsf{nil}) \\ \mathsf{convert}(\mathsf{s}(\mathsf{s}(n)), \mathsf{s}(\mathsf{s}(b))) &\to \mathsf{cons}(\mathsf{mod}(n - b, \mathsf{s}(\mathsf{s}(b))), \\ &\qquad\qquad \mathsf{convert}(\mathsf{s}((n - b) \div \mathsf{s}(\mathsf{s}(b))), \mathsf{s}(\mathsf{s}(b)))) \\ \mathcal{E}: \quad \mathsf{cons}(n, \mathsf{cons}(0, \mathsf{nil})) &\approx \mathsf{cons}(n, \mathsf{nil}) \end{aligned}$$

Already for this system, the set of rules $\mathcal{R}$ is not simply terminating. Of course, we may also add the rules for the auxiliary functions:

$$\begin{aligned} x - 0 &\to x \\ \mathsf{s}(x) - \mathsf{s}(y) &\to x - y \\ \mathsf{lt}(x, 0) &\to \mathsf{false} \\ \mathsf{lt}(0, \mathsf{s}(y)) &\to \mathsf{true} \\ \mathsf{lt}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{lt}(x, y) \\ 0 \div \mathsf{s}(y) &\to 0 \\ \mathsf{s}(x) \div \mathsf{s}(y) &\to \mathsf{if}_{\div}(\mathsf{lt}(x, y), \mathsf{s}(x), \mathsf{s}(y)) \\ \mathsf{if}_{\div}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) &\to 0 \\ \mathsf{if}_{\div}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{s}((x - y) \div \mathsf{s}(y)) \\ \mathsf{mod}(0, \mathsf{s}(y)) &\to 0 \\ \mathsf{mod}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{if}_{\mathsf{mod}}(\mathsf{lt}(x, y), \mathsf{s}(x), \mathsf{s}(y)) \\ \mathsf{if}_{\mathsf{mod}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{s}(x) \\ \mathsf{if}_{\mathsf{mod}}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{mod}(x - y, \mathsf{s}(y)) \end{aligned}$$

We have $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R})) = \mathcal{R}$ and thus, we obtain the following constraints:

$$\begin{aligned}
\mathsf{CONVERT}(\mathsf{s}(\mathsf{s}(n)), \mathsf{s}(\mathsf{s}(b))) &> \mathsf{CONVERT}(\mathsf{s}((n-b) \div \mathsf{s}(\mathsf{s}(b))), \mathsf{s}(\mathsf{s}(b))) \\
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{M}(x, y) \\
\mathsf{LT}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{LT}(x, y) \\
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{IF}_{\div}(\mathsf{lt}(x, y), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{IF}_{\div}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{Q}(x - y, \mathsf{s}(y)) \\
\mathsf{MOD}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{IF}_{\mathsf{mod}}(\mathsf{lt}(x, y), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{IF}_{\mathsf{mod}}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{MOD}(x - y, \mathsf{s}(y)) \\
\mathsf{convert}(\mathsf{0}, \mathsf{s}(\mathsf{s}(b))) &\succsim \mathsf{cons}(\mathsf{0}, \mathsf{nil}) \\
\mathsf{convert}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(b))) &\succsim \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{nil}) \\
\mathsf{convert}(\mathsf{s}(\mathsf{s}(n)), \mathsf{s}(\mathsf{s}(b))) &\succsim \mathsf{cons}(\mathsf{mod}(n - b, \mathsf{s}(\mathsf{s}(b))), \\
&\qquad \mathsf{convert}(\mathsf{s}((n-b) \div \mathsf{s}(\mathsf{s}(b))), \mathsf{s}(\mathsf{s}(b)))) \\
x - \mathsf{0} &\succsim x \\
\mathsf{s}(x) - \mathsf{s}(y) &\succsim x - y \\
\mathsf{lt}(x, \mathsf{0}) &\succsim \mathsf{false} \\
\mathsf{lt}(\mathsf{0}, \mathsf{s}(y)) &\succsim \mathsf{true} \\
\mathsf{lt}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{lt}(x, y) \\
\mathsf{0} \div \mathsf{s}(y) &\succsim \mathsf{0} \\
\mathsf{s}(x) \div \mathsf{s}(y) &\succsim \mathsf{if}_{\div}(\mathsf{lt}(x, y), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{if}_{\div}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{0} \\
\mathsf{if}_{\div}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{s}((x - y) \div \mathsf{s}(y)) \\
\mathsf{mod}(\mathsf{0}, \mathsf{s}(y)) &\succsim \mathsf{0} \\
\mathsf{mod}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{if}_{\mathsf{mod}}(\mathsf{lt}(x, y), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{if}_{\mathsf{mod}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{s}(x) \\
\mathsf{if}_{\mathsf{mod}}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{mod}(x - y, \mathsf{s}(y)) \\
\mathsf{cons}(n, \mathsf{cons}(\mathsf{0}, \mathsf{nil})) &\approx \mathsf{cons}(n, \mathsf{nil})
\end{aligned}$$

When replacing cons, $-$, $\div$, Q, mod, and MOD by their first arguments and $\mathsf{if}_{\div}$, $\mathsf{IF}_{\div}$, $\mathsf{if}_{\mathsf{mod}}$, and $\mathsf{IF}_{\mathsf{mod}}$ by their second arguments, we obtain

$$\begin{aligned}
\mathsf{CONVERT}(\mathsf{s}(\mathsf{s}(n)), \mathsf{s}(\mathsf{s}(b))) &> \mathsf{CONVERT}(\mathsf{s}(n), \mathsf{s}(\mathsf{s}(b))) \\
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{M}(x, y) \\
\mathsf{LT}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{LT}(x, y) \\
\mathsf{s}(x) &\succsim \mathsf{s}(x) \\
\mathsf{s}(x) &> x \\
\mathsf{s}(x) &\succsim \mathsf{s}(x) \\
\mathsf{s}(x) &> x \\
\mathsf{convert}(\mathsf{0}, \mathsf{s}(\mathsf{s}(b))) &\succsim \mathsf{0}
\end{aligned}$$

$$
\begin{aligned}
\mathsf{convert}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(b))) &\succsim \mathsf{s}(0) \\
\mathsf{convert}(\mathsf{s}(\mathsf{s}(n)), \mathsf{s}(\mathsf{s}(b))) &\succsim n \\
x &\succsim x \\
\mathsf{s}(x) &\succsim x \\
\mathsf{lt}(x, 0) &\succsim \mathsf{false} \\
\mathsf{lt}(0, \mathsf{s}(y)) &\succsim \mathsf{true} \\
\mathsf{lt}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{lt}(x, y) \\
0 &\succsim 0 \\
\mathsf{s}(x) &\succsim \mathsf{s}(x) \\
\mathsf{s}(x) &\succsim 0 \\
\mathsf{s}(x) &\succsim \mathsf{s}(x) \\
0 &\succsim 0 \\
\mathsf{s}(x) &\succsim \mathsf{s}(x) \\
\mathsf{s}(x) &\succsim \mathsf{s}(x) \\
\mathsf{s}(x) &\succsim x \\
n &\approx n
\end{aligned}
$$

These constraints are satisfied by the rpo.

A.6 Idempotence

The following example (Ex. 10) is a TRS where the equation states that a function symbol f is idempotent.

$$\mathcal{R}: \quad \mathsf{f}(\mathsf{s}(y)) \to \mathsf{f}(y) \qquad \mathcal{E}: \quad \mathsf{f}(\mathsf{f}(x)) \approx \mathsf{f}(x)$$

As explained in Ex. 10, we have $Ext_{\mathcal{E}}(\mathcal{R}) = \mathcal{R}$ and we also have $Ins_{\mathcal{E}}(\mathcal{R}) = \mathcal{R}$. Thus, the resulting constraints are

$$
\begin{array}{ll}
\mathsf{F}(\mathsf{s}(y)) > \mathsf{F}(y) & \mathsf{f}(\mathsf{f}(x)) \sim \mathsf{f}(x) \\
\mathsf{f}(\mathsf{s}(y)) \gtrsim \mathsf{f}(y) & \mathsf{F}(\mathsf{f}(x)) \sim \mathsf{F}(x)
\end{array}
$$

After replacing f by its argument, these inequalities are satisfied by the rpo.

A.7 Idempotence of flatten

The following example is similar to the previous one, but it illustrates that equations for idempotence can be useful for functions occurring in practice. The following example contains an algorithm to flatten binary trees and the equation states that flatten is idempotent.

$$
\begin{aligned}
\mathcal{R}: \qquad \mathsf{flatten}(\mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{flatten}(\mathsf{cons}(\mathsf{nil}, x)) &\to \mathsf{cons}(\mathsf{nil}, \mathsf{flatten}(x)) \\
\mathsf{flatten}(\mathsf{cons}(\mathsf{cons}(x, y), z)) &\to \mathsf{flatten}(\mathsf{cons}(x, \mathsf{cons}(y, z)))
\end{aligned}
$$

$$\mathcal{E}: \qquad \mathsf{flatten}(\mathsf{flatten}(u)) \approx \mathsf{flatten}(u)$$

By overlapping the subterm $\mathsf{flatten}(u)$ of the equation with the left-hand sides of the first two rules we obtain

$$Ext_{\mathcal{E}}(\mathcal{R}) = \mathcal{R} \cup \{ \begin{array}{r} \mathsf{flatten}(\mathsf{flatten}(\mathsf{nil})) \to \mathsf{flatten}(\mathsf{nil}), \\ \mathsf{flatten}(\mathsf{flatten}(\mathsf{cons}(\mathsf{nil}, x))) \to \mathsf{flatten}(\mathsf{cons}(\mathsf{nil}, x)) \ \}. \end{array}$$

Overlapping $\mathsf{flatten}(u)$ with the left-hand side of the last rule is not needed. This would yield the rule $\mathsf{flatten}(\mathsf{flatten}(\mathsf{cons}(\mathsf{cons}(x, y), z))) \to \mathsf{flatten}(\mathsf{flatten}(\mathsf{cons}(x, \mathsf{cons}(y, z))))$. But the other term of the instantiated equation, $\mathsf{flatten}(\mathsf{cons}(\mathsf{cons}(x, y), z))$ already rewrites to $\mathsf{flatten}(\mathsf{cons}(x, \mathsf{cons}(y, z)))$ which is $\mathcal{E}$-equivalent to $\mathsf{flatten}(\mathsf{flatten}(\mathsf{cons}(x, \mathsf{cons}(y, z))))$. Similarly, it can also be shown that overlapping $\mathsf{flatten}(u)$ with the left-hand sides of the new rules is unnecessary.

We have $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R})) = Ext_{\mathcal{E}}(\mathcal{R})$. The resulting dependency pairs are

$$\langle \mathsf{FLATTEN}(\mathsf{cons}(\mathsf{nil}, x)), \mathsf{FLATTEN}(x) \rangle \tag{8}$$
$$\langle \mathsf{FLATTEN}(\mathsf{cons}(\mathsf{cons}(x, y), z)), \mathsf{FLATTEN}(\mathsf{cons}(x, \mathsf{cons}(y, z))) \rangle \tag{9}$$
$$\langle \mathsf{FLATTEN}(\mathsf{flatten}(\mathsf{nil})), \mathsf{FLATTEN}(\mathsf{nil}) \rangle \tag{10}$$
$$\langle \mathsf{FLATTEN}(\mathsf{flatten}(\mathsf{cons}(\mathsf{nil}, x))), \mathsf{FLATTEN}(\mathsf{cons}(\mathsf{nil}, x)) \rangle \tag{11}$$

Obviously, (10) is not on any cluster and hence, it can be disregarded. The only cluster containing (11) is $\{(8), (11)\}$. Thus, it is sufficient if (11) is only weakly decreasing. We result in the following constraints:

$$\begin{array}{rcl} \mathsf{FLATTEN}(\mathsf{cons}(\mathsf{nil}, x)) & > & \mathsf{FLATTEN}(x) \\ \mathsf{FLATTEN}(\mathsf{cons}(\mathsf{cons}(x, y), z)) & > & \mathsf{FLATTEN}(\mathsf{cons}(x, \mathsf{cons}(y, z))) \\ \mathsf{FLATTEN}(\mathsf{flatten}(\mathsf{cons}(\mathsf{nil}, x))) & \gtrsim & \mathsf{FLATTEN}(\mathsf{cons}(\mathsf{nil}, x)) \\ \mathsf{flatten}(\mathsf{nil}) & \gtrsim & \mathsf{nil} \\ \mathsf{flatten}(\mathsf{cons}(\mathsf{nil}, x)) & \gtrsim & \mathsf{cons}(\mathsf{nil}, \mathsf{flatten}(x)) \\ \mathsf{flatten}(\mathsf{cons}(\mathsf{cons}(x, y), z)) & \gtrsim & \mathsf{flatten}(\mathsf{cons}(x, \mathsf{cons}(y, z))) \\ \mathsf{flatten}(\mathsf{flatten}(u)) & \sim & \mathsf{flatten}(u) \\ \mathsf{FLATTEN}(\mathsf{flatten}(u)) & \sim & \mathsf{FLATTEN}(u) \end{array}$$

After replacing $\mathsf{flatten}$ by its argument, the constraints are satisfied by the lexicographic path ordering [9, 13].

A.8 Addition on Integers

Consider the following equational rewrite system on integers. Here, integer numbers are built from the function symbols $\mathsf{0}$, s, and p. The equation $\mathsf{s}(\mathsf{p}(z)) \approx \mathsf{p}(\mathsf{s}(z))$ states that the function symbols s and p may be interchanged. Thus, equational rewriting is used to model this non-free data structure.

$$\mathcal{R}: \quad \begin{array}{r} 0 + y \to y \\ \mathsf{p}(x) + y \to \mathsf{p}(x + y) \\ \mathsf{s}(x) + y \to \mathsf{s}(x + y) \\ \mathsf{p}(\mathsf{s}(x)) \to x \end{array} \qquad \mathcal{E}: \quad \mathsf{s}(\mathsf{p}(z)) \approx \mathsf{p}(\mathsf{s}(z))$$

We obtain $Ext_{\mathcal{E}}(\mathcal{R}) = \mathcal{R}$. The non-variable proper subterm $\mathsf{p}(z)$ of the equation unifies with the left-hand side $\mathsf{p}(\mathsf{s}(x))$ of the last rule, but the corresponding new rule $\mathsf{s}(\mathsf{p}(\mathsf{s}(x))) \to \mathsf{s}(x)$ does not have to be included in $Ext_{\mathcal{E}}(\mathcal{R})$, since the instantiated other side of the equation, $\mathsf{p}(\mathsf{s}(\mathsf{s}(x)))$, already $\mathcal{R}$-reduces to $\mathsf{s}(x)$. A similar statement holds for the unification of the subterm $\mathsf{s}(z)$ with $\mathsf{p}(\mathsf{s}(x))$.

Moreover, we also have $Ins_{\mathcal{E}}(\mathcal{R}) = \mathcal{R}$. Since dependency pairs with the tuple symbol of $+$ in the left component and the tuple symbols of p or s in the right component do not occur in any cluster of the dependency graph, it is sufficient to satisfy the following constraints (where PLUS denotes the tuple symbol for $+$):

$$\begin{array}{rlrl}
\mathsf{PLUS}(\mathsf{p}(x), y) &> \mathsf{PLUS}(x, y) & \mathsf{s}(\mathsf{p}(z)) &\sim \mathsf{p}(\mathsf{s}(z)) \\
\mathsf{PLUS}(\mathsf{s}(x), y) &> \mathsf{PLUS}(x, y) & \mathsf{S}(\mathsf{p}(z)) &\sim \mathsf{P}(\mathsf{s}(z)) \\
0 + y &\succsim y & & \\
\mathsf{p}(x) + y &\succsim \mathsf{p}(x + y) & & \\
\mathsf{s}(x) + y &\succsim \mathsf{s}(x + y) & & \\
\mathsf{p}(\mathsf{s}(x)) &\succsim x & &
\end{array}$$

These constraints are satisfied by the rpo, where $+$ is greater than p and s in the precedence and both s and p and both S and P are equal in the precedence, respectively.

A.9 Transforming Multisets into Sets

The next example uses equations to model the non-free data structure of multisets. Here, we need an equation $\mathsf{cons}(u, \mathsf{cons}(v, w)) \approx \mathsf{cons}(v, \mathsf{cons}(u, w))$ since it does not matter in which order two elements u and v are inserted into a multiset w. The function set is used to transform a multiset into a set, i.e., to eliminate duplicates from a multiset. It uses the auxiliary function rm (for "remove"), where $\mathsf{rm}(n, x)$ deletes all occurrences of the element n from the multiset x.

$$\begin{array}{rrl}
\mathcal{R}: & \mathsf{eq}(0, 0) &\to \mathsf{true} \\
& \mathsf{eq}(0, \mathsf{s}(y)) &\to \mathsf{false} \\
& \mathsf{eq}(\mathsf{s}(x), 0) &\to \mathsf{false} \\
& \mathsf{eq}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{eq}(x, y) \\
& \mathsf{rm}(n, \mathsf{nil}) &\to \mathsf{nil} \\
& \mathsf{rm}(n, \mathsf{cons}(m, x)) &\to \mathsf{if}(\mathsf{eq}(n, m), n, \mathsf{cons}(m, x)) \\
& \mathsf{if}(\mathsf{true}, n, \mathsf{cons}(m, x)) &\to \mathsf{rm}(n, x) \\
& \mathsf{if}(\mathsf{false}, n, \mathsf{cons}(m, x)) &\to \mathsf{cons}(m, \mathsf{rm}(n, x)) \\
& \mathsf{set}(\mathsf{nil}) &\to \mathsf{nil} \\
& \mathsf{set}(\mathsf{cons}(n, x)) &\to \mathsf{cons}(n, \mathsf{set}(\mathsf{rm}(n, x))) \\
\mathcal{E}: & \mathsf{cons}(u, \mathsf{cons}(v, w)) &\approx \mathsf{cons}(v, \mathsf{cons}(u, w))
\end{array}$$

Since the equation only contains constructors and variables, we obtain $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R})) = \mathcal{R}$. Thus, when using the same ordering for all clusters, it

suffices to satisfy the following constraints:

$$\begin{aligned}
\mathsf{EQ}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{EQ}(x,y)\\
\mathsf{RM}(n,\mathsf{cons}(m,x)) &\succsim \mathsf{IF}(\mathsf{eq}(n,m),n,\mathsf{cons}(m,x))\\
\mathsf{IF}(\mathsf{true},n,\mathsf{cons}(m,x)) &> \mathsf{RM}(n,x)\\
\mathsf{IF}(\mathsf{false},n,\mathsf{cons}(m,x)) &> \mathsf{RM}(n,x)\\
\mathsf{SET}(\mathsf{cons}(n,x)) &> \mathsf{SET}(\mathsf{rm}(n,x))\\
\mathsf{eq}(0,0) &\succsim \mathsf{true}\\
\mathsf{eq}(0,\mathsf{s}(y)) &\succsim \mathsf{false}\\
\mathsf{eq}(\mathsf{s}(x),0) &\succsim \mathsf{false}\\
\mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) &\succsim \mathsf{eq}(x,y)\\
\mathsf{rm}(n,\mathsf{nil}) &\succsim \mathsf{nil}\\
\mathsf{rm}(n,\mathsf{cons}(m,x)) &\succsim \mathsf{if}(\mathsf{eq}(n,m),n,\mathsf{cons}(m,x))\\
\mathsf{if}(\mathsf{true},n,\mathsf{cons}(m,x)) &\succsim \mathsf{rm}(n,x)\\
\mathsf{if}(\mathsf{false},n,\mathsf{cons}(m,x)) &\succsim \mathsf{cons}(m,\mathsf{rm}(n,x))\\
\mathsf{set}(\mathsf{nil}) &\succsim \mathsf{nil}\\
\mathsf{set}(\mathsf{cons}(n,x)) &\succsim \mathsf{cons}(n,\mathsf{set}(\mathsf{rm}(n,x)))\\
\mathsf{cons}(u,\mathsf{cons}(v,w)) &\sim \mathsf{cons}(v,\mathsf{cons}(u,w))
\end{aligned}$$

Before searching for a base ordering satisfying these constraints, we replace rm, RM, if, and IF by their last arguments and we eliminate the first argument of cons. This results in the following constraints:

$$\begin{aligned}
\mathsf{EQ}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{EQ}(x,y)\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{cons}'(x) &> x\\
\mathsf{cons}'(x) &> x\\
\mathsf{SET}(\mathsf{cons}'(x)) &> \mathsf{SET}(x)\\
\mathsf{eq}(0,0) &\succsim \mathsf{true}\\
\mathsf{eq}(0,\mathsf{s}(y)) &\succsim \mathsf{false}\\
\mathsf{eq}(\mathsf{s}(x),0) &\succsim \mathsf{false}\\
\mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) &\succsim \mathsf{eq}(x,y)\\
\mathsf{nil} &\succsim \mathsf{nil}\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{cons}'(x) &\succsim x\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{set}(\mathsf{nil}) &\succsim \mathsf{nil}\\
\mathsf{set}(\mathsf{cons}'(x)) &\succsim \mathsf{cons}'(\mathsf{set}(x))\\
\mathsf{cons}'(\mathsf{cons}'(w)) &\sim \mathsf{cons}'(\mathsf{cons}'(w))
\end{aligned}$$

These resulting constraints are satisfied by the rpo.

A.10 Quicksort on Multisets

The following TRS is used to sort a multiset by the well-known quicksort algorithm. It uses the functions $\mathsf{low}(n,x)$ (resp. $\mathsf{high}(n,x)$) which return the subset of x containing only the elements smaller than or equal to (resp. greater than) n.

The equation in $\mathcal{E}$ is again used to model multisets, i.e., to state that the order of the elements does not matter.

$$
\begin{array}{lrcl}
\mathcal{R}: & \mathsf{le}(0, y) & \to & \mathsf{true} \\
& \mathsf{le}(\mathsf{s}(x), 0) & \to & \mathsf{false} \\
& \mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{le}(x, y) \\
& \mathsf{app}(\mathsf{nil}, y) & \to & y \\
& \mathsf{app}(\mathsf{cons}(n, x), y) & \to & \mathsf{cons}(n, \mathsf{app}(x, y)) \\
& \mathsf{low}(n, \mathsf{nil}) & \to & \mathsf{nil} \\
& \mathsf{low}(n, \mathsf{cons}(m, x)) & \to & \mathsf{if}_{\mathsf{low}}(\mathsf{le}(m, n), n, \mathsf{cons}(m, x)) \\
& \mathsf{if}_{\mathsf{low}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & \to & \mathsf{cons}(m, \mathsf{low}(n, x)) \\
& \mathsf{if}_{\mathsf{low}}(\mathsf{false}, n, \mathsf{cons}(m, x)) & \to & \mathsf{low}(n, x) \\
& \mathsf{high}(n, \mathsf{nil}) & \to & \mathsf{nil} \\
& \mathsf{high}(n, \mathsf{cons}(m, x)) & \to & \mathsf{if}_{\mathsf{high}}(\mathsf{le}(m, n), n, \mathsf{cons}(m, x)) \\
& \mathsf{if}_{\mathsf{high}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & \to & \mathsf{high}(n, x) \\
& \mathsf{if}_{\mathsf{high}}(\mathsf{false}, n, \mathsf{cons}(m, x)) & \to & \mathsf{cons}(m, \mathsf{high}(n, x)) \\
& \mathsf{quicksort}(\mathsf{nil}) & \to & \mathsf{nil} \\
& \mathsf{quicksort}(\mathsf{cons}(n, x)) & \to & \mathsf{app}(\mathsf{quicksort}(\mathsf{low}(n, x)), \\
& & & \quad \mathsf{cons}(n, \mathsf{quicksort}(\mathsf{high}(n, x)))) \\
\mathcal{E}: & \mathsf{cons}(u, \mathsf{cons}(v, w)) & \approx & \mathsf{cons}(v, \mathsf{cons}(u, w))
\end{array}
$$

Again we have $Ins_{\mathcal{E}}(Ext_{\mathcal{E}}(\mathcal{R})) = \mathcal{R}$. So the constraints are

$$
\begin{array}{rcl}
\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{LE}(x, y) \\
\mathsf{APP}(\mathsf{cons}(n, x), y) & > & \mathsf{APP}(x, y) \\
\mathsf{LOW}(n, \mathsf{cons}(m, x)) & \gtrsim & \mathsf{IF}_{\mathsf{low}}(\mathsf{le}(m, n), n, \mathsf{cons}(m, x)) \\
\mathsf{IF}_{\mathsf{low}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & > & \mathsf{LOW}(n, x) \\
\mathsf{IF}_{\mathsf{low}}(\mathsf{false}, n, \mathsf{cons}(m, x)) & > & \mathsf{LOW}(n, x) \\
\mathsf{HIGH}(n, \mathsf{cons}(m, x)) & \gtrsim & \mathsf{IF}_{\mathsf{high}}(\mathsf{le}(m, n), n, \mathsf{cons}(m, x)) \\
\mathsf{IF}_{\mathsf{high}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & > & \mathsf{HIGH}(n, x) \\
\mathsf{IF}_{\mathsf{high}}(\mathsf{false}, n, \mathsf{cons}(m, x)) & > & \mathsf{HIGH}(n, x) \\
\mathsf{QUICKSORT}(\mathsf{cons}(n, x)) & > & \mathsf{QUICKSORT}(\mathsf{low}(n, x)) \\
\mathsf{QUICKSORT}(\mathsf{cons}(n, x)) & > & \mathsf{QUICKSORT}(\mathsf{high}(n, x)) \\
\mathsf{le}(0, y) & \gtrsim & \mathsf{true} \\
\mathsf{le}(\mathsf{s}(x), 0) & \gtrsim & \mathsf{false} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) & \gtrsim & \mathsf{le}(x, y) \\
\mathsf{app}(\mathsf{nil}, y) & \gtrsim & y \\
\mathsf{app}(\mathsf{cons}(n, x), y) & \gtrsim & \mathsf{cons}(n, \mathsf{app}(x, y)) \\
\mathsf{low}(n, \mathsf{nil}) & \gtrsim & \mathsf{nil} \\
\mathsf{low}(n, \mathsf{cons}(m, x)) & \gtrsim & \mathsf{if}_{\mathsf{low}}(\mathsf{le}(m, n), n, \mathsf{cons}(m, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{true}, n, \mathsf{cons}(m, x)) & \gtrsim & \mathsf{cons}(m, \mathsf{low}(n, x)) \\
\mathsf{if}_{\mathsf{low}}(\mathsf{false}, n, \mathsf{cons}(m, x)) & \gtrsim & \mathsf{low}(n, x)
\end{array}
$$

$$
\begin{aligned}
\mathsf{high}(n, \mathsf{nil}) &\succsim \mathsf{nil}\\
\mathsf{high}(n, \mathsf{cons}(m, x)) &\succsim \mathsf{if}_{\mathsf{high}}(\mathsf{le}(m, n), n, \mathsf{cons}(m, x))\\
\mathsf{if}_{\mathsf{high}}(\mathsf{true}, n, \mathsf{cons}(m, x)) &\succsim \mathsf{high}(n, x)\\
\mathsf{if}_{\mathsf{high}}(\mathsf{false}, n, \mathsf{cons}(m, x)) &\succsim \mathsf{cons}(m, \mathsf{high}(n, x))\\
\mathsf{quicksort}(\mathsf{nil}) &\succsim \mathsf{nil}\\
\mathsf{quicksort}(\mathsf{cons}(n, x)) &\succsim \mathsf{app}(\mathsf{quicksort}(\mathsf{low}(n, x)),\\
&\qquad\qquad \mathsf{cons}(n, \mathsf{quicksort}(\mathsf{high}(n, x))))\\
\mathsf{cons}(u, \mathsf{cons}(v, w)) &\sim \mathsf{cons}(v, \mathsf{cons}(u, w))
\end{aligned}
$$

We replace low, LOW, high, HIGH, $\mathsf{if}_{\mathsf{low}}$, $\mathsf{IF}_{\mathsf{low}}$, $\mathsf{if}_{\mathsf{high}}$, and $\mathsf{IF}_{\mathsf{high}}$ by their last arguments and we eliminate the first argument of cons. This results in the following constraints:

$$
\begin{aligned}
\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{LE}(x, y)\\
\mathsf{APP}(\mathsf{cons}'(x), y) &> \mathsf{APP}(x, y)\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{cons}'(x) &> x\\
\mathsf{cons}'(x) &> x\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{cons}'(x) &> x\\
\mathsf{cons}'(x) &> x\\
\mathsf{QUICKSORT}(\mathsf{cons}'(x)) &> \mathsf{QUICKSORT}(x)\\
\mathsf{QUICKSORT}(\mathsf{cons}'(x)) &> \mathsf{QUICKSORT}(x)\\
\mathsf{le}(0, y) &\succsim \mathsf{true}\\
\mathsf{le}(\mathsf{s}(x), 0) &\succsim \mathsf{false}\\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{le}(x, y)\\
\mathsf{app}(\mathsf{nil}, y) &\succsim y\\
\mathsf{app}(\mathsf{cons}'(x), y) &\succsim \mathsf{cons}'(\mathsf{app}(x, y))\\
\mathsf{nil} &\succsim \mathsf{nil}\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{cons}'(x) &\succsim x\\
\mathsf{nil} &\succsim \mathsf{nil}\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{cons}'(x) &\succsim x\\
\mathsf{cons}'(x) &\succsim \mathsf{cons}'(x)\\
\mathsf{quicksort}(\mathsf{nil}) &\succsim \mathsf{nil}\\
\mathsf{quicksort}(\mathsf{cons}'(x)) &\succsim \mathsf{app}(\mathsf{quicksort}(x), \mathsf{cons}'(\mathsf{quicksort}(x)))\\
\mathsf{cons}'(\mathsf{cons}'(w)) &\sim \mathsf{cons}'(\mathsf{cons}'(w))
\end{aligned}
$$

These constraints are satisfied by the rpo using a precedence with quicksort $\sqsupset$ app $\sqsupset$ cons$'$.

References

1. T. Arts and J. Giesl, Automatically Proving Termination where Simplification Orderings Fail, in *Proc. TAPSOFT '97*, LNCS 1214, 261-272, 1997.
2. T. Arts and J. Giesl, Modularity of Termination Using Dependency Pairs, in *Proc. RTA '98*, LNCS 1379, 226-240, 1998.
3. T. Arts and J. Giesl, Termination of Term Rewriting Using Dependency Pairs, *Theoretical Computer Science*, 236:133-178, 2000.
4. T. Arts, System Description: The Dependency Pair Method, in *Proc. RTA '00*, LNCS 1833, 261-264, 2000.
5. F. Baader and W. Snyder, Unification Theory, in *Handbook of Automated Reasoning*, J. A. Robinson and A. Voronkov (eds.), Elsevier. To appear.
6. A. Ben Cherifa and P. Lescanne, Termination of Rewriting Systems by Polynomial Interpretations and its Implementation, *Sc. Comp. Prog.*, 9(2):137-159, 1987.
7. CiME 2. Pre-release available at http://www.lri.fr/~demons/cime-2.0.html.
8. C. Delor and L. Puel, Extension of the Associative Path Ordering to a Chain of Associative Commutative Symbols, in *Proc. RTA '93*, LNCS 690, 389-404, 1993.
9. N. Dershowitz, Termination of Rewriting, *J. Symbolic Computation*, 3:69-116, 1987.
10. N. Dershowitz and J.-P. Jouannaud, Rewrite Systems, *Handbook of Theoretical Computer Science*, Vol. B, pp. 243-320, North-Holland 1990.
11. M. C. F. Ferreira, Dummy Elimination in Equational Rewriting, in *Proc. RTA '96*, LNCS 1103, 78-92, 1996.
12. J.-P. Jouannaud and H. Kirchner, Completion of a Set of Rules Modulo a Set of Equations, *SIAM Journal on Computing*, 15(4):1155-1194, 1986.
13. S. Kamin and J.-J. Lévy, Two Generalizations of the Recursive Path Ordering, Unpublished Note, Dept. of Computer Science, University of Illinois, Urbana, IL, 1980.
14. D. Kapur and G. Sivakumar, A Total Ground Path Ordering for Proving Termination of AC-Rewrite Systems, in *Proc. RTA '97*, LNCS 1231, 142-156, 1997.
15. D. Kapur and G. Sivakumar, Proving Associative-Commutative Termination Using RPO-Compatible Orderings, in *Proc. Automated Deduction in Classical and Non-Classical Logics*, LNAI 1761, 40-62, 2000.
16. K. Kusakari and Y. Toyama, On Proving AC-Termination by AC-Dependency Pairs, Research Report IS-RR-98-0026F, School of Information Science, JAIST, Japan, 1998. Revised version in K. Kusakari, Termination, AC-Termination and Dependency Pairs of Term Rewriting Systems, PhD Thesis, JAIST, Japan, 2000.
17. J.-P. Jouannaud and M. Muñoz, Termination of a Set of Rules Modulo a Set of Equations, in *Proc. 7th CADE*, LNCS 170, 175-193, 1984.
18. C. Marché and X. Urbain, Termination of Associative-Commutative Rewriting by Dependency Pairs, in *Proc. RTA '98*, LNCS 1379, 241-255, 1998.
19. H. Ohsaki, A. Middeldorp, and J. Giesl, Equational Termination by Semantic Labelling, in *Proc. CSL '00*, LNCS 1862, 457-471, 2000.
20. G. E. Peterson and M. E. Stickel, Complete Sets of Reductions for Some Equational Theories, *Journal of the ACM*, 28(2):233-264, 1981.
21. A. Rubio and R. Nieuwenhuis, A Total AC-Compatible Ordering based on RPO, *Theoretical Computer Science*, 142:209-227, 1995.

22. A. Rubio, A Fully Syntactic *AC*-RPO, *Proc. RTA-99*, LNCS 1631, 133-147, 1999.
23. J. Steinbach, Simplification Orderings: History of Results, *Fundamenta Informaticae*, 24:47-87, 1995.
24. L. Vigneron, Positive Deduction modulo Regular Theories, in *Proc. CSL '95*, LNCS 1092, 468-485, 1995.
25. H. Zantema, Termination of Term Rewriting by Semantic Labelling, *Fundamenta Informaticae*, 24:89–105, 1995.