A Relative Dependency Pair Framework*

Christian Sternagel[1] and René Thiemann[2]

1 **School of Information Science**
Japan Advanced Institute of Science and Technology, Japan
c-sterna@jaist.ac.jp
2 **Institute of Computer Science**
University of Innsbruck, Austria
rene.thiemann@uibk.ac.at

1 Introduction

Relative rewriting and the dependency pair framework (DP framework) are two strongly related termination methods. In both formalisms we consider whether the combination of two TRSs allows an infinite derivation:

- Relative termination of $\mathcal{R}/\mathcal{S}$ can be defined as strong normalization of $\to_{\mathcal{R}} \cdot \to_{\mathcal{S}}^*$.
- Finiteness of a DP problem $(\mathcal{P}, \mathcal{R})$ can be defined as strong normalization of $\xrightarrow{\epsilon}_{\mathcal{P}} \cdot \to_{\mathcal{R}}^*$ where $\xrightarrow{\epsilon}$ allows steps only at the top. Moreover, *minimality* can be incorporated by requiring that all terms are terminating w.r.t. $\mathcal{R}$.

The above definitions have two orthogonal distinctions of rules. In both formalisms there are *strict* and *weak* rules: $\mathcal{P}$ and $\mathcal{R}$ are the strict rules of $(\mathcal{P}, \mathcal{R})$ and $\mathcal{R}/\mathcal{S}$, respectively, while $\mathcal{R}$ and $\mathcal{S}$ are the respective weak rules. In the DP framework, there is the additional distinction between rules that may only be applied at the top ($\mathcal{P}$) and those that can be applied at arbitrary positions ($\mathcal{R}$).

Note that the restriction to top rewriting is an important advantage for proving termination in the DP framework. It allows to use non-monotone orders for orienting the strict rules. Furthermore, if minimality is considered, we can use termination techniques (e.g., usable rules or the subterm criterion) that are not available for relative rewriting.

However, also relative rewriting has some advantages which are currently not available in the DP framework: Geser showed that it is always possible to split a relative termination problem into two parts [4]. Relative termination of $(\mathcal{R}_s \cup \mathcal{R}_w)/(\mathcal{S}_s \cup \mathcal{S}_w)$ can be shown by relative termination of both $(\mathcal{R}_s \cup \mathcal{S}_s)/(\mathcal{R}_w \cup \mathcal{S}_w)$ and $\mathcal{R}_w/\mathcal{S}_w$. Hence, it is possible to show (in a first relative termination proof) that the strict rules $\mathcal{R}_s \cup \mathcal{S}_s$ cannot occur infinitely often and afterwards continue (in a second relative termination proof) with the problem $\mathcal{R}_w/\mathcal{S}_w$. A major advantage of this approach is that in the first proof we can apply arbitrary techniques which may increase the size of the TRSs drastically (e.g., semantic labeling [11]), or which may even be incomplete (e.g., string reversal in combination with innermost rewriting, where by reversing the rules we have to forget about the strategy). As long as relative termination of $(\mathcal{R}_s \cup \mathcal{S}_s)/(\mathcal{R}_w \cup \mathcal{S}_w)$ could be proven, we can afterwards continue independently with the problem $\mathcal{R}_w/\mathcal{S}_w$.

Such a split is currently not possible in the DP framework since there are no top weak rules and also no strict rules which can be applied everywhere.

In this paper we generalize the DP framework to a relative DP framework, where such a split is possible. To this end, we consider DP problems of the form $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$, where we have strict and weak, top and non-top rules. (This kind of DP problems were first suggested

* This research is supported by the Austrian Science Fund (FWF): P22767, J3202.

by Jörg Endrullis at the *Workshop on the Certification of Termination Proofs* in 2007 and are already used in his termination tool Jambox [3]. Unfortunately the suggestion did not get much attention back then and we are not aware of any publications on this topic.) In this way, problems that occur in combination with semantic labeling and dependency pairs—which can otherwise be solved by using a dedicated semantics for DP problems [9]—can easily be avoided. Furthermore, the new framework is more general than [9] since it also solves some problems that occur when using other termination techniques like uncurrying [6, 8].

2 A Relative Dependency Pair Framework

We assume familiarity with term rewriting [2] and the DP framework [5].

▶ **Definition 1.** A *relative dependency pair problem* $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$ is a quadruple of TRSs with *pairs* $\mathcal{P} \cup \mathcal{P}_w$ (where pairs from $\mathcal{P}$ are called *strict* and those of $\mathcal{P}_w$ *weak*) and *rules* $\mathcal{R} \cup \mathcal{R}_w$ (where rules of $\mathcal{R}$ are called *strict* and those of $\mathcal{R}_w$ *weak*).

For relative DPPs the notion of chains and finiteness is adapted in the following way.

▶ **Definition 2.** An infinite sequence of pairs $s_1 \to t_1$, $s_2 \to t_2$, ... forms a $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$-*chain* if there exists a corresponding sequence of substitutions σ_1, σ_2, ... such that

$$s_i \to t_i \in \mathcal{P} \cup \mathcal{P}_w \text{ for all } i \tag{1}$$
$$t_i\sigma_i \to^*_{\mathcal{R}\cup\mathcal{R}_w} s_{i+1}\sigma_{i+1} \text{ for all } i \tag{2}$$
$$s_i \to t_i \in \mathcal{P} \text{ or } t_i\sigma_i \to^*_{\mathcal{R}\cup\mathcal{R}_w} \cdot \to_{\mathcal{R}} \cdot \to^*_{\mathcal{R}\cup\mathcal{R}_w} s_{i+1}\sigma_{i+1} \text{ for infinitely many } i \tag{3}$$

For minimal chains, we additionally require

$$\mathsf{SN}_{\mathcal{R}\cup\mathcal{R}_w}(t_i\sigma_i) \text{ for all } i \tag{4}$$

A relative DPP $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$ is *finite*, iff there is no minimal infinite $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$-chain.

Hence, a (minimal) $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$-chain is like a (minimal) $(\mathcal{P} \cup \mathcal{P}_w, \mathcal{R} \cup \mathcal{R}_w)$-chain—as defined in [1]—with the additional demand that there are infinitely many strict steps using $\mathcal{P}$ or $\mathcal{R}$. It is easy to see that $(\mathcal{P}, \mathcal{R})$-chains can be expressed in the new framework.

▶ **Lemma 3.** *The DP problem $(\mathcal{P}, \mathcal{R})$ is finite iff there exists a minimal $(\mathcal{P}, \mathcal{R})$-chain iff there exists a minimal $(\mathcal{P}, \emptyset, \emptyset, \mathcal{R})$-chain iff the relative DPP $(\mathcal{P}, \emptyset, \emptyset, \mathcal{R})$ is finite.*

Note that in contrast to DPPs $(\mathcal{P}, \mathcal{R})$, for relative DPPs, $\mathcal{P} = \emptyset$ does not imply finiteness of $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$.

▶ **Example 4.** The relative DPP $(\emptyset, \{\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{b})\}, \{\mathsf{b} \to \mathsf{a}\}, \emptyset)$ is not finite.

However, a sufficient criterion for finiteness is that there are either no pairs, or that there are neither strict pairs nor strict rules.

▶ **Lemma 5** (Trivially finite relative DPPs). *If $\mathcal{P} \cup \mathcal{P}_w = \emptyset$ or $\mathcal{P} \cup \mathcal{R} = \emptyset$ then $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$ is finite.*

3 Processors in the Relative Dependency Pair Framework

Processors and soundness of processors in the relative DP framework are defined as in the DP framework, but operate on relative DPPs instead of DPPs (a processor is sound if finiteness of all resulting relative DPPs implies finiteness of the given relative DPP).

Note that most processors can easily be adapted to the new framework where most often it suffices to treat the relative DPP $(\mathcal{P}, \mathcal{P}_w, \mathcal{R}, \mathcal{R}_w)$ as the DPP $(\mathcal{P} \cup \mathcal{P}_w, \mathcal{R} \cup \mathcal{R}_w)$.

However, when starting with the initial relative DPP $(DP(\mathcal{R}), \emptyset, \emptyset, \mathcal{R})$ it is questionable whether we ever reach relative DPPs containing weak pairs or strict rules. If this is not the case, then our generalization would be useless. Therefore, in the following we give evidence that the additional flexibility is beneficial.

Easy examples are semantic labeling and uncurrying. Both techniques are transformational techniques where each original step is transformed into one main transformed step together with some auxiliary steps. For the auxiliary steps one uses auxiliary pairs and rules (the decreasing rules and the uncurrying rules, respectively). If there are auxiliary pairs $\mathcal{P}_{aux}$, then in the DP framework, $\mathcal{P}_{aux}$ can only be added as strict pairs, whereas in the relative DP framework, we can add $\mathcal{P}_{aux}$ to the weak pairs, and hence we do not have to delete all pairs of $\mathcal{P}_{aux}$ anymore for proving finiteness.

As another example, we consider top-uncurrying of [8, Def. 19], where some rules $\mathcal{R}$ are turned into pairs. Again, in the DP framework this would turn the weak rules $\mathcal{R}$ into strict pairs, which in fact would demand that we prove termination of $\mathcal{R}$ twice: Once via the original DPs for $\mathcal{R}$, and a second time after the weak rules of $\mathcal{R}$ have been converted into strict pairs. For example, in [8, Ex. 21] termination of the minus-rules is proven twice. This is no longer required in the relative DP framework where one can just turn the weak rules $\mathcal{R}$ into weak pairs $\mathcal{R}$.

Finally, in the relative DP framework we can apply the split technique known from relative rewriting.

▶ **Definition 6** (Split processor). The relative DPP $(\mathcal{P}_s^1 \cup \mathcal{P}_w^1, \mathcal{P}_s^2 \cup \mathcal{P}_w^2, \mathcal{R}_s^1 \cup \mathcal{R}_w^1, \mathcal{R}_s^2 \cup \mathcal{R}_w^2)$ is finite if both $(\mathcal{P}_s^1 \cup \mathcal{P}_s^2, \mathcal{P}_w^1 \cup \mathcal{P}_w^2, \mathcal{R}_s^1 \cup \mathcal{R}_s^2, \mathcal{R}_w^1 \cup \mathcal{R}_w^2)$ and $(\mathcal{P}_w^1, \mathcal{P}_w^2, \mathcal{R}_w^2, \mathcal{R}_w^2)$ are finite.

A more instructive way of putting the above definition for termination tool authors that are used to standard DP problems is as follows. Start from the relative DPP $(\mathcal{P}, \emptyset, \emptyset, \mathcal{R})$. Identify pairs $\mathcal{P}'$ and rules $\mathcal{R}'$ that should be deleted. Then use the split processor to obtain the two relative DPPs $(\mathcal{P}', \mathcal{P} \setminus \mathcal{P}', \mathcal{R}', \mathcal{R} \setminus \mathcal{R}')$ and $(\mathcal{P} \setminus \mathcal{P}', \emptyset, \emptyset, \mathcal{R} \setminus \mathcal{R}')$.

Clearly, the split processor can be used to obtain relative DPPs with strict rules and weak pairs, but the question is how to apply it. We give two possibilities.

Semantic labeling is often used in a way, that after labeling one tries to remove all labeled variants of some rules $\mathcal{R}_s$ and pairs $\mathcal{P}_s$, and afterwards removes the labels again to continue on a smaller unlabeled problem.

▶ **Example 7.** Consider a DP problem $p_1 = (\{1, 2\}, \{3\})$. After applying semantic labeling, all pairs and rules occur in labeled variants $1.x$, $2.x$, and $3.x$, so the resulting DP problem might look like $(\{1.a, 1.b, 2.a, 2.b\}, \{3.a, 3.b, 3.c\})$. Applying standard techniques to remove pairs and rules one might get stuck at $p_2 = (\{2.a, 2.b\}, \{3.a, 3.c\})$. Although p_1 contains less rules than p_2, p_2 is somehow simpler since all rules $1.x$ have been removed. And indeed, after applying unlabeling on p_2 the resulting DP problem $p_3 = (\{2\}, \{3\})$ is smaller than p_1.

Since the removal of labels is problematic for soundness, a special semantics was developed in [9]. This is no longer required in the relative DP framework. After $\mathcal{R}_s$ and $\mathcal{P}_s$ have

been identified, one just applies the split processor to transform $(\mathcal{P}, \emptyset, \emptyset, \mathcal{R})$ into $(\mathcal{P}_s, \mathcal{P} \setminus \mathcal{P}_s, \mathcal{R}_s, \mathcal{R} \setminus \mathcal{R}_s)$ and $(\mathcal{P} \setminus \mathcal{P}_s, \emptyset, \emptyset, \mathcal{R} \setminus \mathcal{R}_s)$. The proof that all labeled variants of rules in $\mathcal{R}_s$ and pairs in $\mathcal{P}_s$ can be dropped, proves finiteness of the first problem, and one can continue on the latter problem without having to apply unlabeling.

▶ **Example 8.** Using split, we can restructure the proof of Example 7 without using unlabeling: We know that in the end, we only get rid of pair 1. Hence, we apply split on p_1 to obtain p_3 and $p_4 = (\{1\}, \{2\}, \emptyset, \{3\})$. Thus, we get the same remaining problem p_3 if we can prove finiteness of p_4. But this can be done by replaying the proof steps in Example 7. Applying the same labeling as before, we obtain $p_5 = (\{1.a, 1.b\}, \{2.a, 2.b\}, \emptyset, \{3.a, 3.b, 3.c\})$. Removing pairs and rules as before, we simplify p_5 to $p_6 = (\emptyset, \{2.a, 2.b\}, \emptyset, \{3.a, 3.c\})$ and this relative DP problem is trivially finite by Lemma 5.

Note that using [9] it was only possible to revert the labeling, but not to revert other techniques like the closure under flat contexts which is used in combination with root-labeling [7]. However, using the split processor this is also easily possible, since one just has to apply the split processor before applying the closure under flat contexts.

A further advantage of the relative DP framework in comparison to [9] can be seen in the combination of semantic labeling with the dependency graph processor.

▶ **Example 9.** Consider a DP problem $p_1 = (\{1, 2\}, \{3, 4\})$ which is transformed into $(\{1.a, 1.b, 2.a, 2.b\}, \{3.a, 3.b, 4.a, 4.b\})$ using semantic labeling. Applying the dependency graph and reduction pairs yields two remaining DP problems $p_2 = (\{2.a\}, \{4.a\})$ and $p_3 = (\{2.b\}, \{3.a, 4.b\})$. Using unlabeling we have to prove finiteness of the two remaining problems $p_4 = (\{2\}, \{4\})$ and $p_5 = (\{2\}, \{3, 4\})$. Note that finiteness of p_5 does not imply finiteness of p_4, so one indeed has to perform two proofs.

However, when using the split processor, only p_5 remains: we observe from p_2 and p_3 that only pair 1 could be removed. So, we start to split p_1 into p_5 and $p_6 = (\{1\}, \{2\}, \emptyset, \{3, 4\})$. Labeling p_6 yields $(\{1.a, 1.b\}, \{2.a, 2.b\}, \emptyset, \{3.a, 3.b, 4.a, 4.b\})$ which is simplified to the two problems $(\emptyset, \{2.a\}, \emptyset, \{4.a\})$ and $(\emptyset, \{2.b\}, \emptyset, \{3.a, 4.b\})$ with the same techniques as before. Both problems are trivially finite by Lemma 5.

Other Techniques may also take advantage of the split processor. For example, the dependency pair transformation of narrowing [1, 5] is not complete in the innermost case but might help to remove some pairs and rules. If it turns out that after some narrowing steps some original pairs and rules can be removed, then one can just insert a split processor before narrowing has been performed. In this way one has obtained progress in proving finiteness and in the remaining system the potential incomplete narrowing steps have not been applied. In other words, the split processor allows to apply incomplete techniques without losing overall completeness.

4 Conclusions and Future Work

We presented the relative DP framework which generalizes the existing DP framework by allowing weak pairs and strict rules. It forms the basis of our proof checker CeTA (since version 2.0) [10] where we additionally integrated innermost rewriting (in the form of $\mathcal{Q}$-restricted rewriting) [5]. One of the main features of the new framework is the possibility to split a DP problem into two DP problems which can be treated independently. Examples to illustrate the new features are provided in the IsaFoR-repository (e.g., `div_uncurry.proof.xml` uses

weak pairs for uncurrying, and in `secret_07_trs_4_top.proof.xml` the split processor is used to avoid unlabeling).

It is an obvious question, whether the relative DP framework can be used to characterize relative termination. In a preliminary version we answered this question positively by presenting a theorem that $\mathcal{R}/\mathcal{S}$ is relative terminating iff there is no infinite $(DP(\mathcal{R}), DP(\mathcal{S}), \mathcal{R}, \mathcal{S})$-chain. However, it was detected that the corresponding proof contained a gap (it was the only proof that we did not formalize in Isabelle/HOL) and that the whole theorem did not hold (by means of a counterexample).

An interesting direction for future work is to unify termination (via relative DP problems) with relative termination. One reason is that this would allow to reduce the formalization effort, since results for termination are expected to be corollaries carrying over from relative termination.

Acknowledgments We would like to thank Jörg Endrullis and an anonymous referee for pointing out that our attempt to characterize relative termination using the relative DP framework is unsound. It remains as interesting open problem to give such a characterization.

References

1 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999.
3 Jörg Endrullis. Jambox. Available at `http://joerg.endrullis.de`.
4 Alfons Geser. *Relative Termination*. PhD thesis, University of Passau, Germany, 1990.
5 Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *LPAR'04*, volume 3452 of *LNAI*, pages 301–331. Springer, 2004.
6 Nao Hirokawa, Aart Middeldorp, and Harald Zankl. Uncurrying for termination. In *LPAR'08*, volume 5330 of *LNAI*, pages 667–681. Springer, 2008.
7 Christian Sternagel and Aart Middeldorp. Root-Labeling. In *RTA'08*, volume 5117 of *LNCS*, pages 336–350. Springer, 2008.
8 Christian Sternagel and René Thiemann. Generalized and formalized uncurrying. In *FroCoS'11*, volume 6989 of *LNAI*, pages 243–258. Springer, 2011.
9 Christian Sternagel and René Thiemann. Modular and certified semantic labeling and unlabeling. In *RTA'11*, volume 10 of *LIPIcs*, pages 329–344. Dagstuhl, 2011.
10 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *TPHOLs'09*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009.
11 Hans Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.