

# A Dependency Pair Framework for Relative Termination of Term Rewriting\*

Jan-Christoph Kassing<sup>✉</sup>, Grigory Vartanyan<sup>✉</sup>, and Jürgen Gies<sup>✉</sup>

LuFG Informatik 2, RWTH Aachen University, Aachen, Germany<sup>1</sup>

**Abstract.** *Dependency pairs* are one of the most powerful techniques for proving termination of term rewrite systems (TRSs), and they are used in almost all tools for termination analysis of TRSs. Problem #106 of the RTA List of Open Problems asks for an adaption of dependency pairs for *relative termination*. Here, infinite rewrite sequences are allowed, but one wants to prove that a certain subset of the rewrite rules cannot be used infinitely often. Dependency pairs were recently adapted to *annotated dependency pairs* (ADPs) to prove almost-sure termination of probabilistic TRSs. In this paper, we develop a novel adaption of ADPs for relative termination. We implemented our new ADP framework in our tool AProVE and evaluate it in comparison to state-of-the-art tools for relative termination of TRSs.

## 1 Introduction

Termination is an important topic in program verification. There is a wealth of work on automatic termination analysis of term rewrite systems (TRSs) which can also be used to analyze termination of programs in many other languages. Essentially all current termination tools for TRSs (e.g., AProVE [11], NaTT [31], MU-TERM [13], T<sub>T</sub>2 [23], etc.) use *dependency pairs* (DPs) [1, 9, 10, 14, 15].

A combination of two TRSs  $\mathcal{R}$  and  $\mathcal{R}^=$  is considered to be “*relatively terminating*” if there is no rewrite sequence that uses infinitely many steps with rules from  $\mathcal{R}$  (whereas rules from  $\mathcal{R}^=$  may be used infinitely often). Relative termination of TRSs has been studied since decades [6], and approaches based on relative rewriting are used for many different applications, e.g., [5, 16, 17, 21, 22, 25, 26, 29, 32].

However, while techniques and tools for analyzing ordinary termination of TRSs are very powerful due to the use of DPs, most approaches for automated analysis of relative termination are quite restricted in power. Therefore, one of the largest open problems regarding DPs is Problem #106 of the RTA List of Open Problems [4]: *Can we use the dependency pair method to prove relative termination?* A first major step towards an answer to this question was presented in [18] by giving criteria for  $\mathcal{R}$  and  $\mathcal{R}^=$  that allow the use of ordinary DPs for relative termination.

Recently, we adapted DPs in order to analyze probabilistic innermost term rewriting, by using so-called *annotated dependency pairs* (ADPs) [20] or *dependency tuples* (DTs) [19] (which were originally proposed for innermost complexity analysis of TRSs [27]).<sup>1</sup> In these adaptations, one considers all *defined* function symbols in the

\* funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 235950644 (Project GI 274/6-2) and DFG Research Training Group 2236 UnRAVeL.

<sup>1</sup> As shown in [20], using ADPs instead of DTs leads to a more elegant, more powerful, and less complicated framework, and to completeness of the underlying *chain criterion*.

right-hand side of a rule at once, whereas ordinary DPs consider them separately. 11

In this paper, we show that considering the defined symbols on right-hand sides 12  
separately (as for DPs) does not suffice for relative termination. On the other hand, 12  
we do not need to consider all of them at once either. Instead, we introduce a new 12  
definition of ADPs that is suitable for relative termination and develop a correspond- 12  
ing ADP framework for automated relative termination proofs of TRSs. Moreover, 12  
while ADPs and DTs were only applicable for *innermost* rewriting in [19, 20, 27], 12  
we now adapt ADPs to *full* (relative) rewriting, i.e., we do not impose any specific 13  
evaluation strategy. So while [18] presented conditions under which the *ordinary* 13  
*classical* DP framework can be used to prove relative termination, in this paper we 13  
develop the first *specific* DP framework for relative termination. 13

**Structure:** We start with preliminaries on relative rewriting in Sect. 2. In Sect. 3 32  
we recapitulate the core processors of the DP framework. Moreover, we state the 33  
main results of [18] on using ordinary DPs for relative termination. Afterwards, 32  
we introduce our novel notion of *annotated dependency pairs* for relative termina- 32  
tion in Sect. 4 and present a corresponding new ADP framework in Sect. 5. We 32  
implemented our framework in the tool AProVE and in Sect. 6, we evaluate our im- 32  
plementation in comparison to other state-of-the-art tools. All proofs can be found 32  
in App. A. 32

## 2 Relative Term Rewriting 0

We assume familiarity with term rewriting [2] and regard (finite) TRSs over a (finite) 15  
signature  $\Sigma$  and a set of variables  $\mathcal{V}$ . 15

*Example 1.* Consider the following TRS  $\mathcal{R}_{\text{divL}}$ , where  $\text{divL}(x, xs)$  computes the num- 16  
ber that results from dividing  $x$  by each element of the list  $xs$ . As usual, natural 16  
numbers are represented by the function symbols  $\mathcal{O}$  and  $s$ , and lists are represented 16  
via  $\text{nil}$  and  $\text{cons}$ . Then  $\text{divL}(s^{24}(\mathcal{O}), \text{cons}(s^4(\mathcal{O}), \text{cons}(s^3(\mathcal{O}), \text{nil})))$  evaluates to  $s^2(\mathcal{O})$ , 16  
because  $(24/4)/3 = 2$ . Here,  $s^2(\mathcal{O})$  stands for  $s(s(\mathcal{O}))$ , etc. 16

$$\begin{array}{ll} \text{minus}(x, \mathcal{O}) \rightarrow x & (1) \quad 20 \quad \text{div}(s(x), s(y)) \rightarrow s(\text{div}(\text{minus}(x, y), s(y))) \quad (4) \quad 151 \\ \text{minus}(s(x), s(y)) \rightarrow \text{minus}(x, y) & (2) \quad 20 \quad \text{divL}(x, \text{nil}) \rightarrow x \quad (5) \quad 151 \\ \text{div}(x, s(\mathcal{O})) \rightarrow x & (3) \quad 20 \quad \text{divL}(x, \text{cons}(y, xs)) \rightarrow \text{divL}(\text{div}(x, y), xs) \quad (6) \quad 151 \end{array}$$

A TRS  $\mathcal{R}$  induces a *rewrite relation*  $\rightarrow_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$  on terms where 22  
 $s \rightarrow_{\mathcal{R}} t$  holds if there is a position  $\pi \in \text{pos}(s)$ , a rule  $\ell \rightarrow r \in \mathcal{R}$ , and a substitution 23  
 $\sigma$  such that  $s|_{\pi} = \ell\sigma$  and  $t = s[r\sigma]_{\pi}$ . For example, we have  $\text{minus}(s(\mathcal{O}), s(\mathcal{O})) \rightarrow_{\mathcal{R}_{\text{divL}}} 23$   
 $\text{minus}(\mathcal{O}, \mathcal{O}) \rightarrow_{\mathcal{R}_{\text{divL}}} \mathcal{O}$ . We call a TRS  $\mathcal{R}$  *strongly normalizing (SN)* or *terminating* 23  
if  $\rightarrow_{\mathcal{R}}$  is well founded. Using the DP framework, one can easily prove that  $\mathcal{R}_{\text{divL}}$  is 23  
SN (see Sect. 3.1). In particular, in each application of the recursive  $\text{divL}$ -rule (6), 23  
the length of the list in  $\text{divL}$ 's second argument is decreased by one. 23

In the relative setting, one considers two TRSs  $\mathcal{R}$  and  $\mathcal{R}^-$ . We say that  $\mathcal{R}$  is 25  
*relatively strongly normalizing* w.r.t.  $\mathcal{R}^-$  (i.e.,  $\mathcal{R}/\mathcal{R}^-$  is SN) if there is no infinite 25  
 $(\rightarrow_{\mathcal{R}} \cup \rightarrow_{\mathcal{R}^-})$ -rewrite sequence that uses an infinite number of  $\rightarrow_{\mathcal{R}}$ -steps. We refer 25  
to  $\mathcal{R}$  as the *main* and  $\mathcal{R}^-$  as the *base* TRS. 25

*Example 2.* For example, let  $\mathcal{R}_{\text{divL}}$  be the *main* TRS. Since the order of the list elements does not affect the termination of  $\mathcal{R}_{\text{divL}}$ , this algorithm also works for multisets. To abstract lists to multisets, we add the *base* TRS  $\mathcal{R}_{\text{mset}}^{\text{=}} = \{(7)\}$ .

$$\text{cons}(x, \text{cons}(y, zs)) \rightarrow \text{cons}(y, \text{cons}(x, zs)) \quad (7) \quad 151$$

$\mathcal{R}_{\text{mset}}^{\text{=}}$  is non-terminating, since it can switch elements in a list arbitrarily often. However,  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset}}^{\text{=}}$  is SN as each application of Rule (6) still reduces the list length.

We will use the following four examples to show why a naive adaption of dependency pairs does not work in the relative setting and why we need our new notion of *annotated dependency pairs*. These examples represent different types of infinite rewrite sequences that lead to non-termination in the relative setting: *redex-duplicating*, *redex-creating* (or “emitting”), and *ordinary infinite sequences*.

*Example 3 (Redex-Duplicating).* Consider the TRSs  $\mathcal{R}_1 = \{a \rightarrow b\}$  and  $\mathcal{R}_1^{\text{=}} = \{f(x) \rightarrow d(f(x), x)\}$ .  $\mathcal{R}_1/\mathcal{R}_1^{\text{=}}$  is not SN due to the infinite rewrite sequence  $f(a) \rightarrow_{\mathcal{R}_1^{\text{=}}} d(f(a), a) \rightarrow_{\mathcal{R}_1} d(f(a), b) \rightarrow_{\mathcal{R}_1^{\text{=}}} d(d(f(a), a), b) \rightarrow_{\mathcal{R}_1} d(d(f(a), b), b) \rightarrow_{\mathcal{R}_1^{\text{=}}} \dots$ . The reason is that  $\mathcal{R}_1^{\text{=}}$  can be used to duplicate an arbitrary  $\mathcal{R}_1$ -redex infinitely often.

*Example 4 (Redex-Creating on Parallel Position).* Next, consider  $\mathcal{R}_2 = \{a \rightarrow b\}$  and  $\mathcal{R}_2^{\text{=}} = \{f \rightarrow d(f, a)\}$ .  $\mathcal{R}_2/\mathcal{R}_2^{\text{=}}$  is not SN as we have the infinite rewrite sequence  $f \rightarrow_{\mathcal{R}_2^{\text{=}}} d(f, a) \rightarrow_{\mathcal{R}_2} d(f, b) \rightarrow_{\mathcal{R}_2^{\text{=}}} d(d(f, a), b) \rightarrow_{\mathcal{R}_2} d(d(f, b), b) \rightarrow_{\mathcal{R}_2^{\text{=}}} \dots$ . Here,  $\mathcal{R}_2^{\text{=}}$  can create an  $\mathcal{R}_2$ -redex infinitely often (where in the right-hand side  $d(f, a)$  of  $\mathcal{R}_2^{\text{=}}$ 's rule, the  $\mathcal{R}_2^{\text{=}}$ -redex  $f$  and the created  $\mathcal{R}_2$ -redex  $a$  are on parallel positions).

*Example 5 (Redex-Creating on Position Above).* Let  $\mathcal{R}_3 = \{a(x) \rightarrow b(x)\}$  and  $\mathcal{R}_3^{\text{=}} = \{f \rightarrow a(f)\}$ .  $\mathcal{R}_3/\mathcal{R}_3^{\text{=}}$  is not SN as we have  $f \rightarrow_{\mathcal{R}_3^{\text{=}}} a(f) \rightarrow_{\mathcal{R}_3} b(f) \rightarrow_{\mathcal{R}_3^{\text{=}}} b(a(f)) \rightarrow_{\mathcal{R}_3} b(b(f)) \rightarrow_{\mathcal{R}_3^{\text{=}}} \dots$ , i.e., again  $\mathcal{R}_3^{\text{=}}$  can be used to create an  $\mathcal{R}_3$ -redex infinitely often. In the right-hand side  $a(f)$  of  $\mathcal{R}_3^{\text{=}}$ 's rule, the position of the created  $\mathcal{R}_3$ -redex  $a(\dots)$  is above the position of the  $\mathcal{R}_3^{\text{=}}$ -redex  $f$ .

*Example 6 (Ordinary Infinite).* Finally, consider  $\mathcal{R}_4 = \{a \rightarrow b\}$  and  $\mathcal{R}_4^{\text{=}} = \{b \rightarrow a\}$ . Here, the base TRS  $\mathcal{R}_4^{\text{=}}$  can neither duplicate nor create an  $\mathcal{R}_4$ -redex infinitely often, but in combination with the main TRS  $\mathcal{R}_4$  we obtain the infinite rewrite sequence  $a \rightarrow_{\mathcal{R}_4} b \rightarrow_{\mathcal{R}_4^{\text{=}}} a \rightarrow_{\mathcal{R}_4} b \rightarrow_{\mathcal{R}_4^{\text{=}}} \dots$ . Thus,  $\mathcal{R}_4/\mathcal{R}_4^{\text{=}}$  is not SN.

### 3 DP Framework 0

We first recapitulate dependency pairs for ordinary (non-relative) rewriting in Sect. 3.1 and summarize existing results on DPs for relative rewriting in Sect. 3.2.

#### 3.1 Dependency Pairs for Ordinary Term Rewriting 107

We recapitulate DPs and the two most important processors of the DP framework and refer to, e.g., [1, 9, 10, 14, 15] for more details. As an example, we show how to prove termination of  $\mathcal{R}_{\text{divL}}$  without the base  $\mathcal{R}_{\text{mset}}^{\text{=}}$ . We decompose the signature  $\Sigma = \mathcal{C} \uplus \mathcal{D}$  of a TRS  $\mathcal{R}$  such that  $f \in \mathcal{D}$  if  $f = \text{root}(\ell)$  for some rule  $\ell \rightarrow r \in \mathcal{R}$ . The

symbols in  $\mathcal{C}$  and  $\mathcal{D}$  are called *constructors* and *defined symbols* of  $\mathcal{R}$ , respectively. For every  $f \in \mathcal{D}$ , we introduce a fresh *annotated symbol*  $f^\#$  of the same arity. Let  $\mathcal{D}^\#$  denote the set of all annotated symbols, and  $\Sigma^\# = \Sigma \uplus \mathcal{D}^\#$ . To ease readability, we often use capital letters like  $F$  instead of  $f^\#$ . For any term  $t = f(t_1, \dots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$  with  $f \in \mathcal{D}$ , let  $t^\# = f^\#(t_1, \dots, t_n)$ . For a rule  $\ell \rightarrow r$  and each subterm  $t$  of  $r$  with defined root symbol, one obtains a *dependency pair*  $\ell^\# \rightarrow t^\#$ . Let  $\mathcal{DP}(\mathcal{R})$  denote the set of all dependency pairs of the TRS  $\mathcal{R}$ .

*Example 7.* For  $\mathcal{R}_{\text{divL}}$  from Ex. 1, we obtain the following five dependency pairs.

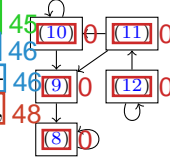
$$\begin{array}{ll} \text{M}(s(x), s(y)) \rightarrow \text{M}(x, y) & (8) \\ \text{D}(s(x), s(y)) \rightarrow \text{M}(x, y) & (9) \\ \text{D}(s(x), s(y)) \rightarrow \text{D}(m(x, y), s(y)) & (10) \\ \text{DL}(x, \text{cons}(y, xs)) \rightarrow \text{D}(x, y) & (11) \\ \text{DL}(x, \text{cons}(y, xs)) \rightarrow \text{DL}(\text{div}(x, y), xs) & (12) \end{array}$$

The DP framework operates on *DP problems*  $(\mathcal{P}, \mathcal{R})$  where  $\mathcal{P}$  is a (finite) set of DPs, and  $\mathcal{R}$  is a (finite) TRS. A (possibly infinite) sequence  $t_0, t_1, t_2, \dots$  with  $t_i \xrightarrow{\varepsilon} t_{i+1}$  for all  $i$  is a  $(\mathcal{P}, \mathcal{R})$ -chain. Here,  $\xrightarrow{\varepsilon}$  denotes rewrite steps at the root. Intuitively, a chain represents subsequent “function calls” in evaluations. Between two function calls (corresponding to steps with  $\mathcal{P}$ , called **p**-steps) one can evaluate the arguments using arbitrary many steps with  $\mathcal{R}$  (called **r**-steps). So **r**-steps are rewrite steps that are needed in order to enable another **p**-step at a position above later on. For example,  $\text{DL}(s(\mathcal{O}), \text{cons}(s(\mathcal{O}), \text{nil})), \text{DL}(s(\mathcal{O}), \text{nil})$  is a  $(\mathcal{DP}(\mathcal{R}_{\text{divL}}), \mathcal{R}_{\text{divL}})$ -chain, as  $\text{DL}(s(\mathcal{O}), \text{cons}(s(\mathcal{O}), \text{nil})) \xrightarrow{\varepsilon} \text{DL}(\text{div}(s(\mathcal{O}), s(\mathcal{O})), \text{nil}) \rightarrow^* \text{DL}(s(\mathcal{O}), \text{nil})$ .

A DP problem  $(\mathcal{P}, \mathcal{R})$  is called *strongly normalizing (SN)* if there is no infinite  $(\mathcal{P}, \mathcal{R})$ -chain. The main result on DPs is the *chain criterion* which states that a TRS  $\mathcal{R}$  is SN iff  $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$  is SN. The key idea of the DP framework is a *divide-and-conquer* approach which applies *DP processors* to transform DP problems into simpler sub-problems. A *DP processor* Proc has the form  $\text{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}_1, \mathcal{R}_1), \dots, (\mathcal{P}_n, \mathcal{R}_n)\}$ , where  $\mathcal{P}, \mathcal{P}_1, \dots, \mathcal{P}_n$  are sets of DPs and  $\mathcal{R}, \mathcal{R}_1, \dots, \mathcal{R}_n$  are TRSs. Proc is *sound* if  $(\mathcal{P}, \mathcal{R})$  is SN whenever  $(\mathcal{P}_i, \mathcal{R}_i)$  is SN for all  $1 \leq i \leq n$ . It is *complete* if  $(\mathcal{P}_i, \mathcal{R}_i)$  is SN for all  $1 \leq i \leq n$  whenever  $(\mathcal{P}, \mathcal{R})$  is SN.

So for a TRS  $\mathcal{R}$ , one starts with the initial DP problem  $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$  and applies sound (and preferably complete) DP processors until all sub-problems are “solved” (i.e., DP processors transform them to the empty set). This allows for modular termination proofs, as different techniques can be applied on each sub-problem  $(\mathcal{P}_i, \mathcal{R}_i)$ .

One of the most important processors is the *dependency graph processor*. The  $(\mathcal{P}, \mathcal{R})$ -*dependency graph* indicates which DPs can be used after each other in chains. Its nodes are  $\mathcal{P}$  and there is an edge from  $s_1 \rightarrow t_1$  to  $s_2 \rightarrow t_2$  if there are substitutions  $\sigma_1, \sigma_2$  with  $t_1\sigma_1 \rightarrow_{\mathcal{R}}^* s_2\sigma_2$ . The  $(\mathcal{DP}(\mathcal{R}_{\text{divL}}), \mathcal{R}_{\text{divL}})$ -dependency graph is on the right. Any infinite  $(\mathcal{P}, \mathcal{R})$ -chain corresponds to an infinite path in the dependency graph, and since the graph is finite, this infinite path must end in a strongly connected component (SCC).<sup>2</sup> Hence, it suffices to consider the SCCs of this graph independently.



<sup>2</sup> Here, a set  $\mathcal{P}'$  of dependency pairs is an *SCC* if it is a maximal cycle, i.e., it is a maximal set such that for any  $s_1 \rightarrow t_1$  and  $s_2 \rightarrow t_2$  in  $\mathcal{P}'$  there is a non-empty path from  $s_1 \rightarrow t_1$  to  $s_2 \rightarrow t_2$  which only traverses nodes from  $\mathcal{P}'$ .

**Theorem 8 (Dep. Graph Processor).** *For the SCCs  $\mathcal{P}_1, \dots, \mathcal{P}_n$  of the  $(\mathcal{P}, \mathcal{R})$ -dependency graph,  $\text{Proc}_{\text{DG}}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}_1, \mathcal{R}), \dots, (\mathcal{P}_n, \mathcal{R})\}$  is sound and complete.*

While the exact dependency graph is not computable in general, there are techniques to over-approximate it automatically, see, e.g., [1, 10, 14]. In our example,  $\text{Proc}_{\text{DG}}(\mathcal{DP}(\mathcal{R}_{\text{divL}}, \mathcal{R}_{\text{divL}}))$  yields  $(\{(8)\}, \mathcal{R}_{\text{divL}})$ ,  $(\{(10)\}, \mathcal{R}_{\text{divL}})$ , and  $(\{(12)\}, \mathcal{R}_{\text{divL}})$ .

The second crucial processor adapts classical reduction orders to DP problems. A *reduction pair*  $(\succsim, \succ)$  consists of two relations on terms such that  $\succsim$  is reflexive, transitive, and closed under contexts and substitutions, and  $\succ$  is a well-founded order that is closed under substitutions but does not have to be closed under contexts. Moreover,  $\succsim$  and  $\succ$  must be compatible, i.e.,  $\succsim \circ \succ \circ \succsim \subseteq \succ$ . The *reduction pair processor* requires that all rules and dependency pairs are weakly decreasing, and it removes those DPs that are strictly decreasing.

**Theorem 9 (Reduction Pair Processor).** *Let  $(\succsim, \succ)$  be a reduction pair such that  $\mathcal{P} \cup \mathcal{R} \subseteq \succsim$ . Then  $\text{Proc}_{\text{RPP}}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P} \setminus \succ, \mathcal{R})\}$  is sound and complete.*

For example, one can use reduction pairs based on polynomial interpretations [24]. A *polynomial interpretation*  $\text{Pol}$  is a  $\Sigma^\#$ -algebra which maps every function symbol  $f \in \Sigma^\#$  to a polynomial  $f_{\text{Pol}} \in \mathbb{N}[\mathcal{V}]$ .  $\text{Pol}(t)$  denotes the *interpretation* of a term  $t$  by the  $\Sigma^\#$ -algebra  $\text{Pol}$ . Then  $\text{Pol}$  induces a reduction pair  $(\succsim, \succ)$  where  $t_1 \succsim t_2$  ( $t_1 \succ t_2$ ) holds if the inequation  $\text{Pol}(t_1) \geq \text{Pol}(t_2)$  ( $\text{Pol}(t_1) > \text{Pol}(t_2)$ ) is true for all instantiations of its variables by natural numbers.

For the three remaining DP problems in our example, we can apply the reduction pair processor using the polynomial interpretation which maps  $\mathcal{O}$  to 0,  $s(x)$  to  $x + 1$ ,  $\text{cons}(y, xs)$  to  $xs + 1$ ,  $\text{DL}(x, xs)$  to  $xs$ , and all other symbols to their first arguments. Since (8), (10), and (12) are strictly decreasing,  $\text{Proc}_{\text{RPP}}$  transforms all three remaining DP problems into DP problems of the form  $(\emptyset, \dots)$ . As  $\text{Proc}_{\text{DG}}(\emptyset, \dots) = \emptyset$  and all processors used are sound, this means that there is no infinite chain for the initial DP problem  $(\mathcal{DP}(\mathcal{R}_{\text{divL}}, \mathcal{R}_{\text{divL}}))$  and thus,  $\mathcal{R}_{\text{divL}}$  is SN.

### 3.2 Dependency Pairs for Relative Termination 0

Up to now, we only considered DPs for ordinary termination of TRSs. The easiest idea to use DPs in the relative setting is to start with the DP problem  $(\mathcal{DP}(\mathcal{R} \cup \mathcal{R}^=), \mathcal{R} \cup \mathcal{R}^=)$ . This would prove termination of  $\mathcal{R} \cup \mathcal{R}^=$ , which implies termination of  $\mathcal{R}/\mathcal{R}^=$ , but ignores that the rules in  $\mathcal{R}^=$  do not have to terminate. Since termination of DP problems is already defined via a relative condition (finite chains can only have finitely many **p**-steps but may have infinitely many **r**-steps), another idea for proving termination of  $\mathcal{R}/\mathcal{R}^=$  is to start with the DP problem  $(\mathcal{DP}(\mathcal{R}), \mathcal{R} \cup \mathcal{R}^=)$ , which only considers the DPs of  $\mathcal{R}$ . However, this is unsound in general.

*Example 10.* The only defined symbol of  $\mathcal{R}_2$  from Ex. 4 is **a**. Since the right-hand side of  $\mathcal{R}_2$ 's rule does not contain defined symbols, we would get the DP problem  $(\emptyset, \mathcal{R}_2 \cup \mathcal{R}_2^=)$ , which is SN as it has no DP. Thus, we would falsely conclude that  $\mathcal{R}_2/\mathcal{R}_2^=$  is SN. Similarly, this approach would also falsely “prove” SN for Ex. 3 and 5.

In [18], it was shown that under certain conditions on  $\mathcal{R}$  and  $\mathcal{R}^-$ , starting with the DP problem  $(\mathcal{DP}(\mathcal{R} \cup \mathcal{R}_a^-), \mathcal{R} \cup \mathcal{R}^-)$  for a subset  $\mathcal{R}_a^- \subseteq \mathcal{R}^-$  is sound for relative termination.<sup>3</sup> The two restrictions on the TRSs are *dominance* and being *non-duplicating*. We say that  $\mathcal{R}$  *dominates*  $\mathcal{R}^-$  if defined symbols of  $\mathcal{R}$  do not occur in the right-hand sides of rules of  $\mathcal{R}^-$ . A TRS is *non-duplicating* if no variable occurs more often on the right-hand side of a rule than on its left-hand side.

**Theorem 11 (First Main Result of [18], Sound and Complete).** *Let  $\mathcal{R}$  and  $\mathcal{R}^-$  be TRSs such that  $\mathcal{R}^-$  is non-duplicating and  $\mathcal{R}$  dominates  $\mathcal{R}^-$ . Then the DP problem  $(\mathcal{DP}(\mathcal{R}), \mathcal{R} \cup \mathcal{R}^-)$  is SN iff  $\mathcal{R}/\mathcal{R}^-$  is SN.*

**Theorem 12 (Second Main Result of [18], only Sound).** *Let  $\mathcal{R}$  and  $\mathcal{R}^- = \mathcal{R}_a^- \uplus \mathcal{R}_b^-$  be TRSs. If  $\mathcal{R}_b^-$  is non-duplicating,  $\mathcal{R} \cup \mathcal{R}_a^-$  dominates  $\mathcal{R}_b^-$ , and the DP problem  $(\mathcal{DP}(\mathcal{R} \cup \mathcal{R}_a^-), \mathcal{R} \cup \mathcal{R}^-)$  is SN, then  $\mathcal{R}/\mathcal{R}^-$  is SN.*

*Example 13.* For the main TRS  $\mathcal{R}_{\text{divL}}$  from Ex. 1 and base TRS  $\mathcal{R}_{\text{mset}}^-$  from Ex. 2 we can apply Thm. 11 and consider the DP problem  $(\mathcal{DP}(\mathcal{R}_{\text{divL}}), \mathcal{R}_{\text{divL}} \cup \mathcal{R}_{\text{mset}}^-)$ , since  $\mathcal{R}_{\text{mset}}^-$  is non-duplicating and  $\mathcal{R}_{\text{divL}}$  dominates  $\mathcal{R}_{\text{mset}}^-$ . As for  $(\mathcal{DP}(\mathcal{R}_{\text{divL}}), \mathcal{R}_{\text{divL}})$ , the DP framework can prove that  $(\mathcal{DP}(\mathcal{R}_{\text{divL}}), \mathcal{R}_{\text{divL}} \cup \mathcal{R}_{\text{mset}}^-)$  is SN. In this way, the tool NaTT which implements the results of [18] proves that  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset}}^-$  is SN. In contrast, a direct application of simplification orders fails to prove SN for  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset}}^-$  because simplification orders already fail to prove termination of  $\mathcal{R}_{\text{divL}}$ .

*Example 14.* If we consider  $\mathcal{R}_{\text{mset2}}^-$  with the rule

$$\text{divL}(z, \text{cons}(x, \text{cons}(y, zs))) \rightarrow \text{divL}(z, \text{cons}(y, \text{cons}(x, zs))) \quad (13)$$

instead of  $\mathcal{R}_{\text{mset}}^-$  as the base TRS, then  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset2}}^-$  remains strongly normalizing, but we cannot use Thm. 11 since  $\mathcal{R}_{\text{divL}}$  does not dominate  $\mathcal{R}_{\text{mset2}}^-$ . If we try to split  $\mathcal{R}_{\text{mset2}}^-$  as in Thm. 12, then  $\emptyset \neq \mathcal{R}_a^- \subseteq \mathcal{R}_{\text{mset2}}^-$  implies  $\mathcal{R}_a^- = \mathcal{R}_{\text{mset2}}^-$ , but  $\mathcal{R}_{\text{mset2}}^-$  is non-terminating. Therefore, all previous tools for relative termination fail in proving that  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset2}}^-$  is SN. In Sect. 4 we will present our novel DP framework which can prove relative termination of relative TRSs like  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset2}}^-$ .

As remarked in [18], Thm. 11 and 12 are unsound if one only considers *minimal* chains, i.e., if for a DP problem  $(\mathcal{P}, \mathcal{R})$  one only considers chains  $t_0, t_1, \dots$ , where all  $t_i$  are  $\mathcal{R}$ -strongly normalizing. In the DP framework for ordinary rewriting, the restriction to minimal chains allows the use of further processors, e.g., based on *usable rules* [10, 15] or the *subterm criterion* [15]. As shown in [18], usable rules and the subterm criterion can nevertheless be applied if  $\mathcal{R}^-$  is *quasi-terminating* [3], i.e., the set  $\{t \mid s \rightarrow_{\mathcal{R}^-}^* t\}$  is finite for every term  $s$ . This restriction would also be needed to integrate processors that rely on minimality into our new framework in Sect. 4.

<sup>3</sup> As before, for the construction of  $\mathcal{DP}(\mathcal{R} \cup \mathcal{R}_a^-)$ , only the root symbols of left-hand sides of  $\mathcal{R} \cup \mathcal{R}_a^-$  are considered to be “defined”.



## 4 Annotated Dependency Pairs for Relative Termination 0

As shown in Sect. 3.2, up to now there only exist criteria [18] that state when it is sound to apply *ordinary* DPs for proving relative termination, but there is no *specific* DP-based technique to analyze relative termination directly. To solve this problem, we now adapt the concept of *annotated dependency pairs* (ADPs) for relative termination. ADPs were introduced in [20] to prove innermost almost-sure termination of probabilistic term rewriting. In the relative setting, we can use similar dependency pairs as in the probabilistic setting, but with a different rewrite relation  $\rightarrow$  to deal with non-innermost rewrite steps. Compared to [18], we can (a) remove the requirement of dominance, which will be handled by the dependency graph processor, and (b) allow for ADP processors that are specifically designed for the relative setting before possibly moving to ordinary DPs. The requirement that  $\mathcal{R}^=$  must be non-duplicating remains, since DPs do not help in analyzing redex-duplicating sequences as in Ex. 3, where the crucial redex  $a$  is not generated from a “function call” in the right-hand side of a rule, but it just corresponds to a duplicated variable. To handle TRSs  $\mathcal{R}/\mathcal{R}^=$  where  $\mathcal{R}_{dup}^= \subseteq \mathcal{R}^=$  is duplicating, one can move the duplicating rules to the main TRS  $\mathcal{R}$  and try to prove relative termination of  $(\mathcal{R} \cup \mathcal{R}_{dup}^=)/(\mathcal{R}^= \setminus \mathcal{R}_{dup}^=)$  instead, or one can try to find a reduction pair  $(\succsim, \succ)$  where  $\succ$  is closed under contexts such that  $\mathcal{R} \cup \mathcal{R}^= \subseteq \succsim$  and  $\mathcal{R}_{dup}^= \subseteq \succ$ . Then it suffices to prove relative termination of  $(\mathcal{R} \setminus \succ, \mathcal{R}^= \setminus \succ)$  instead.

For ordinary termination, we create a separate DP for each occurrence of a defined symbol in the right-hand side of a rule (and no DP is created for rules without defined symbols in their right-hand sides). This would work to detect *ordinary infinite* sequences like the one in Ex. 6 in the relative setting, i.e., such an infinite sequence would give rise to an infinite chain. However, as shown in Ex. 10, this would not suffice to detect infinite redex-creating sequences as in Ex. 4 with  $\mathcal{R}_2 = \{a \rightarrow b\}$  and  $\mathcal{R}_2^= = \{f \rightarrow d(f, a)\}$ :  $f \rightarrow_{\mathcal{R}_2^=} d(f, \underline{a}) \rightarrow_{\mathcal{R}_2} d(\underline{f}, b) \rightarrow_{\mathcal{R}_2^=} d(d(f, \underline{a}), b) \rightarrow_{\mathcal{R}_2} \dots$

Here, (1) we need a DP for the rule  $a \rightarrow b$  to detect the reduction of the created  $\mathcal{R}_2$ -redex  $a$ , although  $b$  is a constructor. Moreover, (2) both defined symbols  $f$  and  $a$  in the right-hand side of  $f \rightarrow d(f, a)$  have to be considered simultaneously: We need  $f$  to create an infinite number of  $\mathcal{R}_2$ -redexes, and we need  $a$  since it is the created  $\mathcal{R}_2$ -redex. Hence, for rules from the base TRS  $\mathcal{R}_2^=$ , we have to consider all possible pairs of defined symbols in their right-hand sides simultaneously.<sup>4</sup> This is not needed for the main TRS  $\mathcal{R}_2$ , i.e., if the  $f$ -rule were in the main TRS, then the  $f$  in the right-hand side could be considered separately from the  $a$  that it generates. Therefore, we distinguish between *main* and *base ADPs* (that are generated from the main and the base TRS, respectively).

As in [20], we now annotate defined symbols directly in the original rewrite rule instead of extracting annotated subterms from its right-hand side. In this way, we may have terms containing several annotated symbols, which allows us to consider pairs of defined symbols in right-hand sides simultaneously.

<sup>4</sup> For relative termination, it suffices to consider *pairs* of defined symbols. The reason is that to “track” a non-terminating reduction, one only has to consider a single redex plus possibly another redex of the base TRS which may later create a redex again.

**Definition 15 (Annotations).** For  $t \in \mathcal{T}(\Sigma^\#, \mathcal{V})$  and  $\Sigma' \subseteq \Sigma^\# \cup \mathcal{V}$ , let  $\text{pos}_{\Sigma'}(t)$  be the set of all positions of  $t$  with symbols or variables from  $\Sigma'$ . For  $\Phi \subseteq \text{pos}_{\mathcal{D} \cup \mathcal{D}^\#}(t)$ ,  $\#_\Phi(t)$  is the variant of  $t$  where the symbols at positions from  $\Phi$  are annotated and all other annotations are removed. Thus,  $\text{pos}_{\mathcal{D}^\#}(\#_\Phi(t)) = \Phi$ , and  $\#_\emptyset(t)$  removes all annotations from  $t$ , where we often write  $\flat(t)$  instead of  $\#_\emptyset(t)$ . Moreover, for a singleton  $\{\pi\}$ , we often write  $\#_\pi$  instead of  $\#_{\{\pi\}}$ . We write  $t \leq_\#^\pi s$  if there is a  $\pi \in \text{pos}_{\mathcal{D}^\#}(s)$  and  $t = \flat(s|_\pi)$  (i.e.,  $t$  results from a subterm of  $s$  with annotated root symbol by removing its annotations). We also write  $\leq_\#$  instead of  $\leq_\#^\pi$ .

*Example 16.* If  $f \in \mathcal{D}$ , then we have  $\#_1(f(f(x))) = \#_1(F(F(x))) = f(F(x))$  and  $\flat(F(F(x))) = f(f(x))$ . Moreover, we have  $f(x) \leq_\#^1 f(F(x))$ .

While in [20] all defined symbols on the right-hand sides of rules were annotated, we now define our novel variant of *annotated dependency pairs* for relative rewriting.

**Definition 17 (Annotated Dependency Pair).** A rule  $\ell \rightarrow r$  with  $\ell \in \mathcal{T}(\Sigma, \mathcal{V}) \setminus \mathcal{V}$ ,  $r \in \mathcal{T}(\Sigma^\#, \mathcal{V})$ , and  $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$  is called an annotated dependency pair (ADP). Let  $\mathcal{D}$  be the defined symbols of  $\mathcal{R} \cup \mathcal{R}^\#$ , and for  $n \in \mathbb{N}$ , let  $\mathcal{A}_n(\ell \rightarrow r) = \{\ell \rightarrow \#_\Phi(r) \mid \Phi \subseteq \text{pos}_{\mathcal{D}}(r), |\Phi| \leq n\}$ . The canonical main ADPs for  $\mathcal{R}$  are  $\mathcal{A}_1(\mathcal{R}) = \bigcup_{\ell \rightarrow r \in \mathcal{R}} \mathcal{A}_1(\ell \rightarrow r)$  and the canonical base ADPs for  $\mathcal{R}^\#$  are  $\mathcal{A}_2(\mathcal{R}^\#) = \bigcup_{\ell \rightarrow r \in \mathcal{R}^\#} \mathcal{A}_2(\ell \rightarrow r)$ .

So the left-hand side of an ADP is just the left-hand side of the original rule. The right-hand side results from the right-hand side of the original rule by replacing certain defined symbols  $f$  with  $f^\#$ . Whenever we have two ADPs  $\ell \rightarrow \#_{\Phi'}(r)$ ,  $\ell \rightarrow \#_\Phi(r)$  with  $\Phi' \subset \Phi$ , then we only consider  $\ell \rightarrow \#_\Phi(r)$  and remove  $\ell \rightarrow \#_{\Phi'}(r)$ .

*Example 18.* The canonical ADPs of Ex. 4 are  $\mathcal{A}_1(\mathcal{R}_2) = \{a \rightarrow b\}$  and  $\mathcal{A}_2(\mathcal{R}_2^\#) = \{f \rightarrow d(F, A)\}$  and for Ex. 5 we get  $\mathcal{A}_1(\mathcal{R}_3) = \{a(x) \rightarrow b(x)\}$  and  $\mathcal{A}_2(\mathcal{R}_3^\#) = \{f \rightarrow A(F)\}$ . For  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset2}}^\#$  from Ex. 1 and 14, the ADPs  $\mathcal{A}_1(\mathcal{R}_{\text{divL}})$  are

$$\begin{array}{ll} \text{minus}(x, \mathcal{O}) \rightarrow x & (14) \\ \text{minus}(s(x), s(y)) \rightarrow M(x, y) & (15) \\ \text{div}(x, s(\mathcal{O})) \rightarrow x & (16) \\ \text{divL}(x, \text{nil}) \rightarrow x & (17) \end{array} \quad \begin{array}{ll} \text{div}(s(x), s(y)) \rightarrow s(D(\text{minus}(x, y), s(y))) & (18) \\ \text{div}(s(x), s(y)) \rightarrow s(\text{div}(M(x, y), s(y))) & (19) \\ \text{divL}(x, \text{cons}(y, xs)) \rightarrow \text{DL}(\text{div}(x, y), xs) & (20) \\ \text{divL}(x, \text{cons}(y, xs)) \rightarrow \text{divL}(D(x, y), xs) & (21) \end{array}$$

and  $\mathcal{A}_2(\mathcal{R}_{\text{mset2}}^\#)$  contains  $\text{divL}(z, \text{cons}(x, \text{cons}(y, zs))) \rightarrow \text{DL}(z, \text{cons}(y, \text{cons}(x, zs)))$  (22)

In [20], ADPs were only used for innermost rewriting. We now modify their rewrite relation and define what happens with annotations inside the substitutions during a rewrite step. To simulate redex-creating sequences as in Ex. 5 with ADPs (where the position of the created redex  $a(\dots)$  is above the position of the creating redex  $f$ ), ADPs should be able to rewrite above annotated arguments without removing their annotation (we will demonstrate that in Ex. 25). Thus, for an ADP  $\ell \rightarrow r$  with  $\ell|_\pi = x$ , we use a *variable reposition function (VRF)* to indicate which occurrence of  $x$  in  $r$  should keep the annotations if one rewrites an instance of  $\ell$  where the subterm at position  $\pi$  is annotated. So a VRF maps positions of variables in the left-hand side of a rule to positions of the same variable in the right-hand side.



**Definition 19 (Variable Reposition Function).** Let  $\ell \rightarrow r$  be an ADP. A function  $\varphi : \text{pos}_V(\ell) \rightarrow \text{pos}_V(r) \cup \{\perp\}$  is called a variable reposition function (VRF) for  $\ell \rightarrow r$  iff  $\ell|_\pi = r|_{\varphi(\pi)}$  whenever  $\varphi(\pi) \neq \perp$ . 88

*Example 20.* For the ADP  $a(x) \rightarrow b(x)$  for  $\mathcal{R}_3$  from Ex. 5, if  $x$  on position 1 of the left-hand side is instantiated by  $F$ , then the VRF  $\varphi(1) = 1$  indicates that this ADP rewrites  $A(F)$  to  $b(F)$ , whereas  $\varphi(1) = \perp$  means that it rewrites  $A(F)$  to  $b(f)$ . 89

With VRFs we can define the rewrite relation for ADPs w.r.t. full rewriting. 90

**Definition 21 ( $\hookrightarrow_P$ ).** Let  $\mathcal{P}$  be a set of ADPs. A term  $s \in \mathcal{T}(\Sigma^\#, V)$  rewrites to  $t$  using  $\mathcal{P}$  (denoted  $s \hookrightarrow_P t$ ) if there is a rule  $\ell \rightarrow r \in \mathcal{P}$ , a substitution  $\sigma$ , a position  $\pi \in \text{pos}_{D \cup D^\#}(s)$  such that  $b(s|_\pi) = \ell\sigma$ , a VRF  $\varphi$  for  $\ell \rightarrow r$ , and 91

$$\begin{aligned} t &= s[\#_\Phi(r\sigma)]_\pi && \text{if } \pi \in \text{pos}_{D^\#}(s) && (\mathbf{pr}) && 138 \\ t &= s[\#_\Psi(r\sigma)]_\pi && \text{if } \pi \in \text{pos}_D(s) && (\mathbf{r}) && 138 \end{aligned}$$

Here,  $\Psi = \{\varphi(\rho).\tau \mid \rho \in \text{pos}_V(\ell), \varphi(\rho) \neq \perp, \rho.\tau \in \text{pos}_{D^\#}(s|_\pi)\}$  and  $\Phi = \text{pos}_{D^\#}(r) \cup \Psi$ . 144

So  $\Psi$  considers all positions of annotated symbols in  $s|_\pi$  that are below positions  $\rho$  of variables in  $\ell$ . If the VRF maps  $\rho$  to a variable position  $\rho'$  in  $r$ , then the annotations below  $\pi.\rho$  in  $s$  are kept in the resulting subterm at position  $\pi.\rho'$  after the rewriting. 93

Rewriting with  $\mathcal{P}$  is like ordinary term rewriting, while considering and modifying annotations. Note that we represent all DPs resulting from a rule as well as the original rule by just one ADP. So the ADP  $\text{div}(s(x), s(y)) \rightarrow s(D(\text{minus}(x, y), s(y)))$  represents both the DP resulting from  $\text{div}$  in the right-hand side of the rule (4), and the rule (4) itself (by simply disregarding all annotations of the ADP). 94

Similar to the classical DP framework, our goal is to track specific reduction sequences. As before, there are **p**-steps where a DP is applied at the position of an annotated symbol. These steps may introduce new annotations. Moreover, between two **p**-steps there can be several **r**-steps. 95

A step of the form **(pr)** at position  $\pi$  in Def. 21 represents a **p**- or an **r**-step (or both), where an **r**-step is only possible if one later rewrites an annotated symbol at a position above  $\pi$ . All annotations are kept during this step except for annotations of subterms that correspond to variables of the applied rule. Here, the used VRF  $\varphi$  determines which of these annotations are kept and which are removed. As an example, with the canonical ADP  $a(x) \rightarrow b(x)$  from  $\mathcal{A}_1(\mathcal{R}_3)$  we can rewrite  $A(F) \hookrightarrow_{\mathcal{A}_1(\mathcal{R}_3)} b(F)$  as in Ex. 20. Here, we have  $\pi = \varepsilon$ ,  $b(s|_\varepsilon) = a(f) = \ell\sigma$ ,  $r = b(x)$ , and the VRF  $\varphi$  with  $\varphi(1) = 1$  such that the annotation of  $F$  in  $A$ 's argument is kept in the argument of  $b$ . 96

A step of the form **(r)** rewrites at the position of a non-annotated defined symbol, and represents just an **r**-step. Hence, we remove all annotations from the right-hand side  $r$  of the ADP. However, we may have to keep the annotations inside the 98

<sup>5</sup> In [20] there were two additional cases in the definition of the corresponding rewrite relation. One of them was needed for processors that restrict the set of rules applicable for **r**-steps (e.g., based on usable rules), and the other case was needed to ensure that the innermost evaluation strategy is not affected by the application of ADP processors. This is unnecessary here since we consider full rewriting. On the other hand, VRFs are new compared to [20], since they are not needed for innermost rewriting. 91

substitution, hence we move them according to the VRF. For example, we obtain the rewrite step  $s(D(\text{minus}(s(\mathcal{O}), s(\mathcal{O})), s(\mathcal{O}))) \hookrightarrow_{\mathcal{A}_1(\mathcal{R}_{\text{divL}})} s(D(\text{minus}(\mathcal{O}, \mathcal{O}), s(\mathcal{O})))$  using the ADP  $\text{minus}(s(x), s(y)) \rightarrow M(x, y)$  (15) and any VRF. 98 98

A (relative) ADP problem has the form  $(\mathcal{P}, \mathcal{P}^=)$ , where  $\mathcal{P}$  and  $\mathcal{P}^=$  are finite sets of ADPs and  $\mathcal{P}^=$  is non-duplicating.  $\mathcal{P}$  is the set of all main ADPs and  $\mathcal{P}^=$  is the set of all base ADPs. Now we can define chains in the relative setting. 100 100

**Definition 22 (Chains and Terminating ADP Problems).** Let  $(\mathcal{P}, \mathcal{P}^=)$  be an ADP problem. A sequence of terms  $t_0, t_1, \dots$  is a  $(\mathcal{P}, \mathcal{P}^=)$ -chain if we have  $t_i \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^=} t_{i+1}$  for all  $i \in \mathbb{N}$ . The chain is called infinite if infinitely many of these rewrite steps use  $\hookrightarrow_{\mathcal{P}}$  with Case (pr). We say that an ADP problem  $(\mathcal{P}, \mathcal{P}^=)$  is strongly normalizing (SN) if there is no infinite  $(\mathcal{P}, \mathcal{P}^=)$ -chain. 101 101 101 101

Note the two different forms of relativity in Def. 22: In a finite chain, we may not only use infinitely many steps with  $\mathcal{P}^=$  but also infinitely many steps with  $\mathcal{P}$  where Case (r) applies. Thus, an ADP problem  $(\mathcal{P}, \mathcal{P}^=)$  without annotated symbols or without any main ADPs (i.e., where  $\mathcal{P} = \emptyset$ ) is obviously SN. Finally, we obtain our desired chain criterion. 102 102 102 102

**Theorem 23 (Chain Criterion for Relative Rewriting).** Let  $\mathcal{R}$  and  $\mathcal{R}^=$  be TRSs such that  $\mathcal{R}^=$  is non-duplicating. Then  $\mathcal{R}/\mathcal{R}^=$  is SN iff the ADP problem  $(\mathcal{A}_1(\mathcal{R}), \mathcal{A}_2(\mathcal{R}^=))$  is SN. 103 103

*Example 24.* The infinite rewrite sequence of Ex. 4 can be simulated by the following infinite chain using  $\mathcal{A}_1(\mathcal{R}_2) = \{a \rightarrow b\}$  and  $\mathcal{A}_2(\mathcal{R}_2^=) = \{f \rightarrow d(F, A)\}$ . 104 104

$$E \hookrightarrow_{\mathcal{A}_2(\mathcal{R}_2^=)} d(F, \underline{A}) \hookrightarrow_{\mathcal{A}_1(\mathcal{R}_2)} d(\underline{E}, b) \hookrightarrow_{\mathcal{A}_2(\mathcal{R}_2^=)} d(d(F, \underline{A}), b) \hookrightarrow_{\mathcal{A}_1(\mathcal{R}_2)} \dots \quad 129$$

The steps with  $\hookrightarrow_{\mathcal{A}_2(\mathcal{R}_2^=)}$  use Case (pr) at the position of the annotated symbol  $F$  and the steps with  $\hookrightarrow_{\mathcal{A}_1(\mathcal{R}_2)}$  use (pr) as well. For this infinite chain, we indeed need two annotated symbols in the right-hand side of the base ADP: If  $A$  were not annotated (i.e., if we had the ADP  $f \rightarrow d(F, a)$ ), then the step with  $\hookrightarrow_{\mathcal{A}_1(\mathcal{R}_2)}$  would just use Case (r) and the chain would not be considered “infinite”. If  $F$  were not annotated (i.e., if we had the ADP  $f \rightarrow d(f, A)$ ), then we would have the step  $f \hookrightarrow_{\mathcal{A}_2(\mathcal{R}_2^=)} d(f, a)$  which uses Case (r) and removes all annotations from the right-hand side. Hence, again the chain would not be considered “infinite”. 104 104 104 104 104

*Example 25.* The infinite rewrite sequence of Ex. 5 is simulated by the following chain with  $\mathcal{A}_1(\mathcal{R}_3) = \{a(x) \rightarrow b(x)\}$  and  $\mathcal{A}_2(\mathcal{R}_3^=) = \{f \rightarrow A(F)\}$ . 105 105

$$E \hookrightarrow_{\mathcal{A}_2(\mathcal{R}_3^=)} \underline{A}(F) \hookrightarrow_{\mathcal{A}_1(\mathcal{R}_3)} b(\underline{E}) \hookrightarrow_{\mathcal{A}_2(\mathcal{R}_3^=)} b(\underline{A}(F)) \hookrightarrow_{\mathcal{A}_1(\mathcal{R}_3)} b(b(\underline{E})) \hookrightarrow_{\mathcal{A}_2(\mathcal{R}_3^=)} \dots \quad 129$$

Here, it is important to use the VRF  $\varphi(1) = 1$  for  $a(x) \rightarrow b(x)$  which keeps the annotation of  $A$ ’s argument  $F$  during the rewrite steps with  $\mathcal{A}_1(\mathcal{R}_3)$ , i.e., these steps must yield  $b(F)$  instead of  $b(f)$  to generate further subterms  $A(\dots)$  afterwards. 105 105 105

## 5 The Relative ADP Framework 132

Now we present processors for our novel relative ADP framework. An *ADP processor* Proc has the form  $\text{Proc}(\mathcal{P}, \mathcal{P}^-) = \{(\mathcal{P}_1, \mathcal{P}_1^-), \dots, (\mathcal{P}_n, \mathcal{P}_n^-)\}$ , where  $\mathcal{P}, \mathcal{P}_1, \dots, \mathcal{P}_n, \mathcal{P}_1^-, \dots, \mathcal{P}_n^-$  are sets of ADPs. Proc is *sound* if  $(\mathcal{P}, \mathcal{P}^-)$  is SN whenever  $(\mathcal{P}_i, \mathcal{P}_i^-)$  is SN for all  $1 \leq i \leq n$ . It is *complete* if  $(\mathcal{P}_i, \mathcal{P}_i^-)$  is SN for all  $1 \leq i \leq n$  whenever  $(\mathcal{P}, \mathcal{P}^-)$  is SN. To prove relative termination of  $\mathcal{R}/\mathcal{R}^-$ , we start with the canonical ADP problem  $(\mathcal{A}_1(\mathcal{R}), \mathcal{A}_2(\mathcal{R}^-))$  and apply sound (and preferably complete) ADP processors until all sub-problems are transformed to the empty set. 106

In Sect. 5.1, we present two processors to remove (base) ADPs, and in Sect. 5.2 and 5.3, we adapt the main processors of the classical DP framework from Sect. 3.1 to the relative setting. As mentioned, the soundness and completeness proofs for our processors and the chain criterion (Thm. 23) can be found in App. A. 107

### 5.1 Derelativifying Processors 0

The following two *derelativifying* processors can be used to switch from ADPs to ordinary DPs, similar to Thm. 11 and 12. We extend  $\flat$  to ADPs and sets of ADPs  $\mathcal{S}$  by defining  $\flat(\ell \rightarrow r) = \ell \rightarrow \flat(r)$  and  $\flat(\mathcal{S}) = \{\ell \rightarrow \flat(r) \mid \ell \rightarrow r \in \mathcal{S}\}$ . 108

If the ADPs in  $\mathcal{P}^-$  contain no annotations anymore, then it suffices to use ordinary DPs. The corresponding set of DPs for a set of ADPs  $\mathcal{P}$  is defined as  $\text{DP}(\mathcal{P}) = \{\ell^\# \rightarrow t^\# \mid \ell \rightarrow r \in \mathcal{P}, t \trianglelefteq_\# r\}$ . 110

**Theorem 26 (Derelativifying Processor (1)).** *Let  $(\mathcal{P}, \mathcal{P}^-)$  be an ADP problem such that  $\flat(\mathcal{P}^-) = \mathcal{P}^-$ . Then  $\text{Proc}_{\text{DRP1}}(\mathcal{P}, \mathcal{P}^-) = \emptyset$  is sound and complete iff the ordinary DP problem  $(\text{DP}(\mathcal{P}), \flat(\mathcal{P} \cup \mathcal{P}^-))$  is SN.* 142

Furthermore, similar to Thm. 12, we can always move ADPs from  $\mathcal{P}^-$  to  $\mathcal{P}$ , but such a processor is only sound and not complete. However, it may help to satisfy the requirements of Thm. 26 by moving ADPs with annotations from  $\mathcal{P}^-$  to  $\mathcal{P}$  such that the ordinary DP framework can be used afterwards. 112

**Theorem 27 (Derelativifying Processor (2)).** *Let  $(\mathcal{P}, \mathcal{P}^-)$  be an ADP problem, and let  $\mathcal{P}^- = \mathcal{P}_a^- \uplus \mathcal{P}_b^-$ . Then  $\text{Proc}_{\text{DRP2}}(\mathcal{P}, \mathcal{P}^-) = \{(\mathcal{P} \cup \text{split}(\mathcal{P}_a^-), \mathcal{P}_b^-)\}$  is sound. Here,  $\text{split}(\mathcal{P}_a^-) = \{\ell \rightarrow \#_\pi(r) \mid \ell \rightarrow r \in \mathcal{P}_a^-, \pi \in \text{pos}_{\mathcal{D}^\#}(r)\}$ .* 113

So if  $\mathcal{P}_a^-$  contains an ADP with two annotations, then we split it into two ADPs, where each only contains a single annotation. 114

*Example 28.* There are also examples that are redex-creating and terminating, e.g.,  $\mathcal{R}_2 = \{a \rightarrow b\}$  and the base TRS  $\mathcal{R}_2^- = \{f(s(y)) \rightarrow d(f(y), a)\}$ . Relative (and full) termination of this example can easily be shown by using the second derelativifying processor from Thm. 27 to replace the base ADP  $f(s(y)) \rightarrow d(f(y), A)$  by the main ADPs  $f(s(y)) \rightarrow d(F(y), a)$  and  $f(s(y)) \rightarrow d(f(y), A)$ . Then one can use the processor of Thm. 26 to switch to the ordinary DPs  $F(s(y)) \rightarrow F(y)$  and  $F(s(y)) \rightarrow A$ . 115

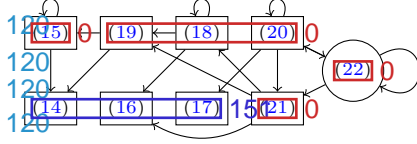
## 5.2 Relative Dependency Graph Processor 107

Next, we develop a dependency graph processor in the relative setting. The definition of the dependency graph is analogous to the one in the standard setting and thus, the same techniques can be used to over-approximate it automatically. 116

**Definition 29 (Relative Dependency Graph).** Let  $(\mathcal{P}, \mathcal{P}^-)$  be an ADP problem. The  $(\mathcal{P}, \mathcal{P}^-)$ -dependency graph has the nodes  $\mathcal{P} \cup \mathcal{P}^-$  and there is an edge from  $\ell_1 \rightarrow r_1$  to  $\ell_2 \rightarrow r_2$  if there exist substitutions  $\sigma_1, \sigma_2$  and a term  $t \trianglelefteq_{\#} r_1$  such that  $t^{\#}\sigma_1 \rightarrow_{b(\mathcal{P} \cup \mathcal{P}^-)}^* \ell_2^{\#}\sigma_2$ . 117

So similar to the standard dependency graph, there is an edge from an ADP  $\ell_1 \rightarrow r_1$  to  $\ell_2 \rightarrow r_2$  if the rules of  $b(\mathcal{P} \cup \mathcal{P}^-)$  (without annotations) can reduce an instance of a subterm  $t$  of  $r_1$  to an instance of  $\ell_2$ , if one only annotates the roots of  $t$  and  $\ell_2$  (i.e., then the rules can only be applied below the root). 118

**Example 30.** The dependency graph for the ADP problem  $(\mathcal{A}_1(\mathcal{R}_{\text{divL}}), \mathcal{A}_2(\mathcal{R}_{\text{mset2}}^-))$  from Ex. 18 is shown on the right. Here, nodes from  $\mathcal{A}_1(\mathcal{R}_{\text{divL}})$  are denoted by rectangles and the node from  $\mathcal{A}_2(\mathcal{R}_{\text{mset2}}^-)$  is a circle. 120



To detect possible ordinary infinite rewrite sequences as in Ex. 6, we again have to regard SCCs of the dependency graph, where we only need to consider SCCs that contain a node from  $\mathcal{P}$ , because otherwise, all steps in the SCC are relative. However, in the relative ADP framework, non-termination can also be due to chains representing redex-creating sequences. Here, it does not suffice to look at SCCs. Thus, the relative dependency graph processor differs substantially from the corresponding processor for ordinary rewriting (and also from the corresponding processor for the probabilistic ADP framework in [20]). 121

**Example 31 (Dependency Graph for Redex-Creating TRSs).** For  $\mathcal{R}_2$  and  $\mathcal{R}_2^-$  from Ex. 4, the dependency graph for  $(\mathcal{A}_1(\mathcal{R}_2), \mathcal{A}_2(\mathcal{R}_2^-))$  from Ex. 24 can be seen on the right. Here, we cannot regard the SCC  $\{f \rightarrow d(F, A)\}$  separately, as we need the rule  $a \rightarrow b$  from  $\mathcal{A}_1(\mathcal{R}_2)$  to reduce the created redex. To find the ADPs that can reduce the created redexes, we have to regard the outgoing paths from the SCCs of  $\mathcal{P}^-$  to ADPs of  $\mathcal{P}$ . 122



The structure that we are looking for in the redex-creating case is a path from an SCC to a node from  $\mathcal{P}$  (i.e., a form of a *lasso*), which is *minimal* in the sense that if we reach a node from  $\mathcal{P}$ , then we stop and do not move further along the edges of the graph. Moreover, the SCC needs to contain an ADP with more than one annotated symbol, as otherwise the generation of the infinitely many  $\mathcal{P}$ -redexes would not be possible. Here, it suffices to look at SCCs in the graph restricted to only  $\mathcal{P}^-$ -nodes (i.e., to SCCs in the  $(b(\mathcal{P}), \mathcal{P}^-)$ -dependency graph). The reason is that if the SCC contains a node from  $\mathcal{P}$ , then as mentioned above, we have to prove anyway that the SCC does not give rise to infinite chains. 125

**Definition 32** ( $\text{SCC}_{\mathcal{P}, \mathcal{P}^=}$ , Lasso). *Let  $(\mathcal{P}, \mathcal{P}^=)$  be an ADP problem. For any  $\mathcal{P}' \subseteq \mathcal{P} \cup \mathcal{P}^=$ , let  $\text{SCC}_{\mathcal{P}', \mathcal{P}^=}$  denote the set of all SCCs of the  $(\mathcal{P}, \mathcal{P}^=)$ -dependency graph that contain an ADP from  $\mathcal{P}'$ . Moreover, let  $\mathcal{P}_{>1}^= \subseteq \mathcal{P}^=$  denote the set of all ADPs from  $\mathcal{P}^=$  with more than one annotation. Then the set of all minimal lassos is defined as  $\text{Lasso} = \{\mathcal{Q} \cup \{n_1, \dots, n_k\} \mid \mathcal{Q} \in \text{SCC}_{\mathcal{P}_{>1}^=, \mathcal{P}^=}, n_1, \dots, n_k \text{ is a path such that } n_1 \in \mathcal{Q}, n_k \in \mathcal{P}, \text{ and } n_i \notin \mathcal{P} \text{ for all } 1 \leq i \leq k-1\}$ .*

We remove the annotations of ADPs which do not have to be considered anymore for  $\mathbf{p}$ -steps due to the dependency graph, but we keep the ADPs for possible  $\mathbf{r}$ -steps and thus, consider them as relative (base) ADPs.

**Theorem 33 (Dep. Graph Processor).** *Let  $(\mathcal{P}, \mathcal{P}^=)$  be an ADP problem. Then*

$$\text{Proc}_{\text{DG}}(\mathcal{P}, \mathcal{P}^=) = \{(\mathcal{P} \cap \mathcal{Q}, (\mathcal{P}^= \cap \mathcal{Q}) \cup \mathfrak{b}((\mathcal{P} \cup \mathcal{P}^=) \setminus \mathcal{Q})) \mid \mathcal{Q} \in \text{SCC}_{\mathcal{P}_{>1}^=, \mathcal{P}^=}, \mathcal{Q} \notin \text{Lasso}\}$$

*is sound and complete.*

*Example 34.* For  $(\mathcal{A}_1(\mathcal{R}_{\text{divL}}), \mathcal{A}_2(\mathcal{R}_{\text{mset2}}^=))$  from Ex. 30 we have three SCCs  $\{(15)\}$ ,  $\{(18)\}$ , and  $\{(20), (22)\}$  containing nodes from  $\mathcal{A}_1(\mathcal{R}_{\text{divL}})$ . The set  $\{(22)\}$  is the only SCC of  $(\mathfrak{b}(\mathcal{A}_1(\mathcal{R}_{\text{divL}})), \mathcal{A}_2(\mathcal{R}_{\text{mset2}}^=))$  and there are paths from that SCC to the ADPs (20) and (21) of  $\mathcal{P}$ . However, they are not in Lasso, because the SCC  $\{(22)\}$  does not contain an ADP with more than one annotation. Hence, we result in the three new ADP problems  $(\{(15)\} \cup \mathfrak{b}(\mathcal{A}_1(\mathcal{R}_{\text{divL}}) \setminus \{(15)\}), \mathfrak{b}(22))$ ,  $(\{(18)\} \cup \mathfrak{b}(\mathcal{A}_1(\mathcal{R}_{\text{divL}}) \setminus \{(18)\}), \mathfrak{b}(22))$ , and  $(\{(20)\} \cup \mathfrak{b}(\mathcal{A}_1(\mathcal{R}_{\text{divL}}) \setminus \{(20)\}), \mathfrak{b}(22))$ . For the first two of these new ADP problems, we can use the derelativizing processor of Thm. 26 and prove SN via ordinary DPs, since their base ADP  $\mathfrak{b}(22)$  does not contain any annotated symbols anymore.

The dependency graph processor in combination with the derelativizing processors of Thm. 26 and 27 already subsumes the techniques of Thm. 11 and 12. The reason is that if  $\mathcal{R}$  dominates  $\mathcal{R}^=$ , then there is no edge from an ADP of  $\mathcal{A}_2(\mathcal{R}^=)$  to any ADP of  $\mathcal{A}_1(\mathcal{R})$  in the  $(\mathcal{A}_1(\mathcal{R}), \mathcal{A}_2(\mathcal{R}^=))$ -dependency graph. Hence, there are no minimal lassos and the dependency graph processor just creates ADP problems from the SCCs of  $\mathcal{A}_1(\mathcal{R})$  where the base ADPs do not have any annotations anymore. Then Thm. 26 allows us to switch to ordinary DPs. For example, if we consider  $\mathcal{R}_{\text{mset}}^=$  instead of  $\mathcal{R}_{\text{mset2}}^=$ , then the dependency graph processor only yields the two subproblems for the SCCs  $\{(15)\}$  and  $\{(18)\}$ , where the base ADPs do not contain any annotations anymore. Then, we can move to ordinary DPs via Thm. 26.

Compared to Thm. 11 and 12, the dependency graph allows for more precise over-approximations than just “dominance” in order to detect when the base ADPs do not depend on the main ADPs. Moreover, the derelativizing processors of Thm. 26 and 27 allow us to switch to the ordinary DP framework also for subproblems which result from the application of other processors of our relative ADP framework. In other words, Thm. 26 and 27 allow us to apply this switch in a modular way, even if their prerequisites do not hold for the initial canonical ADP problem (i.e., even if the prerequisites of Thm. 11 and 12 do not hold for the whole TRSs).

### 5.3 Relative Reduction Pair Processor 107

Next, we adapt the reduction pair processor to ADPs for relative rewriting. While the reduction pair processor for ADPs in the probabilistic setting [20] was restricted to polynomial interpretations, we now allow arbitrary reduction pairs using a similar idea as in the reduction pair processor from [27] for complexity analysis via dependency tuples. 136

To find out which ADPs cannot be used for infinitely many  $\mathbf{p}$ -steps, the idea is not to compare the annotated left-hand side with the whole right-hand side, but just with the set of its annotated subterms. To combine these subterms in the case of ADPs with two or no annotated symbols, we extend the signature by two fresh compound symbols  $c_0$  and  $c_2$  of arity 0 and 2, respectively. Similar to [27], we have to use  $\mathbf{c}$ -monotonic and  $\mathbf{c}$ -invariant reduction pairs. 137

**Definition 35 (c-Monotonic, c-Invariant).** For  $r \in \mathcal{T}(\Sigma^\#, \mathcal{V})$ , we define  $\text{ann}(r) = c_0$  if  $r$  does not contain any annotation,  $\text{ann}(r) = t^\#$  if  $t \not\leq_\# r$  and  $r$  only contains one annotated symbol, and  $\text{ann}(r) = c_2(r_1^\#, r_2^\#)$  if  $r_1 \leq_\# t$ ,  $r_2 \leq_\# t$ , and  $\pi_1 <_{lex} \pi_2$  where  $<_{lex}$  is the (total) lexicographic order on positions. 163

A reduction pair  $(\succsim, \succ)$  is called  $\mathbf{c}$ -monotonic if  $c_2(s_1, t) \succ c_2(s_2, t)$  and  $c_2(t, s_1) \succ c_2(t, s_2)$  for all  $s_1, s_2, t \in \mathcal{T}(\Sigma^\#, \mathcal{V})$  with  $s_1 \succ s_2$ . Moreover, it is  $\mathbf{c}$ -invariant if  $c_2(x, y) \sim c_2(y, x)$  and  $c_2(x, c_2(y, z)) \sim c_2(c_2(x, y), z)$  for  $\sim = \succsim \cap \precsim$ . 163

So for example, reduction pairs based on polynomial interpretations are  $\mathbf{c}$ -monotonic and  $\mathbf{c}$ -invariant if  $c_2(x, y)$  is interpreted by  $x + y$ . 139

For an ADP problem  $(\mathcal{P}, \mathcal{P}^-)$ , now the reduction pair processor has to orient the non-annotated rules  $\mathbf{b}(\mathcal{P} \cup \mathcal{P}^-)$  weakly and for all ADPs  $\ell \rightarrow r$ , it compares the annotated left-hand side  $\ell^\#$  with  $\text{ann}(r)$ . In strictly decreasing ADPs, one can then remove all annotations and consider them as relative (base) ADPs again. 141

**Theorem 36 (Reduction Pair Processor).** Let  $(\mathcal{P}, \mathcal{P}^-)$  be an ADP problem and let  $(\succsim, \succ)$  be a  $\mathbf{c}$ -monotonic and  $\mathbf{c}$ -invariant reduction pair such that  $\mathbf{b}(\mathcal{P} \cup \mathcal{P}^-) \subseteq \succsim$  and  $\ell^\# \succ \text{ann}(r)$  for all  $\ell \rightarrow r \in \mathcal{P} \cup \mathcal{P}^-$ . Moreover, let  $\mathcal{P}_\succ \subseteq \mathcal{P} \cup \mathcal{P}^-$  such that  $\ell^\# \succ \text{ann}(r)$  for all  $\ell \rightarrow r \in \mathcal{P}_\succ$ . Then  $\text{Proc}_{\text{RPP}}(\mathcal{P}, \mathcal{P}^-) = \{(\mathcal{P} \setminus \mathcal{P}_\succ, (\mathcal{P}^- \setminus \mathcal{P}_\succ) \cup \mathbf{b}(\mathcal{P}_\succ))\}$  is sound and complete. 142

*Example 37.* For the remaining ADP problem  $(\{(20)\} \cup \mathbf{b}(\mathcal{A}_1(\mathcal{R}_{\text{divL}}) \setminus \{(20)\}), \{(22)\})$  from Ex. 34, we can apply the reduction pair processor using the polynomial interpretation from the end of Sect. 3.1 which maps  $\mathcal{O}$  to 0,  $s(x)$  to  $x + 1$ ,  $\text{cons}(y, xs)$  to  $xs + 1$ ,  $\text{DL}(x, xs)$  to  $xs$ , and all other symbols to their first arguments. Then, (20) is oriented strictly (i.e., it is in  $\mathcal{P}_\succ$ ) and (22) is oriented weakly. Hence, we remove the annotation from (20) and move it to the base ADPs. Now there is no SCC with a main ADP anymore in the dependency graph, and thus the dependency graph processor returns  $\emptyset$ . This proves SN for  $(\mathcal{A}_1(\mathcal{R}_{\text{divL}}), \mathcal{A}_2(\mathcal{R}_{\text{mset2}}^-))$ , hence  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset2}}^-$  is also SN. 143

*Example 38.* Regard the ADPs  $\mathbf{a} \rightarrow \mathbf{b}$  and  $\mathbf{f} \rightarrow \mathbf{d}(\mathbf{F}, \mathbf{A})$  for the redex-creating Ex. 4 again. When using a polynomial interpretation  $\text{Pol}$  that maps  $c_0$  to 0 and  $c_2(x, y)$  to  $x + y$ , then for the reduction pair processor one has to satisfy  $\text{Pol}(\mathbf{A}) \geq 0$  and  $\text{Pol}(\mathbf{F}) \geq \text{Pol}(\mathbf{F}) + \text{Pol}(\mathbf{A})$ , i.e., one cannot make any of the ADPs strictly decreasing. 144



In contrast, for the variant with the terminating base rule  $f(s(y)) \rightarrow d(f(y), a)$  from Ex. 28, we have the ADPs  $a \rightarrow b$  and  $f(s(y)) \rightarrow d(f(y), A)$ . Here, the second constraint is  $\text{Pol}(F(s(y))) \geq \text{Pol}(F(y)) + \text{Pol}(A)$ . To make one of the ADPs strictly decreasing, one can set  $\text{Pol}(F(x)) = x$ ,  $\text{Pol}(s(x)) = x + 1$ , and  $\text{Pol}(A) = 1$  or  $\text{Pol}(A) = 0$ . Then the reduction pair processor removes the annotations from the strictly decreasing ADP and the dependency graph processor proves SN.

## 6 Evaluation and Conclusion

In this paper, we introduced the first notion of (annotated) dependency pairs and the first DP framework for relative termination, which also features suitable dependency graph and reduction pair processors for relative ADPs. Of course, further classical DP processors can be adapted to our relative ADP framework as well. For example, in our implementation of the novel ADP framework in our tool AProVE [11], we also included a straightforward adaption of the classical *rule removal processor* [9], see App. A.<sup>6</sup> In future work, we will investigate how to use our new form of ADPs for full (instead of innermost) rewriting also in the probabilistic setting and for complexity analysis.

To evaluate the new relative ADP framework, we compared its implementation in “new AProVE” to all other tools that participated in the most recent *termination competition (TermComp 2023)* [12] on relative rewriting, i.e., NaTT [31], TTT2 [23], MultumNonMultum [7], and “old AProVE” which did not yet contain the contributions of the current paper. In *TermComp 2023*, 98 benchmarks were used for relative termination. However, these benchmarks only consist of examples where the main TRS  $\mathcal{R}$  dominates the base TRS  $\mathcal{R}^=$  (i.e., which can be handled by Thm. 11 from [18]) or which can already be solved via simplification orders directly. Therefore, we extended the collection by 17 new examples, including both  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset}}^=$  from Ex. 1 and 2, and our leading example  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset2}}^=$  from Ex. 14 (where only new AProVE can prove SN). Except for  $\mathcal{R}_{\text{divL}}/\mathcal{R}_{\text{mset}}^=$ , in these examples  $\mathcal{R}$  does not dominate  $\mathcal{R}^=$ . Most of these examples adapt well-known classical TRSs from the *Termination Problem Data Base* [28] used at *TermComp* to the relative setting. In the following table, the number in the “YES” (“NO”) row indicates for how many of the 115 examples the respective tool could prove (disprove) relative termination and “MAYBE” refers to the benchmarks where the tool could not solve the problem within the timeout of 300 s per example. The numbers in brackets are the respective results when only considering our new 17 examples. “AVG(s)” gives the average runtime of the tool on solved examples in seconds.

	new AProVE	NaTT	old AProVE	TTT2	MultumNonMultum
YES	78 (17)	65 (7)	47 (4)	39 (3)	0 (0)
NO	13 (0)	5 (0)	13 (0)	7 (0)	13 (0)
MAYBE	24 (0)	45 (10)	55 (13)	69 (14)	102 (17)
AVG(s)	6.68	0.38	3.67	1.61	1.28

<sup>6</sup> This processor works analogously to the preprocessing at the beginning of Sect. 4 which can be used to remove duplicating rules: For an ADP problem  $(\mathcal{P}, \mathcal{P}^=)$ , it tries to find a reduction pair  $(\succsim, \succ)$  where  $\succ$  is closed under contexts such that  $b(\mathcal{P} \cup \mathcal{P}^=) \subseteq \succsim$ . Then for  $\mathcal{P}_\succ \subseteq \mathcal{P} \cup \mathcal{P}^=$  with  $b(\mathcal{P}_\succ) \subseteq \succ$ , the processor replaces the ADP by  $(\mathcal{P} \setminus \mathcal{P}_\succ, \mathcal{P}^= \setminus \mathcal{P}_\succ)$ .

The table clearly shows that while *old* AProVE was already the second most powerful tool for relative termination, the integration of the ADP framework in *new* AProVE yields a substantial advance in power (i.e., it only fails on 24 of the examples, compared to 45 and 55 failures of NaTT and *old* AProVE, respectively). In particular, previous tools (including *old* AProVE) often have problems with relative TRSs where the main TRS does not dominate the base TRS, whereas the ADP framework can handle such examples.

A special form of relative TRSs are *relative string rewrite systems (SRSs)*, where all function symbols have arity 1. Due to the base ADPs with two annotated symbols on the right-hand side, here the ADP framework is less powerful than dedicated techniques for string rewriting. For the 403 relative SRSs at *TermComp 2023*, the ADP framework only finds 71 proofs, mostly due to the dependency graph and the rule removal processor, while termination analysis via AProVE's standard strategy for relative SRSs succeeds on 209 examples, and the two most powerful tools for relative SRSs at *TermComp 2023* (MultumNonMultia and Matchbox [30]) succeed on 274 and 269 examples, respectively.

Another special form of relative rewriting is *equational rewriting*, where one has a set of equations  $E$  which correspond to relative rules that can be applied in both directions. In [8], DPs were adapted to equational rewriting. However, this approach requires  $E$ -unification to be decidable and finitary (i.e., for (certain) pairs of terms, it has to compute finite complete sets of  $E$ -unifiers). This works well if  $E$  are AC- or C-axioms, and for this special case, dedicated techniques like [8] are more powerful than our new ADP framework for relative termination. For example, on the 76 AC- and C-benchmarks for equational rewriting at *TermComp 2023*, the relative ADP framework finds 36 proofs, while dedicated tools for AC-rewriting like AProVE's equational strategy or MU-TERM [13] succeed on 66 and 64 examples, respectively. However, in general, the requirement of a finitary  $E$ -unification algorithm is a hard restriction. In contrast to existing tools for equational rewriting, our new ADP framework can be used for arbitrary (non-duplicating) relative rules.

For details on our experiments, our collection of examples, and for instructions on how to run our implementation in AProVE via its *web interface* or locally, see <https://aprove-developers.github.io/RelativeDTFramework/>

## References 0

1. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. *Theoretical Computer Science* **236**(1-2), 133–178 (2000). [https://doi.org/10.1016/S0304-3975\(99\)00207-8](https://doi.org/10.1016/S0304-3975(99)00207-8)
2. Baader, F., Nipkow, T.: *Term Rewriting and All That*. Cambridge University Press (1998). <https://doi.org/10.1017/CBO9781139172752>
3. Dershowitz, N.: Termination of rewriting. *Journal of Symbolic Computation* **3**(1), 69–115 (1987). [https://doi.org/10.1016/S0747-7171\(87\)80022-6](https://doi.org/10.1016/S0747-7171(87)80022-6)
4. Dershowitz, N., Treinen, R.: The RTA list of open problems. <https://www.win.tue.nl/rtaloop/>
5. Fuhs, C.: Transforming derivational complexity of term rewriting to run-time complexity. In: *Proc. FroCoS '19*. pp. 348–364. LNCS 11715 (2019). [https://doi.org/10.1007/978-3-030-29007-8\\_20](https://doi.org/10.1007/978-3-030-29007-8_20)

6. Geser, A.: Relative Termination. Ph.D. thesis, University of Passau, Germany (1990).  
[https://www.uni-ulm.de/fileadmin/website\\_uni\\_ulm/iui/Ulmer\\_Informatik\\_Berichte/1991/UIB-1991-03.pdf](https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui/Ulmer_Informatik_Berichte/1991/UIB-1991-03.pdf) 1
7. Geser, A., Hofbauer, D., Waldmann, J.: Sparse tiling through overlap closures for  
termination of string rewriting. In: Proc. FSCD '19. pp. 21:1–21:21. LIPIcs 131 (2019).  
<https://doi.org/10.4230/LIPICS.FSCD.2019.21> 0
8. Giesl, J., Kapur, D.: Dependency pairs for equational rewriting. In: Proc. RTA '01. pp.  
93–108. LNCS 2051 (2001). [https://doi.org/10.1007/3-540-45127-7\\_9](https://doi.org/10.1007/3-540-45127-7_9) 0
9. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Com-  
bining techniques for automated termination proofs. In: Proc. LPAR '04. pp. 301–331.  
LNCS 3452 (2004). [https://doi.org/10.1007/978-3-540-32275-7\\_21](https://doi.org/10.1007/978-3-540-32275-7_21) 120
10. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improv-  
ing dependency pairs. Journal of Automated Reasoning **37**(3), 155–203 (2006).  
<https://doi.org/10.1007/s10817-006-9057-7> 0
11. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel,  
J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.:  
Analyzing program termination and complexity automatically with AProVE. Journal of  
Automated Reasoning **58**(1), 3–31 (2017). <https://doi.org/10.1007/s10817-016-9388-y> 0
12. Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termina-  
tion and complexity competition. In: Proc. TACAS '19. pp. 156–166. LNCS  
11429 (2019). [https://doi.org/10.1007/978-3-030-17502-3\\_10](https://doi.org/10.1007/978-3-030-17502-3_10), website of *TermComp*:  
[https://termination-portal.org/wiki/Termination\\_Competition](https://termination-portal.org/wiki/Termination_Competition) 0
13. Gutiérrez, R., Lucas, S.: MU-TERM: Verify Termination Properties Automatically  
(System Description). In: Proc. IJCAR '20. pp. 436–447. LNCS 12167 (2020).  
[https://doi.org/10.1007/978-3-030-51054-1\\_28](https://doi.org/10.1007/978-3-030-51054-1_28) 0
14. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. Information  
and Computation **199**(1-2), 172–199 (2005). <https://doi.org/10.1016/j.ic.2004.10.004> 145
15. Hirokawa, N., Middeldorp, A.: Tyrolean termination tool: Techniques  
and features. Information and Computation **205**(4), 474–511 (2007).  
<https://doi.org/10.1016/J.IC.2006.08.010> 145
16. Hirokawa, N., Middeldorp, A.: Decreasing diagrams and relative ter-  
mination. Journal of Automated Reasoning **47**(4), 481–501 (2011).  
<https://doi.org/10.1007/S10817-011-9238-X> 5
17. Iborra, J., Nishida, N., Vidal, G.: Goal-directed and relative dependency pairs for  
proving the termination of narrowing. In: Proc. LOPSTR '09. pp. 52–66. LNCS 6037  
(2009). [https://doi.org/10.1007/978-3-642-12592-8\\_5](https://doi.org/10.1007/978-3-642-12592-8_5) 69
18. Iborra, J., Nishida, N., Vidal, G., Yamada, A.: Relative termination via  
dependency pairs. Journal of Automated Reasoning **58**(3), 391–411 (2017).  
<https://doi.org/10.1007/S10817-016-9373-5> 5
19. Kassing, J.C., Giesl, J.: Proving almost-sure innermost termination of probabilistic  
term rewriting using dependency pairs. In: Proc. CADE '23. pp. 344–364. LNCS 14132  
(2023). [https://doi.org/10.1007/978-3-031-38499-8\\_20](https://doi.org/10.1007/978-3-031-38499-8_20) 3
20. Kassing, J.C., Dollase, S., Giesl, J.: A complete dependency pair framework for almost-  
sure innermost termination of probabilistic term rewriting. In: Proc. FLOPS '24. LNCS  
(2024). <https://doi.org/10.48550/arXiv.2309.00344>, to appear. Long version at *CoRR*  
[abs/2309.00344](https://arxiv.org/abs/2309.00344) 145
21. Klein, D., Hirokawa, N.: Confluence of non-left-linear trss via rela-  
tive termination. In: Proc. LPAR '18. pp. 258–273. LNCS 7180 (2012).  
[https://doi.org/10.1007/978-3-642-28717-6\\_21](https://doi.org/10.1007/978-3-642-28717-6_21) 158
22. Koprowski, A., Zantema, H.: Proving liveness with fairness using rewriting. In: Proc.  
FroCoS '05. pp. 232–247. LNCS 3717 (2005). [https://doi.org/10.1007/11559306\\_13](https://doi.org/10.1007/11559306_13) 42

23. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean termi-  
nation tool 2. In: Proc. RTA '09. pp. 295–304. LNCS 5595 (2009).  
[https://doi.org/10.1007/978-3-642-02348-4\\_21](https://doi.org/10.1007/978-3-642-02348-4_21) 0
24. Lankford, D.S.: On proving term rewriting systems are Noetherian. Memo  
mtp-3, math. dept., Louisiana Technical University, Ruston, LA (1979).  
[http://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford\\_Poly\\_Term.pdf](http://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford_Poly_Term.pdf) 1
25. Nagele, J., Felgenhauer, B., Zankl, H.: Certifying confluence proofs via relative ter-  
mination and rule labeling. Logical Methods in Computer Science **13**(2) (2017).  
[https://doi.org/10.23638/LMCS-13\(2:4\)2017](https://doi.org/10.23638/LMCS-13(2:4)2017) 56
26. Nishida, N., Vidal, G.: Termination of narrowing via termination of rewriting. Appli-  
cable Algebra in Engineering, Communication and Computing **21**(3), 177–225 (2010).  
<https://doi.org/10.1007/S00200-010-0122-4> 69
27. Noschinski, L., Emmes, F., Giesl, J.: Analyzing innermost runtime complexity of term  
rewriting by dependency pairs. Journal of Automated Reasoning **51**, 27–56 (2013).  
[https://doi.org/10.1007/978-3-642-22438-6\\_32](https://doi.org/10.1007/978-3-642-22438-6_32) 10
28. TPDB (Termination Problem Data Base), <https://github.com/TermCOMP/TPDB> 3
29. Vidal, G.: Termination of narrowing in left-linear constructor sys-  
tems. In: Proc. FLOPS '08. pp. 113–129. LNCS 4989 (2008).  
[https://doi.org/10.1007/978-3-540-78969-7\\_10](https://doi.org/10.1007/978-3-540-78969-7_10) 3
30. Waldmann, J.: Matchbox: A tool for match-bounded string rewriting. In: Proc. RTA '04.  
pp. 85–94. LNCS 3091 (2004). [https://doi.org/10.1007/978-3-540-25979-4\\_6](https://doi.org/10.1007/978-3-540-25979-4_6) 0
31. Yamada, A., Kusakari, K., Sakabe, T.: Nagoya Termination  
Tool. In: Proc. RTA-TLCA '14. pp. 466–475. LNCS 8560 (2014).  
[https://doi.org/10.1007/978-3-319-08918-8\\_32](https://doi.org/10.1007/978-3-319-08918-8_32) 0
32. Zankl, H., Korp, M.: Modular complexity analysis for term rewriting. Logical Methods  
in Computer Science **10**(1) (2014). [https://doi.org/10.2168/LMCS-10\(1:19\)2014](https://doi.org/10.2168/LMCS-10(1:19)2014) 69

## A Appendix 5

In this appendix, we give all proofs for our new results and observations, and present an additional rule removal processor for our ADP framework (Thm. 42) that results from a straightforward adaption of the corresponding processor from the classical DP framework [9].

Before we start with the proof of the chain criterion, for any infinite rewrite sequence  $t_0 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} t_1 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \dots$  we define *origin graphs* that indicate which subterm of  $t_{i+1}$  “originates” from which subterm of  $t_i$ . Therefore, these graphs also indicate how annotations can move in the chains that correspond to this rewrite sequence. Note that due to the choice of the VRF and due to the fact that at most two defined symbols are annotated in the right-hand sides of ADPs, there are multiple possibilities for the annotations to move in a chain. Each origin graph corresponds to one possible way that the annotations can move.

**Definition 39 (Origin Graph).** Let  $\mathcal{R}$  and  $\mathcal{R}^=$  be two TRSs and let  $\Theta : t_0 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} t_1 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \dots$  be a rewrite sequence. A graph with the nodes  $(i, \pi)$  for all  $i \in \mathbb{N}$  and all  $\pi \in \text{pos}(t_i)$  is called an *origin graph* for  $\Theta$  if the edges are defined as follows: For  $i \in \mathbb{N}$ , let the rewrite step  $t_i \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} t_{i+1}$  be performed using the rule  $\ell \rightarrow r \in \mathcal{R} \cup \mathcal{R}^=$ , the position  $\tau$ , and the substitution  $\sigma$ , i.e.,  $t_i|_\tau = \ell\sigma$  and  $t_{i+1} = t_i[r\sigma]_\tau$ . Furthermore, let  $\pi \in \text{pos}(t_i)$ .

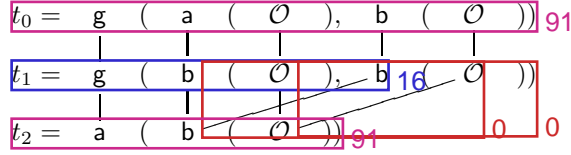
- (a) If  $\pi < \tau$  or  $\pi \perp \tau$  (i.e.,  $\pi$  is above or parallel to  $\tau$ ), then there is an edge from  $(i, \pi)$  to  $(i+1, \pi)$ . The reason is that if position  $\pi$  is annotated in  $t_i$ , then in a chain it would remain annotated in  $t_{i+1}$ .
- (b) For  $\pi = \tau$ , there are at most two outgoing edges from  $(i, \pi)$  if the rule is in  $\mathcal{R}^=$  and at most one edge if the rule is in  $\mathcal{R}$ . These edges lead to nodes of the form  $(i+1, \pi.\alpha)$  for  $\alpha \in \text{pos}_\Sigma(r)$ . The reason is that rules in  $\mathcal{A}_1(\mathcal{R})$  contain at most one annotation and rules in  $\mathcal{A}_2(\mathcal{R})$  contain at most two annotations in their right-hand sides. Moreover, if  $\pi$  is annotated in  $t_i$ , then we may create annotations at the positions  $\pi.\alpha$  in the right-hand side of the used ADP.
- (c) For every variable position  $\alpha_\ell \in \text{pos}_V(\ell)$ , either there are no outgoing edges from any node of the form  $(i, \tau.\alpha_\ell.\beta)$  with  $\beta \in \mathbb{N}^*$ , or there is a position  $\alpha_r \in \text{pos}_V(r)$  with  $r|_{\alpha_r} = \ell|_{\alpha_\ell}$  and for all  $\beta \in \mathbb{N}^*$  with  $\alpha_\ell.\beta \in \text{pos}(\ell\sigma)$ , there is an edge from  $(i, \tau.\alpha_\ell.\beta)$  to  $(i+1, \tau.\alpha_r.\beta)$ . This captures the behavior of VRFs.
- (d) For all other positions  $\pi \in \text{pos}(t_i)$ , there is no outgoing edge from the node  $(i, \pi)$ .

Due to Def. 39, we have the following connection between origin graphs and chains. For every origin graph for a rewrite sequence  $t_0 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} t_1 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \dots$  and every  $t_0$  such that  $b(t_0) = t_0$ , there is a chain  $t_0 \hookrightarrow_{\mathcal{A}_1(\mathcal{R}) \cup \mathcal{A}_2(\mathcal{R}^=)} t_1 \hookrightarrow_{\mathcal{A}_1(\mathcal{R}) \cup \mathcal{A}_2(\mathcal{R}^=)} \dots$  with  $b(\tilde{t}_i) = t_i$  for all  $i \in \mathbb{N}$  such that:

$$\pi \in \text{pos}_{\mathcal{D}^\#}(\tilde{t}_i) \quad \text{iff} \quad \text{there is a path in the origin graph from } (0, \tau) \text{ to } (i, \pi) \text{ for some } \tau \in \text{pos}_{\mathcal{D}^\#}(\tilde{t}_0)$$

**Example 40.** Let  $\mathcal{R} \cup \mathcal{R}^=$  have the rules  $a(x) \rightarrow b(x)$ ,  $g(x, x) \rightarrow a(x)$ ,  $b(x) \rightarrow b(x)$ , and consider the rewrite sequence  $\Theta : g(a(\mathcal{O}), b(\mathcal{O})) \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} g(b(\mathcal{O}), b(\mathcal{O})) \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \dots$

$a(b(\mathcal{O})) \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \dots$ . So here we have  $t_0 = g(a(\mathcal{O}), b(\mathcal{O}))$ ,  $t_1 = g(b(\mathcal{O}), b(\mathcal{O}))$ , and  $t_2 = a(b(\mathcal{O}))$ . The following is a possible origin graph for  $\Theta$ .



We can now prove the chain criterion in the relative setting. In the following, we often use the notation  $\text{pos}_f(t)$  instead of  $\text{pos}_{\{f\}}(t)$  for a term  $t$  and a single symbol or variable  $f \in \Sigma \cup \mathcal{V}$ .

**Theorem 23 (Chain Criterion for Relative Rewriting).** *Let  $\mathcal{R}$  and  $\mathcal{R}^=$  be TRSs such that  $\mathcal{R}^=$  is non-duplicating. Then  $\mathcal{R}/\mathcal{R}^=$  is SN iff the ADP problem  $(\mathcal{A}_1(\mathcal{R}), \mathcal{A}_2(\mathcal{R}^=))$  is SN.*

*Proof.* Completeness: Assume that the ADP problem  $(\mathcal{A}_1(\mathcal{R}), \mathcal{A}_2(\mathcal{R}^=))$  is not SN. Then, there exists an infinite  $(\mathcal{A}_1(\mathcal{R}), \mathcal{A}_2(\mathcal{R}^=))$ -chain, i.e., an infinite rewrite sequence  $t_0 \hookrightarrow_{\mathcal{A}_1(\mathcal{R}) \cup \mathcal{A}_2(\mathcal{R}^=)} t_1 \hookrightarrow_{\mathcal{A}_1(\mathcal{R}) \cup \mathcal{A}_2(\mathcal{R}^=)} \dots$  that uses an infinite number of rewrite steps with  $\mathcal{A}_1(\mathcal{R})$  and Case (pr).

By removing all annotations we obtain the rewrite sequence  $b(t_0) \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} b(t_1) \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \dots$  that uses an infinite number of rewrite steps with  $\mathcal{R}$ . Hence,  $\mathcal{R}/\mathcal{R}^=$  is not SN either.

Soundness: Assume that  $\mathcal{R}/\mathcal{R}^=$  is not SN. Then there exists an infinite sequence  $\Theta : t_0 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} t_1 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \dots$  that uses an infinite number of  $\mathcal{R}$ -rewrite steps.

We will define a sequence  $\tilde{t}_0, \tilde{t}_1, \dots$  of annotated terms such that  $b(\tilde{t}_i) = t_i$  and  $\tilde{t}_i \hookrightarrow_{\mathcal{A}_1(\mathcal{R}) \cup \mathcal{A}_2(\mathcal{R}^=)} \tilde{t}_{i+1}$  for all  $i \in \mathbb{N}$ , where we use an infinite number of rewrite steps with  $\mathcal{A}_1(\mathcal{R})$  and Case (pr). This is an infinite  $(\mathcal{A}_1(\mathcal{R}), \mathcal{A}_2(\mathcal{R}^=))$ -chain, and hence,  $(\mathcal{A}_1(\mathcal{R}), \mathcal{A}_2(\mathcal{R}^=))$  is not SN either.

W.l.o.g., let  $t_0$  be a minimal term that is non-terminating w.r.t.  $\mathcal{R}/\mathcal{R}^=$ , i.e., there exists no proper subterm of  $t_0$  that starts a rewrite sequence which uses an infinite number of  $\mathcal{R}$ -steps. Such a minimal term exists, since there are only finitely many subterms of  $t_0$ . Next, we prove that there exists an origin graph for  $\Theta$  with a path from  $(0, \varepsilon)$  to some node  $(i, \pi)$  and a path from  $(0, \varepsilon)$  to some node  $(i+1, \pi')$  with  $i \in \mathbb{N}$  such that the rewrite step  $t_i \rightarrow_{\mathcal{R}} t_{i+1}$  takes place at position  $\pi$ , and  $\tilde{t}_{i+1}|_{\pi'}$  is a minimal non-terminating term w.r.t.  $\mathcal{R}/\mathcal{R}^=$ . Then, according to the definition of the origin graph, there exists a chain  $t_0, \dots, t_i, t_{i+1}$  with  $b(t_j) = t_j$  for all  $1 \leq j \leq i+1$  such that the positions  $\pi$  in  $t_i$  and  $\pi'$  in  $t_{i+1}$  are annotated. Hence, the rewrite step  $\tilde{t}_i \hookrightarrow_{\mathcal{A}_1(\mathcal{R}) \cup \mathcal{A}_2(\mathcal{R}^=)} \tilde{t}_{i+1}$  is performed with  $\mathcal{A}_1(\mathcal{R})$  and Case (pr). Furthermore, there is a minimal non-terminating subterm of  $\tilde{t}_{i+1}$  whose root is annotated. Thus, we can perform the whole construction again to create another chain starting in  $\tilde{t}_{i+1}$  that ends in a rewrite step with  $\mathcal{A}_1(\mathcal{R})$  and Case (pr). By repeating this construction infinitely often, we generate our desired infinite chain.

It remains to prove that such an origin graph exists for every minimal non-terminating term  $t_0$ . Let  $k$  be the arity of  $t_0$ 's root symbol. Then for  $i \in \mathbb{N}$ , by induction we now define  $t_i^1, \dots, t_i^k$  such that  $t_i^1, \dots, t_i^k \triangleleft t_i$  at parallel positions and



such that  $t_i \rightarrow_{\mathcal{R} \cup \mathcal{R}^=}^1 t_{i+1} \rightarrow_{\mathcal{R} \cup \mathcal{R}^=}^1 \dots$  for all  $1 \leq j \leq k$ , where  $\rightarrow_{\mathcal{R} \cup \mathcal{R}^=}^1 = (\rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \cup =)$ , i.e.,  $\rightarrow_{\mathcal{R} \cup \mathcal{R}^=}^1$  is the reflexive closure of  $\rightarrow_{\mathcal{R} \cup \mathcal{R}^=}$ . In addition, for  $i \in \mathbb{N}$  we define a non-empty context  $C_i$  such that  $t_i = C_i[q_{1,1}, \dots, q_{1,h_1}, \dots, q_{k,1}, \dots, q_{k,h_k}]$  for subterms  $q_{j,1}, \dots, q_{j,h_j} \leq t_i$  at parallel positions for all  $1 \leq j \leq k$ . Moreover, if  $i > 0$ , then for all pairs of positions  $\pi_1, \pi_2 \in \text{pos}_\Sigma(C_i)$  we show that one can construct an origin graph with paths from  $(0, \varepsilon)$  to  $(i, \pi_1)$  and from  $(0, \varepsilon)$  to  $(i, \pi_2)$ . Note that there exists an  $i \in \mathbb{N}$  such that the rewrite step from  $t_i$  to  $t_{i+1}$  is done with an  $\mathcal{R}$ -rule at a position in  $C_i$ . The reason is that otherwise, infinitely many  $\mathcal{R}$ -steps would be applied on terms of the form  $q_{j,b}$ . But this would mean that there exists a  $t_i$  such that the sequence  $t_i \rightarrow_{\mathcal{R} \cup \mathcal{R}^=}^1 t_{i+1} \rightarrow_{\mathcal{R} \cup \mathcal{R}^=}^1 \dots$  has infinitely many  $\mathcal{R}$ -steps. However, this would be a contradiction to the minimality of  $t_0$  because  $t_i$  is a proper subterm of  $t_0$ . So we define  $t_i^1, \dots, t_i^k$  and  $C_i$  for all  $i \geq 0$  until we reach the first  $i$  where an  $\mathcal{R}$ -step is performed with a redex at a position in  $C_i$ .

We start with the case  $i = 0$ . Let  $t_0^1, \dots, t_0^k$  be the subterms of  $t_0$  at positions  $1, \dots, k$  (i.e.,  $t_0^j = t_0|_j$  for all  $1 \leq j \leq k$ ). Then, we have  $t_0 = C_0[t_0^1, \dots, t_0^k]$  for the context  $C_0$  that only consists of  $t_0$ 's root symbol applied to  $k$  holes.

In the induction step, we have  $t_i \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} t_{i+1}$  using a position  $\pi$ , a substitution  $\sigma$ , and a rule  $\ell \rightarrow r$  with  $t_i|_\pi = \ell\sigma$  and  $t_{i+1} = t_i[r\sigma]_\pi$ . Here, we have two cases:

- If  $\pi \notin \text{pos}(C_i) \setminus \text{pos}_\square(C_i)$ , then  $\pi$  must be in some  $q_{i,b} \leq t_i$ . So there is an  $\alpha \in \text{pos}_\square(C_i)$  such that  $\pi = \alpha.\beta$  for some  $\beta \in \mathbb{N}^*$ . Hence, we have  $t_i|_\pi = C_i[q_{1,1}, \dots, q_{1,h_1}, \dots, q_{k,1}, \dots, q_{k,h_k}]|_\pi = q_{j,b}|_\beta$  for some  $q_{j,b} = t_i|_j$ . Here, we simply perform the rewrite step on this term, and the context and the other terms remain the same. Hence, we have  $C_{i+1} = C_i$ ,  $t_{i+1}^j = t_i^j|_\beta$  and  $t_{i+1}^j = t_i^j$  for all  $1 \leq j \leq k$  with  $j' \neq j$ . Using the subterms  $q'_{1,1}, \dots, q'_{1,h_1}, \dots, q'_{k,1}, \dots, q'_{k,h_k}$  with  $q'_{j,b} = q_{j,b}[r\sigma]_\beta$  and  $q'_{c,d} = q_{c,d}$  for all  $1 \leq c \leq k$  and  $1 \leq d \leq h_c$  with  $(c,d) \neq (j,b)$ , we finally get  $t_{i+1} = C_{i+1}[q'_{1,1}, \dots, q'_{1,h_1}, \dots, q'_{k,1}, \dots, q'_{k,h_k}]$  as desired.

It remains to prove that our claim on the paths in the origin graph is still satisfied. For all  $\tau_1, \tau_2 \in \text{pos}_\Sigma(C_{i+1}) = \text{pos}_\Sigma(C_i)$ , by the induction hypothesis there exists an origin graph with paths from  $(0, \varepsilon)$  to  $(i, \tau_1)$  and from  $(0, \varepsilon)$  to  $(i, \tau_2)$ . Since the origin graph has edges from  $(i, \tau_1)$  to  $(i+1, \tau_1)$  and from  $(i, \tau_2)$  to  $(i+1, \tau_2)$  by Def. 39 since  $\tau_1$  and  $\tau_2$  are above or parallel to  $\pi$ , there are also paths from  $(0, \varepsilon)$  to  $(i+1, \tau_1)$  and from  $(0, \varepsilon)$  to  $(i+1, \tau_2)$ .

- Now we consider the case  $\pi \in \text{pos}(C_i) \setminus \text{pos}_\square(C_i)$ . If the step  $t_i \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} t_{i+1}$  is an  $\mathcal{R}$ -step, then we stop, because we reached the first  $\mathcal{R}$ -step where the redex is at a position in  $C_i$ .

So we now have  $t_i \rightarrow_{\mathcal{R}^=} t_{i+1}$ . We define  $t_{i+1}^j = t_i^j$  for all  $1 \leq j \leq k$ , but we still need to define the context and the subterms for each  $t_{i+1}^j$ . We will now define a suitable VRF  $\varphi : \text{pos}_\Sigma(\ell) \rightarrow \text{pos}_\Sigma(r) \cup \{\perp\}$  step by step, where we initialize  $\varphi$  to yield  $\perp$  for all arguments. For every  $\rho \in \text{pos}_\Sigma(\ell)$ , if possible, we let  $\varphi(\rho) \in \text{pos}_\Sigma(r)$  be a position of  $r$  that is not yet in the image of  $\varphi$  and where  $\ell|_\rho = r|_{\varphi(\rho)}$ . If there is no such position of  $r$ , then we keep  $\varphi(\rho) = \perp$ .

Let  $\varphi|_{\text{pos}_\Sigma(r)}$  denote the restriction of  $\varphi$  to the codomain  $\text{pos}_\Sigma(r)$  (i.e.,  $\varphi|_{\text{pos}_\Sigma(r)}$  is only defined on those  $\rho \in \text{pos}_\Sigma(\ell)$  where  $\varphi(\rho) \neq \perp$ ). Then  $\varphi|_{\text{pos}_\Sigma(r)}$  is surjective, since rules of  $\mathcal{R}^=$  must not be duplicating, and injective, since we only extend

the function  $\varphi$  if a position of a variable in  $r$  was not already in the image of  $\varphi$ . Let  $\{\rho_1, \dots, \rho_w\} \subseteq \text{pos}_V(\ell)$  be those positions of variables from  $\ell$  in the context  $C_i$  that are no holes (i.e., where  $\pi.\rho_z \in \text{pos}(C_i) \setminus \text{pos}_\square(C_i)$ ) and where  $\varphi(\rho_z) \neq \perp$  for all  $1 \leq z \leq w$ . Then we define the new context  $C_{i+1}$  as  $C_{i+1} = C_i[r\delta_\square]_\pi[C_i|_{\pi.\rho_1}]_{\pi.\varphi(\rho_1)} \dots [C_i|_{\pi.\rho_w}]_{\pi.\varphi(\rho_w)}$  using the substitution  $\delta_\square$  that maps every variable to  $\square$ . So  $C_{i+1}$  results from  $C_i$  by replacing the subterm at position  $\pi$  by  $r$  where all variables are substituted with  $\square$ , and then restoring the part of the context that was inside the substitution. 176

Next, we need to define the subterms  $q'_{j,1}, \dots, q'_{j,h}$  of  $t_{i+1}$  such that  $t_{i+1} = C_{i+1}[q'_{1,1}, \dots, q'_{1,h}, q'_{k,1}, \dots, q'_{k,h}]$ . Let  $\kappa$  be the position of some  $q_{j,b}$  in  $t_i$ . First, consider the case that for some  $\alpha_\ell \in \text{pos}_V(\ell)$  and  $\beta \in \mathbb{N}^*$  we have  $\pi.\alpha_\ell.\beta = \kappa$ , i.e.,  $q_{j,b}$  is completely inside the substitution. If  $\varphi(\alpha_\ell) = \perp$ , then we remove the subterm  $q_{j,b}$  in  $t_{i+1}$ . Otherwise,  $q_{j,b}$  is one of the subterms  $q'_{j,b'}$  and it moves from position  $\pi.\alpha_\ell.\beta$  in  $t_i$  to position  $\pi.\varphi(\alpha_\ell).\beta$  in  $t_{i+1}$ . Note that there are no two such  $q_{j_1,b_1}, q_{j_2,b_2}$  that move to the same position, due to injectivity of  $\varphi$ . Second,  $q_{j,b}$  is also one of the subterms  $q'_{j,b'}$  if the position  $\kappa$  is parallel to  $\pi$ . Here, the position of  $q_{j,b}$  in  $t_{i+1}$  remains the same as in  $t_i$ . Third, we consider the case that there exist some  $\alpha_\ell \in \text{pos}_V(\ell)$  such that  $\kappa < \pi.\alpha_\ell$ , i.e.,  $q_{j,b}$  is a subterm of the redex but not completely inside the substitution. For all such  $\alpha_\ell$ , let  $\chi_{\alpha_\ell} \in \mathbb{N}^*$  such that  $\kappa.\chi_{\alpha_\ell} = \pi.\alpha_\ell$ . Instead of the subterm  $q_{j,b}$ , we now use the subterms  $q_{j,b}|_{\chi_{\alpha_\ell}}$  for all those  $\alpha_\ell$  with  $\varphi(\alpha_\ell) \neq \perp$ . Now  $q_{j,b}|_{\chi_{\alpha_\ell}}$  is a subterm of  $t_{i+1}$  at position  $\pi.\varphi(\alpha_\ell)$ . All in all, we get  $t_{i+1} = C_{i+1}[q'_{1,1}, \dots, q'_{1,h}, q'_{k,1}, \dots, q'_{k,h}]$  and  $q'_{j,1}, \dots, q'_{j,h}$  at parallel positions for all  $1 \leq j \leq k$ . 170

Finally, we prove that our claim on the paths in the origin graph is still satisfied. Consider two positions  $\tau_1, \tau_2 \in \text{pos}_\Sigma(C_{i+1})$ , and let  $\tau \in \{\tau_1, \tau_2\}$ . If  $\tau$  is above or parallel to  $\pi$ , then by the induction hypothesis there exists an origin graph with a path from  $(0, \varepsilon)$  to  $(i, \tau)$ . Since the origin graph has an edge from  $(i, \tau)$  to  $(i+1, \tau)$  by Def. 39, there is a path from  $(0, \varepsilon)$  to  $(i+1, \tau)$ . If  $\tau$  has the form  $\pi.\alpha$  with  $\alpha \in \text{pos}_\Sigma(r)$  in  $C_{i+1}$ , then by the induction hypothesis there is an origin graph with a path from  $(0, \varepsilon)$  to  $(i, \pi)$ . Since the origin graph can be chosen to have an edge from  $(i, \pi)$  to  $(i+1, \pi.\alpha)$  by Def. 39, there is a path from  $(0, \varepsilon)$  to  $(i+1, \pi.\alpha)$ . Note that if both  $\tau_1$  and  $\tau_2$  have such a form (i.e.,  $\tau_1 = \pi.\alpha_1$  and  $\tau_2 = \pi.\alpha_2$  with  $\alpha_1, \alpha_2 \in \text{pos}_\Sigma(r)$ ), then the origin graph can be chosen to have edges from  $(i, \pi)$  to both  $(i+1, \pi.\alpha_1)$  and  $(i+1, \pi.\alpha_2)$ , as we can have two such edges for a rewrite step with a relative rule from  $\mathcal{R}^-$ . Finally, if  $\tau$  has the form  $\pi.\alpha_r.\beta$  with  $\alpha_r \in \text{pos}_V(r)$  in  $C_{i+1}$ , then since the image of  $\varphi$  consists of all variable positions from  $\text{pos}_V(r)$  (due to non-duplication of  $\mathcal{R}^-$ ), there is a  $\rho \in \text{pos}_V(\ell)$  with  $\ell|_\rho = r|_{\varphi(\rho)}$  where  $\varphi(\rho) = \alpha_r$ . By the induction hypothesis, there is an origin graph with a path from  $(0, \varepsilon)$  to  $(i, \pi.\rho.\beta)$  and by Def. 39, the origin graph can be chosen to have an edge from  $(i, \pi.\rho.\beta)$  to  $(i+1, \pi.\alpha_r.\beta)$ . All in all, there exists an origin graph with paths from  $(0, \varepsilon)$  to  $(i, \tau_1)$  and from  $(0, \varepsilon)$  to  $(i, \tau_2)$ . 176

So we have shown that there exists an origin graph for  $\Theta$  with a path from  $(0, \varepsilon)$  to some node  $(i, \pi)$  such that the rewrite step  $t_i \rightarrow_{\mathcal{R}} t_{i+1}$  takes place at position  $\pi$  in the context  $C_i$ . Moreover, for any further position  $\pi' \in \text{pos}_\Sigma(C_i)$ , in this

origin graph there is also a path from  $(0, \varepsilon)$  to some node  $(i, \pi')$ . In a similar way, one can also extend the construction one step further to define  $t_{i+1}^1, \dots, t_{i+1}^k$  and a non-empty context  $C_{i+1}$  such that  $t_{i+1} = C_{i+1}[q_{1,1}, \dots, q_{1,h_1}, \dots, q_{k,1}, \dots, q_{k,h_k}]$  for subterms  $q_{j,1}, \dots, q_{j,h_j} \leq t_{i+1}^j$  (not necessarily at parallel positions, since the used  $\mathcal{R}$ -step may be duplicating). Moreover, for all positions  $\pi_1 \in \text{pos}_\Sigma(C_i)$  and  $\pi_2 \in \text{pos}_\Sigma(C_{i+1})$  one can construct an origin graph with paths from  $(0, \varepsilon)$  to  $(i, \pi_1)$  and from  $(0, \varepsilon)$  to  $(i+1, \pi_2)$ . For this last step, the cases of the proof are exactly the same, the only difference is that from node  $(i, \pi)$ , there may only be a single outgoing edge (since the used rule was in  $\mathcal{R}$ ). But as we are only looking for a single position  $\pi_2$  in  $\text{pos}_\Sigma(C_{i+1})$  (and no pairs of positions anymore), this suffices for the construction. 177

Then we have shown that the desired origin graph exists, by choosing  $\pi_1$  to be  $\pi$  (the position of the redex in the  $\mathcal{R}$ -step from  $t_i$  to  $t_{i+1}$ ) and by choosing  $\pi_2$  to be the position of a minimal non-terminating subterm of  $t_{i+1}$ . Note that this position must be in  $\text{pos}_\Sigma(C_{i+1})$ , because all subterms  $q_{j,b}$  are terminating w.r.t.  $\mathcal{R}/\mathcal{R}^\infty$ . 177

For the following proof, recall that  $t \leq_\#^\tau s$  if  $\tau \in \text{pos}_{\mathcal{D}^\#}(s)$  and  $t = b(s|_\tau)$ . 179

**Theorem 26 (Derelativizing Processor (1)).** *Let  $(\mathcal{P}, \mathcal{P}^\infty)$  be an ADP problem such that  $b(\mathcal{P}^\infty) = \mathcal{P}^\infty$ . Then  $\text{Proc}_{\text{DRP1}}(\mathcal{P}, \mathcal{P}^\infty) = \emptyset$  is sound and complete iff the ordinary DP problem  $(\text{DP}(\mathcal{P}), b(\mathcal{P} \cup \mathcal{P}^\infty))$  is SN.* 142

*Proof.* Completeness of any processor that yields the empty set is trivial. So we only have to consider soundness. 180

Only if: Assume that  $(\text{DP}(\mathcal{P}), b(\mathcal{P} \cup \mathcal{P}^\infty))$  is not SN. Then there exists an infinite sequence  $t_0, t_1, t_2, \dots$  with  $t_i \xrightarrow{\varepsilon}_{\text{DP}(\mathcal{P})} \circ \rightarrow_{b(\mathcal{P} \cup \mathcal{P}^\infty)}^* t_{i+1}$  for all  $i \in \mathbb{N}$ . We now create a sequence  $s_0, s_1, \dots$  of annotated terms such that  $s_i \xrightarrow{\text{(pr)}}_{\mathcal{P}}^* s_{i+1}$  and  $b(t_i) \leq_\# s_i$  for all  $i \in \mathbb{N}$ , which by Thm. 23 implies that  $(\mathcal{P}, \mathcal{P}^\infty)$  is not SN, i.e., the processor is not sound. Initially, we start with the term  $s_0 = t_0$ . We have 181

$$t_0 \xrightarrow{\varepsilon}_{\text{DP}(\mathcal{P})} t_{0,1} \rightarrow_{b(\mathcal{P} \cup \mathcal{P}^\infty)} t_{0,2} \rightarrow_{b(\mathcal{P} \cup \mathcal{P}^\infty)} \dots \rightarrow_{b(\mathcal{P} \cup \mathcal{P}^\infty)} t_1$$

In the first rewrite step, there is a DP  $\ell^\# \rightarrow t^\# \in \text{DP}(\mathcal{P})$  and a substitution  $\sigma$  such that  $t_0 = \ell^\# \sigma$  and  $t_{0,1} = t^\# \sigma$ . This DP  $\ell^\# \rightarrow t^\#$  results from some ADP  $\ell \rightarrow r \in \mathcal{P}$  with  $t \leq_\#^\tau r$  for some position  $\tau \in \text{pos}(r)$ . We can rewrite  $s_0$  with  $\ell \rightarrow r$  and the substitution  $\sigma$  at the root resulting in  $s_{0,1} = r\sigma$  with  $b(t_{0,1}) \leq_\#^\tau r\sigma = s_{0,1}$ . Then, we mirror each step that takes place at position  $\pi$  in  $t_{0,i}$  at position  $\tau.\pi$  in  $s_{0,i}$ . To be precise, if we have  $t_{0,i+1} = t_{0,i}[r'\sigma']_\pi$  using a rule  $\ell' \rightarrow b(r') \in \mathcal{P} \cup \mathcal{P}^\infty$ , the substitution  $\sigma'$ , and the position  $\pi$ , then we have  $b(t_{0,i}) \leq_\#^\tau s_{0,i}$ , and can rewrite  $s_{0,i}$  with the ADP  $\ell' \rightarrow r'$ , the substitution  $\sigma'$ , and the position  $\tau.\pi$ . We get  $b(s_{0,i+1}) = b(s_{0,i}[r'\sigma']_{\tau.\pi})$  with  $b(t_{0,i+1}) = b(t_{0,i}[r'\sigma']_\pi) \leq_\#^\tau s_{0,i+1}$ . In the end, we have  $b(t_1) \leq_\# s_1$ . 181

We can now repeat this for each  $i \in \mathbb{N}$  and result in our desired chain. 169

If: Assume that the processor is not sound and that  $(\mathcal{P}, \mathcal{P}^\infty)$  is not SN. Then, by Thm. 23 there exists an infinite sequence  $t_0, t_1, t_2, \dots$  of annotated terms such that  $t_i \xrightarrow{\text{(pr)}}_{\mathcal{P}}^* t_{i+1}$  for all  $i \in \mathbb{N}$ . W.l.o.g., let  $t_0$  contain only a single annotation (we only need the annotation for the position that leads to infinitely many  $\mathcal{P}$ -steps with Case (pr)). We now create a sequence  $s_0, s_1, \dots$  such that 181

$s_i \xrightarrow{\varepsilon}_{\text{DP}(\mathcal{P})} \circ \rightarrow_{b(\mathcal{P} \cup \mathcal{P}^=)}^* s_{i+1}$  and  $b(s_i) \leq_{\#} t_i$  for all  $i \in \mathbb{N}$ , which implies that  $(\text{DP}(\mathcal{P}), b(\mathcal{P} \cup \mathcal{P}^=))$  is not SN. Due to the condition  $b(\mathcal{P}^=) = \mathcal{P}^=$ , we have

$$t_0 \hookrightarrow_{\mathcal{P}}^{(\mathbf{pr})} t_{0,1} \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^=}^{(\mathbf{r})} t_{0,2} \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^=}^{(\mathbf{r})} \dots \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^=}^{(\mathbf{r})} t_1$$

The reason that we have no  $\hookrightarrow_{\mathcal{P}^=}^{(\mathbf{pr})}$ -step is that all terms  $t_i$  and  $t_{i,j}$  only contain a single annotation. Since  $b(\mathcal{P}^=) = \mathcal{P}^=$ , we only have annotations in  $\mathcal{P}$ , where every right-hand side of a rule only has at most a single annotation. Thus, no term  $t_i$  or  $t_{i,j}$  can have more than one annotation. If we would rewrite at the position of this annotation with a rule without annotations (e.g., from  $\mathcal{P}^=$ ), then we would not have any annotation left and there would be no future  $\hookrightarrow_{\mathcal{P}^=}^{(\mathbf{pr})}$ -step possible anymore.

In the first rewrite step, there is an ADP  $\ell \rightarrow r \in \mathcal{P}$ , a substitution  $\sigma$ , and a position  $\pi$  such that  $t_0|_{\pi} = \ell^{\#}\sigma$  and  $t_{0,1} = t_0[\#_{\rho}(r\sigma)]_{\pi}$  for some  $\rho \in \text{pos}(r)$ . Initially, we start with the term  $s_0 = \ell^{\#}\sigma$ . We can rewrite  $s_0$  with  $\ell^{\#} \rightarrow t^{\#} \in \text{DP}(\mathcal{P})$  for  $t \leq_{\#} r$  and the substitution  $\sigma$  resulting in  $s_{0,1} = t^{\#}\sigma$  with  $b(s_{0,1}) = t\sigma \leq_{\#} \#_{\rho}(r\sigma)$ , i.e.,  $b(s_{0,1}) \leq_{\#}^{\kappa} t_0[\#_{\rho}(r\sigma)]_{\pi} = t_{0,1}$  for  $\kappa = \pi.\rho$ . Then, each rewrite step in  $t_{0,i}$  is mirrored in  $s_{0,i}$ .

To be precise, let  $b(t_{0,i+1}) = b(t_{0,i}[r'\sigma']_{\tau})$  using a rule  $\ell' \rightarrow r' \in b(\mathcal{P} \cup \mathcal{P}^=)$ , the substitution  $\sigma'$ , and the position  $\tau$ . If  $\tau$  is above or parallel to the position  $\kappa$  of the subterm that corresponds to our classical chain, then we do nothing and set  $s_{0,i} = s_{0,i+1}$ . However, a rewrite step on or above  $\kappa$  might change the position of this subterm, i.e., the position that we denoted with  $\kappa$  may be modified in each such step. Note however, that this subterm cannot be erased by such a step because then we would remove the only annotated symbol and the chain would become finite. If  $\tau$  is below  $\kappa$  (i.e., we have  $\tau = \kappa.\pi'$  for some position  $\pi'$ ), then we have  $b(s_{0,i}) \leq_{\#}^{\kappa} t_{0,i}$ , and can rewrite  $s_{0,i}$  with the rule  $\ell' \rightarrow b(r')$ , the substitution  $\sigma'$ , and the position  $\pi'$ . If  $\kappa$  denotes the current (possibly changed) position of the corresponding subterm, then we get  $b(s_{0,i+1}) = b(s_{0,i}[r'\sigma']_{\pi'}) \leq_{\#}^{\kappa} t_{0,i+1}$ . Finally,  $\tau$  cannot be  $\kappa$ , since then this would not be a  $(\mathbf{r})$ -rewrite step but a  $(\mathbf{pr})$ -rewrite step. In the end, we obtain  $b(s_1) \leq_{\#} t_1$ .

We can now repeat this for each  $i \in \mathbb{N}$  and result in our desired chain.  $\square$

**Theorem 27 (Derelativifying Processor (2)).** *Let  $(\mathcal{P}, \mathcal{P}^=)$  be an ADP problem, and let  $\mathcal{P}^= = \mathcal{P}_a^- \uplus \mathcal{P}_b^-$ . Then  $\text{Proc}_{\text{DRP2}}(\mathcal{P}, \mathcal{P}^=) = \{(\mathcal{P} \cup \text{split}(\mathcal{P}_a^-), \mathcal{P}_b^-)\}$  is sound. Here,  $\text{split}(\mathcal{P}_a^-) = \{\ell \rightarrow \#_{\pi}(r) \mid \ell \rightarrow r \in \mathcal{P}_a^-, \pi \in \text{pos}_{\mathcal{D}\#}(r)\}$ .*

*Proof. Soundness:* By Def. 22,  $(\mathcal{P}, \mathcal{P}^=)$  is not SN iff there exists an infinite  $(\mathcal{P}, \mathcal{P}^=)$ -chain. Such a chain would also be an infinite  $(\mathcal{P} \cup \mathcal{P}_a^-, \mathcal{P}_b^-)$ -chain. There is an infinite  $(\mathcal{P} \cup \mathcal{P}_a^-, \mathcal{P}_b^-)$ -chain iff there is an infinite  $(\mathcal{P} \cup \text{split}(\mathcal{P}_a^-), \mathcal{P}_b^-)$ -chain as we only need at most a single annotation in the main ADPs. By Def. 22, this is equivalent to non-termination of  $(\mathcal{P} \cup \text{split}(\mathcal{P}_a^-), \mathcal{P}_b^-)$ .  $\square$

Before we prove the soundness and completeness of the dependency graph processor, we present another definition regarding origin graphs. In Def. 39 we have seen that for a *non-annotated* rewrite sequence  $t_0 \rightarrow_{\mathcal{R} \cup \mathcal{R}^=} \dots$  one can obtain several different origin graphs. Now, we define the *canonical* origin graph for an *annotated* rewrite sequence  $t_0 \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^=} \dots$ . This graph represents the flow of the annotations in this sequence.

**Definition 41 (Canonical Origin Graph).** Let  $(\mathcal{P}, \mathcal{P}^-)$  be an ADP problem and let  $\Theta : t_0 \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^-} t_1 \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^-} \dots$ . The canonical origin graph for  $\Theta$  has the nodes  $(i, \pi)$  for all  $i \in \mathbb{N}$  and all  $\pi \in \text{pos}(t_i)$ , and its edges are defined as follows: For  $i \in \mathbb{N}$ , let the step  $t_i \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^-} t_{i+1}$  be performed using the rule  $\ell \rightarrow r \in \mathcal{P} \cup \mathcal{P}^-$ , the position  $\tau$ , the substitution  $\sigma$ , and the VRF  $\varphi$ . Furthermore, let  $\pi \in \text{pos}(t_i)$ .

- (a) If  $\pi < \tau$  or  $\pi \perp \tau$  (i.e.,  $\pi$  is above or parallel to  $\tau$ ), then there is an edge from  $(i, \pi)$  to  $(i+1, \pi)$ .
- (b) For  $\pi = \tau$ , there is an edge from  $(i, \pi)$  to  $(i+1, \pi.\alpha)$  for all  $\alpha \in \text{pos}_{D^\#}(r)$ .
- (c) If  $\pi = \tau.\alpha_\ell.\beta$  for a variable position  $\alpha_\ell \in \text{pos}_V(\ell)$  and  $\beta \in \mathbb{N}^*$ , then there is an edge from  $(i, \tau.\alpha_\ell.\beta)$  to  $(i+1, \tau.\varphi(\alpha_\ell).\beta)$  if  $\varphi(\alpha_\ell) \neq \perp$ .
- (d) For all other positions  $\pi \in \text{pos}(t_i)$ , there is no outgoing edge from the node  $(i, \pi)$ .

Moreover, if an ADP is applied with Case (pr) at position  $\tau$  in  $t_i$  in  $\Theta$ , then all edges originating in  $(i, \tau)$  are labeled with this ADP. All other edges are not labeled.

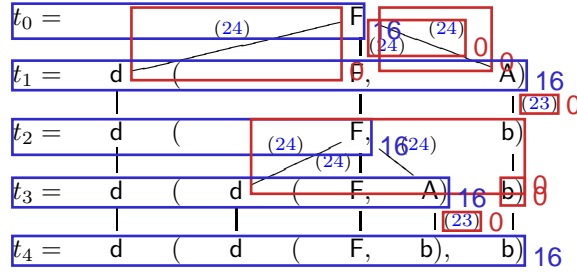
So for  $\mathcal{R}_2$  from Ex. 4 where  $\mathcal{A}_1(\mathcal{R}_2)$  consists of

$$a \rightarrow b \quad (23) \quad 0$$

and  $\mathcal{A}_2(\mathcal{R}_2^-)$  consists of

$$f \rightarrow d(F, A) \quad (24) \quad 0$$

the chain from Ex. 24 yields the following canonical origin graph:



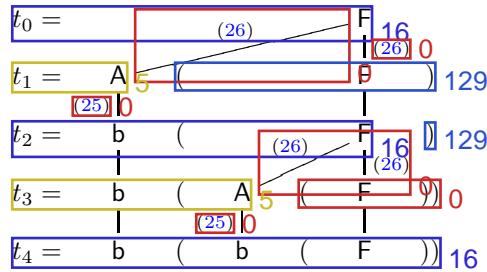
For  $\mathcal{R}_3$  from Ex. 5 where  $\mathcal{A}_1(\mathcal{R}_3)$  consists of

$$a(x) \rightarrow b(x) \quad (25) \quad 0$$

and  $\mathcal{A}_2(\mathcal{R}_3^-)$  consists of

$$f \rightarrow A(F) \quad (26) \quad 0$$

the chain from Ex. 25 yields the following canonical origin graph:



**Theorem 33 (Dep. Graph Processor).** *Let  $(\mathcal{P}, \mathcal{P}^=)$  be an ADP problem. Then* 128

$$\text{Proc}_{\text{DG}}(\mathcal{P}, \mathcal{P}^=) = \{(\mathcal{P} \cap \mathcal{Q}, (\mathcal{P}^= \cap \mathcal{Q}) \cup \mathfrak{b}((\mathcal{P} \cup \mathcal{P}^=) \setminus \mathcal{Q})) \mid \mathcal{Q} \in \text{SCC}_{\mathcal{P}}^{(\mathcal{P}, \mathcal{P}^=)} \cup \text{Lasso}\} \quad 0$$

*is sound and complete.* 49

*Proof.* Completeness: For every  $(\mathcal{P}', \mathcal{P}'=)$ -chain with  $(\mathcal{P}', \mathcal{P}'=) \in \text{Proc}_{\text{DG}}(\mathcal{P}, \mathcal{P}^=)$  191  
there exists a  $(\mathcal{P}, \mathcal{P}^=)$ -chain with the same terms and possibly more annotations. 191  
Hence, if some ADP problem in  $\text{Proc}_{\text{DG}}(\mathcal{P}, \mathcal{P}^=)$  is not SN, then  $(\mathcal{P}, \mathcal{P}^=)$  is not SN 191  
either. 191

**Soundness:** By the definition of  $\hookrightarrow$ , whenever there is a path from an edge labeled 191  
with an ADP  $\ell \rightarrow r$  to an edge labeled with an ADP  $\ell' \rightarrow r'$  in the canonical origin 191  
graph, then there is a path from  $\ell \rightarrow r$  to  $\ell' \rightarrow r'$  in the dependency graph. Since 191  
there only exist finitely many ADPs and the chain uses ADPs from  $\mathcal{P}$  with Case 191  
(**pr**) infinitely many times, there are two cases: 191

Either there exists a path in the canonical origin graph where infinitely many 191  
edges are labeled with an ADP from  $\mathcal{P}$ . Then after finitely many steps, this path 191  
only uses edges labeled with ADPs from an SCC  $\mathcal{Q}$  of the dependency graph that 191  
contains an ADP from  $\mathcal{P}$  (i.e., from a  $\mathcal{Q} \in \text{SCC}_{\mathcal{P}}^{(\mathcal{P}, \mathcal{P}^=)}$ ). 191

Otherwise, there is no such path in the canonical origin graph, but then there is 191  
a path in the canonical origin graph where infinitely many edges are labeled with 191  
ADPs from  $\mathcal{P}^=$  and this path generates infinitely many paths that lead to an edge 191  
labeled with an ADP from  $\mathcal{P}$ . Then after finitely many steps, this path only uses 191  
edges labeled with ADPs from a minimal lasso of the dependency graph (i.e., from 191  
a  $\mathcal{Q} \in \text{Lasso}$ ). 191

So in both cases, there exists a  $\mathcal{Q} \in \text{SCC}_{\mathcal{P}}^{(\mathcal{P}, \mathcal{P}^=)} \cup \text{Lasso}$  such that the infinite 97  
path gives rise to an infinite  $((\mathcal{P} \cap \mathcal{Q}) \cup \mathfrak{b}(\mathcal{P} \setminus \mathcal{Q}), (\mathcal{P}^= \cap \mathcal{Q}) \cup \mathfrak{b}(\mathcal{P}^= \setminus \mathcal{Q}))$ -chain. 97  
Since the ADPs  $\mathfrak{b}(\mathcal{P} \setminus \mathcal{Q})$  are never used for steps with Case (**pr**) in this infinite 97  
chain, they can also be moved to the base ADPs. Thus, this is also an infinite 97  
 $(\mathcal{P} \cap \mathcal{Q}, (\mathcal{P}^= \cap \mathcal{Q}) \cup \mathfrak{b}((\mathcal{P} \cup \mathcal{P}^=) \setminus \mathcal{Q}))$ -chain, i.e.,  $(\mathcal{P} \cap \mathcal{Q}, (\mathcal{P}^= \cap \mathcal{Q}) \cup \mathfrak{b}((\mathcal{P} \cup \mathcal{P}^=) \setminus \mathcal{Q}))$  97  
is not SN either. 97  $\square$  97

**Theorem 36 (Reduction Pair Processor).** *Let  $(\mathcal{P}, \mathcal{P}^=)$  be an ADP problem 142  
and let  $(\succ, \succ^=)$  be a  $\mathfrak{c}$ -monotonic and  $\mathfrak{c}$ -invariant reduction pair such that  $\mathfrak{b}(\mathcal{P} \cup \mathcal{P}^=)$  142  
 $\subseteq \succ$  and  $\ell^\# \succ \text{ann}(r)$  for all  $\ell \rightarrow r \in \mathcal{P} \cup \mathcal{P}^=$ . Moreover, let  $\mathcal{P}_\succ \subseteq \mathcal{P} \cup \mathcal{P}^=$  such that 142  
 $\ell^\# \succ \text{ann}(r)$  for all  $\ell \rightarrow r \in \mathcal{P}_\succ$ . Then  $\text{Proc}_{\text{RPP}}(\mathcal{P}, \mathcal{P}^=) = \{(\mathcal{P} \setminus \mathcal{P}_\succ, (\mathcal{P}^= \setminus \mathcal{P}_\succ) \cup 142$   
 $\mathfrak{b}(\mathcal{P}_\succ))\}$  is sound and complete. 142*

*Proof.* Completeness: For every  $(\mathcal{P} \setminus \mathcal{P}_\succ, (\mathcal{P}^= \setminus \mathcal{P}_\succ) \cup \mathfrak{b}(\mathcal{P}_\succ))$ -chain there exists 191  
a  $(\mathcal{P}, \mathcal{P}^=)$ -chain with the same terms and possibly more annotations. Hence, if 191  
 $(\mathcal{P} \setminus \mathcal{P}_\succ, (\mathcal{P}^= \setminus \mathcal{P}_\succ) \cup \mathfrak{b}(\mathcal{P}_\succ))$  is not SN, then  $(\mathcal{P}, \mathcal{P}^=)$  is not SN either. 191

**Soundness:** We start by showing that the conditions of the theorem extend to rewrite 161  
steps instead of just ADPs: 161

- (a) If  $s, t \in \mathcal{T}(\Sigma^\#, \mathcal{V})$  with  $s \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^=} t$ , then  $\text{ann}(s) \succ \text{ann}(t)$ . 162
- (b) If  $s, t \in \mathcal{T}(\Sigma^\#, \mathcal{V})$  with  $s \hookrightarrow_{\mathcal{P}_\succ} t$  using Case (**pr**), then  $\text{ann}(s) \succ \text{ann}(t)$ . 162



For this, we extend  $\text{ann}(t)$  to terms with possibly more than two annotations by  
 defining  $\text{ann}(t) = c_2(r_1^\#, \dots, c_2(r_n^\#, r_n^\#) \dots)$  if  $r_i \leq_\# r$  for  $n \geq 2$  and the positions  
 $\pi_1, \dots, \pi_n$  with  $\pi_i <_{lex} \pi_{i+1}$  for all  $1 \leq i < n$ .

(a) Assume that we have  $s \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^=} t$  using the ADP  $\ell \rightarrow r$ , the VRF  $\varphi$ , the  
 position  $\pi$ , and the substitution  $\sigma$ . So we have  $b(s|_\pi) = \ell\sigma$ . Furthermore, let  
 $\text{ann}(s)$  contain the terms  $s_1^\#, \dots, s_n^\#$  with annotated root symbols. If  $n = 0$ ,  
 then we have  $\text{ann}(s) = c_0 = \text{ann}(t)$  which proves the claim.

Otherwise, we partition  $s_1, \dots, s_n$  into several disjoint groups:

Let  $s_{i_1}, \dots, s_{i_{n_1}}$  be all those  $s_i$  that are at positions above  $\pi$  in  $s$  (i.e., these are  
 those  $s_i$  where  $\pi_i < \pi$ ). Let  $s_{i_{n_1+1}}, \dots, s_{i_{n_2}}$  be all those  $s_i$  that are at positions  
 parallel to  $\pi$  or on or below a variable position of  $\ell$  that is “considered” by  
 the VRF (i.e., those  $s_i$  where  $\pi_i \perp \pi$  or  $\pi.\alpha \leq \pi_i$  for some  $\alpha \in \text{pos}_V(\ell)$  with  
 $\varphi(\alpha) \neq \perp$ ). Finally, let  $s_{i_{n_2+1}}, \dots, s_{i_{n_3}}$  be all those  $s_i$  that are below position  $\pi$ ,  
 but not on or below a variable position of  $\ell$  that is “considered” by the VRF  
 (i.e., those  $s_i$  where  $\pi_i = \pi.\alpha$  for some  $\alpha \in \text{pos}_V(\ell)$  with  $\alpha \neq \varepsilon$  or  $\pi.\alpha \leq \pi_i$  for  
 some  $\alpha \in \text{pos}_V(\ell)$  with  $\varphi(\alpha) = \perp$ ).

If the rewrite step takes place at a position that is not annotated (i.e., the sym-  
 bol at position  $\pi$  in  $s$  is not annotated), then we have  $n_3 = n$  and  $\{i_1, \dots, i_{n_3}\} =$   
 $\{1, \dots, n\}$ . Otherwise, we have  $n_3 = n-1$  and  $\{s_{i_1}, \dots, s_{i_{n_3}}, \ell^\#\sigma\} = \{s_1, \dots, s_n\}$ .  
 After the rewrite step with  $\hookrightarrow_{\mathcal{P} \cup \mathcal{P}^=}$ ,  $\text{ann}(t)$  contains the following terms with  
 annotated root symbols: The terms  $s_{i_{n_1+1}}, \dots, s_{i_{n_2}}$  are unchanged and still con-  
 tained in  $\text{ann}(t)$ . The terms  $s_{i_{n_2+1}}, \dots, s_{i_{n_3}}$  are removed, i.e., they are no longer  
 in  $\text{ann}(t)$ . The terms  $s_{i_1}, \dots, s_{i_{n_1}}$  are replaced by  $s_{i_1}[b(r)\sigma]_{\tau_1}, \dots, s_{i_{n_1}}[b(r)\sigma]_{\tau_{n_1}}$   
 for appropriate positions  $\tau_i \neq \varepsilon$ . Furthermore, if the symbol at position  $\pi$  in  $s$   
 was annotated, then in addition,  $\text{ann}(t)$  contains the terms  $r_1^\#, \dots, r_m^\# \sigma$  where  
 $r_j$  for  $1 \leq j \leq m$  are all terms with  $r_j \leq_\# r$ . Hence, if the symbol at position  $\pi$   
 in  $s$  was annotated, then there exist contexts  $C, C', C''$  containing no function  
 symbol except  $c_2$  such that

$$\begin{aligned}
 \text{ann}(s) &= C[s_{i_1}, \dots, s_{i_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, s_{i_{n_2+1}}, \dots, s_{i_{n_3}}, \ell^\#\sigma] \\
 &\lesssim C'[s_{i_1}, \dots, s_{i_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, s_{i_{n_2+1}}, \dots, s_{i_{n_3}}, r_1^\#, \dots, r_m^\#\sigma] \\
 &\quad \text{by } \ell^\# \lesssim \text{ann}(r), \text{ c-invariance, and c-monotonicity} \\
 &\lesssim C''[s_{i_1}, \dots, s_{i_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, r_1^\#, \dots, r_m^\#\sigma] \\
 &\lesssim C'''[s_{i_1}[b(r)\sigma]_{\tau_1}, \dots, s_{i_{n_1}}[b(r)\sigma]_{\tau_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, r_1^\#, \dots, r_m^\#\sigma] \\
 &\quad \text{by } \ell \lesssim b(r), \text{ c-invariance, and c-monotonicity} \\
 &= \text{ann}(t)
 \end{aligned}$$

If the symbol at position  $\pi$  in  $s$  was not annotated, then we obtain

$$\begin{aligned}
 \text{ann}(s) &= C[s_{i_1}, \dots, s_{i_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, s_{i_{n_2+1}}, \dots, s_{i_{n_3}}] \\
 &\lesssim C'[s_{i_1}, \dots, s_{i_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}] \\
 &\lesssim C''[s_{i_1}[b(r)\sigma]_{\tau_1}, \dots, s_{i_{n_1}}[b(r)\sigma]_{\tau_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}] \\
 &\quad \text{by } \ell \lesssim b(r), \text{ c-invariance, and c-monotonicity} \\
 &= \text{ann}(t)
 \end{aligned}$$

(b) Assume that we have  $s \hookrightarrow_{\mathcal{P}_\succ} t$  using the ADP  $\ell \rightarrow r$ , the VRF  $\varphi$ , the position  $\pi$ ,  
the substitution  $\sigma$ , and we rewrite at an annotated position. So here,  $\text{ann}(s) \neq c_0$ .  
Using the same notations as in (a), we get

$$\begin{aligned}
 \text{ann}(s) &= C[s_{i_1}, \dots, s_{i_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, s_{i_{n_2+1}}, \dots, s_{i_{n_3}}, \ell^\# \sigma] \\
 &\succ C'[s_{i_1}, \dots, s_{i_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, s_{i_{n_2+1}}, \dots, s_{i_{n_3}}, r_1^\# \sigma, \dots, r_m^\# \sigma] \\
 &\quad \text{by } \ell^\# \succ \text{ann}(r), \text{ c-invariance, and c-monotonicity} \\
 &\succ C''[s_{i_1}, \dots, s_{i_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, r_1^\# \sigma, \dots, r_m^\# \sigma] \\
 &\succ C'''[s_{i_1} [b(r) \sigma]_{\tau_1}, \dots, s_{i_{n_1}} [b(r) \sigma]_{\tau_{n_1}}, s_{i_{n_1+1}}, \dots, s_{i_{n_2}}, r_1^\# \sigma, \dots, r_m^\# \sigma] \\
 &\quad \text{by } \ell \succ b(r), \text{ c-invariance, and c-monotonicity} \\
 &= \text{ann}(t)
 \end{aligned}$$

We can now prove soundness. Assume that  $(\mathcal{P}, \mathcal{P}^\#)$  is not SN. Then there exists  
an infinite  $(\mathcal{P}, \mathcal{P}^\#)$ -chain  $t_0 \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^\#} t_1 \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^\#} t_2 \dots$ . If the chain uses an infinite  
number of rewrite steps with rules from  $\mathcal{P}_\succ$  and Case (pr), then  $\text{ann}(t_0) \succ \text{ann}(t_1) \succ \dots$   
 $\text{ann}(t_2) \succ \dots$  would contain an infinite number of steps where the strict relation  $\succ$   
holds, which is a contradiction to well-foundedness of  $\succ$ , as  $\succ$  is compatible with  
 $\succsim$ .

Hence, the chain only contains a finite number of  $\hookrightarrow_{\mathcal{P}_\succ}$ -steps with Case (pr). So  
there is an infinite suffix of the chain where only ADPs from  $(\mathcal{P} \setminus \mathcal{P}_\succ) \cup (\mathcal{P}^\# \setminus \mathcal{P}_\succ)$  are  
used with Case (pr). This means that  $t_i \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^\#} t_{i+1} \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^\#} t_{i+2} \dots$  is an infinite  
 $((\mathcal{P} \setminus \mathcal{P}_\succ) \cup b(\mathcal{P} \cap \mathcal{P}_\succ), (\mathcal{P}^\# \setminus \mathcal{P}_\succ) \cup b(\mathcal{P}^\# \cap \mathcal{P}_\succ))$ -chain, as ADPs that are only used for  
steps with Case (r) do not need annotations. Since the ADPs  $b(\mathcal{P} \cap \mathcal{P}_\succ)$  are never  
used for steps with Case (pr), they can also be moved to the base ADPs. Thus, this  
is also an infinite  $(\mathcal{P} \setminus \mathcal{P}_\succ, (\mathcal{P}^\# \setminus \mathcal{P}_\succ) \cup b(\mathcal{P}_\succ))$ -chain, i.e.,  $(\mathcal{P} \setminus \mathcal{P}_\succ, (\mathcal{P}^\# \setminus \mathcal{P}_\succ) \cup b(\mathcal{P}_\succ))$   
is not SN either.  $\square$

Since ADPs are ordinary rewrite rules with annotations, we can also use ordinary  
reduction orderings (that are closed under contexts) to remove rules from an ADP  
problem completely.

**Theorem 42 (Rule Removal Processor).** *Let  $(\mathcal{P}, \mathcal{P}^\#)$  be an ADP problem, and  
let  $(\succsim, \succ)$  be a reduction pair where  $\succ$  is closed under contexts such that  $b(\mathcal{P} \cup \mathcal{P}^\#) \subseteq \succsim$ .  
Moreover, let  $\mathcal{P}_\succ \subseteq \mathcal{P} \cup \mathcal{P}^\#$  such that  $b(\mathcal{P}_\succ) \subseteq \succ$ . Then  $\text{Proc}_{\text{RR}}(\mathcal{P}, \mathcal{P}^\#) =$   
 $\{(\mathcal{P} \setminus \mathcal{P}_\succ, \mathcal{P}^\# \setminus \mathcal{P}_\succ)\}$  is sound and complete.*

*Proof.* Completeness: Every  $(\mathcal{P} \setminus \mathcal{P}_\succ, \mathcal{P}^\# \setminus \mathcal{P}_\succ)$ -chain is also a  $(\mathcal{P}, \mathcal{P}^\#)$ -chain. Hence,  
if  $(\mathcal{P} \setminus \mathcal{P}_\succ, \mathcal{P}^\# \setminus \mathcal{P}_\succ)$  is not SN, then  $(\mathcal{P}, \mathcal{P}^\#)$  is not SN either.

Soundness: Assume that  $(\mathcal{P}, \mathcal{P}^\#)$  is not SN. Let  $t_0, t_1, \dots$  be an infinite  $(\mathcal{P}, \mathcal{P}^\#)$ -  
chain. Then,  $t_i \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^\#} t_{i+1}$  for all  $i \in \mathbb{N}$ . Since  $b(\mathcal{P} \cup \mathcal{P}^\#) \subseteq \succsim$  and  $\succsim$  is closed under  
contexts and substitutions, we obtain  $b(t_0) \succsim b(t_1) \succsim \dots$ . Assume for a contradiction  
that we use infinitely many steps with  $\hookrightarrow_{\mathcal{P}_\succ}$ . Then,  $b(t_0) \succ b(t_1) \succ \dots$  would contain  
an infinite number of steps where the strict relation  $\succ$  holds, because  $b(\mathcal{P}_\succ) \subseteq \succ$   
and  $\succ$  is also closed under contexts and substitutions. This is a contradiction to  
well-foundedness of  $\succ$ , as  $\succ$  is compatible with  $\succsim$ .

Hence, the chain only contains a finite number of  $\hookrightarrow_{\mathcal{P}_\succ}$ -steps. So there is an  
infinite suffix of the chain where only ADPs from  $(\mathcal{P} \cup \mathcal{P}^\#) \setminus \mathcal{P}_\succ$  are used. This  
means that  $t_i \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^\#} t_{i+1} \hookrightarrow_{\mathcal{P} \cup \mathcal{P}^\#} t_{i+2} \dots$  is an infinite  $(\mathcal{P} \setminus \mathcal{P}_\succ, \mathcal{P}^\# \setminus \mathcal{P}_\succ)$ -chain.  
Hence,  $(\mathcal{P} \setminus \mathcal{P}_\succ, \mathcal{P}^\# \setminus \mathcal{P}_\succ)$  is not SN either.  $\square$