A Complete Dependency Pair Framework for Almost-Sure Innermost Termination of Probabilistic Term Rewriting

Abstract. Recently, the well-known dependency pair (DP) framework was adapted to a *dependency tuple* framework in order to prove almost-sure innermost termination (iAST) of probabilistic term rewrite systems. While this approach was *incomplete*, in this paper, we improve it into a *complete* criterion for iAST by presenting a new, more elegant definition of DPs for probabilistic term rewriting. Based on this, we extend the probabilistic DP framework by new *transformations*. Our implementation in the tool AProVE shows that they increase its power considerably.

1 Introduction

Termination of term rewrite systems (TRSs) is studied for decades and TRSs are used for automated termination analysis of many programming languages. One of the most powerful techniques implemented in essentially all current termination tools for TRSs is the *dependency pair* (DP) framework [3, 14, 15, 20] which allows modular proofs that apply different techniques in different sub-proofs.

In [9], term rewriting was extended to the probabilistic setting. Probabilistic programs describe randomized algorithms and probability distributions, with applications in many areas. In the probabilistic setting, there are several notions of "termination". A program is *almost-surely terminating* (AST) if the probability of termination is 1. A strictly stronger notion is *positive AST* (PAST), which requires that the expected runtime is finite. While numerous techniques exist to prove (P)AST of imperative programs on numbers (e.g., [2, 5, 10, 13, 18, 21–23, 28–31]), there are only few automatic approaches for programs with complex non-tail recursive structure [8, 11, 25]. The approaches that are also suitable for algorithms on recursive data structures [4, 7, 27, 33] are mostly specialized for specific data structures and cannot easily be adjusted to other (possibly user-defined) ones, or are not yet fully automated. In contrast, our goal is a fully automatic termination analysis for (arbitrary) probabilistic TRSs (PTRSs).

Up to now, only two approaches for automatic termination analysis of PTRSs were developed [4, 24]. In [4], orderings based on interpretations were adapted to prove PAST. However, already for non-probabilistic TRSs such a direct application of orderings is limited in power. To obtain a powerful approach, one should combine such orderings in a modular way, as in the DP framework.

Indeed, in [24], the DP framework was adapted to the probabilistic setting in order to prove innermost AST (iAST), i.e., AST for rewrite sequences which follow the innermost evaluation strategy. However, in contrast to the DP framework for ordinary TRSs, the probabilistic *dependency tuple (DT)* framework in [24] is *incomplete*, i.e., there are PTRSs which are iAST but where this cannot be proved with DTs. In this paper, we introduce a new concept of probabilistic DPs and a corresponding new rewrite relation. In this way, we obtain a novel *complete* criterion for iAST via DPs while maintaining soundness for all processors that were developed in the probabilistic DT framework of [24]. Moreover, our improvement allows us to introduce additional more powerful "transformational" probabilistic DP processors which were not possible in the framework of [24].

We briefly recapitulate the DP framework for non-probabilistic TRSs in Sect. 2. Then, we present our novel ADPs (*annotated dependency pairs*) for probabilistic TRSs in Sect. 3. In Sect. 4, we show how to adapt the processors from the framework of [24] to our new probabilistic ADP framework. But in addition, our new framework allows for the definition of new processors which *transform* ADPs. As an example, in Sect. 5 we adapt the *rewriting processor* to the probabilistic setting, which benefits from our new, more precise rewrite relation. The implementation of our approach in the tool AProVE is evaluated in Sect. 6. We refer to [1] for all proofs.

2 The DP Framework

We assume familiarity with term rewriting [6] and recapitulate the DP framework along with its core processors (see e.g., [3, 14, 15, 20] for more details). We regard finite TRSs $\mathcal{R}$ over a finite signature Σ and let $\mathcal{T}(\Sigma, \mathcal{V})$ denote the set of terms over Σ and a set of variables $\mathcal{V}$. We decompose $\Sigma = \mathcal{D} \uplus \mathcal{C}$ such that $f \in \mathcal{D}$ if $f = \operatorname{root}(\ell)$ for some rule $\ell \to r \in \mathcal{R}$. The symbols in $\mathcal{D}$ are called *defined symbols*. For every $f \in \mathcal{D}$, we introduce a fresh *annotated symbol* $f^{\#}$ of the same arity.[1] To ease readability, we often write F instead of $\mathsf{f}^{\#}$. Let $\mathcal{D}^{\#}$ denote the set of all annotated symbols and $\Sigma^{\#} = \mathcal{D}^{\#} \uplus \Sigma$. For any term $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ with $f \in \mathcal{D}$, let $t^{\#} = f^{\#}(t_1, \ldots, t_n)$. For every rule $\ell \to r$ and every subterm t of r with defined root symbol, one obtains a dependency pair (DP) $\ell^{\#} \to t^{\#}$. $\mathcal{DP}(\mathcal{R})$ denotes the set of all dependency pairs of $\mathcal{R}$. As an example, consider $\mathcal{R}_{\mathsf{ex}} = \{(1), (2)\}$ with its dependency pairs $\mathcal{DP}(\mathcal{R}_{\mathsf{ex}}) = \{(3), (4)\}$.

$$\mathsf{f}(\mathsf{s}(x)) \to \mathsf{c}(\mathsf{f}(\mathsf{g}(x))) \quad (1) \qquad \mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(\mathsf{g}(x)) \quad (3)$$

$$\mathsf{g}(x) \to \mathsf{s}(x) \quad (2) \qquad \mathsf{F}(\mathsf{s}(x)) \to \mathsf{G}(x) \quad (4)$$

The DP framework uses *DP problems* $(\mathcal{P}, \mathcal{R})$ where $\mathcal{P}$ is a set of DPs and $\mathcal{R}$ is a TRS. A (possibly infinite) sequence $t_0, t_1, t_2, \ldots$ with $t_i \xrightarrow{\mathsf{i}}_{\mathcal{P},\mathcal{R}} \circ \xrightarrow{\mathsf{i}}^*_{\mathcal{R}} t_{i+1}$ for all i is an (innermost) $(\mathcal{P}, \mathcal{R})$-*chain* which represents subsequent "function calls" in evaluations. Throughout the paper, we restrict ourselves to innermost rewriting, because our adaption of dependency pairs to the probabilistic setting relies on this evaluation strategy. Here, steps with $\xrightarrow{\mathsf{i}}_{\mathcal{P},\mathcal{R}}$ are called P-*steps*, where $\xrightarrow{\mathsf{i}}_{\mathcal{P},\mathcal{R}}$ is the restriction of $\to_{\mathcal{P}}$ to rewrite steps where the used redex is in $\mathtt{NF}_{\mathcal{R}}$ (the set of normal forms w.r.t. $\mathcal{R}$). Steps with $\xrightarrow{\mathsf{i}}^*_{\mathcal{R}}$ are called R-*steps* and are used to evaluate the arguments of an annotated function

[1] The symbols $f^{\#}$ were called *tuple symbols* in the original DP framework [15] and also in [24], as they represent the tuple of arguments of the original defined symbol f.

symbol. So an infinite chain consists of an infinite number of P-steps with a finite number of R-steps between consecutive P-steps. For example, $\mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{s}(x)), \ldots$ is an infinite $(\mathcal{DP}(\mathcal{R}_{\mathsf{ex}}), \mathcal{R}_{\mathsf{ex}})$-chain, as $\mathsf{F}(\mathsf{s}(x)) \xrightarrow{\mathrm{i}}_{\mathcal{DP}(\mathcal{R}_{\mathsf{ex}}),\mathcal{R}_{\mathsf{ex}}} \mathsf{F}(\mathsf{g}(x)) \xrightarrow{\mathrm{i}}{}^{*}_{\mathcal{R}_{\mathsf{ex}}} \mathsf{F}(\mathsf{s}(x))$. A DP problem $(\mathcal{P}, \mathcal{R})$ is called *innermost terminating* (iTerm) if there is no infinite innermost $(\mathcal{P}, \mathcal{R})$-chain. The main result on dependency pairs is the *chain criterion* which states that there is no infinite sequence $t_1 \xrightarrow{\mathrm{i}}_{\mathcal{R}} t_2 \xrightarrow{\mathrm{i}}_{\mathcal{R}} \ldots$, i.e., $\mathcal{R}$ is iTerm, iff $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$ is iTerm. 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 $\mathrm{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}_1, \mathcal{R}_1), \ldots, (\mathcal{P}_n, \mathcal{R}_n)\}$, where $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_n$ are sets of DPs and $\mathcal{R}, \mathcal{R}_1, \ldots, \mathcal{R}_n$ are TRSs. A processor Proc is *sound* if $(\mathcal{P}, \mathcal{R})$ is iTerm whenever $(\mathcal{P}_i, \mathcal{R}_i)$ is iTerm for all $1 \leq i \leq n$. It is *complete* if $(\mathcal{P}_i, \mathcal{R}_i)$ is iTerm for all $1 \leq i \leq n$ whenever $(\mathcal{P}, \mathcal{R})$ is iTerm.

So given 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 repeatedly until all sub-problems are "solved" (i.e., sound processors transform them to the empty set). This gives a modular framework for termination proofs, as different techniques can be used for different "sub-problems" $(\mathcal{P}_i, \mathcal{R}_i)$. The following three theorems recapitulate the three most important processors of the DP framework.

The (innermost) $(\mathcal{P}, \mathcal{R})$-*dependency graph* is a control flow graph that indicates which DPs can be used after each other in a chain. Its node set is $\mathcal{P}$ and there is an edge from $\ell_1^\# \to t_1^\#$ to $\ell_2^\# \to t_2^\#$ if there exist substitutions σ_1, σ_2 such that $t_1^\# \sigma_1 \xrightarrow{\mathrm{i}}{}^{*}_{\mathcal{R}} \ell_2^\# \sigma_2$ and $\ell_1^\# \sigma_1, \ell_2^\# \sigma_2 \in \mathtt{NF}_{\mathcal{R}}$. 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 some strongly connected component (SCC).[2] Hence, it suffices to consider the SCCs of this graph independently.

Theorem 1 (Dep. Graph Processor). *For the SCCs $\mathcal{P}_1, ..., \mathcal{P}_n$ of the $(\mathcal{P}, \mathcal{R})$-dependency graph,* $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}_1, \mathcal{R}), ..., (\mathcal{P}_n, \mathcal{R})\}$ *is sound and complete.*

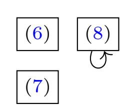
Example 2 (Dependency Graph). Consider the TRS $\mathcal{R}_{\mathsf{ffg}} = \{(5)\}$ with $\mathcal{DP}(\mathcal{R}_{\mathsf{ffg}}) = \{(6), (7), (8)\}$. The $(\mathcal{DP}(\mathcal{R}_{\mathsf{ffg}}), \mathcal{R}_{\mathsf{ffg}})$-dependency graph is on the right.

$$\mathsf{f}(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x))))) \quad (5)$$

$$\mathsf{F}(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{F}(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(x))))) \quad (6)$$

$$\mathsf{F}(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{F}(\mathsf{g}(\mathsf{f}(x))) \quad (7)$$

$$\mathsf{F}(\mathsf{f}(\mathsf{g}(x))) \to \mathsf{F}(x) \quad (8)$$



While the exact dependency graph is not computable in general, there are several techniques to over-approximate it automatically, see, e.g., [3, 15, 20]. In our example, $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{DP}(\mathcal{R}_{\mathsf{ffg}}), \mathcal{R}_{\mathsf{ffg}})$ yields the DP problem $(\{(8)\}, \mathcal{R}_{\mathsf{ffg}})$.

The next processor removes rules that cannot be used for right-hand sides of dependency pairs when their variables are instantiated with normal forms.

[2] Here, a set $\mathcal{P}'$ of DPs is an *SCC* if it is a maximal cycle, i.e., it is a maximal set such that for any $\ell_1^\# \to t_1^\#$ and $\ell_2^\# \to t_2^\#$ in $\mathcal{P}'$ there is a non-empty path from $\ell_1^\# \to t_1^\#$ to $\ell_2^\# \to t_2^\#$ which only traverses nodes from $\mathcal{P}'$.

Theorem 3 (Usable Rules Processor). *Let $\mathcal{R}$ be a TRS. For every $f \in \Sigma^{\#}$ let* $\mathrm{Rules}_{\mathcal{R}}(f) = \{\ell \to r \in \mathcal{R} \mid \mathrm{root}(\ell) = f\}$*. For any $t \in \mathcal{T}(\Sigma^{\#}, \mathcal{V})$, its* usable rules $\mathcal{U}_{\mathcal{R}}(t)$ *are the smallest set such that $\mathcal{U}_{\mathcal{R}}(x) = \varnothing$ for all $x \in \mathcal{V}$ and* $\mathcal{U}_{\mathcal{R}}(f(t_1, \ldots, t_n)) = \mathrm{Rules}_{\mathcal{R}}(f) \cup \bigcup_{i=1}^{n} \mathcal{U}_{\mathcal{R}}(t_i) \cup \bigcup_{\ell \to r \in \mathrm{Rules}_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}}(r)$*. The* usable rules *for the DP problem $(\mathcal{P}, \mathcal{R})$ are $\mathcal{U}(\mathcal{P}, \mathcal{R}) = \bigcup_{\ell^{\#} \to t^{\#} \in \mathcal{P}} \mathcal{U}_{\mathcal{R}}(t^{\#})$. Then* $\mathrm{Proc}_{\mathtt{UR}}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}, \mathcal{U}(\mathcal{P}, \mathcal{R}))\}$ *is sound but not complete.*[3]

$\mathrm{Proc}_{\mathtt{UR}}(\{(8)\}, \mathcal{R}_{\mathsf{ffg}})$ yields the problem $(\{(8)\}, \varnothing)$, i.e., it removes all rules, because the right-hand side of (8) does not contain the defined symbol f.

A *polynomial interpretation* Pol is a Σ-algebra which maps every function symbol $f \in \Sigma$ to a polynomial $f_{\mathrm{Pol}} \in \mathbb{N}[\mathcal{V}]$, see [26]. $\mathrm{Pol}(t)$ denotes the *interpretation* of a term t by the Σ-algebra Pol. An arithmetic inequation like $\mathrm{Pol}(t_1) > \mathrm{Pol}(t_2)$ *holds* if it is true for all instantiations of its variables by natural numbers. The reduction pair processor[4] allows us to use *weakly monotonic* polynomial interpretations that do not have to depend on all of their arguments, i.e., $x \geq y$ implies $f_{\mathrm{Pol}}(\ldots, x, \ldots) \geq f_{\mathrm{Pol}}(\ldots, y, \ldots)$ for all $f \in \Sigma^{\#}$. The processor requires that all rules and DPs are weakly decreasing and it removes those DPs that are strictly decreasing.

Theorem 4 (Reduction Pair Processor with Polynomial Interpretations). *Let $\mathrm{Pol} : \mathcal{T}(\Sigma^{\#}, \mathcal{V}) \to \mathbb{N}[\mathcal{V}]$ be a weakly monotonic polynomial interpretation. Let $\mathcal{P} = \mathcal{P}_{\geq} \uplus \mathcal{P}_{>}$ with $\mathcal{P}_{>} \neq \varnothing$ such that:*

(1) For every $\ell \to r \in \mathcal{R}$, we have $\mathrm{Pol}(\ell) \geq \mathrm{Pol}(r)$.
(2) For every $\ell^{\#} \to t^{\#} \in \mathcal{P}$, we have $\mathrm{Pol}(\ell^{\#}) \geq \mathrm{Pol}(t^{\#})$.
(3) For every $\ell^{\#} \to t^{\#} \in \mathcal{P}_{>}$, we have $\mathrm{Pol}(\ell^{\#}) > \mathrm{Pol}(t^{\#})$.

Then $\mathrm{Proc}_{\mathtt{RP}}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}_{\geq}, \mathcal{R})\}$ is sound and complete.

For $(\{(8)\}, \varnothing)$, the reduction pair processor uses the polynomial interpretation that maps $\mathsf{f}(x)$ to $x + 1$ and both $\mathsf{F}(x)$ and $\mathsf{g}(x)$ to x, i.e., $\mathrm{Proc}_{\mathtt{RP}}(\{(8)\}, \varnothing) = \{(\varnothing, \varnothing)\}$. As $\mathrm{Proc}_{\mathtt{DG}}(\varnothing, \ldots) = \varnothing$ and all processors used are sound, this means that there is no infinite innermost chain for the initial DP problem $(\mathcal{DP}(\mathcal{R}_{\mathsf{ffg}}), \mathcal{R}_{\mathsf{ffg}})$ and thus, $\mathcal{R}_{\mathsf{ffg}}$ is innermost terminating.

3 Probabilistic Annotated Dependency Pairs

In this section we present our novel adaption of DPs to the probabilistic setting. As in [4, 9, 12, 24], the rules of a probabilistic TRS have finite multi-distributions on the right-hand sides. A finite *multi-distribution* μ on a set $A \neq \varnothing$ is a finite multiset of pairs $(p : a)$, where $0 < p \leq 1$ is a probability and $a \in A$, with $\sum_{(p:a) \in \mu} p = 1$. $\mathrm{FDist}(A)$ is the set of all finite multi-distributions on A. For $\mu \in \mathrm{FDist}(A)$, its *support* is the multiset $\mathrm{Supp}(\mu) = \{a \mid (p : a) \in \mu \text{ for some } p\}$.

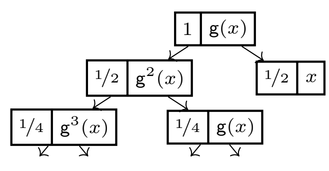
[3] See [14] for a complete version of this processor. It extends DP problems by an additional set to store the left-hand sides of all rules (including the non-usable ones) to determine whether a rewrite step is innermost. We omit this here for readability.

[4] In this paper, we only regard the reduction pair processor with polynomial interpretations, because for most other classical orderings it is not clear how to extend them to probabilistic TRSs, where one has to consider "expected values of terms".

A pair $\ell \to \mu \in \mathcal{T}(\Sigma, \mathcal{V}) \times \mathrm{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ such that $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ for every $r \in \mathrm{Supp}(\mu)$ is a *probabilistic rewrite rule*. A *probabilistic TRS* (PTRS) is a finite set of probabilistic rewrite rules. As an example, consider the PTRS $\mathcal{R}_{\mathsf{rw}}$ with the rule $\mathsf{g}(x) \to \{1/2 : \mathsf{g}(\mathsf{g}(x)),\ 1/2 : x\}$, which corresponds to a symmetric random walk. Let $\mathsf{g}^2(x)$ abbreviate $\mathsf{g}(\mathsf{g}(x))$, etc.

A PTRS $\mathcal{R}$ induces a *rewrite relation* $\to_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathrm{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ where $s \to_{\mathcal{R}} \{p_1 : t_1, \ldots, p_k : t_k\}$ if there is a position π of s, a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{R}$, and a substitution σ such that $s|_\pi = \ell\sigma$ and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$. We call $s \to_{\mathcal{R}} \mu$ an *innermost* rewrite step (denoted $s \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mu$) if $\ell\sigma \in \mathtt{ANF}_{\mathcal{R}}$, where $\mathtt{ANF}_{\mathcal{R}}$ is the set of all *terms in argument normal form w.r.t.* $\mathcal{R}$, i.e., all terms t where every proper subterm of t is in normal form w.r.t. $\mathcal{R}$.

To track all possible rewrite sequences (up to non-determinism) with their probabilities, we *lift* $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ to *(innermost) rewrite sequence trees (RSTs)*. An (innermost) $\mathcal{R}$-RST is a tree whose nodes v are labeled by pairs (p_v, t_v) of a probability p_v and a term t_v such that the edge relation represents a probabilistic innermost rewrite step. More precisely, $\mathfrak{T} = (V, E, L)$ is an (innermost) $\mathcal{R}$-*RST* if (1) (V, E) is a (possibly infinite) directed tree with nodes $V \neq \varnothing$ and directed edges $E \subseteq V \times V$ where $vE = \{w \mid (v, w) \in E\}$ is finite for every $v \in V$, (2) $L : V \to (0, 1] \times \mathcal{T}(\Sigma, \mathcal{V})$ labels every node v by a probability p_v and a term t_v where $p_v = 1$ for the root $v \in V$ of the tree, and (3) for all $v \in V$: if $vE = \{w_1, \ldots, w_k\} \neq \varnothing$, then $t_v \xrightarrow{\mathrm{i}}_{\mathcal{R}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \ldots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$. For any innermost $\mathcal{R}$-RST $\mathfrak{T}$ we define $|\mathfrak{T}|_{\mathtt{Leaf}} = \sum_{v \in \mathtt{Leaf}} p_v$, where $\mathtt{Leaf}$ is the set of $\mathfrak{T}$'s leaves. An RST $\mathfrak{T}$ is *innermost almost-surely terminating (iAST)* if $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$. Similarly, a PTRS $\mathcal{R}$ is *iAST* if all innermost $\mathcal{R}$-RSTs are iAST. While $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$ for every finite RST $\mathfrak{T}$, for infinite RSTs $\mathfrak{T}$ we may have $|\mathfrak{T}|_{\mathtt{Leaf}} < 1$, and even $|\mathfrak{T}|_{\mathtt{Leaf}} = 0$ if $\mathfrak{T}$ has no leaf at all. This notion is equivalent to the notions of AST in [4, 24], where one uses a lifting to multisets instead of trees. For example, the infinite $\mathcal{R}_{\mathsf{rw}}$-RST $\mathfrak{T}$ on the side has $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$. In fact, $\mathcal{R}_{\mathsf{rw}}$ is iAST, because $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$ holds for all innermost $\mathcal{R}_{\mathsf{rw}}$-RSTs $\mathfrak{T}$.



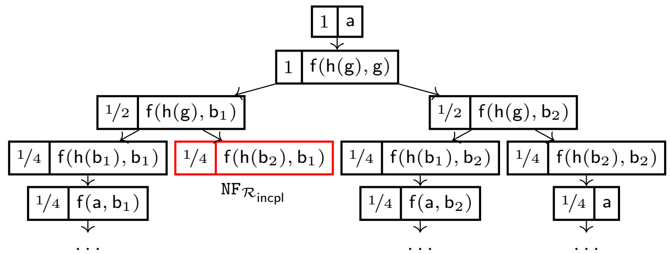
As shown in [24], to adapt the DP framework in order to prove iAST of PTRSs, one has to regard all DPs resulting from the same rule *at once*. Otherwise, one would not be able to distinguish between the DPs of the TRS with the rule $\mathsf{a} \to \{1/2 : \mathsf{b}, 1/2 : \mathsf{c}(\mathsf{a}, \mathsf{a})\}$ which is iAST and the rule $\mathsf{a} \to \{1/2 : \mathsf{b}, 1/2 : \mathsf{c}(\mathsf{a}, \mathsf{a}, \mathsf{a})\}$, which is not iAST. For that reason, in the adaption of the DP framework to PTRSs in [24], one constructs *dependency tuples* (DTs) whose right-hand sides combine the right-hand sides of all dependency pairs resulting from one rule. However, a drawback of this approach is that the resulting chain criterion is not complete, i.e., it allows for chains that do not correspond to any rewrite sequence of the original PTRS $\mathcal{R}$.

Example 5. Consider the PTRS $\mathcal{R}_{\mathsf{incpl}}$ with the rules

$$\mathsf{a} \to \{1 : \mathsf{f}(\mathsf{h}(\mathsf{g}), \mathsf{g})\} \quad (9) \qquad \mathsf{h}(\mathsf{b}_1) \to \{1 : \mathsf{a}\} \quad (11)$$

$$\mathsf{g} \to \{1/2 : \mathsf{b}_1, 1/2 : \mathsf{b}_2\} \quad (10) \qquad \mathsf{f}(x, \mathsf{b}_2) \to \{1 : \mathsf{a}\} \quad (12)$$

and the $\mathcal{R}_{\mathsf{incpl}}$-RST below. So a can be rewritten to the normal form $\mathsf{f}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_1)$



with probability $1/4$ and to the terms $\mathsf{f}(\mathsf{a}, \mathsf{b}_1)$ and a that contain the redex a with a probability of $1/4 + 1/4 = 1/2$. In the term $\mathsf{f}(\mathsf{a}, \mathsf{b}_2)$, one can rewrite the subterm a, and if that ends in a normal form, one can still rewrite the outer f which will yield a again. So to over-approximate the probability of non-termination, one could consider the term $\mathsf{f}(\mathsf{a}, \mathsf{b}_2)$ as if one had two occurrences of a. Then this would correspond to a random walk where the number of a symbols is decreased by 1 with probability $1/4$, increased by 1 with probability $1/4$, and kept the same with probability $1/2$. Such a random walk is AST, and since a similar observation holds for all $\mathcal{R}_{\mathsf{incpl}}$-RSTs, $\mathcal{R}_{\mathsf{incpl}}$ is iAST (we will prove iAST of $\mathcal{R}_{\mathsf{incpl}}$ with our new ADP framework in Sect. 4 and 5).

In contrast, the DT framework from [24] fails on this example. As mentioned, the right-hand sides of DTs combine the right-hand sides of all dependency pairs resulting from one rule. So the right-hand side of the DT for (9) contains the term $\mathsf{com}_4(\mathsf{F}(\mathsf{h}(\mathsf{g}), \mathsf{g}), \mathsf{H}(\mathsf{g}), \mathsf{G}, \mathsf{G})$, where com_4 is a special compound symbol of arity 4. However, here it is no longer clear which occurrence of the annotated symbol G corresponds to which occurrences of g. Therefore, when rewriting an occurrence of G, in the "chains" of [24] one may also rewrite arbitrary occurrences of g simultaneously. (For that reason, in [24] one also couples the DT together with its original rule.) In particular, [24] also allows a simultaneous rewrite step of all underlined symbols in $\mathsf{com}(\mathsf{F}(\mathsf{h}(\mathsf{g}), \underline{\mathsf{g}}), \mathsf{H}(\underline{\mathsf{g}}), \underline{\mathsf{G}}, \mathsf{G})$ even though the underlined G cannot correspond to both underlined gs. As shown in [1], this leads to a chain that is not iAST and that does not correspond to any rewrite sequence. To avoid this problem, one would have to keep track of the connections between annotated symbols and the corresponding original subterms. However, such an improvement would become very complicated in the formalization of [24].

Therefore, in contrast to [24], in our new notion of DPs, we annotate defined symbols directly in the original rewrite rule instead of extracting annotated subterms from its right-hand side. This makes the definition easier, more elegant, and more readable, and allows us to solve the incompleteness problem of [24].

Definition 7 (Annotations). *Let $t \in \mathcal{T}\left(\Sigma^{\#}, \mathcal{V}\right)$ be an* annotated term *and for $\Sigma' \subseteq \Sigma^{\#}$ let $\mathrm{pos}_{\Sigma'}(t)$ be all positions of t with symbols from Σ'. For a set of positions $\Phi \subseteq \mathrm{pos}_{\mathcal{D} \cup \mathcal{D}^{\#}}(t)$, let $\#_{\Phi}(t)$ be the variant of t where the symbols at positions from Φ in t are annotated and all other annotations are removed. Thus, $\mathrm{pos}_{\mathcal{D}^{\#}}(\#_{\Phi}(t)) = \Phi$, and $\#_{\varnothing}(t)$ removes all annotations from t, where we often write $\flat(t)$ instead of $\#_{\varnothing}(t)$. We extend $\flat$ to multi-distributions, rules, and sets of rules by removing the annotations of all occurring terms. We write $\#_{\mathcal{D}}(t)$ instead of $\#_{\mathrm{pos}_{\mathcal{D}}(t)}(t)$ to annotate all defined symbols in t. Moreover, let $\flat^{\uparrow}_{\pi}(t)$ result from removing all annotations from t that are strictly above the position π. Finally, we write $t \trianglelefteq_{\#} s$ if $t \trianglelefteq s$ (i.e., t is a subterm of s) and $\mathrm{root}(t) \in \mathcal{D}^{\#}$.*

Example 8. So if $\mathsf{g} \in \mathcal{D}$, then we have $\#_{\{1\}}(\mathsf{g}(\mathsf{g}(x))) = \#_{\{1\}}(\mathsf{G}(\mathsf{G}(x))) = \mathsf{g}(\mathsf{G}(x))$, $\#_{\mathcal{D}}(\mathsf{g}(\mathsf{g}(x))) = \#_{\{\varepsilon,1\}}(\mathsf{g}(\mathsf{g}(x))) = \mathsf{G}(\mathsf{G}(x))$, and $\flat(\mathsf{G}(\mathsf{G}(x))) = \mathsf{g}(\mathsf{g}(x))$. Moreover, $\flat^{\uparrow}_{1}(\mathsf{G}(\mathsf{G}(x))) = \mathsf{g}(\mathsf{G}(x))$ and $\mathsf{G}(x) \trianglelefteq_{\#} \mathsf{g}(\mathsf{G}(x))$.

Next, we define the *canonical annotated dependency pairs* for a given PTRS.

Definition 9 (Canonical Annotated Dependency Pairs). *For a rule $\ell \to \mu = \{p_1 : r_1, \ldots, p_k : r_k\}$, its canonical* annotated dependency pair *(ADP) is*

$$\mathcal{DP}(\ell \to \mu) \quad = \quad \ell \to \{p_1 : \#_{\mathcal{D}}(r_1), \ldots, p_k : \#_{\mathcal{D}}(r_k)\}^{\mathsf{true}}$$

The canonical ADPs of a PTRS $\mathcal{R}$ are $\mathcal{DP}(\mathcal{R}) = \{\mathcal{DP}(\ell \to \mu) \mid \ell \to \mu \in \mathcal{R}\}$.

Example 10. For $\mathcal{R}_{\mathsf{rw}}$, the canonical ADP for $\mathsf{g}(x) \to \{1/2 : \mathsf{g}(\mathsf{g}(x))), 1/2 : x\}$ is $\mathsf{g}(x) \to \{1/2 : \mathsf{G}(\mathsf{G}(x)), 1/2 : x\}^{\mathsf{true}}$ instead of the (complicated) DT from [24]:

$$\mathcal{DT}(\mathcal{R}_{\mathsf{rw}}) = \{\langle \mathsf{G}(x), \mathsf{g}(x)\rangle \to \{1/2 : \langle \mathsf{com}_2(\mathsf{G}(\mathsf{g}(x)), \mathsf{G}(x)), \mathsf{g}^2(x)\rangle, 1/2 : \langle \mathsf{com}_0, x\rangle\}\}$$

So the left-hand side of an ADP is just the left-hand side of the original rule. The right-hand side of the ADP results from the right-hand side of the original rule by replacing all $f \in \mathcal{D}$ with $f^{\#}$. Moreover, every ADP has a flag $m \in \{\mathsf{true}, \mathsf{false}\}$ to indicate whether this ADP may be used for an R-step before a P-step at a position above. (This flag will later be modified by our usable rules processor.) In general, we work with the following rewrite systems in our framework.

Definition 11 (Annotated Dependency Pairs, $\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}}$). *An* ADP *has the form $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$, where $\ell \in \mathcal{T}(\Sigma, \mathcal{V})$ with $\ell \notin \mathcal{V}$, $m \in \{\mathsf{true}, \mathsf{false}\}$, and for all $1 \leq j \leq k$ we have $r_j \in \mathcal{T}\left(\Sigma^{\#}, \mathcal{V}\right)$ with $\mathcal{V}(r_j) \subseteq \mathcal{V}(\ell)$.*

Let $\mathcal{P}$ be a finite set of ADPs (a so-called ADP problem*). An annotated term $s \in \mathcal{T}\left(\Sigma^{\#}, \mathcal{V}\right)$ rewrites with $\mathcal{P}$ to $\mu = \{p_1 : t_1, \ldots, p_k : t_k\}$ (denoted $s \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \mu$) if there is a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$, a substitution σ, and a $\pi \in \mathrm{pos}_{\mathcal{D}\cup\mathcal{D}^{\#}}(s)$ such that $\flat(s|_\pi) = \ell\sigma \in \mathtt{ANF}_{\mathcal{P}}$, and for all $1 \leq j \leq k$ we have*

$$\begin{array}{lll}
t_j = \phantom{\flat^{\uparrow}_{\pi}(} s[r_j\sigma]_\pi & \text{if } \pi \in \mathrm{pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{true} & (PR)\\
t_j = \flat^{\uparrow}_{\pi}(\ s[r_j\sigma]_\pi) & \text{if } \pi \in \mathrm{pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{false} & (P)\\
t_j = \phantom{\flat^{\uparrow}_{\pi}(} s[\flat(r_j)\sigma]_\pi & \text{if } \pi \notin \mathrm{pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{true} & (R)\\
t_j = \flat^{\uparrow}_{\pi}(\ s[\flat(r_j)\sigma]_\pi) & \text{if } \pi \notin \mathrm{pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{false} & (IRR)
\end{array}$$

To highlight the position π of the redex, we also write $s \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P},\pi} t$. Again, $\mathtt{ANF}_{\mathcal{P}}$ is the set of all annotated terms in argument normal form w.r.t. $\mathcal{P}$.

Rewriting with $\mathcal{P}$ can be seen as ordinary term rewriting while considering and modifying annotations. In the ADP framework, we represent all DPs resulting from a rule as well as the original rule by just one ADP. So for example, the ADP $\mathsf{g}(x) \to \{1/2 : \mathsf{G}(\mathsf{G}(x)), 1/2 : x\}^{\mathsf{true}}$ for the rule $\mathsf{g}(x) \to \{1/2 : \mathsf{g}(\mathsf{g}(x)), 1/2 : x\}$ represents both DPs resulting from the two occurrences of g on the right-hand side, and the rule itself (by simply disregarding all annotations of the ADP).

As in the classical DP framework, our goal is to track specific reduction sequences where (1) the root symbols of the terms are annotated, (2) there are *P-steps* where a DP is applied at the root position, and (3) between two P-steps there can be several *R-steps* where rules are applied below the root.

A step of the form (PR) at position π in Def. 11 can represent either a P- or an R-step. So all annotations are kept during this step except for annotations of the subterms that correspond to variables of the applied rule. These subterms are always in normal form due to the innermost evaluation strategy and we erase their annotations in order to handle rewriting with non-left-linear rules correctly. If one later considers an annotated symbol at a position above π, then this (PR)-step has played the role of an R-step, and otherwise it has played the role of a P-step. As an example, for a PTRS $\mathcal{R}_{\mathsf{ex2}}$ with the rules $\mathsf{g}(x,x) \to \{1 : \mathsf{f}(x)\}$ and $\mathsf{f}(\mathsf{a}) \to \{1 : \mathsf{f}(\mathsf{b})\}$, we have the canonical ADPs $\mathsf{g}(x,x) \to \{1 : \mathsf{F}(x)\}^{\mathsf{true}}$ and $\mathsf{f}(\mathsf{a}) \to \{1 : \mathsf{F}(\mathsf{b})\}^{\mathsf{true}}$, and we can rewrite $\mathsf{G}(\mathsf{F}(\mathsf{b}),\mathsf{f}(\mathsf{b})) \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{R}_{\mathsf{ex2}})} \{1 : \mathsf{F}(\mathsf{f}(\mathsf{b}))\}$ using the first ADP. Here, we have $\pi = \varepsilon$, $\flat(s|_{\varepsilon}) = \mathsf{g}(\mathsf{f}(\mathsf{b}),\mathsf{f}(\mathsf{b})) = \ell\sigma$ where σ instantiates x with the normal form $\mathsf{f}(\mathsf{b})$, and $r_1 = \mathsf{F}(x)$.

A step of the form (R) rewrites at the position of a non-annotated defined symbol. So this represents an R-step and thus, we remove all annotations from the right-hand side r_j. As an example, we have $\mathsf{G}(\mathsf{F}(\mathsf{b}),\mathsf{f}(\mathsf{a})) \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{R}_{\mathsf{ex2}})} \{1 : \mathsf{G}(\mathsf{F}(\mathsf{b}),\mathsf{f}(\mathsf{b}))\}$ using the ADP $\mathsf{f}(\mathsf{a}) \to \{1 : \mathsf{F}(\mathsf{b})\}^{\mathsf{true}}$.

A step of the form (P) represents a P-step. Thus, we remove all annotations above the position π, because no P-steps are possible above π. So if $\mathcal{P}$ contains $\mathsf{f}(\mathsf{a}) \to \{1 : \mathsf{F}(\mathsf{b})\}^{\mathsf{false}}$, then $\mathsf{G}(\mathsf{F}(\mathsf{b}),\mathsf{F}(\mathsf{a})) \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{1 : \mathsf{g}(\mathsf{F}(\mathsf{b}),\mathsf{F}(\mathsf{b}))\}$.

Finally, a step of the form (IRR) is an R-step that is irrelevant for proving iAST, because due to the flag $m = \mathsf{false}$, afterwards there cannot be a P-step at a position above. For example, if $\mathcal{P}$ again contains $\mathsf{f}(\mathsf{a}) \to \{1 : \mathsf{F}(\mathsf{b})\}^{\mathsf{false}}$, then we have $\mathsf{G}(\mathsf{F}(\mathsf{b}),\mathsf{f}(\mathsf{a})) \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{1 : \mathsf{g}(\mathsf{F}(\mathsf{b}),\mathsf{f}(\mathsf{b}))\}$. Steps of the form (IRR) are needed to ensure that all rewrite steps with $\mathcal{R}$ are also possible with the ADP problems $\mathcal{P}$ that result from $\mathcal{DP}(\mathcal{R})$ when applying ADP processors. These processors only modify the annotations, but keep the rest of the rules unchanged. So for all these ADP problems $\mathcal{P}$, we have $\mathcal{R} = \flat(\mathcal{P})$ and $\flat(t) \in \mathtt{ANF}_{\mathcal{R}}$ iff $t \in \mathtt{ANF}_{\mathcal{P}}$ for all $t \in \mathcal{T}\left(\Sigma^{\#},\mathcal{V}\right)$, i.e., the innermost evaluation strategy is not affected by the application of ADP processors. This is different from the classical DP framework, where the usable rules processor reduces the number of rules. This may result in new redexes that are allowed for innermost rewriting. Thus, the usable rules processor in our new ADP framework is *complete*, whereas in [14], one has to extend DP problems by an additional component in order to achieve completeness of this processor (see Footnote 3).

Now, $s \overset{\mathrm{i}}{\to}_{\mathcal{R}} \{p_1 : t_1, \ldots, p_k : t_k\}$ essentially[5] implies $\#_{\mathcal{D}}(s) \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{R})} \{p_1 : \#_{\mathcal{D}}(t_1), \ldots, p_k : \#_{\mathcal{D}}(t_k)\}$, and we got rid of any ambiguities in the rewrite relation, that led to incompleteness in [24]. While our ADPs are much simpler than the DTs of [24], due to their annotations they still contain all information that is needed to define the required DP processors.

Instead of chains of DPs, in the probabilistic setting one works with *chain trees* [24], where P- and R-steps are indicated by P- and R-nodes in the tree.

[5] We have $\#_{\mathcal{D}}(s) \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{R})} \{p_1 : t'_1, \ldots, p_k : t'_k\}$ where t'_j and $\#_{\mathcal{D}}(t_j)$ are the same up to some annotations of subterms that are $\mathcal{DP}(\mathcal{R})$-normal forms. The reason is that as mentioned above, annotations of the subterms (in normal form) that correspond to variables of the rule are erased. So for example, rewriting $\mathsf{G}(\mathsf{F}(\mathsf{b}),\mathsf{F}(\mathsf{b}))$ with $\mathcal{DP}(\mathcal{R}_{\mathsf{ex2}})$ yields $\{1 : \mathsf{F}(\mathsf{f}(\mathsf{b}))\}$ and not $\{1 : \mathsf{F}(\mathsf{F}(\mathsf{b}))\}$.

Chain trees are defined analogously to RSTs, but the crucial requirement is that every infinite path of the tree must contain infinitely many steps of the forms (PR) or (P). Thus, in our setting $\mathfrak{T} = (V, E, L, P)$ is a $\mathcal{P}$*-chain tree (CT)* if

1. (V, E) is a (possibly infinite) directed tree with nodes $V \neq \varnothing$ and directed edges $E \subseteq V \times V$ where $vE = \{w \mid (v, w) \in E\}$ is finite for every $v \in V$.
2. $L : V \to (0, 1] \times \mathcal{T}\left(\Sigma^{\#}, \mathcal{V}\right)$ labels every node v by a probability p_v and a term t_v. For the root $v \in V$ of the tree, we have $p_v = 1$.
3. $P \subseteq V \setminus \texttt{Leaf}$ (where $\texttt{Leaf}$ are all leaves) is a subset of the inner nodes to indicate whether we use (PR) or (P) for the next rewrite step. $R = V \setminus (\texttt{Leaf} \cup P)$ are all inner nodes that are not in P, i.e., where we rewrite using (R) or (IRR).
4. For all $v \in P$: if $vE = \{w_1, \ldots, w_k\}$, then $t_v \stackrel{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \ldots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$ using Case (PR) or (P).
5. For all $v \in R$: if $vE = \{w_1, \ldots, w_k\}$, then $t_v \stackrel{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \ldots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$ using Case (R) or (IRR).
6. Every infinite path in $\mathfrak{T}$ contains infinitely many nodes from P.

Let $|\mathfrak{T}|_{\texttt{Leaf}} = \sum_{v \in \texttt{Leaf}} p_v$. We define that $\mathcal{P}$ is iAST if $|\mathfrak{T}|_{\texttt{Leaf}} = 1$ for all $\mathcal{P}$-CTs $\mathfrak{T}$. So Conditions 1–5 ensure that the chain tree corresponds to an RST and Condition 6 requires that one may only use finitely many R-steps before the next P-step. This yields a chain criterion as in the non-probabilistic setting, where (in contrast to the chain criterion of [24]) we again have "iff" instead of "if".

Theorem 12 (Chain Criterion). *$\mathcal{R}$ is iAST iff $\mathcal{DP}(\mathcal{R})$ is iAST.*

Since ADPs only add annotations to already existing rules, our chain criterion is complete ("only if"), because every $\mathcal{DP}(\mathcal{R})$-CT can be turned into an $\mathcal{R}$-RST by omitting all annotations. To prove soundness ("if"), one has to show that every $\mathcal{R}$-RST $\mathfrak{T}$ can be simulated by a $\mathcal{DP}(\mathcal{R})$-CT. As mentioned, all proofs can be found in [1].

4 The ADP Framework

The new (probabilistic) ADP framework again uses a *divide-and-conquer* approach which applies *ADP processors* to transform an ADP problem into simpler sub-problems. An *ADP processor* Proc has the form $\mathrm{Proc}(\mathcal{P}) = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, where $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_n$ are ADP problems. Proc is *sound* if $\mathcal{P}$ is iAST whenever $\mathcal{P}_i$ is iAST for all $1 \leq i \leq n$. It is *complete* if $\mathcal{P}_i$ is iAST for all $1 \leq i \leq n$ whenever $\mathcal{P}$ is iAST. Given a PTRS $\mathcal{R}$, one starts with the canonical ADP problem $\mathcal{DP}(\mathcal{R})$ and applies sound (and preferably complete) ADP processors repeatedly until all ADPs in all sub-problems contain no annotations anymore. Such an ADP problem is trivially iAST. The framework again allows for modular termination proofs, since different techniques can be applied on each sub-problem $\mathcal{P}_i$.

We now adapt the processors from [24] to our new framework. The (innermost) $\mathcal{P}$*-dependency graph* is a control flow graph between ADPs from $\mathcal{P}$, indicating whether an ADP α may lead to an application of another ADP α' on an annotated subterm introduced by α. This possibility is not related to the probabilities.

Hence, we can use the *non-probabilistic variant* $\mathrm{np}(\mathcal{P}) = \{\ell \to \flat(r_j) \mid \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{\mathsf{true}} \in \mathcal{P}, 1 \leq j \leq k\}$, which is an ordinary TRS over the signature Σ. Note that for $\mathrm{np}(\mathcal{P})$ we only need to consider rules with the flag true, since only such rules can be used before a P-step at a position above.

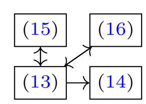
Definition 13 (Dep. Graph). *The* $\mathcal{P}$-dependency graph *has the nodes* $\mathcal{P}$ *and there is an edge from* $\ell_1 \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$ *to* $\ell_2 \to \ldots$ *if there are substitutions* σ_1, σ_2 *and a* $t \trianglelefteq_{\#} r_j$ *for some* $1 \leq j \leq k$ *such that* $\#_{\{\varepsilon\}}(t)\sigma_1 \xrightarrow{\mathrm{i}}{}^{*}_{\mathrm{np}(\mathcal{P})} \#_{\{\varepsilon\}}(\ell_2)\sigma_2$ *and both* $\ell_1\sigma_1$ *and* $\ell_2\sigma_2$ *are in* $\mathtt{ANF}_{\mathcal{P}}$*.*

So there is an edge from an ADP α to an ADP α' if after a step of the form (PR) or (P) with α at the root of the term there may eventually come another step of the form (PR) or (P) with α'. Hence, for every path in a $\mathcal{P}$-CT from a P-node where an annotated subterm $f^{\#}(\ldots)$ is introduced to the next P-node where the subterm $f^{\#}(\ldots)$ at this position is rewritten, there is a corresponding edge in the $\mathcal{P}$-dependency graph. Since every infinite path in a CT contains infinitely many nodes from P, every such path traverses a cycle of the dependency graph infinitely often. Thus, it suffices to consider the SCCs of the dependency graph separately. In our framework, this means that we remove the annotations from all rules except those that are in the SCC that we want to analyze. As in [24], to automate the following two processors, the same over-approximation techniques as for the non-probabilistic dependency graph can be used.

Theorem 14 (Prob. Dep. Graph Processor). *For the SCCs* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *of the* $\mathcal{P}$*-dependency graph,* $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{P}) = \{\mathcal{P}_1 \cup \flat(\mathcal{P} \setminus \mathcal{P}_1), \ldots, \mathcal{P}_n \cup \flat(\mathcal{P} \setminus \mathcal{P}_n)\}$ *is sound and complete.*

Example 15. Consider the PTRS $\mathcal{R}_{\mathsf{incpl}}$ from Ex. 5 with the canonical ADPs

$$\mathsf{a} \to \{1 : \mathsf{F}(\mathsf{H}(\mathsf{G}), \mathsf{G})\}^{\mathsf{true}} \quad (13) \qquad \mathsf{h}(\mathsf{b}_1) \to \{1 : \mathsf{A}\}^{\mathsf{true}} \quad (15)$$
$$\mathsf{g} \to \{1/2 : \mathsf{b}_1, 1/2 : \mathsf{b}_2\}^{\mathsf{true}} \quad (14) \qquad \mathsf{f}(x, \mathsf{b}_2) \to \{1 : \mathsf{A}\}^{\mathsf{true}} \quad (16)$$



The $\mathcal{DP}(\mathcal{R}_{\mathsf{incpl}})$-dependency graph can be seen on the right. As (14) is the only ADP not contained in the SCC, we can remove all of its annotations. However, since (14) already does not contain any annotation, here the dependency graph processor does not change $\mathcal{DP}(\mathcal{R}_{\mathsf{incpl}})$.

To remove the annotations of *non-usable* terms like G in (13) that lead out of the SCCs of the dependency graph, one can apply the *usable terms processor*.

Theorem 16 (Usable Terms Processor). *Let* $\ell_1 \in \mathcal{T}(\Sigma, \mathcal{V})$ *and* $\mathcal{P}$ *be an ADP problem. We call* $t \in \mathcal{T}(\Sigma^{\#}, \mathcal{V})$ *with* $\mathrm{root}(t) \in \mathcal{D}^{\#}$ usable *w.r.t.* ℓ_1 *and* $\mathcal{P}$ *if there are substitutions* σ_1, σ_2 *and an* $\ell_2 \to \mu_2 \in \mathcal{P}$ *where* μ_2 *contains an annotated symbol, such that* $\#_{\varepsilon}(t)\sigma_1 \xrightarrow{\mathrm{i}}{}^{*}_{\mathrm{np}(\mathcal{P})} \#_{\varepsilon}(\ell_2)\sigma_2$ *and both* $\ell_1\sigma_1$ *and* $\ell_2\sigma_2$ *are in* $\mathtt{ANF}_{\mathcal{P}}$*. Let* $\flat_{\ell,\mathcal{P}}(s)$ *be the variant of* s *where all annotations of those subterms of* s *are removed that are not usable w.r.t.* ℓ *and* $\mathcal{P}$*. The transformation that removes all annotations from non-usable terms in the right-hand sides of ADPs is* $\mathcal{T}_{\mathtt{UT}}(\mathcal{P}) = \{\ell \to \{p_1 : \flat_{\ell,\mathcal{P}}(r_1), \ldots, p_k : \flat_{\ell,\mathcal{P}}(r_k)\}^m \mid \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}\}$*. Then* $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{P}) = \{\mathcal{T}_{\mathtt{UT}}(\mathcal{P})\}$ *is sound and complete.*

So for $\mathcal{DP}(\mathcal{R}_{\mathsf{incpl}})$, $\mathrm{Proc}_{\mathtt{UT}}$ replaces (13) by $\mathsf{a} \to \{1 : \mathsf{F}(\mathsf{H}(\mathsf{g}), \mathsf{g})\}^{\mathsf{true}}$ (13′).

Again, the idea of the *usable rules processor* is to find rules that cannot be used below an annotation in right-hand sides of ADPs when their variables are instantiated with normal forms.

Theorem 17 (Probabilistic Usable Rules Processor). *Let $\mathcal{P}$ be an ADP problem. For every $f \in \Sigma^{\#}$ let $\mathrm{Rules}_{\mathcal{P}}(f) = \{\ell \to \mu^m \in \mathcal{P} \mid \mathrm{root}(\ell) = f\}$. For any term $t \in \mathcal{T}\left(\Sigma^{\#}, \mathcal{V}\right)$, its* usable rules *$\mathcal{U}_{\mathcal{P}}(t)$ are the smallest set such that $\mathcal{U}_{\mathcal{P}}(x) = \varnothing$ for all $x \in \mathcal{V}$ and $\mathcal{U}_{\mathcal{P}}(f(t_1, \ldots, t_n)) = \mathrm{Rules}_{\mathcal{P}}(f) \cup \bigcup_{i=1}^{n} \mathcal{U}_{\mathcal{P}}(t_i) \cup \bigcup_{\ell \to \mu^m \in \mathrm{Rules}_{\mathcal{P}}(f), r \in \mathrm{Supp}(\mu)} \mathcal{U}_{\mathcal{P}}(\flat(r))$, otherwise. The* usable rules *for $\mathcal{P}$ are $\mathcal{U}(\mathcal{P}) = \bigcup_{\ell \to \mu^m \in \mathcal{P}, r \in \mathrm{Supp}(\mu), t \trianglelefteq_{\#} r} \mathcal{U}_{\mathcal{P}}(\#_{\{\varepsilon\}}(t))$. Then $\mathrm{Proc}_{\mathtt{UR}}(\mathcal{P}) = \{\mathcal{U}(\mathcal{P}) \cup \{\ell \to \mu^{\mathsf{false}} \mid \ell \to \mu^m \in \mathcal{P} \setminus \mathcal{U}(\mathcal{P})\}\}$ is sound and complete, i.e., we turn the flag of all non-usable rules to* false.

Example 18. For our ADP problem $\{(13'), (14), (15), (16)\}$, (16) is not usable because neither f nor F occur below annotated symbols on right-hand sides. Hence, $\mathrm{Proc}_{\mathtt{UR}}$ replaces (16) by $\mathsf{f}(x, \mathsf{b}_2) \to \{1 : \mathsf{A}\}^{\mathsf{false}}$ (16′). As discussed after Def. 11, in contrast to the processor of Thm. 3, our usable rules processor is complete since we do not remove non-usable rules but only set their flag to false.

Finally, we adapt the reduction pair processor. Here, (1) for every rule with the flag true (which can therefore be used for R-steps), the expected value must be weakly decreasing when removing the annotations. Since rules can also be used for P-steps, (2) we also require a weak decrease when comparing the annotated left-hand side with the expected value of all annotated subterms in the right-hand side. Since we sum up the values of the annotated subterms of each right-hand side, we can again use *weakly monotonic* interpretations. As in [4, 24], to ensure "monotonicity" w.r.t. expected values we have to restrict ourselves to interpretations with multilinear polynomials, where all monomials have the form $c \cdot x_1^{e_1} \cdot \ldots \cdot x_n^{e_n}$ with $c \in \mathbb{N}$ and $e_1, \ldots, e_n \in \{0, 1\}$. The processor then removes the annotations from those ADPs where (3) in addition there is at least one right-hand side r_j with an annotated subterm t that is strictly decreasing.[6]

Theorem 19 (Probabilistic Reduction Pair Processor). *Let $\mathrm{Pol} : \mathcal{T}(\Sigma^{\#}, \mathcal{V}) \to \mathbb{N}[\mathcal{V}]$ be a weakly monotonic, multilinear polynomial interpretation. Let $\mathcal{P} = \mathcal{P}_{\geq} \uplus \mathcal{P}_{>}$ such that:*

(1) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{\mathsf{true}} \in \mathcal{P}$, we have $\mathrm{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(\flat(r_j))$.
(2) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{m} \in \mathcal{P}$, we have $\mathrm{Pol}(\#_{\{\varepsilon\}}(\ell)) \geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_{\#} r_j} \mathrm{Pol}(\#_{\{\varepsilon\}}(t))$.

[6] In addition, the corresponding non-annotated term right-hand side $\flat(r_j)$ must be at least weakly decreasing. The reason is that in contrast to the original DP framework, we now may have nested annotated symbols and thus, we have to ensure that they behave "monotonically". So we have to ensure that $\mathrm{Pol}(A) > \mathrm{Pol}(B)$ also implies that the measure of $F(A)$ is greater than $F(B)$. Every term r is "measured" as $\sum_{t \trianglelefteq_{\#} r} \mathrm{Pol}(\#_{\{\varepsilon\}}(t))$, i.e., $F(A)$ is measured as $\mathrm{Pol}(F(a)) + \mathrm{Pol}(A)$. Hence, in this example we must ensure that $\mathrm{Pol}(A) > \mathrm{Pol}(B)$ implies $\mathrm{Pol}(F(a)) + \mathrm{Pol}(A) > \mathrm{Pol}(F(b)) + \mathrm{Pol}(B)$. For that reason, we also have to require $\mathrm{Pol}(a) \geq \mathrm{Pol}(b)$.

(3) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}_>$*, there exists a* $1 \leq j \leq k$ *with* $\mathrm{Pol}(\#_{\{\varepsilon\}}(\ell)) > \sum_{t \trianglelefteq_\# r_j} \mathrm{Pol}(\#_{\{\varepsilon\}}(t))$.
If $m = \mathsf{true}$*, then we additionally have* $\mathrm{Pol}(\ell) \geq \mathrm{Pol}(\flat(r_j))$.

Then $\mathrm{Proc}_{\mathsf{RP}}(\mathcal{P}) = \{\mathcal{P}_\geq \cup \flat(\mathcal{P}_>)\}$ *is sound and complete.*

Example 20. In Sect. 5, we will present a new *rewriting processor* and show how the ADP (13′) can be transformed into

$$\mathsf{a} \to \{{}^1\!/\!{}_4 : \mathsf{f}(\mathsf{H}(\mathsf{b}_1), \mathsf{b}_1), {}^1\!/\!{}_4 : \mathsf{f}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_1), {}^1\!/\!{}_4 : \mathsf{F}(\mathsf{H}(\mathsf{b}_1), \mathsf{b}_2), {}^1\!/\!{}_4 : \mathsf{F}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_2)\}^{\mathsf{true}} \quad (13'')$$

For the resulting ADP problem $\{(13''), (14), (15), (16')\}$ with

$$\mathsf{g} \to \{{}^1\!/\!{}_2 : \mathsf{b}_1, {}^1\!/\!{}_2 : \mathsf{b}_2\}^{\mathsf{true}} \ (14) \quad \mathsf{h}(\mathsf{b}_1) \to \{1 : \mathsf{A}\}^{\mathsf{true}} \ (15) \quad \mathsf{f}(x, \mathsf{b}_2) \to \{1 : \mathsf{A}\}^{\mathsf{false}} \ (16')$$

we use the reduction pair processor with the polynomial interpretation that maps A, F, and H to 1 and all other symbols to 0, to remove all annotations from the a-ADP (because it contains the right-hand side $\mathsf{f}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_1)$ without annotations and thus, $\mathrm{Pol}(\mathsf{A}) = 1 > \sum_{t \trianglelefteq_\# \mathsf{f}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_1)} \mathrm{Pol}(\#_{\{\varepsilon\}}(t)) = 0$). Another application of the usable terms processor removes the remaining A-annotations. Since there are no more annotations left, this proves iAST of $\mathcal{R}_{\mathsf{incpl}}$.

Finally, in proofs with the ADP framework, one may obtain ADP problems $\mathcal{P}$ that have a non-probabilistic structure, i.e., every ADP has the form $\ell \to \{1 : r\}^m$. The *probability removal processor* then allows us to switch to ordinary DPs.

Theorem 21 (Probability Removal Processor). *Let* $\mathcal{P}$ *be an ADP problem where every ADP in* $\mathcal{P}$ *has the form* $\ell \to \{1 : r\}^m$*. Let* $\mathrm{dp}(\mathcal{P}) = \{\#_{\{\varepsilon\}}(\ell) \to \#_{\{\varepsilon\}}(t) \mid \ell \to \{1 : r\}^m \in \mathcal{P}, t \trianglelefteq_\# r\}$*. Then* $\mathcal{P}$ *is iAST iff the non-probabilistic DP problem* $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ *is iTerm. So if* $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ *is iTerm, then the processor* $\mathrm{Proc}_{\mathsf{PR}}(\mathcal{P}) = \varnothing$ *is sound and complete.*

5 Transforming ADPs

Compared to the DT framework for PTRSs in [24], our new ADP framework is not only easier, more elegant, and yields a complete chain criterion, but it also has important practical advantages, because every processor that performs a rewrite step benefits from our novel definition of rewriting with ADPs (whereas the rewrite relation with DTs in [24] was an "incomplete over-approximation" of the rewrite relation of the original TRS). To illustrate this, we adapt the *rewriting* processor from the original DP framework [15] to the probabilistic setting, which allows us to prove iAST of $\mathcal{R}_{\mathsf{incpl}}$ from Ex. 5. (Such transformational processors had not been adapted in the DT framework of [24].) One could also adapt the rewriting processor to the probabilistic setting of [24], but then it would be substantially weaker, and we would fail in proving iAST of $\mathcal{R}_{\mathsf{incpl}}$. We refer to [1] for our adaption of the remaining transformational processors from [15] (based on *instantiation*, *forward instantiation*, and *narrowing*) to the probabilistic setting.

In the non-probabilistic setting, the rewriting processor may rewrite a redex in the right-hand side of a DP if this does not affect the construction of chains. To ensure that, the usable rules for this redex must be non-overlapping (NO). If the DP occurs in a chain, then this redex is weakly innermost terminating, hence by NO also terminating and confluent, and thus, it has a unique normal form [19].

For the probabilistic rewriting processor, to ensure that the probabilities for the normal forms stay the same, in addition to NO we require that the rule used for the rewrite step is linear (L) (i.e., every variable occurs at most once in the left-hand side and in each term of the multi-distribution μ on the right-hand side) and non-erasing (NE) (i.e., each variable of the left-hand side occurs in each term of $\mathrm{Supp}(\mu)$).

Definition 22 (Rewriting Processor). *Let $\mathcal{P}$ be an ADP problem with $\mathcal{P} = \mathcal{P}' \uplus \{\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m\}$. Let $\tau \in \mathrm{pos}_{\mathcal{D}}(r_j)$ for some $1 \leq j \leq k$ such that $r_j|_\tau \in \mathcal{T}(\Sigma, \mathcal{V})$, i.e., there exists no annotation below or at the position τ. If $r_j \hookrightarrow_{\mathcal{P},\tau} \{q_1 : e_1, \ldots, q_h : e_h\}$, where $\hookrightarrow_{\mathcal{P},\tau}$ is defined like $\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P},\tau}$ but the used redex $r_j|_\tau$ does not have to be in $\mathtt{ANF}_{\mathcal{P}}$, then we define*

$$\mathrm{Proc}_{\mathtt{r}}(\mathcal{P}) = \left\{ \begin{array}{l} \mathcal{P}' \cup \{\ell \to \{p_1 : \flat(r_1), \ldots, p_k : \flat(r_k)\}^m\} \\ \quad \cup \left\{ \begin{array}{l} \ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \setminus \{p_j : r_j\} \\ \qquad \cup \{p_j \cdot q_1 : e_1, \ldots, p_j \cdot q_h : e_h\}^m \end{array} \right\} \end{array} \right\}$$

In the non-probabilistic DP framework, one only transforms the DPs by rewriting, but the rules are left unchanged. But since our ADPs represent both DPs and rules, when rewriting an ADP, we add a copy of the original ADP without any annotations (i.e., this corresponds to the original rule which can now only be used for "R-steps"). Another change to the non-probabilistic rewriting processor is the requirement that there exists no annotation below τ. Otherwise, rewriting would potentially remove annotations from r_j. For the soundness of the processor, we have to ensure that this cannot happen.

Theorem 23 (Soundness[7] of the Rewriting Processor). $\mathrm{Proc}_{\mathtt{r}}$ *as in Def. 22 is* sound *if one of the following cases holds:*

1. *$\mathcal{U}_{\mathcal{P}}(r_j|_\tau)$ is NO, and the rule used for rewriting $r_j|_\tau$ is L and NE.*
2. *$\mathcal{U}_{\mathcal{P}}(r_j|_\tau)$ is NO, and all its rules have the form $\ell' \to \{1 : r'\}^{m'}$.*
3. *$\mathcal{U}_{\mathcal{P}}(r_j|_\tau)$ is NO, $r_j|_\tau$ is a ground term, and $r_j \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P},\tau} \{q_1 : e_1, \ldots, q_h : e_h\}$ is an innermost step.*

We refer to [1] for a discussion on the requirements L and NE in the first case. The second case corresponds to the original rewrite processor where all usable rules of $r_j|_\tau$ are non-probabilistic. In the last case, for any instantiation only a single innermost rewrite step is possible for $r_j|_\tau$. The restriction to innermost rewrite steps is only useful if $r_j|_\tau$ is ground. Otherwise, an innermost step on $r_j|_\tau$ might become a non-innermost step when instantiating $r_j|_\tau$'s variables.

The rewriting processor benefits from our ADP framework, because it applies the rewrite relation $\hookrightarrow_{\mathcal{P}}$. In contrast, a rewriting processor in the DT framework of [24] may have to replace a DT by *multiple* new DTs, due to the ambiguities in

[7] For completeness in the non-probabilistic setting [15], one uses a different definition of "non-terminating" (or "infinite") DP problems. In future work, we will examine whether a similar definition would also yield completeness in the probabilistic case.

their rewrite relation. Such a rewriting processor would fail for $\mathcal{R}_{\mathsf{incpl}}$ whereas with the processor of Thm. 23 we can now prove that $\mathcal{R}_{\mathsf{incpl}}$ is iAST.

Example 24. After applying the usable terms and the usable rules processor to $\mathcal{DP}(\mathcal{R}_{\mathsf{incpl}})$, we obtained:

$$\mathsf{a} \to \{1 : \mathsf{F}(\mathsf{H}(\mathsf{g}), \mathsf{g})\}^{\mathsf{true}} \quad (13') \qquad \mathsf{h}(\mathsf{b}_1) \to \{1 : \mathsf{A}\}^{\mathsf{true}} \quad (15)$$

$$\mathsf{g} \to \{1/2 : \mathsf{b}_1, 1/2 : \mathsf{b}_2\}^{\mathsf{true}} \quad (14) \qquad \mathsf{f}(x, \mathsf{b}_2) \to \{1 : \mathsf{A}\}^{\mathsf{false}} \quad (16)$$

Now we can apply the rewriting processor on (13′) repeatedly until all gs are rewritten and replace it by the ADP $\mathsf{a} \to \{1/4 : \mathsf{F}(\mathsf{H}(\mathsf{b}_1), \mathsf{b}_1), 1/4 : \mathsf{F}(\mathsf{H}(\mathsf{b}_2), \mathsf{b}_1), 1/4 : \mathsf{F}(\mathsf{H}(\mathsf{b}_1), \mathsf{b}_2), 1/4 : \mathsf{F}(\mathsf{H}(\mathsf{b}_2), \mathsf{b}_2)\}^{\mathsf{true}}$ as well as several resulting ADPs $\mathsf{a} \to \ldots$ without annotations. Now the annotations in the terms $\mathsf{F}(\ldots, \mathsf{b}_1)$ and $\mathsf{H}(\mathsf{b}_2)$ are removed by the usable terms processor, as they cannot rewrite to instances of left-hand sides of ADPs. So the a-ADP is changed to $\mathsf{a} \to \{1/4 : \mathsf{f}(\mathsf{H}(\mathsf{b}_1), \mathsf{b}_1), 1/4 : \mathsf{f}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_1), 1/4 : \mathsf{F}(\mathsf{H}(\mathsf{b}_1), \mathsf{b}_2), 1/4 : \mathsf{F}(\mathsf{h}(\mathsf{b}_2), \mathsf{b}_2)\}^{\mathsf{true}}$ (13″). We can now use the reduction pair processor as described in Ex. 20 to conclude that $\mathcal{R}_{\mathsf{incpl}}$ is iAST.

6 Conclusion and Evaluation

In this paper, we developed a new ADP framework, which advances the work of [24] into a *complete* criterion for almost-sure innermost termination by using annotated dependency pairs instead of dependency tuples, which also simplifies the framework substantially. Moreover, we adapted the *rewriting* processor of the DP framework to the probabilistic setting. Similarly, we also adapted the other transformational processors of the original non-probabilistic DP framework, see [1]. The soundness proofs for the adapted processors are much more involved than in the non-probabilistic setting, due to the more complex structure of chain trees. However, the processors themselves are analogous to their non-probabilistic counterparts, and thus, existing implementations of the processors can easily be adapted to their probabilistic versions.

We implemented our new contributions in the termination prover AProVE [16] and compared the new probabilistic ADP framework with transformational processors (ADP) to the DT framework from [24] (DT) and to AProVE's techniques for ordinary non-probabilistic TRSs (AProVE-NP), which include many additional processors and which benefit from using separate dependency pairs instead of ADPs or DTs. For the processors in Sect. 4, we could re-use the existing implementation from [24] for our ADP framework. The main goal for probabilistic termination analysis is to become as powerful as termination analysis in the non-probabilistic setting. Therefore, in our first experiment, we considered the non-probabilistic TRSs of the *TPDB* [32], the benchmark set used in the annual *Termination and Complexity Competition (TermComp)* [17] and compared ADP and DT with AProVE-NP, because at the current *TermComp*, AProVE-NP was the most powerful tool for termination of ordinary non-probabilistic TRSs. Clearly, a TRS can be represented as a PTRS with trivial probabilities, and then (innermost) AST is the same as (innermost) termination. While both ADP and DT have a probability removal processor to switch to the classical DP framework for such problems, we disabled that processor in this experiment. Since ADP and DT can only deal with innermost evaluation, we used the benchmarks from the "TRS Innermost" and "TRS Standard" categories of the *TPDB*, but only considered innermost evaluation for all examples. We used a timeout of 300 seconds for each example. The "TRS Innermost" category contains 366 benchmarks where AProVE-NP proves innermost termination for 293, DT is able to prove it for 133 (45% of AProVE-NP), and for ADP this number rises to 159 (54%). For the 1512 benchmarks from the "TRS Standard" category, AProVE-NP can prove innermost termination for 1114, DT for 611 (55% of AProVE-NP), and ADP for 723 (65%). This shows that the transformations are very important for automatic termination proofs as we get around 10% closer to AProVE-NP's results in both categories.

As a second experiment, we extended the PTRS benchmark set from [24] by 33 new PTRSs for typical probabilistic programs, including some examples with complicated probabilistic structure. For instance, we added the following PTRS $\mathcal{R}_{\mathsf{qsrt}}$ for probabilistic quicksort. Here, we write r instead of $\{1 : r\}$ for readability.

$$\begin{aligned}
\mathsf{rotate}(\mathsf{cons}(x, xs)) &\to \{{}^1\!/\!_2 : \mathsf{cons}(x, xs),\ {}^1\!/\!_2 : \mathsf{rotate}(\mathsf{app}(xs, \mathsf{cons}(x, \mathsf{nil})))\} \\
\mathsf{qsrt}(xs) &\to \mathsf{if}(\mathsf{empty}(xs),\ \mathsf{low}(\mathsf{hd}(xs), \mathsf{tl}(xs)),\ \mathsf{hd}(xs),\ \mathsf{high}(\mathsf{hd}(xs), \mathsf{tl}(xs))) \\
\mathsf{if}(\mathsf{true}, xs, x, ys) &\to \mathsf{nil} \qquad \mathsf{empty}(\mathsf{nil}) \to \mathsf{true} \qquad \mathsf{empty}(\mathsf{cons}(x, xs)) \to \mathsf{false} \\
\mathsf{if}(\mathsf{false}, xs, x, ys) &\to \mathsf{app}(\mathsf{qsrt}(\mathsf{rotate}(xs)),\ \mathsf{cons}(x,\ \mathsf{qsrt}(\mathsf{rotate}(ys)))) \\
\mathsf{hd}(\mathsf{cons}(x, xs)) &\to x \qquad \mathsf{tl}(\mathsf{cons}(x, xs)) \to xs
\end{aligned}$$

The rotate-rules rotate a list randomly often (they are AST, but not terminating). Thus, by choosing the first element of the resulting list, one obtains random pivot elements for the recursive calls of qsrt in the second if-rule. In addition to the rules above, $\mathcal{R}_{\mathsf{qsrt}}$ contains rules for list concatenation (app), and rules such that $\mathsf{low}(x, xs)$ ($\mathsf{high}(x, xs)$) returns all elements of the list xs that are smaller (greater or equal) than x, see [1]. In contrast to the quicksort example in [24], proving iAST of the above rules requires transformational processors to instantiate and rewrite the empty-, hd-, and tl-subterms in the right-hand side of the qsrt-rule. So while DT fails for this example, ADP can prove iAST of $\mathcal{R}_{\mathsf{qsrt}}$.

90 of the 100 PTRSs in our set are iAST, and DT succeeds for 54 of them (60 %) with the technique of [24] that does not use transformational processors. Adding the new processors in ADP increases this number to 77 (86 %), which demonstrates their power for PTRSs with non-trivial probabilities. For details on our experiments and for instructions on how to run our implementation in AProVE via its *web interface* or locally, see: https://aprove-developers.github.io/ProbabilisticADPs/

There, we also performed experiments where we disabled individual transformational processors of the ADP framework, which shows the usefulness of each new processor. In addition to the ADP and DT framework, an alternative technique to analyze PTRSs via a direct application of interpretations was presented in [4]. However, [4] analyzes PAST (or rather *strong* AST), and a comparison between the DT framework and their technique can be found in [24]. In future work, we will adapt more processors of the original DP framework to the probabilistic setting. Moreover, we work on analyzing AST also for full instead of innermost rewriting.

References

[1] *A Complete Dependency Pair Framework for Almost-Sure Innermost Termination of Probabilistic Term Rewriting*. Long version with all proofs. Available from https://aprove-developers.github.io/ProbabilisticADPs/. 2023.

[2] S. Agrawal, K. Chatterjee, and P. Novotný. "Lexicographic Ranking Supermartingales: An Efficient Approach to Termination of Probabilistic Programs". In: *Proc. ACM Program. Lang.* 2.POPL (2017). DOI: 10.1145/3158122.

[3] T. Arts and J. Giesl. "Termination of Term Rewriting Using Dependency Pairs". In: *Theor. Comput. Sc.* 236.1-2 (2000), pp. 133–178. DOI: 10.1016/S0304-3975(99)00207-8.

[4] M. Avanzini, U. Dal Lago, and A. Yamada. "On Probabilistic Term Rewriting". In: *Sci. Comput. Program.* 185 (2020). DOI: 10.1016/j.scico.2019.102338.

[5] M. Avanzini, G. Moser, and M. Schaper. "A Modular Cost Analysis for Probabilistic Programs". In: *Proc. ACM Program. Lang.* 4.OOPSLA (2020). DOI: 10.1145/3428240.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. DOI: 10.1017/CBO9781139172752.

[7] K. Batz, B. L. Kaminski, J.-P. Katoen, C. Matheja, and L. Verscht. "A Calculus for Amortized Expected Runtimes". In: *Proc. ACM Program. Lang.* 7.POPL (2023). DOI: 10.1145/3571260.

[8] R. Beutner and L. Ong. "On Probabilistic Termination of Functional Programs with Continuous Distributions". In: *Proc. PLDI '21*. 2021, pp. 1312–1326. DOI: 10.1145/3453483.3454111.

[9] O. Bournez and F. Garnier. "Proving Positive Almost-Sure Termination". In: *Proc. RTA '05*. LNCS 3467. 2005, pp. 323–337. DOI: 10.1007/978-3-540-32033-3_24.

[10] K. Chatterjee, H. Fu, and P. Novotný. "Termination Analysis of Probabilistic Programs with Martingales". In: *Foundations of Probabilistic Programming*. Ed. by G. Barthe, J. Katoen, and A. Silva. Cambridge University Press, 2020, 221–258. DOI: 10.1017/9781108770750.008.

[11] U. Dal Lago and C. Grellois. "Probabilistic Termination by Monadic Affine Sized Typing". In: *Proc. ESOP '17*. LNCS 10201. 2017, pp. 393–419. DOI: 10.1007/978-3-662-54434-1_15.

[12] C. Faggian. "Probabilistic Rewriting and Asymptotic Behaviour: On Termination and Unique Normal Forms". In: *Log. Methods in Comput. Sci.* 18.2 (2022). DOI: 10.46298/lmcs-18(2:5)2022.

[13] L. M. Ferrer Fioriti and H. Hermanns. "Probabilistic Termination: Soundness, Completeness, and Compositionality". In: *Proc. POPL '15*. 2015, pp. 489–501. DOI: 10.1145/2676726.2677001.

[14] J. Giesl, R. Thiemann, and P. Schneider-Kamp. "The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs".

In: *Proc. LPAR '04*. LNCS 3452. 2004, pp. 301–331. DOI: 10.1007/978-3-540-32275-7_21.

[15] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. "Mechanizing and Improving Dependency Pairs". In: *J. Autom. Reason.* 37.3 (2006), pp. 155–203. DOI: 10.1007/s10817-006-9057-7.

[16] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. "Analyzing Program Termination and Complexity Automatically with AProVE". In: *J. Autom. Reason.* 58.1 (2017), pp. 3–31. DOI: 10.1007/s10817-016-9388-y.

[17] J. Giesl, A. Rubio, C. Sternagel, J. Waldmann, and A. Yamada. "The Termination and Complexity Competition". In: *Proc. TACAS '19*. LNCS 11429. 2019, pp. 156–166. DOI: 10.1007/978-3-030-17502-3_10.

[18] J. Giesl, P. Giesl, and M. Hark. "Computing Expected Runtimes for Constant Probability Programs". In: *Proc. CADE '19*. LNCS 11716. 2019, pp. 269–286. DOI: 10.1007/978-3-030-29436-6_16.

[19] B. Gramlich. "Abstract Relations between Restricted Termination and Confluence Properties of Rewrite Systems". In: *Fundamenta Informaticae* 24 (1995), pp. 2–23.

[20] N. Hirokawa and A. Middeldorp. "Automating the Dependency Pair Method". In: *Inf. Comput.* 199.1-2 (2005), pp. 172–199. DOI: 10.1016/j.ic.2004.10.004.

[21] M. Huang, H. Fu, K. Chatterjee, and A. K. Goharshady. "Modular Verification for Almost-Sure Termination of Probabilistic Programs". In: *Proc. ACM Program. Lang.* 3.OOPSLA (2019). DOI: 10.1145/3360555.

[22] B. L. Kaminski, J.-P. Katoen, C. Matheja, and F. Olmedo. "Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms". In: *J. ACM* 65 (2018), pp. 1–68. DOI: 10.1145/3208102.

[23] B. L. Kaminski, J. Katoen, and C. Matheja. "Expected Runtime Analyis by Program Verification". In: *Foundations of Probabilistic Programming*. Ed. by G. Barthe, J. Katoen, and A. Silva. Cambridge University Press, 2020, 185–220. DOI: 10.1017/9781108770750.007.

[24] J.-C. Kassing and J. Giesl. "Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs". In: *Proc. CADE '23*. LNCS 14132. 2023, pp. 344–364. DOI: 10.1007/978-3-031-38499-8_20.

[25] U. D. Lago, C. Faggian, and S. R. Della Rocca. "Intersection Types and (Positive) Almost-Sure Termination". In: *Proc. ACM Program. Lang.* 5.POPL (2021). DOI: 10.1145/3434313.

[26] D. S. Lankford. *On Proving Term Rewriting Systems are Noetherian.* Memo MTP-3, Math. Dept., Louisiana Technical University, Ruston, LA, 1979. URL: http://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford_Poly_Term.pdf.

[27] L. Leutgeb, G. Moser, and F. Zuleger. "Automated Expected Amortised Cost Analysis of Probabilistic Data Structures". In: *Proc. CAV '22*. LNCS 13372. 2022, pp. 70–91. DOI: 10.1007/978-3-031-13188-2_4.

[28] A. McIver, C. Morgan, B. L. Kaminski, and J.-P. Katoen. "A New Proof Rule for Almost-Sure Termination". In: *Proc. ACM Program. Lang.* 2.POPL (2018). DOI: 10.1145/3158121.

[29] F. Meyer, M. Hark, and J. Giesl. "Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes". In: *Proc. TACAS '21*. LNCS 12651. 2021, pp. 250–269. DOI: 10.1007/978-3-030-72016-2_14.

[30] M. Moosbrugger, E. Bartocci, J.-P. Katoen, and L. Kovács. "Automated Termination Analysis of Polynomial Probabilistic Programs". In: *Proc. ESOP '21*. LNCS 12648. 2021, pp. 491–518. DOI: 10.1007/978-3-030-72019-3_18.

[31] V. C. Ngo, Q. Carbonneaux, and J. Hoffmann. "Bounded Expectations: Resource Analysis for Probabilistic Programs". In: *Proc. PLDI '18*. 2018, pp. 496–512. DOI: 10.1145/3192366.3192394.

[32] *Termination Problem Data Base*. https://github.com/TermCOMP/TPDB.

[33] D. Wang, D. M. Kahn, and J. Hoffmann. "Raising Expectations: Automating Expected Cost Analysis with Types". In: *Proc. ACM Program. Lang.* 4.ICFP (2020). DOI: 10.1145/3408992.