Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs*

Jan-Christoph Kassing(✉) and Jürgen Giesl(✉)

LuFG Informatik 2, RWTH Aachen University, Aachen, Germany

Abstract. Dependency pairs are one of the most powerful techniques to analyze termination of term rewrite systems (TRSs) automatically. We adapt the dependency pair framework to the probabilistic setting in order to prove almost-sure innermost termination of probabilistic TRSs. To evaluate its power, we implemented the new framework in our tool AProVE.

1 Introduction

Techniques and tools to analyze innermost termination of term rewrite systems (TRSs) automatically are successfully used for termination analysis of programs in many languages (e.g., Java [10, 35, 38], Haskell [18], and Prolog [19]). While there exist several classical orderings for proving termination of TRSs (e.g., based on polynomial interpretations [30]), a *direct* application of these orderings is usually too weak for TRSs that result from actual programs. However, these orderings can be used successfully within the *dependency pair* (DP) framework [2, 16, 17]. This framework allows for modular termination proofs (e.g., which apply different orderings in different sub-proofs) and is one of the most powerful techniques for termination analysis of TRSs that is used in essentially all current termination tools for TRSs, e.g., AProVE [20], MU-TERM [22], NaTT [40], TTT2 [29], etc.

On the other hand, *probabilistic* programs are used to describe randomized algorithms and probability distributions, with applications in many areas. To use TRSs also for such programs, *probabilistic term rewrite systems* (PTRSs) were introduced in [8, 9]. In the probabilistic setting, there are several notions of "termination". A program is *almost-surely terminating* (AST) if the probability for termination is 1. As remarked in [24]: "AST is the classical and most widely-studied problem that extends termination of non-probabilistic programs, and is considered as a core problem in the programming languages community". A strictly stronger notion is *positive almost-sure termination* (PAST), which requires that the expected runtime is finite. While there exist many automatic approaches to prove (P)AST of imperative programs on numbers (e.g., [1, 4, 11, 15, 21, 24–26, 32–34, 36]), there are only few automatic approaches for programs with complex non-tail recursive structure [7, 12], and even less approaches which

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

are also suitable for algorithms on recursive data structures [3, 6, 31, 39]. The approach of [39] focuses on algorithms on lists and [31] mainly targets algorithms on trees, but they cannot easily be adjusted to other (possibly user-defined) data structures. The calculus of [6] considers imperative programs with stack, heap, and pointers, but it is not yet automated. Moreover, the approaches of [3, 6, 31, 39] analyze expected runtime, while we focus on AST.

PTRSs can be used to model algorithms (possibly with complex recursive structure) operating on algebraic data types. While PTRSs were introduced in [8, 9], the first (and up to now only) tool to analyze their termination automatically was presented in [3], where orderings based on interpretations were adapted to prove PAST. Moreover, [14] extended general concepts of abstract rewrite systems (e.g., confluence and uniqueness of normal forms) to the probabilistic setting.

As mentioned, already for non-probabilistic TRSs a *direct* application of orderings (as in [3]) is limited in power. To obtain a powerful approach, one should combine such orderings in a modular way, as in the DP framework. In this paper, we show for the first time that an adaption of dependency pairs to the probabilistic setting is possible and present the first DP framework for probabilistic term rewriting. Since the crucial idea of dependency pairs is the modularization of the termination proof, we analyze AST instead of PAST, because it is well known that AST is compositional, while PAST is not (see, e.g., [25]). We also present a novel adaption of the technique from [3] for the direct application of polynomial interpretations in order to prove AST (instead of PAST) of PTRSs.

We start by briefly recapitulating the DP framework for non-probabilistic TRSs in Sect. 2. Then we recall the definition of PTRSs based on [3, 9, 14] in Sect. 3 and introduce a novel way to prove AST using polynomial interpretations automatically. In Sect. 4 we present our new probabilistic DP framework. The implementation of our approach in the tool AProVE is evaluated in Sect. 5. We refer to [28] for all proofs (which are much more involved than the original proofs for the non-probabilistic DP framework from [2, 16, 17]).

2 The DP Framework

We assume familiarity with term rewriting [5] and regard TRSs over a finite signature Σ and a set of variables $\mathcal{V}$. A *polynomial interpretation* Pol is a Σ-algebra with carrier set $\mathbb{N}$ which maps every function symbol $f \in \Sigma$ to a polynomial $f_{\text{Pol}} \in \mathbb{N}[\mathcal{V}]$. For a term $t \in \mathcal{T}(\Sigma, \mathcal{V})$, $\text{Pol}(t)$ denotes the interpretation of t by the Σ-algebra Pol. An arithmetic inequation like $\text{Pol}(t_1) > \text{Pol}(t_2)$ *holds* if it is true for all instantiations of its variables by natural numbers.

Theorem 1 (Termination With Polynomial Interpretations [30]). *Let $\mathcal{R}$ be a TRS and let $\text{Pol} : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathbb{N}[\mathcal{V}]$ be a monotonic polynomial interpretation (i.e., $x > y$ implies $f_{\text{Pol}}(\ldots, x, \ldots) > f_{\text{Pol}}(\ldots, y, \ldots)$ for all $f \in \Sigma$). If for every $\ell \to r \in \mathcal{R}$, we have $\text{Pol}(\ell) > \text{Pol}(r)$, then $\mathcal{R}$ is terminating.*

The search for polynomial interpretations is usually automated by SMT solving. Instead of polynomials over the naturals, Thm. 1 (and the other termination criteria in the paper) can also be extended to polynomials over the

non-negative reals, by requiring that whenever a term is "strictly decreasing", then its interpretation decreases at least by a certain fixed amount $\delta > 0$.

Example 2. Consider the TRS $\mathcal{R}_{\mathsf{div}} = \{(1), \dots, (4)\}$ for division from [2].

$$\mathsf{minus}(x, \mathcal{O}) \to x \quad (1) \qquad \mathsf{div}(\mathcal{O}, \mathsf{s}(y)) \to \mathcal{O} \quad (3)$$

$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{minus}(x, y) \quad (2) \qquad \mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{s}\,(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y))) \quad (4)$$

Termination of $\mathcal{R}_{\mathsf{minus}} = \{(1), (2)\}$ can be proved by the polynomial interpretation that maps $\mathsf{minus}(x, y)$ to $x + y + 1$, $\mathsf{s}(x)$ to $x + 1$, and $\mathcal{O}$ to 0. However, a direct application of classical techniques like polynomial interpretations fails for $\mathcal{R}_{\mathsf{div}}$. These techniques correspond to so-called (quasi-)simplification orderings [13] which cannot handle rules like (4) where the right-hand side is embedded in the left-hand side if y is instantiated with $\mathsf{s}(x)$. In contrast, the dependency pair framework is able to prove termination of $\mathcal{R}_{\mathsf{div}}$ automatically.

We now recapitulate the DP framework and its core processors, and refer to, e.g., [2, 16, 17, 23] for more details. In this paper, we restrict ourselves to the DP framework for *innermost* rewriting (denoted "$\xrightarrow[\mathcal{R}]{\mathsf{i}}$"), because our adaption to the probabilistic setting relies on this evaluation strategy (see Sect. 4.1).

Definition 3 (Dependency Pair). *Let $\mathcal{R}$ be a (finite) TRS. We decompose its signature $\Sigma = \Sigma_C \uplus \Sigma_D$ such that $f \in \Sigma_D$ if $f = \mathrm{root}(\ell)$ for some rule $\ell \to r \in \mathcal{R}$. The symbols in Σ_C and Σ_D are called* constructors *and* defined symbols*, respectively. For every $f \in \Sigma_D$, we introduce a fresh* tuple symbol *$f^\#$ of the same arity. Let $\Sigma^\#$ denote the set of all tuple symbols. To ease readability, we often write F instead of $\mathsf{f}^\#$. For any term $t = f(t_1, \dots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ with $f \in \Sigma_D$, let $t^\# = f^\#(t_1, \dots, t_n)$. Moreover, for any $r \in \mathcal{T}(\Sigma, \mathcal{V})$, let $\mathrm{Sub}_D(r)$ be the set of all subterms of r with defined root symbol. For a rule $\ell \to r$ with $\mathrm{Sub}_D(r) = \{t_1, \dots, t_n\}$, one obtains the n dependency pairs (DPs) $\ell^\# \to t_i^\#$ with $1 \le i \le n$. $\mathcal{DP}(\mathcal{R})$ denotes the set of all dependency pairs of $\mathcal{R}$.*

Example 4. For the TRS $\mathcal{R}_{\mathsf{div}}$ from Ex. 2, we get the following dependency pairs.

$$\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{M}(x, y) \quad (5) \qquad \mathsf{D}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{M}(x, y) \quad (6)$$

$$\mathsf{D}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{D}(\mathsf{minus}(x, y), \mathsf{s}(y)) \quad (7)$$

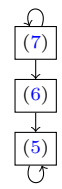
The DP framework uses *DP problems* $(\mathcal{D}, \mathcal{R})$ where $\mathcal{D}$ is a (finite) set of DPs and $\mathcal{R}$ is a (finite) TRS. A (possibly infinite) sequence $t_0^\#, t_1^\#, t_2^\#, \dots$ with $t_i^\# \xrightarrow[\mathcal{D},\mathcal{R}]{\mathsf{i}} \circ \xrightarrow[\mathcal{R}]{\mathsf{i}}{}^{*} t_{i+1}^\#$ for all i is an (innermost) $(\mathcal{D}, \mathcal{R})$-*chain*. Here, $\xrightarrow[\mathcal{D},\mathcal{R}]{\mathsf{i}}$ is the restriction of $\to_{\mathcal{D}}$ to rewrite steps where the used redex is in normal form w.r.t. $\mathcal{R}$. A chain represents subsequent "function calls" in evaluations. Between two function calls (corresponding to steps with $\mathcal{D}$) one can evaluate the arguments with $\mathcal{R}$. For example, $\mathsf{D}(\mathsf{s}^2(\mathcal{O}), \mathsf{s}(\mathcal{O}))$, $\mathsf{D}(\mathsf{s}(\mathcal{O}), \mathsf{s}(\mathcal{O}))$ is a $(\mathcal{DP}(\mathcal{R}_{\mathsf{div}}), \mathcal{R}_{\mathsf{div}})$-chain, as $\mathsf{D}(\mathsf{s}^2(\mathcal{O}), \mathsf{s}(\mathcal{O})) \xrightarrow[\mathcal{DP}(\mathcal{R}_{\mathsf{div}}),\mathcal{R}_{\mathsf{div}}]{\mathsf{i}} \mathsf{D}(\mathsf{minus}(\mathsf{s}(\mathcal{O}), \mathcal{O}), \mathsf{s}(\mathcal{O})) \xrightarrow[\mathcal{R}_{\mathsf{div}}]{\mathsf{i}}{}^{*} \mathsf{D}(\mathsf{s}(\mathcal{O}), \mathsf{s}(\mathcal{O}))$, where $\mathsf{s}^2(\mathcal{O})$ is $\mathsf{s}(\mathsf{s}(\mathcal{O}))$.

A DP problem $(\mathcal{D}, \mathcal{R})$ is called *innermost terminating* (iTerm) if there is no infinite innermost $(\mathcal{D}, \mathcal{R})$-chain. The main result on dependency pairs is the *chain criterion* which states that a TRS $\mathcal{R}$ 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 $\text{Proc}(\mathcal{D},\mathcal{R}) = \{(\mathcal{D}_1,\mathcal{R}_1),\dots,(\mathcal{D}_n,\mathcal{R}_n)\}$, where $\mathcal{D},\mathcal{D}_1,\dots,\mathcal{D}_n$ are sets of dependency pairs and $\mathcal{R},\mathcal{R}_1,\dots,\mathcal{R}_n$ are TRSs. A processor Proc is *sound* if $(\mathcal{D},\mathcal{R})$ is iTerm whenever $(\mathcal{D}_i,\mathcal{R}_i)$ is iTerm for all $1 \le i \le n$. It is *complete* if $(\mathcal{D}_i,\mathcal{R}_i)$ is iTerm for all $1 \le i \le n$ whenever $(\mathcal{D},\mathcal{R})$ 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 allows for modular termination proofs, since different techniques can be applied on each resulting "sub-problem" $(\mathcal{D}_i,\mathcal{R}_i)$. The following three theorems recapitulate the three most important processors of the DP framework.

The (innermost) $(\mathcal{D},\mathcal{R})$-*dependency graph* is a control flow graph that indicates which dependency pairs can be used after each other in a chain. Its node set is $\mathcal{D}$ 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 $t_1^\#\sigma_1 \xrightarrow{i}{}^*_{\mathcal{R}} \ell_2^\#\sigma_2$, and both $\ell_1^\#\sigma_1$ and $\ell_2^\#\sigma_2$ are in normal form w.r.t. $\mathcal{R}$. Any infinite $(\mathcal{D},\mathcal{R})$-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).[1] Hence, it suffices to consider the SCCs of this graph independently. The $(\mathcal{DP}(\mathcal{R}_{\mathsf{div}}),\mathcal{R}_{\mathsf{div}})$-dependency graph can be seen on the right.



Theorem 5 (Dep. Graph Processor). *For the SCCs* $\mathcal{D}_1,\dots,\mathcal{D}_n$ *of the* $(\mathcal{D},\mathcal{R})$*-dependency graph,* $\text{Proc}_{\texttt{DG}}(\mathcal{D},\mathcal{R}) = \{(\mathcal{D}_1,\mathcal{R}),\dots,(\mathcal{D}_n,\mathcal{R})\}$ *is sound and complete.*

While the exact dependency graph is not computable in general, there are several techniques to over-approximate it automatically, see, e.g., [2, 17, 23]. In our example, applying $\text{Proc}_{\texttt{DG}}$ to the initial problem $(\mathcal{DP}(\mathcal{R}_{\mathsf{div}}),\mathcal{R}_{\mathsf{div}})$ results in the smaller problems $\big(\{(5)\},\mathcal{R}_{\mathsf{div}}\big)$ and $\big(\{(7)\},\mathcal{R}_{\mathsf{div}}\big)$ that can be treated separately.

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

Theorem 6 (Usable Rules Processor). *Let* $\mathcal{R}$ *be a TRS. For every* $f \in \Sigma \uplus \Sigma^\#$ *let* $\text{Rules}_{\mathcal{R}}(f) = \{\ell \to r \in \mathcal{R} \mid \text{root}(\ell) = f\}$*. For any* $t \in \mathcal{T}\big(\Sigma \uplus \Sigma^\#,\mathcal{V}\big)$*, 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* $\mathcal{U}_{\mathcal{R}}(f(t_1,\dots,t_n)) = \text{Rules}_{\mathcal{R}}(f) \cup \bigcup_{i=1}^n \mathcal{U}_{\mathcal{R}}(t_i) \cup \bigcup_{\ell\to r\in\text{Rules}_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}}(r)$*. The* usable rules *for the DP problem* $(\mathcal{D},\mathcal{R})$ *are* $\mathcal{U}(\mathcal{D},\mathcal{R}) = \bigcup_{\ell^\#\to t^\#\in\mathcal{D}} \mathcal{U}_{\mathcal{R}}(t^\#)$*. Then* $\text{Proc}_{\texttt{UR}}(\mathcal{D},\mathcal{R}) = \{(\mathcal{D},\mathcal{U}(\mathcal{D},\mathcal{R}))\}$ *is sound but not complete.*[2]

For the DP problem $\big(\{(7)\},\mathcal{R}_{\mathsf{div}}\big)$ only the minus-rules are usable and thus $\text{Proc}_{\texttt{UR}}\big(\{(7)\},\mathcal{R}_{\mathsf{div}}\big) = \{\{(7)\},\{(1),(2)\}\}$. For $\big(\{(5)\},\mathcal{R}_{\mathsf{div}}\big)$ there are no usable rules at all, and thus $\text{Proc}_{\texttt{UR}}\big(\{(5)\},\mathcal{R}_{\mathsf{div}}\big) = \{\big(\{(5)\},\varnothing\big)\}$.

[1] Here, a set $\mathcal{D}'$ of dependency pairs 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 $\mathcal{D}'$ there is a non-empty path from $\ell_1^\# \to t_1^\#$ to $\ell_2^\# \to t_2^\#$ which only traverses nodes from $\mathcal{D}'$.

[2] For a complete version of the usable rules processor, one has to use a more involved notion of DP problems with more components that we omit here for readability [16].

The last processor adapts classical orderings like polynomial interpretations to DP problems.[3] In contrast to their direct application in Thm. 1, we may now use weakly monotonic polynomials f_{Pol} that do not have to depend on all of their arguments. The reduction pair processor requires that all rules and dependency pairs are weakly decreasing and it removes those DPs that are strictly decreasing.

Theorem 7 (Reduction Pair Processor with Polynomial Interpretations). *Let* $\mathrm{Pol} : \mathcal{T}\left(\Sigma \uplus \Sigma^{\#}, \mathcal{V}\right) \to \mathbb{N}[\mathcal{V}]$ *be a weakly monotonic polynomial interpretation (i.e.,* $x \geq y$ *implies* $f_{\mathrm{Pol}}(\ldots, x, \ldots) \geq f_{\mathrm{Pol}}(\ldots, y, \ldots)$ *for all* $f \in \Sigma \uplus \Sigma^{\#}$*). Let* $\mathcal{D} = \mathcal{D}_{\geq} \uplus \mathcal{D}_{>}$ *with* $\mathcal{D}_{>} \neq \varnothing$ *such that:*

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

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

The constraints of the reduction pair processor for the remaining DP problems $(\{(7)\}, \{(1), (2)\})$ and $(\{(5)\}, \varnothing)$ are satisfied by the polynomial interpretation which maps $\mathcal{O}$ to 0, $\mathsf{s}(x)$ to $x + 1$, and all other non-constant function symbols to the projection on their first arguments. Since (7) and (5) are strictly decreasing, $\mathrm{Proc}_{\mathtt{RP}}$ transforms both $(\{(7)\}, \{(1), (2)\})$ and $(\{(5)\}, \varnothing)$ into DP problems of the form $(\varnothing, \ldots)$. 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 $(\mathcal{DP}(\mathcal{R}_{\mathsf{div}}), \mathcal{R}_{\mathsf{div}})$ and thus, $\mathcal{R}_{\mathsf{div}}$ is innermost terminating.

3 Probabilistic Term Rewriting

Now we recapitulate *probabilistic TRSs* [3, 9, 14] and present a novel criterion to prove almost-sure termination automatically by adapting the direct application of polynomial interpretations from Thm. 1 to PTRSs. In contrast to TRSs, a PTRS has finite[4] multi-distributions on the right-hand side of rewrite rules.

Definition 8 (Multi-Distribution). *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* $a \in A$*, such that* $\sum_{(p:a)\in\mu} p = 1$*.* $\mathrm{FDist}(A)$ *is the set of all finite multi-distributions on* A*. For* $\mu \in \mathrm{FDist}(A)$*, its* support *is the multiset* $\mathrm{Supp}(\mu) = \{a \mid (p:a) \in \mu \text{ for some } p\}$*.*

Definition 9 (PTRS). *A* probabilistic rewrite rule *is 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* $r \in \mathrm{Supp}(\mu)$*. A* probabilistic TRS *(PTRS) is a finite set* $\mathcal{R}$ *of probabilistic rewrite rules. Similar to*

[3] 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".

[4] Since our goal is the automation of termination analysis, in this paper we restrict ourselves to finite PTRSs with finite multi-distributions.

TRSs, the 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* $s \to_{\mathcal{R}} \{p_1 : t_1, \dots, p_k : t_k\}$ *if there is a position* π*, a rule* $\ell \to \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{R}$*, and a substitution* σ *such that* $s|_\pi = \ell\sigma$ *and* $t_j = s[r_j\sigma]_\pi$ *for all* $1 \le j \le k$*. We call* $s \to_{\mathcal{R}} \mu$ *an* innermost *rewrite step (denoted* $s \overset{\mathsf{i}}{\to}_{\mathcal{R}} \mu$*) if every proper subterm of the used redex* $\ell\sigma$ *is in normal form w.r.t.* $\mathcal{R}$*.*

Example 10. As an example, consider the PTRS $\mathcal{R}_{\mathsf{rw}}$ with the only rule $\mathsf{g}(x) \to \{1/2 : x,\ 1/2 : \mathsf{g}(\mathsf{g}(x))\}$, which corresponds to a symmetric random walk.

As proposed in [3], we *lift* $\to_{\mathcal{R}}$ to a rewrite relation between multi-distributions in order to track all probabilistic rewrite sequences (up to non-determinism) at once. For any $0 < p \le 1$ and any $\mu \in \mathrm{FDist}(A)$, let $p \cdot \mu = \{(p \cdot q : a) \mid (q : a) \in \mu\}$.

Definition 11 (Lifting). *The* lifting $\rightrightarrows \subseteq \mathrm{FDist}(\mathcal{T}(\Sigma,\mathcal{V})) \times \mathrm{FDist}(\mathcal{T}(\Sigma,\mathcal{V}))$ *of a relation* $\to \subseteq \mathcal{T}(\Sigma,\mathcal{V}) \times \mathrm{FDist}(\mathcal{T}(\Sigma,\mathcal{V}))$ *is the smallest relation with:*

- *If* $t \in \mathcal{T}(\Sigma,\mathcal{V})$ *is in normal form w.r.t.* $\to$*, then* $\{1 : t\} \rightrightarrows \{1 : t\}$*.*
- *If* $t \to \mu$*, then* $\{1 : t\} \rightrightarrows \mu$*.*
- *If for all* $1 \le j \le k$ *there are* $\mu_j, \nu_j \in \mathrm{FDist}(\mathcal{T}(\Sigma,\mathcal{V}))$ *with* $\mu_j \rightrightarrows \nu_j$ *and* $0 < p_j \le 1$ *with* $\sum_{1 \le j \le k} p_j = 1$*, then* $\bigcup_{1 \le j \le k} p_j \cdot \mu_j \rightrightarrows \bigcup_{1 \le j \le k} p_j \cdot \nu_j$*.*

For a PTRS $\mathcal{R}$, we write $\rightrightarrows_{\mathcal{R}}$ and $\overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}}$ for the liftings of $\to_{\mathcal{R}}$ and $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$, respectively.

Example 12. For instance, we obtain the following $\overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_{\mathsf{rw}}}$-rewrite sequence:

$$\{1 : \mathsf{g}(\mathcal{O})\} \overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_{\mathsf{rw}}} \{1/2 : \mathcal{O}, 1/2 : \mathsf{g}^2(\mathcal{O})\} \overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_{\mathsf{rw}}} \{1/2 : \mathcal{O}, 1/4 : \mathsf{g}(\mathcal{O}), 1/4 : \mathsf{g}^3(\mathcal{O})\}$$
$$\overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_{\mathsf{rw}}} \{1/2 : \mathcal{O}, 1/8 : \mathcal{O}, 1/8 : \mathsf{g}^2(\mathcal{O}), 1/8 : \mathsf{g}^2(\mathcal{O}), 1/8 : \mathsf{g}^4(\mathcal{O})\}$$

Note that the two occurrences of $\mathcal{O}$ and $\mathsf{g}^2(\mathcal{O})$ in the multi-distribution above could be rewritten differently if the PTRS had rules resulting in different terms. So it should be distinguished from $\{5/8 : \mathcal{O}, 1/4 : \mathsf{g}^2(\mathcal{O}), 1/8 : \mathsf{g}^4(\mathcal{O})\}$.

To express the concept of almost-sure termination, one has to determine the probability for normal forms in a multi-distribution.

Definition 13 ($|\mu|_{\mathcal{R}}$). *For a PTRS* $\mathcal{R}$*,* $\mathtt{NF}_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma,\mathcal{V})$ *denotes the set of all normal forms w.r.t.* $\mathcal{R}$*. For any* $\mu \in \mathrm{FDist}(\mathcal{T}(\Sigma,\mathcal{V}))$*, let* $|\mu|_{\mathcal{R}} = \sum_{(p:t) \in \mu, t \in \mathtt{NF}_{\mathcal{R}}} p$*.*

Example 14. Consider the multi-distribution $\{1/2 : \mathcal{O}, 1/8 : \mathcal{O}, 1/8 : \mathsf{g}^2(\mathcal{O}), 1/8 : \mathsf{g}^2(\mathcal{O}), 1/8 : \mathsf{g}^4(\mathcal{O})\}$ from Ex. 12 and $\mathcal{R}_{\mathsf{rw}}$ from Ex. 10. Then $|\mu|_{\mathcal{R}_{\mathsf{rw}}} = 1/2 + 1/8 = 5/8$.

Definition 15 ((Innermost) AST). *Let* $\mathcal{R}$ *be a PTRS and* $(\mu_n)_{n \in \mathbb{N}}$ *be an infinite* $\rightrightarrows_{\mathcal{R}}$*-rewrite sequence, i.e.,* $\mu_n \rightrightarrows_{\mathcal{R}} \mu_{n+1}$ *for all* $n \in \mathbb{N}$*. Note that* $\lim_{n \to \infty} |\mu_n|_{\mathcal{R}}$ *exists, since* $|\mu_n|_{\mathcal{R}} \le |\mu_{n+1}|_{\mathcal{R}} \le 1$ *for all* $n \in \mathbb{N}$*.* $\mathcal{R}$ *is* almost-surely terminating (AST) *(*innermost almost-surely terminating (iAST)*) if* $\lim_{n \to \infty} |\mu_n|_{\mathcal{R}} = 1$ *holds for every infinite* $\rightrightarrows_{\mathcal{R}}$*-rewrite sequence (*$\overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}}$*-rewrite sequence)* $(\mu_n)_{n \in \mathbb{N}}$*.*

Example 16. For the (unique) infinite extension of the $\overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_{\mathsf{rw}}}$-rewrite sequence $(\mu_n)_{n \in \mathbb{N}}$ in Ex. 12, we have $\lim_{n \to \infty} |\mu_n|_{\mathcal{R}} = 1$. Indeed, $\mathcal{R}_{\mathsf{rw}}$ is AST (but not PAST, i.e., the expected number of rewrite steps is infinite for every term containing g).

Thm. 17 introduces a novel technique to prove AST automatically using a direct application of polynomial interpretations.

Theorem 17 (Proving AST with Polynomial Interpretations). *Let $\mathcal{R}$ be a PTRS, let* $\mathrm{Pol} : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathbb{N}[\mathcal{V}]$ *be a monotonic, multilinear*[5] *polynomial interpretation (i.e., for all $f \in \Sigma$, all monomials of $f_{\mathrm{Pol}}(x_1, \ldots, x_n)$ have the form $c \cdot x_1^{e_1} \cdot \ldots \cdot x_n^{e_n}$ with $c \in \mathbb{N}$ and $e_1, \ldots, e_n \in \{0, 1\}$). If for every rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{R}$,*

(1) there exists a $1 \leq j \leq k$ with $\mathrm{Pol}(\ell) > \mathrm{Pol}(r_j)$ and
(2) $\mathrm{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(r_j)$,

then $\mathcal{R}$ is AST.

In [3], it was shown that PAST can be proved by using multilinear polynomials and requiring a strict decrease in the expected value of each rule. In contrast, we only require a weak decrease of the expected value in (2) and in addition, at least one term in the support of the right-hand side must become strictly smaller (1). As mentioned, the proof for Thm. 17 (and for all our other new results and observations) can be found in [28]. The proof idea is based on [32], but it extends their approach from while-programs on integers to terms. However, in contrast to [32], PTRSs can only deal with constant probabilities, since all variables stand for terms, not for numbers. Note that the constraints (1) and (2) of our new criterion in Thm. 17 are equivalent to the constraint of the classical Thm. 1 in the special case where the PTRS is in fact a TRS (i.e., all rules have the form $\ell \to \{1 : r\}$).

Example 18. To prove that $\mathcal{R}_{\mathsf{rw}}$ is AST with Thm. 17, we can use the polynomial interpretation that maps $\mathsf{g}(x)$ to $x + 1$ and $\mathcal{O}$ to 0.

4 Probabilistic Dependency Pairs

We introduce our new adaption of DPs to the probabilistic setting in Sect. 4.1. Then we present the processors for the probabilistic DP framework in Sect. 4.2.

4.1 Dependency Tuples and Chains for Probabilistic Term Rewriting

We first show why straightforward adaptions are unsound. A natural idea to define DPs for probabilistic rules $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{R}$ would be (8) or (9):

$$\{\, \ell^{\#} \to \{p_1 : r_1, \ldots, p_i : t_j^{\#}, \ldots, p_k : r_k\} \mid t_j \in \mathrm{Sub}_D(r_j) \text{ with } 1 \leq j \leq k \,\} \quad (8)$$

$$\{\, \ell^{\#} \to \{p_1 : t_1^{\#}, \ldots, p_k : t_k^{\#}\} \mid t_j \in \mathrm{Sub}_D(r_j) \text{ for all } 1 \leq j \leq k \,\} \quad (9)$$

For (9), if $\mathrm{Sub}_D(r_j) = \varnothing$, then we insert a fresh constructor $\bot$ into $\mathrm{Sub}_D(r_j)$ that does not occur in $\mathcal{R}$. So in both (8) and (9), we replace r_j by a single term $t_j^{\#}$ in the right-hand side. The following example shows that this notion of probabilistic DPs does not yield a sound chain criterion. Consider the PTRSs $\mathcal{R}_1$ and $\mathcal{R}_2$:

$$\mathcal{R}_1 = \{\mathsf{g} \to \{1/2 : \mathcal{O}, 1/2 : \mathsf{f}(\mathsf{g}, \mathsf{g})\}\} \qquad \mathcal{R}_2 = \{\mathsf{g} \to \{1/2 : \mathcal{O}, 1/2 : \mathsf{f}(\mathsf{g}, \mathsf{g}, \mathsf{g})\}\} \quad (10)$$

[5] As in [3], multilinearity ensures "monotonicity" w.r.t. expected values, since multilinearity implies $f_{\mathrm{Pol}}(\ldots, \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(r_j), \ldots) = \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(f(\ldots, r_j, \ldots))$.

$\mathcal{R}_1$ is AST since it corresponds to a symmetric random walk stopping at 0, where the number of gs denotes the current position. In contrast, $\mathcal{R}_2$ is not AST as it corresponds to a random walk where there is an equal chance of reducing the number of gs by 1 or increasing it by 2. For both $\mathcal{R}_1$ and $\mathcal{R}_2$, (8) and (9) would result in the only dependency pair $\mathsf{G} \to \{1/2 : \mathcal{O}, 1/2 : \mathsf{G}\}$ and $\mathsf{G} \to \{1/2 : \bot, 1/2 : \mathsf{G}\}$, resp. Rewriting with this DP is clearly AST, since it corresponds to a program that flips a coin until one gets head and then terminates. So the definitions (8) and (9) would not yield a sound approach for proving AST.

$\mathcal{R}_1$ and $\mathcal{R}_2$ show that the number of occurrences of the same subterm in the right-hand side r of a rule matters for AST. Thus, we now regard the *multiset* $\mathrm{MSub}_D(r)$ of all subterms of r with defined root symbol to ensure that multiple occurrences of the same subterm in r are taken into account. Moreover, instead of pairs we regard *dependency tuples* which consider all subterms with defined root in r at once. Dependency tuples were already used when adapting DPs for complexity analysis of (non-probabilistic) TRSs [37]. We now adapt them to the probabilistic setting and present a novel rewrite relation for dependency tuples.

Definition 19 (Transformation *dp*). *If* $\mathrm{MSub}_D(r) = \{t_1, \ldots, t_n\}$*, then we define* $dp(r) = \mathsf{c}_n(t_1^\#, \ldots, t_n^\#)$*. To make* $dp(r)$ *unique, we use the lexicographic ordering* $<$ *on positions where* $t_i = r|_{\pi_i}$ *and* $\pi_1 < \ldots < \pi_n$*. Here, we extend* Σ_C *by fresh* compound *constructor symbols* c_n *of arity* n *for* $n \in \mathbb{N}$*.*

When rewriting a subterm $t_i^\#$ of $\mathsf{c}_n(t_1^\#, \ldots, t_n^\#)$ with a dependency tuple, one obtains terms with nested compound symbols. To abstract from nested compound symbols and from the order of their arguments, we introduce the following normalization.

Definition 20 (Normalizing Compound Terms). *For any term* t*, its* content $cont(t)$ *is the multiset defined by* $cont(\mathsf{c}_n(t_1, \ldots, t_n)) = cont(t_1) \cup \ldots \cup cont(t_n)$ *and* $cont(t) = \{t\}$ *otherwise. For any term* t *with* $cont(t) = \{t_1, \ldots, t_n\}$*, the term* $\mathsf{c}_n(t_1, \ldots, t_n)$ *is a* normalization *of* t*. For two terms* t, t'*, we define* $t \approx t'$ *if* $cont(t) = cont(t')$*. We define* $\approx$ *on multi-distributions in a similar way: whenever* $t_j \approx t'_j$ *for all* $1 \leq j \leq k$*, then* $\{p_1 : t_1, \ldots, p_k : t_k\} \approx \{p_1 : t'_1, \ldots, p_k : t'_k\}$*.*

So for example, $\mathsf{c}_3(x, x, y)$ is a normalization of $\mathsf{c}_2(\mathsf{c}_1(x), \mathsf{c}_2(x, y))$. We do not distinguish between terms and multi-distributions that are equal w.r.t. $\approx$ and we write $\mathsf{c}_n(t_1, \ldots, t_n)$ for any term t with a compound root symbol where $cont(t) = \{t_1, \ldots, t_n\}$, i.e., we consider all such t to be normalized.

For any rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{R}$, the natural idea would be to define its *dependency tuple (DT)* as $\ell^\# \to \{p_1 : dp(r_1), \ldots, p_k : dp(r_k)\}$. Then innermost *chains* in the probabilistic setting would result from alternating a DT-step with an arbitrary number of $\mathcal{R}$-steps (using $\overset{\mathsf{i}}{\Rightarrow}^*_{\mathcal{R}}$). However, such chains would not necessarily correspond to the original rewrite sequence and thus, the resulting chain criterion would not be sound.

Example 21. Consider the PTRS $\mathcal{R}_3 = \{\mathsf{f}(\mathcal{O}) \to \{1 : \mathsf{f}(\mathsf{a})\}, \mathsf{a} \to \{1/2 : \mathsf{b}_1, 1/2 : \mathsf{b}_2\}, \mathsf{b}_1 \to \{1 : \mathcal{O}\}, \mathsf{b}_2 \to \{1 : \mathsf{f}(\mathsf{a})\}\}$. Its DTs would be $\mathcal{D}_3 = \{\mathsf{F}(\mathcal{O}) \to \{1 : \mathsf{c}_2(\mathsf{F}(\mathsf{a}), \mathsf{A})\}, \mathsf{A} \to \{1/2 : \mathsf{c}_1(\mathsf{B}_1), 1/2 : \mathsf{c}_1(\mathsf{B}_2)\}, \mathsf{B}_1 \to \{1 : \mathsf{c}_0\}, \mathsf{B}_2 \to \{1 : \mathsf{c}_2(\mathsf{F}(\mathsf{a}), \mathsf{A})\}\}$. $\mathcal{R}_3$ is not iAST, because one can extend the rewrite sequence

$$\{1{:}\mathsf{f}(\mathcal{O})\} \overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_3} \{1{:}\mathsf{f}(\mathsf{a})\} \overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_3} \{1/2{:}\mathsf{f}(\mathsf{b}_1), 1/2{:}\mathsf{f}(\mathsf{b}_2)\} \overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_3} \{1/2{:}\mathsf{f}(\mathcal{O}), 1/2{:}\mathsf{f}(\mathsf{f}(\mathsf{a}))\} \quad (11)$$

to an infinite sequence without normal forms. The resulting chain starts with

$$\begin{array}{ll} & \{\quad 1 : \mathsf{c}_1(\mathsf{F}(\mathcal{O}))\} \\ \overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{D}_3} & \{\quad 1 : \mathsf{c}_2(\mathsf{F}(\mathsf{a}), \mathsf{A})\} \\ \overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{D}_3} & \{\ 1/2 : \mathsf{c}_2(\mathsf{F}(\mathsf{a}), \mathsf{B}_1), 1/2 : \mathsf{c}_2(\mathsf{F}(\mathsf{a}), \mathsf{B}_2)\} \\ \overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_3} & \{1/4 : \mathsf{c}_2(\mathsf{F}(\mathsf{b}_1), \mathsf{B}_1), 1/4 : \underline{\mathsf{c}_2(\mathsf{F}(\mathsf{b}_2), \mathsf{B}_1)}, 1/4 : \mathsf{c}_2(\mathsf{F}(\mathsf{b}_1), \mathsf{B}_2), 1/4 : \mathsf{c}_2(\mathsf{F}(\mathsf{b}_2), \mathsf{B}_2).\} \end{array}$$

The second and third term in the last distribution do not correspond to terms in the original rewrite sequence (11). After the next $\mathcal{D}_3$-step which removes B_1, no further $\mathcal{D}_3$-step can be applied to the underlined term anymore, because b_2 cannot be rewritten to $\mathcal{O}$. Thus, the resulting chain criterion would be unsound, as every chain $(\mu_n)_{n\in\mathbb{N}}$ in this example contains such $\mathcal{D}_3$-normal forms and therefore, it is AST (i.e., $\lim_{n\to\infty} |\mu_n|_{\mathcal{D}_3} = 1$ where $|\mu_n|_{\mathcal{D}_3}$ is the probability for $\mathcal{D}_3$-normal forms in μ_n). So we have to ensure that when A is rewritten to B_1 via a DT from $\mathcal{D}_3$, then the "copy" a of the redex A is rewritten via $\mathcal{R}_3$ to the corresponding term b_1 instead of b_2. Thus, after the step with $\overset{\mathsf{i}}{\rightrightarrows}_{\mathcal{R}_3}$ we should have $\mathsf{c}_2(\mathsf{F}(\mathsf{b}_1), \mathsf{B}_1)$ and $\mathsf{c}_2(\mathsf{F}(\mathsf{b}_2), \mathsf{B}_2)$, but not $\mathsf{c}_2(\mathsf{F}(\mathsf{b}_2), \mathsf{B}_1)$ or $\mathsf{c}_2(\mathsf{F}(\mathsf{b}_1), \mathsf{B}_2)$.

Therefore, for our new adaption of DPs to the probabilistic setting, we operate on *pairs*. Instead of having a rule $\ell \to \{p_1 : r_1, \dots, p_k : r_k\}$ from $\mathcal{R}$ and its corresponding dependency tuple $\ell^\# \to \{p_1 : dp(r_1), \dots, p_k : dp(r_k)\}$ separately, we couple them together to $\langle \ell^\#, \ell\rangle \to \{p_1 : \langle dp(r_1), r_1\rangle, \dots, p_k : \langle dp(r_k), r_k\rangle\}$. This type of rewrite system is called a *probabilistic pair term rewrite system (PPTRS)*, and its rules are called *coupled dependency tuples*. Our new DP framework works on *(probabilistic) DP problems* $(\mathcal{P}, \mathcal{S})$, where $\mathcal{P}$ is a PPTRS and $\mathcal{S}$ is a PTRS.

Definition 22 (Coupled Dependency Tuple). *Let $\mathcal{R}$ be a PTRS. For every $\ell \to \mu = \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{R}$, its* coupled dependency tuple *(or simply* dependency tuple, DT*) is $\mathcal{DT}(\ell \to \mu) = \langle \ell^\#, \ell\rangle \to \{p_1 : \langle dp(r_1), r_1\rangle, \dots, p_k : \langle dp(r_k), r_k\rangle\}$. The set of all coupled dependency tuples of $\mathcal{R}$ is denoted by $\mathcal{DT}(\mathcal{R})$.*

Example 23. The following PTRS $\mathcal{R}_{\mathsf{pdiv}}$ adapts $\mathcal{R}_{\mathsf{div}}$ to the probabilistic setting.

$$\mathsf{minus}(x, \mathcal{O}) \to \{1 : x\} \quad (12) \qquad \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \to \{1 : \mathsf{minus}(x, y)\} \quad (13)$$

$$\mathsf{div}(\mathcal{O}, \mathsf{s}(y)) \to \{1 : \mathcal{O}\} \quad (14)$$

$$\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) \to \{1/2 : \mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)), 1/2 : \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))\} \quad (15)$$

In (15), we now do the actual rewrite step with a chance of $1/2$ or the terms stay the same. Our new probabilistic DP framework can prove automatically that $\mathcal{R}_{\mathsf{pdiv}}$ is iAST, while (as in the non-probabilistic setting) a direct application of polynomial interpretations via Thm. 17 fails. We get $\mathcal{DT}(\mathcal{R}_{\mathsf{pdiv}}) = \{(16), \dots, (19)\}$:

$$\langle \mathsf{M}(x, \mathcal{O}), \mathsf{minus}(x, \mathcal{O})\rangle \to \{1 : \langle \mathsf{c}_0,\ x\rangle\} \quad (16)$$

$$\langle \mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y))\rangle \to \{1 : \langle \mathsf{c}_1(\mathsf{M}(x, y)),\ \mathsf{minus}(x, y)\rangle\} \quad (17)$$

$$\langle \mathsf{D}(\mathcal{O}, \mathsf{s}(y)), \mathsf{div}(\mathcal{O}, \mathsf{s}(y))\rangle \to \{1 : \langle \mathsf{c}_0,\ \mathcal{O}\rangle\} \quad (18)$$

$$\begin{aligned} \langle \mathsf{D}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{div}(\mathsf{s}(x), \mathsf{s}(y))\rangle \to \{&1/2 : \langle \mathsf{c}_1(\mathsf{D}(\mathsf{s}(x), \mathsf{s}(y))),\ \mathsf{div}(\mathsf{s}(x), \mathsf{s}(y))\rangle, \\ &1/2 : \langle \mathsf{c}_2(\mathsf{D}(\mathsf{minus}(x, y), \mathsf{s}(y)), \mathsf{M}(x, y)),\ \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))\rangle\} \end{aligned} \quad (19)$$

Definition 24 (PPTRS, $\overset{\mathsf{i}}{\to\!\triangleright}_{\mathcal{P},\mathcal{S}}$). *Let $\mathcal{P}$ be a finite set of rules of the form $\langle \ell^{\#}, \ell\rangle \to \{p_1 : \langle d_1, r_1\rangle, \ldots, p_k : \langle d_k, r_k\rangle\}$. For every such rule, let $\mathrm{proj}_1(\mathcal{P})$ contain $\ell^{\#} \to \{p_1 : d_1, \ldots, p_k : d_k\}$ and let $\mathrm{proj}_2(\mathcal{P})$ contain $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}$. If $\mathrm{proj}_2(\mathcal{P})$ is a PTRS and $cont(d_j) \subseteq cont(dp(r_j))$ holds[6] for all $1 \le j \le k$, then $\mathcal{P}$ is a* probabilistic pair term rewrite system (PPTRS).

Let $\mathcal{S}$ be a PTRS. Then a normalized term $\mathsf{c}_n(s_1, \ldots, s_n)$ rewrites *with the PPTRS $\mathcal{P}$ to $\{p_1 : b_1, \ldots, p_k : b_k\}$ w.r.t. $\mathcal{S}$ (denoted $\overset{\mathsf{i}}{\to\!\triangleright}_{\mathcal{P},\mathcal{S}}$) if there are an $1 \le i \le n$, an $\langle \ell^{\#}, \ell\rangle \to \{p_1 : \langle d_1, r_1\rangle, \ldots, p_k : \langle d_k, r_k\rangle\} \in \mathcal{P}$, a substitution σ with $s_i = \ell^{\#}\sigma \in \mathtt{NF}_{\mathcal{S}}$, and for all $1 \le j \le k$ we have $b_j = \mathsf{c}_n(t_1^j, \ldots, t_n^j)$ where*

- *$t_i^j = d_j\sigma$ for all $1 \le j \le k$, i.e., we rewrite the term s_i using $\mathrm{proj}_1(\mathcal{P})$.*
- *For every $1 \le i' \le n$ with $i \ne i'$ we have*
  (i) *$t_{i'}^j = s_{i'}$ for all $1 \le j \le k$ or*
  (ii) *$t_{i'}^j = s_{i'}[r_j\sigma]_\tau$ for all $1 \le j \le k$,*
  if $s_{i'}|_\tau = \ell\sigma$ for some position τ and if $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{S}$.
  So $s_{i'}$ stays the same in all b_j or we can apply the rule from $\mathrm{proj}_2(\mathcal{P})$ to rewrite $s_{i'}$ in all b_j, provided that this rule is also contained in $\mathcal{S}$. Note that even if the rule is applicable, the term $s_{i'}$ can still stay the same in all b_j.

Example 25. For $\mathcal{R}_3$ from Ex. 21, the (coupled) dependency tuple for the f-rule is $\langle \mathsf{F}(\mathcal{O}), \mathsf{f}(\mathcal{O})\rangle \to \{1 : \langle \mathsf{c}_2(\mathsf{F}(\mathsf{a}), \mathsf{A}), \mathsf{f}(\mathsf{a})\rangle\}$ and the DT for the a-rule is $\langle \mathsf{A}, \mathsf{a}\rangle \to \{1/2 : \langle \mathsf{c}_1(\mathsf{B}_1), \mathsf{b}_1\rangle, 1/2 : \langle \mathsf{c}_1(\mathsf{B}_2), \mathsf{b}_2\rangle\}$. With the lifting $\overset{\mathsf{i}}{\Rightarrow}_{\mathcal{P},\mathcal{S}}$ of $\overset{\mathsf{i}}{\to\!\triangleright}_{\mathcal{P},\mathcal{S}}$, we get the following sequence which corresponds to the rewrite sequence (11) from Ex. 21.

$$\begin{aligned}\{1 : \mathsf{c}_1(\mathsf{F}(\mathcal{O}))\} &\overset{\mathsf{i}}{\Rightarrow}_{\mathcal{DT}(\mathcal{R}_3),\mathcal{R}_3} \{1 : \mathsf{c}_2(\mathsf{F}(\mathsf{a}), \mathsf{A})\} \\ &\overset{\mathsf{i}}{\Rightarrow}_{\mathcal{DT}(\mathcal{R}_3),\mathcal{R}_3} \{1/2 : \mathsf{c}_2(\mathsf{F}(\mathsf{b}_1), \mathsf{B}_1), 1/2 : \mathsf{c}_2(\mathsf{F}(\mathsf{b}_2), \mathsf{B}_2)\}\end{aligned} \tag{20}$$

So with the PPTRS, when rewriting A to B_1 in the second step, we can simultaneously rewrite the inner subterm a of $\mathsf{F}(\mathsf{a})$ to b_1 or keep a unchanged, but we cannot rewrite a to b_2. This is ensured by b_1 in the second component of $\langle \mathsf{A}, \mathsf{a}\rangle \to \{1/2 : \langle \mathsf{c}_1(\mathsf{B}_1), \mathsf{b}_1\rangle, \ldots\}$, since by Def. 24, if $s_{i'}$ contains $\ell\sigma$ at some arbitrary position τ, then one can (only) use the rule in the second component of the DT to rewrite $\ell\sigma$ (i.e., here we have $s_{i'} = \mathsf{F}(\mathsf{a})$, $s_i = \mathsf{A}$, and $s_{i'}|_\tau = \mathsf{a}$). A similar observation holds when rewriting A to B_2. Recall that with the notion of chains in Ex. 21, one *cannot simulate* every possible rewrite sequence, which leads to unsoundness. In contrast, with the notion of coupled DTs and PPTRSs, every possible rewrite sequence *can be simulated* which ensures soundness of the chain criterion. Of course, due to the ambiguity in (i) and (ii) of Def. 24, one could also create other "unsuitable" $\overset{\mathsf{i}}{\Rightarrow}_{\mathcal{DT}(\mathcal{R}_3),\mathcal{R}_3}$-sequences where a is not reduced to b_1 and b_2 in the second step, but is kept unchanged. This does not affect the soundness of the chain criterion, since every rewrite sequence of the original PTRS can be simulated by a "suitable" chain. To obtain completeness of the chain criterion, one would have to avoid such "unsuitable" sequences.

We also introduce an analogous rewrite relation for PTRSs, where we can apply the same rule simultaneously to the same subterms in a single rewrite step.

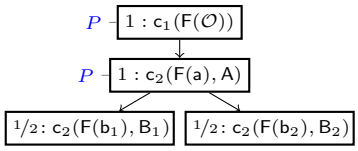
[6] The reason for $cont(d_j) \subseteq cont(dp(r_j))$ instead of $cont(d_j) = cont(dp(r_j))$ is that in this way processors can remove terms from the right-hand sides of DTs, see Thm. 32.

Definition 26 ($\xrightarrow{\mathsf{i}}_{\mathcal{S}}$). *For a PTRS $\mathcal{S}$ and a normalized term $\mathsf{c}_n(s_1,\dots,s_n)$, we define $\mathsf{c}_n(s_1,\dots,s_n) \xrightarrow{\mathsf{i}}_{\mathcal{S}} \{p_1 : b_1, \dots, p_k : b_k\}$ if there are an $1 \le i \le n$, an $\ell \to \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{S}$, a position π, a substitution σ with $s_i|_\pi = \ell\sigma$ such that every proper subterm of $\ell\sigma$ is in $\mathtt{NF}_{\mathcal{S}}$, and for all $1 \le j \le k$ we have $b_j = \mathsf{c}_n(t_1^j, \dots, t_n^j)$ where*

- *$t_i^j = s_i[r_j\sigma]_\pi$ for all $1 \le j \le k$, i.e., we rewrite the term s_i using $\mathcal{S}$.*
- *For every $1 \le i' \le n$ with $i \ne i'$ we have*
  (i) *$t_{i'}^j = s_{i'}$ for all $1 \le j \le k$ or*
  (ii) *$t_{i'}^j = s_{i'}[r_j\sigma]_\tau$ for all $1 \le j \le k$, if $s_{i'}|_\tau = \ell\sigma$ for some position τ.*

So for example, the lifting $\xrightarrow{\mathsf{i}}_{\mathcal{S}}$ of $\xrightarrow{\mathsf{i}}_{\mathcal{S}}$ for $\mathcal{S} = \mathcal{R}_3$ rewrites $\{1 : \mathsf{c}_2(\mathsf{f}(\mathsf{a}), \mathsf{a})\}$ to both $\{1/2 : \mathsf{c}_2(\mathsf{f}(\mathsf{b}_1), \mathsf{b}_1), 1/2 : \mathsf{c}_2(\mathsf{f}(\mathsf{b}_2), \mathsf{b}_2)\}$ and $\{1/2 : \mathsf{c}_2(\mathsf{f}(\mathsf{a}), \mathsf{b}_1), 1/2 : \mathsf{c}_2(\mathsf{f}(\mathsf{a}), \mathsf{b}_2)\}$.

A straightforward adaption of "chains" to the probabilistic setting using $\xrightarrow{\mathsf{i}}_{\mathcal{P},\mathcal{S}} \circ \xrightarrow{\mathsf{i}}_{\mathcal{S}}^*$ would force us to use steps with DTs from $\mathcal{P}$ at the same time for all terms in a multi-distribution. Therefore, instead we view a rewrite sequence on multi-distributions as a tree (e.g., the tree representation of the rewrite sequence (20) from Ex. 25 is on the right). Regarding the paths in this tree (which represent rewrite sequences of terms with certain probabilities) allows us to adapt the idea of chains, i.e., that one uses only finitely many $\mathcal{S}$-steps before the next step with a DT from $\mathcal{P}$.



Definition 27 (Chain Tree). *$\mathfrak{T} = (V, E, L, P)$ is an (innermost) $(\mathcal{P},\mathcal{S})$-chain tree if*

1. *$V \ne \varnothing$ is a possibly infinite set of nodes and $E \subseteq V \times V$ is a set of directed edges, such that (V, E) is a (possibly infinite) directed tree where $vE = \{w \mid (v, w) \in E\}$ is finite for every $v \in V$.*
2. *$L : V \to (0, 1] \times \mathcal{T}\left(\Sigma \uplus \Sigma^{\#}, \mathcal{V}\right)$ 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. *$P \subseteq V \setminus \mathrm{Leaf}$ (where Leaf are all leaves) is a subset of the inner nodes to indicate whether we use the PPTRS $\mathcal{P}$ or the PTRS $\mathcal{S}$ for the rewrite step. $S = V \setminus (\mathrm{Leaf} \cup P)$ are all inner nodes that are not in P. Thus, $V = P \uplus S \uplus \mathrm{Leaf}$.*
4. *For all $v \in P$: If $vE = \{w_1, \dots, w_k\}$, then $t_v \xrightarrow{\mathsf{i}}_{\mathcal{P},\mathcal{S}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \dots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$.*
5. *For all $v \in S$: If $vE = \{w_1, \dots, w_k\}$, then $t_v \xrightarrow{\mathsf{i}}_{\mathcal{S}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \dots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$.*
6. *Every infinite path in $\mathfrak{T}$ contains infinitely many nodes from P.*

Conditions 1–5 ensure that the tree represents a valid rewrite sequence and the last condition is the main property for chains.

Definition 28 ($|\mathfrak{T}|_{\mathrm{Leaf}}$, iAST). *For any innermost $(\mathcal{P},\mathcal{S})$-chain tree $\mathfrak{T}$ we define $|\mathfrak{T}|_{\mathrm{Leaf}} = \sum_{v \in \mathrm{Leaf}} p_v$. We say that $(\mathcal{P},\mathcal{S})$ is* iAST *if we have $|\mathfrak{T}|_{\mathrm{Leaf}} = 1$ for every innermost $(\mathcal{P},\mathcal{S})$-chain tree $\mathfrak{T}$.*

While we have $|\mathfrak{T}|_{\mathrm{Leaf}} = 1$ for every finite chain tree $\mathfrak{T}$, for infinite chain trees $\mathfrak{T}$ we may have $|\mathfrak{T}|_{\mathrm{Leaf}} < 1$ or even $|\mathfrak{T}|_{\mathrm{Leaf}} = 0$ if $\mathfrak{T}$ has no leaf at all.

With this new type of DTs and chain trees, we now obtain an analogous chain criterion to the non-probabilistic setting.

Theorem 29 (Chain Criterion). *A PTRS $\mathcal{R}$ is iAST if $(\mathcal{DT}(\mathcal{R}), \mathcal{R})$ is iAST.*

In contrast to the non-probabilistic case, our chain criterion as presented in the paper is *sound* but not *complete* (i.e., we do not have "iff" in Thm. 29). However, we also developed a refinement where our chain criterion is made complete by also storing the positions of the defined symbols in $dp(r)$ [27]. In this way, one can avoid "unsuitable" chain trees, as discussed at the end of Ex. 25.

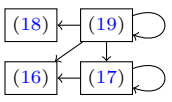
Our notion of DTs and chain trees is only suitable for *innermost* evaluation. To see this, consider the PTRSs $\mathcal{R}_1'$ and $\mathcal{R}_2'$ which both contain $\mathsf{g} \to \{1/2 : \mathcal{O}, 1/2 : \mathsf{h}(\mathsf{g})\}$, but in addition $\mathcal{R}_1'$ has the rule $\mathsf{h}(x) \to \{1 : \mathsf{f}(x, x)\}$ and $\mathcal{R}_2'$ has the rule $\mathsf{h}(x) \to \{1 : \mathsf{f}(x, x, x)\}$. Similar to $\mathcal{R}_1$ and $\mathcal{R}_2$ in (10), $\mathcal{R}_1'$ is AST while $\mathcal{R}_2'$ is not. In contrast, both $\mathcal{R}_1'$ and $\mathcal{R}_2'$ are iAST, since the innermost evaluation strategy prevents the application of the h-rule to terms containing g. Our DP framework handles $\mathcal{R}_1'$ and $\mathcal{R}_2'$ in the same way, as both have the same DT $\langle \mathsf{G}, \mathsf{g}\rangle \to \{1/2 : \langle \mathsf{c}_0, \mathcal{O}\rangle, 1/2 : \langle \mathsf{c}_2(\mathsf{H}(\mathsf{g}), \mathsf{G}), \mathsf{h}(\mathsf{g})\rangle\}$ and a DT $\langle \mathsf{H}(x), \mathsf{h}(x)\rangle \to \{1 : \langle \mathsf{c}_0, \mathsf{f}(\ldots)\rangle\}$. Even if we allowed the application of the second DT to terms of the form $\mathsf{H}(\mathsf{g})$, we would still obtain $|\mathfrak{T}|_{\text{Leaf}} = 1$ for every chain tree $\mathfrak{T}$. So a DP framework to analyze "full" instead of innermost AST would be considerably more involved.

4.2 The Probabilistic DP Framework

Now we introduce the probabilistic dependency pair framework which keeps the core ideas of the non-probabilistic framework. So instead of applying one ordering for a PTRS directly as in Thm. 17, we want to benefit from modularity. Now a *DP processor* Proc is of the form $\text{Proc}(\mathcal{P}, \mathcal{S}) = \{(\mathcal{P}_1, \mathcal{S}_1), \ldots, (\mathcal{P}_n, \mathcal{S}_n)\}$, where $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_n$ are PPTRSs and $\mathcal{S}, \mathcal{S}_1, \ldots, \mathcal{S}_n$ are PTRSs. A processor Proc is *sound* if $(\mathcal{P}, \mathcal{S})$ is iAST whenever $(\mathcal{P}_i, \mathcal{S}_i)$ is iAST for all $1 \leq i \leq n$. It is *complete* if $(\mathcal{P}_i, \mathcal{S}_i)$ is iAST for all $1 \leq i \leq n$ whenever $(\mathcal{P}, \mathcal{S})$ is iAST. In the following, we adapt the three main processors from Thm. 5 to 7 to the probabilistic setting and present two additional processors.

The (innermost) $(\mathcal{P}, \mathcal{S})$-*dependency graph* indicates which DTs from $\mathcal{P}$ can rewrite to each other using the PTRS $\mathcal{S}$. The possibility of rewriting with $\mathcal{S}$ is not related to the probabilities. Thus, for the dependency graph, we can use the *non-probabilistic variant* $\text{np}(\mathcal{S}) = \{\ell \to r_j \mid \ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{S}, 1 \leq j \leq k\}$.

Definition 30 (Dep. Graph). *The node set of the* $(\mathcal{P}, \mathcal{S})$-dependency graph *is* $\mathcal{P}$ *and there is an edge from* $\langle \ell_1^\#, \ell_1\rangle \to \{p_1 : \langle d_1, r_1\rangle, \ldots, p_k : \langle d_k, r_k\rangle\}$ *to* $\langle \ell_2^\#, \ell_2\rangle \to \ldots$ *if there are substitutions* σ_1, σ_2 *and* $t^\# \in \mathit{cont}(d_j)$ *for some* $1 \leq j \leq k$ *such that* $t^\#\sigma_1 \xrightarrow{\text{i}}{}^*_{\text{np}(\mathcal{S})} \ell_2^\#\sigma_2$ *and both* $\ell_1^\#\sigma_1$ *and* $\ell_2^\#\sigma_2$ *are in* $\mathtt{NF}_{\mathcal{S}}$.



For $\mathcal{R}_{\mathsf{pdiv}}$ from Ex. 23, the $(\mathcal{DT}(\mathcal{R}_{\mathsf{pdiv}}), \mathcal{R}_{\mathsf{pdiv}})$-dependency graph is on the side. In the non-probabilistic DP framework, every step with $\xrightarrow{\text{i}}_{\mathcal{D},\mathcal{R}}$ corresponds to an edge in the $(\mathcal{D}, \mathcal{R})$-dependency graph. Similarly, in the probabilistic setting, every path from one node of P to the next node of P in a $(\mathcal{P}, \mathcal{S})$-chain tree corresponds to an edge in the $(\mathcal{P}, \mathcal{S})$-dependency graph. Since every infinite path in a chain tree contains infinitely many nodes from P, when tracking the arguments of the compound symbols, every such path traverses a cycle of the dependency graph infinitely often. Thus, it again suffices to consider the SCCs of the dependency graph separately. So for our

example, we obtain $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{DT}(\mathcal{R}_{\mathsf{pdiv}}), \mathcal{R}_{\mathsf{pdiv}}) = \{(\{(17)\}, \mathcal{R}_{\mathsf{pdiv}}), (\{(19)\}, \mathcal{R}_{\mathsf{pdiv}})\}$. To automate the following two processors, the same over-approximation techniques as for the non-probabilistic dependency graph can be used.

Theorem 31 (Prob. Dep. Graph Processor). *For the SCCs $\mathcal{P}_1, ..., \mathcal{P}_n$ of the $(\mathcal{P}, \mathcal{S})$-dependency graph, $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{P}, \mathcal{S}) = \{(\mathcal{P}_1, \mathcal{S}), ..., (\mathcal{P}_n, \mathcal{S})\}$ is sound and complete.*

Next, we introduce a new *usable terms processor* (a similar processor was also proposed for the DTs in [37]). Since we regard dependency *tuples* instead of pairs, after applying $\mathrm{Proc}_{\mathtt{DG}}$, the right-hand sides of DTs $\langle \ell_1^\#, \ell_1 \rangle \to \dots$ might still contain terms $t^\#$ where no instance $t^\#\sigma_1$ rewrites to an instance $\ell_2^\#\sigma_2$ of a left-hand side of a DT (where we only consider instantiations such that $\ell_1^\#\sigma_1$ and $\ell_2^\#\sigma_2$ are in $\mathtt{NF}_{\mathcal{S}}$, because only such instantiations are regarded in chain trees). Then $t^\#$ can be removed from the right-hand side of the DT. For example, in the DP problem $(\{(19)\}, \mathcal{R}_{\mathsf{pdiv}})$, the only DT (19) has the left-hand side $\mathsf{D}(\mathsf{s}(x), \mathsf{s}(y))$. As the term $\mathsf{M}(x, y)$ in (19)'s right-hand side cannot "reach" $\mathsf{D}(\dots)$, the following processor removes it, i.e., $\mathrm{Proc}_{\mathtt{UT}}(\{(19)\}, \mathcal{R}_{\mathsf{pdiv}}) = \{(\{(21)\}, \mathcal{R}_{\mathsf{pdiv}})\}$, where (21) is

$$\langle \mathsf{D}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) \rangle \to \{1/2 : \langle \mathsf{c}_1(\mathsf{D}(\mathsf{s}(x), \mathsf{s}(y))), \mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) \rangle, \\ 1/2 : \langle \mathsf{c}_1(\mathsf{D}(\mathsf{minus}(x, y), \mathsf{s}(y))), \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y))) \rangle\}. \quad (21)$$

So both Thm. 31 and 32 are needed to fully simulate the dependency graph processor in the probabilistic setting, i.e., they are both necessary to guarantee that the probabilistic DP processors work analogously to the non-probabilistic ones (which in turn ensures that the probabilistic DP framework is similar in power to its non-probabilistic counterpart). This is also confirmed by our experiments in Sect. 5 which show that disabling the processor of Thm. 32 affects the power of our approach. For example, without Thm. 32, the proof that $\mathcal{R}_{\mathsf{pdiv}}$ is iAST in the probabilistic DP framework would require a more complicated polynomial interpretation. In contrast, when using both processors of Thm. 31 and 32, then one can prove iAST of $\mathcal{R}_{\mathsf{pdiv}}$ with the same polynomial interpretation that was used to prove iTerm of $\mathcal{R}_{\mathsf{div}}$ (see Ex. 36).

Theorem 32 (Usable Terms Processor). *Let $\ell_1^\#$ be a term and $(\mathcal{P}, \mathcal{S})$ be a DP problem. We call a term $t^\#$* usable *w.r.t. $\ell_1^\#$ and $(\mathcal{P}, \mathcal{S})$ if there is a $\langle \ell_2^\#, \ell_2 \rangle \to \dots \in \mathcal{P}$ and substitutions σ_1, σ_2 such that $t^\#\sigma_1 \xrightarrow{\mathrm{i}}{}^*_{\mathrm{np}(\mathcal{S})} \ell_2^\#\sigma_2$ and both $\ell_1^\#\sigma_1$ and $\ell_2^\#\sigma_2$ are in $\mathtt{NF}_{\mathcal{S}}$. If $d = \mathsf{c}_n(t_1^\#, \dots, t_n^\#)$, then $\mathcal{UT}(d)_{\ell_1^\#, \mathcal{P}, \mathcal{S}}$ denotes the term $\mathsf{c}_m(t_{i_1}^\#, \dots, t_{i_m}^\#)$, where $1 \le i_1 < \dots < i_m \le n$ are the indices of all terms $t_i^\#$ that are usable w.r.t. $\ell_1^\#$ and $(\mathcal{P}, \mathcal{S})$. The transformation that removes all non-usable terms in the right-hand sides of dependency tuples is denoted by:*

$$\mathcal{T}_{\mathtt{UT}}(\mathcal{P}, \mathcal{S}) = \{\langle \ell^\#, \ell \rangle \to \{p_1 : \langle \mathcal{UT}(d_1)_{\ell^\#, \mathcal{P}, \mathcal{S}}, r_1 \rangle, \dots, p_k : \langle \mathcal{UT}(d_k)_{\ell^\#, \mathcal{P}, \mathcal{S}}, r_k \rangle\} \\ \mid \langle \ell^\#, \ell \rangle \to \{p_1 : \langle d_1, r_1 \rangle, \dots, p_k : \langle d_k, r_k \rangle\} \in \mathcal{P}\}$$

Then $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{P}, \mathcal{S}) = \{(\mathcal{T}_{\mathtt{UT}}(\mathcal{P}, \mathcal{S}), \mathcal{S})\}$ is sound and complete.

To adapt the *usable rules processor*, we adjust the definition of usable rules such that it regards every term in the support of the distribution on the right-hand side of a rule. The usable rules processor only deletes non-usable rules from

$\mathcal{S}$, but not from $\text{proj}_2(\mathcal{P})$. This is sufficient, because according to Def. 24, rules from $\text{proj}_2(\mathcal{P})$ can only be applied if they also occur in $\mathcal{S}$.

Theorem 33 (Probabilistic Usable Rules Processor). *Let $(\mathcal{P},\mathcal{S})$ be a DP problem. For every $f \in \Sigma \uplus \Sigma^\#$ let $\text{Rules}_\mathcal{S}(f) = \{\ell \to \mu \in \mathcal{S} \mid \text{root}(\ell) = f\}$. For any term $t \in \mathcal{T}\left(\Sigma \uplus \Sigma^\#, \mathcal{V}\right)$, its* usable rules $\mathcal{U}_\mathcal{S}(t)$ *are the smallest set such that $\mathcal{U}_\mathcal{S}(x) = \varnothing$ for all $x \in \mathcal{V}$ and $\mathcal{U}_\mathcal{S}(f(t_1,\ldots,t_n)) = \text{Rules}_\mathcal{S}(f) \cup \bigcup_{i=1}^n \mathcal{U}_\mathcal{S}(t_i) \cup \bigcup_{\ell\to\mu\in\text{Rules}_\mathcal{S}(f), r\in\text{Supp}(\mu)} \mathcal{U}_\mathcal{S}(r)$. The* usable rules *for $(\mathcal{P},\mathcal{S})$ are $\mathcal{U}(\mathcal{P},\mathcal{S}) = \bigcup_{\ell^\#\to\mu\in\text{proj}_1(\mathcal{P}), d\in\text{Supp}(\mu)} \mathcal{U}_\mathcal{S}(d)$. Then $\text{Proc}_{\mathtt{UR}}(\mathcal{P},\mathcal{S}) = \{(\mathcal{P},\mathcal{U}(\mathcal{P},\mathcal{S}))\}$ is sound.*

Example 34. For the DP problem $(\{(21)\},\mathcal{R}_{\mathsf{pdiv}})$ only the minus-rules are usable and thus $\text{Proc}_{\mathtt{UR}}(\{(21)\},\mathcal{R}_{\mathsf{pdiv}}) = \{(\{(21)\},\{(12),(13)\})\}$. For $(\{(17)\},\mathcal{R}_{\mathsf{pdiv}})$ there are no usable rules at all, hence $\text{Proc}_{\mathtt{UR}}(\{(17)\},\mathcal{R}_{\mathsf{pdiv}}) = \{(\{(17)\},\varnothing)\}$.

For the *reduction pair processor*, we again restrict ourselves to multilinear polynomials and use analogous constraints as in our new criterion for the direct application of polynomial interpretations to PTRSs (Thm. 17), but adapted to DP problems $(\mathcal{P},\mathcal{S})$. Moreover, as in the original reduction pair processor of Thm. 7, the polynomials only have to be weakly monotonic. For every rule in $\mathcal{S}$ or $\text{proj}_1(\mathcal{P})$, we require that the expected value is weakly decreasing. The reduction pair processor then removes those DTs $\langle\ell^\#,\ell\rangle \to \{p_1 : \langle d_1, r_1\rangle, \ldots, p_k : \langle d_k, r_k\rangle\}$ from $\mathcal{P}$ where in addition there is at least one term d_j that is strictly decreasing. Recall that we can also rewrite with the original rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}$ from $\text{proj}_2(\mathcal{P})$, provided that it is also contained in $\mathcal{S}$. Therefore, to remove the dependency tuple, we also have to require that the rule $\ell \to r_j$ is weakly decreasing. Finally, we have to use c-*additive* interpretations (with $\mathsf{c}_{n\text{Pol}}(x_1,\ldots,x_n) = x_1 + \ldots + x_n$) to handle compound symbols and their normalization correctly.

Theorem 35 (Probabilistic Reduction Pair Processor). *Let $\text{Pol} : \mathcal{T}(\Sigma \uplus \Sigma^\#, \mathcal{V}) \to \mathbb{N}[\mathcal{V}]$ be a weakly monotonic, multilinear, and c-additive polynomial interpretation. Let $\mathcal{P} = \mathcal{P}_\geq \uplus \mathcal{P}_>$ with $\mathcal{P}_> \neq \varnothing$ such that:*

(1) For every $\ell \to \{p_1 : r_1, ..., p_k : r_k\} \in \mathcal{S}$, we have $\text{Pol}(\ell) \geq \sum_{1\leq j\leq k} p_j \cdot \text{Pol}(r_j)$.
(2) For every $\langle\ell^\#,\ell\rangle \to \{p_1 : \langle d_1, r_1\rangle, \ldots, p_k : \langle d_k, r_k\rangle\} \in \mathcal{P}$, we have $\text{Pol}(\ell^\#) \geq \sum_{1\leq j\leq k} p_j \cdot \text{Pol}(d_j)$.
(3) For every $\langle\ell^\#,\ell\rangle \to \{p_1 : \langle d_1, r_1\rangle, \ldots, p_k : \langle d_k, r_k\rangle\} \in \mathcal{P}_>$, there exists a $1 \leq j \leq k$ with $\text{Pol}(\ell^\#) > \text{Pol}(d_j)$.
If $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{S}$, then we additionally have $\text{Pol}(\ell) \geq \text{Pol}(r_j)$.

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

Example 36. The constraints of the reduction pair processor for the two DP problems from Ex. 34 are satisfied by the c-additive polynomial interpretation which again maps $\mathcal{O}$ to 0, $\mathsf{s}(x)$ to $x + 1$, and all other non-constant function symbols to the projection on their first arguments. As in the non-probabilistic case, this results in DP problems of the form $(\varnothing,\ldots)$ and subsequently, $\text{Proc}_{\mathtt{DG}}(\varnothing,\ldots)$ yields $\varnothing$. By the soundness of all processors, this proves that $\mathcal{R}_{\mathsf{pdiv}}$ is iAST.

So with the new probabilistic DP framework, the proof that $\mathcal{R}_{\mathsf{pdiv}}$ is iAST is analogous to the proof that $\mathcal{R}_{\mathsf{div}}$ is iTerm in the original DP framework (the

proofs even use the same polynomial interpretation in the respective reduction pair processors). This indicates that our novel framework for PTRSs has the same essential concepts and advantages as the original DP framework for TRSs. This is different from our previous adaption of dependency pairs for complexity analysis of TRSs, which also relies on dependency tuples [37]. There, the power is considerably restricted, because one does not have full modularity as one cannot decompose the proof according to the SCCs of the dependency graph.

In proofs with the probabilistic DP framework, one may obtain DP problems $(\mathcal{P}, \mathcal{S})$ that have a non-probabilistic structure (i.e., every DT in $\mathcal{P}$ has the form $\langle \ell^\#, \ell \rangle \to \{1 : \langle d, r \rangle\}$ and every rule in $\mathcal{S}$ has the form $\ell' \to \{1 : r'\}$). We now introduce a processor that allows us to switch to the original non-probabilistic DP framework for such (sub-)problems. This is advantageous, because due to the use of dependency *tuples* instead of pairs in $\mathcal{P}$, in general the constraints of the probabilistic reduction pair processor of Thm. 35 are harder than the ones of the reduction pair processor of Thm. 7. Moreover, Thm. 7 is not restricted to multilinear polynomial interpretations and the original DP framework has many additional processors that have not yet been adapted to the probabilistic setting.

Theorem 37 (Probability Removal Processor). *Let $(\mathcal{P}, \mathcal{S})$ be a probabilistic DP problem where every DT in $\mathcal{P}$ has the form $\langle \ell^\#, \ell \rangle \to \{1 : \langle d, r \rangle\}$ and every rule in $\mathcal{S}$ has the form $\ell' \to \{1 : r'\}$. Let $\mathrm{np}(\mathcal{P}) = \{\ell^\# \to t^\# \mid \ell^\# \to \{1 : d\} \in \mathrm{proj}_1(\mathcal{P}), t^\# \in \mathit{cont}(d)\}$. Then $(\mathcal{P}, \mathcal{S})$ is iAST iff the non-probabilistic DP problem $(\mathrm{np}(\mathcal{P}), \mathrm{np}(\mathcal{S}))$ is iTerm. So if $(\mathrm{np}(\mathcal{P}), \mathrm{np}(\mathcal{S}))$ is iTerm, then the processor $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{P}, \mathcal{S}) = \varnothing$ is sound and complete.*

5 Conclusion and Evaluation

Starting with a new "direct" technique to prove almost-sure termination of probabilistic TRSs (Thm. 17), we presented the first adaption of the dependency pair framework to the probabilistic setting in order to prove innermost AST automatically. This is not at all obvious, since most straightforward ideas for such an adaption are unsound (as discussed in Sect. 4.1). So the challenge was to find a suitable definition of dependency pairs (resp. tuples) and chains (resp. chain trees) such that one can define DP processors which are sound and work analogously to the non-probabilistic setting (in order to obtain a framework which is similar in power to the non-probabilistic one). While the soundness proofs for our new processors are much more involved than in the non-probabilistic case, the new processors themselves are quite analogous to their non-probabilistic counterparts and thus, adapting an existing implementation of the non-probabilistic DP framework to the probabilistic one does not require much effort.

We implemented our contributions in our termination prover AProVE, which yields the first tool to prove almost-sure innermost termination of PTRSs on arbitrary data structures (including PTRSs that are not PAST). In our experiments, we compared the direct application of polynomials for proving AST (via our new Thm. 17) with the probabilistic DP framework. We evaluated AProVE on a collection of 67 PTRSs which includes many typical probabilistic algorithms.

For example, it contains the following PTRS $\mathcal{R}_{\mathsf{qs}}$ for probabilistic quicksort.

$$\begin{aligned}
\mathsf{rotate}(\mathsf{cons}(x, xs)) &\to \{1/2 : \mathsf{cons}(x, xs), 1/2 : \mathsf{rotate}(\mathsf{app}(xs, \mathsf{cons}(x, \mathsf{nil})))\}\\
\mathsf{qs}(\mathsf{nil}) &\to \{1 : \mathsf{nil}\}\\
\mathsf{qs}(\mathsf{cons}(x, xs)) &\to \{1 : \mathsf{qsHelp}(\mathsf{rotate}(\mathsf{cons}(x, xs)))\}\\
\mathsf{qsHelp}(\mathsf{cons}(x, xs)) &\to \{1 : \mathsf{app}(\mathsf{qs}(\mathsf{low}(x, xs)), \mathsf{cons}(x, \mathsf{qs}(\mathsf{high}(x, xs))))\}
\end{aligned}$$

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 a random pivot element for the recursive call of quicksort. In addition to the rules above, $\mathcal{R}_{\mathsf{qs}}$ contains rules for list concatenation (app), and rules such that $\mathsf{low}(x, xs)$ (resp. $\mathsf{high}(x, xs)$) returns all elements of the list xs that are smaller (resp. greater or equal) than x, see [28]. Using the probabilistic DP framework, AProVE can prove iAST of $\mathcal{R}_{\mathsf{qs}}$ and many other typical programs.

61 of the 67 examples in our collection are iAST and AProVE can prove iAST for 53 (87%) of them. Here, the DP framework proves iAST for 51 examples and the direct application of polynomial interpretations via Thm. 17 succeeds for 27 examples. (In contrast, proving PAST via the direct application of polynomial interpretations as in [3] only works for 22 examples.) The average runtime of AProVE per example was 2.88 s (where no example took longer than 8 s). So our experiments indicate that the power of the DP framework can now also be used for probabilistic TRSs.

We also performed experiments where we disabled individual processors of the probabilistic DP framework. More precisely, we disabled either the usable terms processor (Thm. 32), both the dependency graph and the usable terms processor (Thm. 31 and 32), or all processors except the reduction pair processor of Thm. 35. Our experiments show that disabling processors indeed affects the power of the approach, in particular for larger examples with several defined symbols (e.g., then AProVE cannot prove iAST of $\mathcal{R}_{\mathsf{qs}}$ anymore). So all of our processors are needed to obtain a powerful technique for termination analysis of PTRSs.

Due to the use of dependency *tuples* instead of pairs, the probabilistic DP framework does not (yet) subsume the direct application of polynomials completely (two examples in our collection can only be proved by the latter, see [28]). Therefore, currently AProVE uses the direct approach of Thm. 17 in addition to the probabilistic DP framework. In future work, we will adapt further processors of the original DP framework to the probabilistic setting, which will also allow us to integrate the direct approach of Thm. 17 into the probabilistic DP framework in a modular way. Moreover, we will develop processors to prove AST of full (instead of innermost) rewriting. Further work may also include processors to disprove (i)AST and possible extensions to analyze PAST and expected runtimes as well. Finally, one could also modify the formalism of PTRSs in order to allow non-constant probabilities which depend on the sizes of terms.

For details on our experiments and for instructions on how to run our implementation in AProVE via its *web interface* or locally, we refer to https://aprove-developers.github.io/ProbabilisticTermRewriting/.

Acknowledgements. We are grateful to Marcel Hark, Dominik Meier, and Florian Frohn for help and advice.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] O. Bournez and C. Kirchner. "Probabilistic Rewrite Strategies. Applications to ELAN". In: *Proc. RTA '02*. LNCS 2378. 2002, pp. 252–266. DOI: 10.1007/3-540-45610-4_18.

[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] M. Brockschmidt, R. Musiol, C. Otto, and J. Giesl. "Automated Termination Proofs for Java Programs with Cyclic Data". In: *Proc. CAV '12*. LNCS 7358. 2012, pp. 105–122. DOI: 10.1007/978-3-642-31424-7_13.

[11] 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.

[12] 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.

[13] N. Dershowitz. "Orderings for Term-Rewriting Systems". In: *Theor. Comput. Sci.* 17 (1982), pp. 279–301. DOI: 10.1016/0304-3975(82)90026-3.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. "Automated Termination Proofs for Haskell by Term Rewriting". In: *ACM Trans. Program. Lang. Syst.* 33.2 (2011). DOI: 10.1145/1890028.1890030.

[19] J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, and C. Fuhs. "Symbolic Evaluation Graphs and Term Rewriting: A General Methodology for Analyzing Logic Programs". In: *Proc. PPDP '12*. 2012, pp. 1–12. DOI: 10.1145/2370776.2370778.

[20] 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.

[21] 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.

[22] R. Gutiérrez and S. Lucas. "MU-TERM: Verify Termination Properties Automatically (System Description)". In: *Proc. IJCAR '20*. LNCS 12167. 2020, pp. 436–447. DOI: 10.1007/978-3-030-51054-1_28.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] J.-C. Kassing. "Using Dependency Pairs for Proving Almost-Sure Termination of Probabilistic Term Rewriting". MA thesis. RWTH Aachen University, 2022. URL: https://verify.rwth-aachen.de/da/Kassing-Masterthesis.pdf.

[28] J.-C. Kassing and J. Giesl. "Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs". In: *CoRR* abs/2305.11741 (2023). DOI: 10.48550/arXiv.2305.11741.

[29] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. "Tyrolean Termination Tool 2". In: *Proc. RTA '09*. LNCS 5595. 2009, pp. 295–304. DOI: 10.1007/978-3-642-02348-4_21.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] G. Moser and M. Schaper. "From Jinja Bytecode to Term Rewriting: A Complexity Reflecting Transformation". In: *Inf. Comput.* 261 (2018), pp. 116–143. DOI: 10.1016/j.ic.2018.05.007.

[36] 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.

[37] L. Noschinski, F. Emmes, and J. Giesl. "Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs". In: *J. Autom. Reason.* 51 (2013), pp. 27–56. DOI: 10.1007/978-3-642-22438-6_32.

[38] C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. "Automated Termination Analysis of Java Bytecode by Term Rewriting". In: *Proc. RTA '10*. LIPIcs 6. 2010, pp. 259–276. DOI: 10.4230/LIPIcs.RTA.2010.259.

[39] 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.

[40] A. Yamada, K. Kusakari, and T. Sakabe. "Nagoya Termination Tool". In: *Proc. RTA-TLCA '14*. LNCS 8560. 2014, pp. 466–475. DOI: 10.1007/978-3-319-08918-8_32.