Annotated Dependency Pairs for Full Almost-Sure Termination of Probabilistic Term Rewriting*

Jan-Christoph Kassing and Jürgen Giesl

RWTH Aachen University, Aachen, Germany
{kassing,giesl}@cs.rwth-aachen.de

Abstract. Dependency pairs (DPs) are one of the most powerful techniques for automated termination analysis of term rewrite systems. Recently, we adapted the DP framework to the probabilistic setting to prove almost-sure termination (AST) via annotated DPs (ADPs). However, this adaption only handled AST w.r.t. the *innermost* evaluation strategy. In this paper, we improve the ADP framework to prove AST for *full* rewriting. Moreover, we refine the framework for rewrite sequences that start with *basic* terms containing a single defined function symbol. We implemented and evaluated the new framework in our tool AProVE.

1 Introduction

Term rewrite systems (TRSs) are used for automated termination analysis of many programming languages. There exist numerous powerful tools to prove termination of TRSs, e.g., [19,34,49,22]. Dependency pairs (DPs, see e.g., [2,17,18,23,24]) are one of the main concepts used in all these tools.

In [9,10,4,14], TRSs were extended to the probabilistic setting. Probabilistic programs describe randomized algorithms and probability distributions, with applications in many areas, see, e.g., [21]. Instead of only considering ordinary termination (i.e., absence of infinite evaluation sequences), in the probabilistic setting one is interested in *almost-sure termination* (AST), where infinite evaluation sequences are allowed, but their probability is 0. A strictly stronger notion is *positive* AST (PAST), which requires that the expected runtime is finite [10,43].

There exist numerous techniques to prove (P)AST of imperative programs on numbers (like the probabilistic guarded command language pGCL [35,38]), e.g., [26,39,40,1,11,27,15,20,25,41,5,42]. In contrast, *probabilistic TRSs* (PTRSs) are especially suitable for modeling and analyzing functional programs and algorithms operating on (user-defined) data structures like lists, trees, etc. Up to now, there exist only few automatic approaches to analyze (P)AST of probabilistic programs with complex non-tail recursive structure [8,12,13]. The approaches that are suitable for algorithms on recursive data structures [48,37,7] 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

* funded by the DFG Research Training Group 2236 UnRAVeL

automatic termination analysis for arbitrary PTRSs.

For PTRSs, orderings based on interpretations were adapted to prove `PAST` of *full* rewriting (w.r.t. any evaluation strategy) in [4], and we presented a related technique to prove `AST` in [28]. However, already for non-probabilistic TRSs, such a direct application of orderings is limited in power. To obtain a powerful approach, one should combine orderings in a modular way, as in the DP framework.

Indeed, based on initial work in [28], in [31] we adapted the DP framework to the probabilistic setting to prove innermost AST (`iAST`) of PTRSs via so-called *annotated dependency pairs* (ADPs). However, this adaption is restricted to *innermost* rewriting, i.e., one only considers sequences that rewrite at innermost positions of terms. Already for non-probabilistic TRSs, innermost termination is easier to prove than *full* termination, and this remains true in the probabilistic setting.

In the current paper, we adapt the definition of ADPs to use them for any evaluation strategy. As our running example, we transform Alg. 1 on the right (written in `pGCL`) into an equivalent PTRS and show how our new ADP framework proves `AST`. Here, $\oplus_{1/2}$ denotes probabilistic choice, and $\square$ denotes demonic non-determinism. Note that there are proof rules (e.g., [39]) and tools (e.g., [41]) that can prove `AST` for both loops of Alg. 1 individually, and hence for the whole algorithm. Moreover, the tool `Caesar` [44] can prove `AST` if one provides super-martingales for the two loops. However, to the best of our knowledge there exist no automatic techniques to handle similar algorithms on arbitrary algebraic data structures, i.e., (non-deterministic) algorithms that first create a random data object y in a first loop and then access or modify it in a second loop, whereas this is possible with our new ADP framework.[1] Note that while Alg. 1 is `AST`, its expected runtime is infinite, i.e., it is not `PAST`.[2]

Algorithm 1:

$x \leftarrow 0$
while $x = 0$ **do**
  {
    $x \leftarrow 0 \oplus_{1/2} x \leftarrow 1;$
    $y \leftarrow 2 \cdot y;$
  }$\square${
    $x \leftarrow 0 \oplus_{1/3} x \leftarrow 1;$
    $y \leftarrow 3 \cdot y;$
  }
while $y > 0$ **do**
  $y \leftarrow y - 1;$

In [30], we developed the first criteria for classes of PTRSs where `iAST` implies `AST`. So for PTRSs from these classes, one can use our ADP framework for `iAST` in order to conclude `AST`. However, these criteria exclude non-probabilistic non-determinism, i.e., they require that the rules of the PTRS must be non-overlapping. In addition, they impose linearity restrictions on both sides of the rewrite rules. In contrast, our novel ADP framework can be applied to overlapping PTRSs and it also weakens the linearity requirements considerably.

We start with preliminaries on (probabilistic) term rewriting in Sect. 2. In Sect. 3 we recapitulate annotated dependency pairs for innermost `AST` [31], explain why they cannot prove `AST` for *full* rewriting, and adapt them accordingly. We present the probabilistic ADP framework in Sect. 4, illustrate its main processors, and show how to adapt them from `iAST` to `AST`. Finally, in Sect. 5 we evaluate the

[1] Such examples can be found in our benchmark set, see Sect. 5 and App. A.

[2] This already holds for the program where only the first possibility of the first **while**-loop is considered (i.e., where y is always doubled in its body). Then for the initial value $y = 1$, the expected number of iterations of the second **while**-loop which decrements y is $\frac{1}{2} \cdot 2 + \frac{1}{4} \cdot 4 + \frac{1}{8} \cdot 8 + \ldots = 1 + 1 + 1 + \ldots = \infty$.

implementation of our approach in the tool AProVE [19]. We refer to App. A to illustrate our approach on examples with non-numerical data structures like lists or trees, and to App. B for all proofs.

2 Preliminaries

Sect. 2.1 to 2.3 recapitulate classic [6] and probabilistic [4,9,10,28] term rewriting, and results on PTRSs where iAST and AST are equivalent, respectively.

2.1 Term Rewriting

We regard a (finite) signature $\Sigma = \biguplus_{n \in \mathbb{N}} \Sigma_n$ and a set of variables $\mathcal{V}$. The set of *terms* $\mathcal{T}(\Sigma, \mathcal{V})$ (or simply $\mathcal{T}$) is the smallest set with $\mathcal{V} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$, and if $f \in \Sigma_n$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$ then $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$. We say that s is a *subterm* of t (denoted $s \trianglelefteq t$) if $s = t$, or $t = f(t_1, \ldots, t_n)$ and $s \trianglelefteq t_i$ for some $1 \leq i \leq n$. It is a *proper* subterm (denoted $s \triangleleft t$) if $s \trianglelefteq t$ and $s \neq t$. A *substitution* is a function $\sigma : \mathcal{V} \to \mathcal{T}$ with $\sigma(x) = x$ for all but finitely many $x \in \mathcal{V}$. We often write $x\sigma$ instead of $\sigma(x)$. Substitutions can also be applied to terms: If $t = f(t_1, \ldots, t_n) \in \mathcal{T}$ then $t\sigma = f(t_1\sigma, \ldots, t_n\sigma)$. For a term $t \in \mathcal{T}$, the set of *positions* $\mathrm{Pos}(t)$ is the smallest subset of $\mathbb{N}^*$ satisfying $\varepsilon \in \mathrm{Pos}(t)$, and if $t = f(t_1, \ldots, t_n)$ then for all $1 \leq i \leq n$ and all $\pi \in \mathrm{Pos}(t_i)$ we have $i.\pi \in \mathrm{Pos}(t)$. If $\pi \in \mathrm{Pos}(t)$ then $t|_\pi$ denotes the subterm at position π, where we have $t|_\varepsilon = t$ for the *root position* ε and $f(t_1, \ldots, t_n)|_{i.\pi} = t_i|_\pi$. The *root symbol* at position ε is also denoted by $\mathrm{root}(t)$. If $r \in \mathcal{T}$ and $\pi \in \mathrm{Pos}(t)$ then $t[r]_\pi$ denotes the term that results from replacing the subterm $t|_\pi$ with the term r.

A *rewrite rule* $\ell \to r \in \mathcal{T} \times \mathcal{T}$ is a pair with $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$. A *term rewrite system* (TRS) is a (finite) set of rewrite rules. For example, $\mathcal{R}_{\mathsf{d}}$ with the only rule $\mathsf{d}(x) \to \mathsf{c}(x, x)$ is a TRS. A TRS $\mathcal{R}$ induces a *rewrite relation* $\to_{\mathcal{R}} \subseteq \mathcal{T} \times \mathcal{T}$ where $s \to_{\mathcal{R}} t$ holds if there are an $\ell \to r \in \mathcal{R}$, a substitution σ, and a $\pi \in \mathrm{Pos}(s)$ such that $s|_\pi = \ell\sigma$ and $t = s[r\sigma]_\pi$. A term s is in *normal form* w.r.t. $\mathcal{R}$ (denoted $s \in \mathtt{NF}_{\mathcal{R}}$) if there is no term t with $s \to_{\mathcal{R}} t$, and in *argument normal form* w.r.t. $\mathcal{R}$ (denoted $s \in \mathtt{ANF}_{\mathcal{R}}$) if $s' \in \mathtt{NF}_{\mathcal{R}}$ for all proper subterms $s' \triangleleft s$. A rewrite step $s \to_{\mathcal{R}} t$ is *innermost* (denoted $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$) if the used *redex* $\ell\sigma$ is in argument normal form. For example, $\mathsf{d}(\mathsf{d}(\mathsf{0})) \xrightarrow{\mathsf{i}}_{\mathcal{R}_{\mathsf{d}}} \mathsf{d}(\mathsf{c}(\mathsf{0}, \mathsf{0}))$, but $\mathsf{d}(\mathsf{d}(\mathsf{0})) \to_{\mathcal{R}_{\mathsf{d}}} \mathsf{c}(\mathsf{d}(\mathsf{0}), \mathsf{d}(\mathsf{0}))$ is not an innermost step. A TRS $\mathcal{R}$ is *(innermost) terminating* if $(\xrightarrow{\mathsf{i}}_{\mathcal{R}}) \to_{\mathcal{R}}$ is well founded.

Two rules $\ell_1 \to r_1, \ell_2 \to r_2 \in \mathcal{R}$ with renamed variables such that $\mathcal{V}(\ell_1) \cap \mathcal{V}(\ell_2) = \varnothing$ are *overlapping* if there exists a non-variable position π of ℓ_1 such that $\ell_1|_\pi$ and ℓ_2 are unifiable, i.e., there exists a substitution σ such that $\ell_1|_\pi\sigma = \ell_2\sigma$. If $(\ell_1 \to r_1) = (\ell_2 \to r_2)$, then we require that $\pi \neq \varepsilon$. $\mathcal{R}$ is *non-overlapping* if it has no overlapping rules (e.g., $\mathcal{R}_{\mathsf{d}}$ is non-overlapping). A TRS is *left-linear* (*right-linear*) if every variable occurs at most once in the left-hand side (right-hand side) of a rule. Finally, a TRS is *non-duplicating* if for every rule, every variable occurs at most as often in the right-hand side as in the left-hand side. As an example, $\mathcal{R}_{\mathsf{d}}$ is left-linear, not right-linear, and hence duplicating.

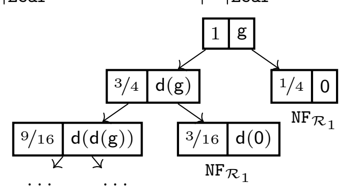
2.2 Probabilistic Term Rewriting

In contrast to TRSs, a *probabilistic TRS* (PTRS) [4,9,10,28] has (finite) multi-distributions on the right-hand sides of its rewrite rules. 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 $\mathrm{Supp}(\mu) = \{a \mid (p:a) \in \mu \text{ for some } p\}$. A *probabilistic rewrite rule* $\ell \to \mu \in \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ is a pair such that $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ for every $r \in \mathrm{Supp}(\mu)$. Examples for probabilistic rewrite rules are

$$\mathsf{g} \to \{{}^{3}\!/\!_{4} : \mathsf{d}(\mathsf{g}),\ {}^{1}\!/\!_{4} : \mathsf{0}\} \quad (1) \qquad \mathsf{d}(x) \to \{1 : \mathsf{c}(x,x)\} \quad (2)$$
$$\mathsf{d}(\mathsf{d}(x)) \to \{1 : \mathsf{c}(x,\mathsf{g})\} \quad (3) \qquad \mathsf{d}(x) \to \{1 : \mathsf{0}\} \quad (4)$$

A *probabilistic TRS* is a finite set $\mathcal{R}$ of probabilistic rewrite rules, e.g., $\mathcal{R}_1 = \{(1)\}$, $\mathcal{R}_2 = \{(1),(2)\}$, or $\mathcal{R}_3 = \{(1),(3),(4)\}$. Similar to TRSs, a PTRS $\mathcal{R}$ induces a *rewrite relation* $\to_{\mathcal{R}} \subseteq \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ where $s \to_{\mathcal{R}} \{p_1 : t_1, \ldots, p_k : t_k\}$ if there are an $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{R}$, a substitution σ, and a $\pi \in \mathrm{Pos}(s)$ such that $s|_\pi = \ell\sigma$ and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$. The step is *innermost* (denoted $s \xrightarrow{\mathrm{i}}_{\mathcal{R}} \{p_1 : r_1, \ldots, p_k : r_k\}$) if $\ell\sigma \in \mathtt{ANF}_{\mathcal{R}}$. So the PTRS $\mathcal{R}_1$ can be interpreted as a biased coin flip that terminates in each step with a chance of ${}^{1}\!/\!_{4}$.

To track all possible rewrite sequences (up to non-determinism) with their corresponding probabilities, as in [31] we *lift* $\to_{\mathcal{R}}$ to *rewrite sequence trees (RSTs)*. The nodes v of an $\mathcal{R}$-RST are labeled by pairs $(p_v : t_v)$ of a probability p_v and a term t_v, where the root is always labeled with the probability 1. For each node v with the successors $w_1, \ldots, w_k$, the edge relation represents a probabilistic rewrite step, i.e., $t_v \to_{\mathcal{R}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \ldots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$. An $\mathcal{R}$-RST is an *innermost* $\mathcal{R}$-RST if the edge relation represents only innermost steps. For an $\mathcal{R}$-RST $\mathfrak{T}$ we define $|\mathfrak{T}|_{\mathtt{Leaf}} = \sum_{v \in \mathtt{Leaf}} p_v$, where $\mathtt{Leaf}$ is the set of all its leaves, and we say that a PTRS $\mathcal{R}$ is *almost-surely terminating (*`AST`*)* (*almost-surely innermost terminating (*`iAST`*)*) if $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$ holds for all $\mathcal{R}$-RSTs (innermost $\mathcal{R}$-RSTs) $\mathfrak{T}$. While $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$ for every finite RST $\mathfrak{T}$, for infinite RSTs $\mathfrak{T}$ we may have $|\mathfrak{T}|_{\mathtt{Leaf}} < 1$ or even $|\mathfrak{T}|_{\mathtt{Leaf}} = 0$ if $\mathfrak{T}$ has no leaf at all. This notion of `AST` is equivalent to the ones in [10,4,28], where `AST` is defined via a lifting of $\to_{\mathcal{R}}$ to multisets or via stochastic processes. The infinite $\mathcal{R}_1$-RST $\mathfrak{T}$ on the side has $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$. As this holds for all $\mathcal{R}_1$-RSTs, $\mathcal{R}_1$ is `AST`.



Example 1. $\mathcal{R}_2$ is not `AST`. If we always apply (2) directly after (1), this corresponds to the rule $\mathsf{g} \to \{{}^{3}\!/\!_{4} : \mathsf{c}(\mathsf{g},\mathsf{g}), {}^{1}\!/\!_{4} : \mathsf{0}\}$, which represents a random walk on the number of g's in a term biased towards non-termination (as $\frac{3}{4} > \frac{1}{4}$). $\mathcal{R}_3$ is not `AST` either, because if we always apply (3) after two applications of (1), this corresponds to $\mathsf{g} \to \{{}^{9}\!/\!_{16} : \mathsf{c}(\mathsf{g},\mathsf{g}),\ {}^{3}\!/\!_{16} : \mathsf{0},\ {}^{1}\!/\!_{4} : \mathsf{0}\}$, which is also biased towards non-termination (as $\frac{9}{16} > \frac{3}{16} + \frac{1}{4}$).

However, in innermost evaluations, the d-rule (2) can only duplicate normal forms, and hence $\mathcal{R}_2$ is `iAST`, see [30]. $\mathcal{R}_3$ is `iAST` as well, as (3) is not applicable in innermost evaluations. For both $\mathcal{R}_2$ and $\mathcal{R}_3$, `iAST` can also be proved automatically by our implementation of the ADP framework for `iAST` in AProVE [28,31].

Example 2. The following PTRS $\mathcal{R}_{\mathsf{alg}}$ corresponds to Alg. 1. Here, the non-determinism is modeled by the non-deterministic choice between the overlapping rules (5) and (6). In Sect. 4, we will prove that $\mathcal{R}_{\mathsf{alg}}$ is AST via our new notion of ADPs.

$$\mathsf{loop1}(y) \to \{1/2 : \mathsf{loop1}(\mathsf{double}(y)),\ 1/2 : \mathsf{loop2}(\mathsf{double}(y))\} \quad (5)$$

$$\mathsf{loop1}(y) \to \{1/3 : \mathsf{loop1}(\mathsf{triple}(y)),\ 2/3 : \mathsf{loop2}(\mathsf{triple}(y))\} \quad (6)$$

$$\mathsf{loop2}(\mathsf{s}(y)) \to \{1 : \mathsf{loop2}(y)\}$$

$$\mathsf{double}(\mathsf{s}(y)) \to \{1 : \mathsf{s}(\mathsf{s}(\mathsf{double}(y)))\}$$

$$\mathsf{double}(\mathsf{0}) \to \{1 : \mathsf{0}\}$$

$$\mathsf{triple}(\mathsf{s}(y)) \to \{1 : \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{triple}(y))))\}$$

$$\mathsf{triple}(\mathsf{0}) \to \{1 : \mathsf{0}\}$$

A PTRS $\mathcal{R}$ is *right-linear* iff the TRS $\{\ell \to r \mid \ell \to \mu \in \mathcal{R}, r \in \mathrm{Supp}(\mu)\}$ is right-linear. Left-linearity and being non-overlapping can be lifted to PTRSs directly, as their rules also have just a single term on their left-hand sides.

For a PTRS $\mathcal{R}$, we decompose its signature $\Sigma = \mathcal{C} \uplus \mathcal{D}$ such that $f \in \mathcal{D}$ iff $f = \mathrm{root}(\ell)$ for some $\ell \to \mu \in \mathcal{R}$. The symbols in $\mathcal{C}$ and $\mathcal{D}$ are called *constructors* and *defined symbols*, respectively. For $\mathcal{R}_2$ we have $\mathcal{C} = \{\mathsf{c}, \mathsf{0}\}$ and $\mathcal{D} = \{\mathsf{g}, \mathsf{d}\}$. A term $t \in \mathcal{T}$ is *basic* if $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{D}$ and $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ for all $1 \leq i \leq n$. So a basic term represents an algorithm f applied to arguments t_i which only represent data and do not contain executable functions.

Finally, we define *spareness* [16], which prevents the duplication of redexes if the evaluation starts with a basic term. A rewrite step $\ell\sigma \to_{\mathcal{R}} \mu\sigma$ is *spare* if $\sigma(x) \in \mathtt{NF}_{\mathcal{R}}$ for every $x \in \mathcal{V}$ that occurs more than once in some $r \in \mathrm{Supp}(\mu)$. An $\mathcal{R}$-RST is spare if all rewrite steps corresponding to its edges are spare. A PTRS $\mathcal{R}$ is spare if each $\mathcal{R}$-RST that starts with $\{1 : t\}$ for a basic term t is spare. So for example, $\mathcal{R}_2$ is not spare, because the basic term g starts a rewrite sequence where the redex g is duplicated by Rule (2). Computable sufficient conditions for spareness were presented in [16].

2.3 Existing Techniques for Proving Full AST

In order to prove AST automatically, one can either use orderings directly on the whole PTRS [4,28], or check whether the PTRS $\mathcal{R}$ belongs to a class where it is known that $\mathcal{R}$ is AST iff $\mathcal{R}$ is iAST. Then, it suffices to analyze iAST, and to this end, one can use the existing ADP framework [31]. In [30], we introduced the following first criteria for classes of PTRSs where iAST is equivalent to AST.

Theorem 3 (From iAST to AST (1) [30]). *If a PTRS $\mathcal{R}$ is non-overlapping, left-linear, and right-linear, then $\mathcal{R}$ is* AST *iff $\mathcal{R}$ is* iAST.

Moreover, if one restricts the analysis to basic start terms, then we can weaken right-linearity to spareness. In the following, "b(i)AST" (basic (i)AST) means that one only considers rewrite sequences that start with $\{1 : t\}$ for basic terms t.

Theorem 4 (From iAST to AST (2) [30]). *If a PTRS $\mathcal{R}$ is non-overlapping, left-linear, and spare, then $\mathcal{R}$ is* bAST *iff $\mathcal{R}$ is* biAST.

Since iAST obviously implies biAST, under the conditions of Thm. 4 it suffices to analyze iAST to prove bAST.[3] In addition to Thm. 3 and 4, [30] presented another criterion to weaken the left-linearity condition. We do not recapitulate it here, as our novel approach in Sect. 3 and 4 will not require left-linearity anyway.

$\mathcal{R}_{\mathsf{alg}}$ from Ex. 2 is left- and right-linear, but overlapping. Hence, $\mathcal{R}_{\mathsf{alg}}$ does not belong to any known class of PTRSs where iAST is equivalent to AST. Thus, to prove AST of such PTRSs, one needs a new approach, e.g., as in Sect. 3 and 4.

3 Probabilistic Annotated Dependency Pairs

In Sect. 3.1 we recapitulate annotated dependency pairs (ADPs) [31] which adapt DPs in order to prove iAST. Then in Sect. 3.2 we introduce our novel adaption of ADPs for full probabilistic rewriting w.r.t. any evaluation strategy.

3.1 ADPs and Chains - Innermost Rewriting

Instead of comparing left- and right-hand sides of rules to prove termination, ADPs only consider the subterms with defined root symbols in the right-hand sides, as only these subterms might be evaluated further. In the probabilistic setting, we use annotations to mark which subterms in right-hand sides could potentially lead to a non-(i)AST evaluation. For every $f \in \mathcal{D}$, we introduce a fresh *annotated symbol* $f^\#$ of the same arity. Let $\mathcal{D}^\#$ denote the set of all annotated symbols, $\Sigma^\# = \mathcal{D}^\# \uplus \Sigma$, and $\mathcal{T}^\# = \mathcal{T}\left(\Sigma^\#, \mathcal{V}\right)$. To ease readability, we often use capital letters like F instead of $f^\#$. For any $t = f(t_1, \ldots, t_n) \in \mathcal{T}$ with $f \in \mathcal{D}$, let $t^\# = f^\#(t_1, \ldots, t_n)$. For $t \in \mathcal{T}^\#$ and $\mathcal{X} \subseteq \Sigma^\# \cup \mathcal{V}$, let $\mathrm{Pos}_{\mathcal{X}}(t)$ be all positions of t with symbols or variables from $\mathcal{X}$. 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 $\#_\varnothing(t)$. Moreover, let $\flat^\uparrow_\pi(t)$ result from removing all annotations from t that are strictly above the position π. So for $\mathcal{R}_2$, we have $\#_{\{1\}}(\mathsf{d}(\mathsf{g})) = \#_{\{1\}}(\mathsf{D}(\mathsf{G})) = \mathsf{d}(\mathsf{G})$, $\flat(\mathsf{D}(\mathsf{G})) = \mathsf{d}(\mathsf{g})$, and $\flat^\uparrow_1(\mathsf{D}(\mathsf{G})) = \mathsf{d}(\mathsf{G})$. To transform the rules of a PTRS into ADPs, initially we annotate all $f \in \mathcal{D}$ occurring in right-hand sides.

Every ADP also has a flag $m \in \{\mathsf{true}, \mathsf{false}\}$ to indicate whether this ADP may be applied to rewrite at a position below an annotated symbol in non-(i)AST evaluations. This flag will be modified and used by the processors in Sect. 4.

Definition 5 (ADPs). *An* annotated dependency pair (ADP) *has the form* $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$*, where* $\ell \in \mathcal{T}$ *with* $\ell \notin \mathcal{V}$*,* $m \in \{\mathsf{true}, \mathsf{false}\}$*, and for all* $1 \leq j \leq k$ *we have* $r_j \in \mathcal{T}^\#$ *with* $\mathcal{V}(r_j) \subseteq \mathcal{V}(\ell)$*.*

[3] Instead of restricting start terms to basic terms, one could allow start terms in argument normal form (denoted ANF-AST). Both Thm. 4 as well as our results on the ADP framework in Sect. 3 and 4 also hold for ANF-AST (ANF-iAST) instead of bAST (biAST). While ANF-iAST is equivalent to iAST, the requirement of start terms in $\mathtt{ANF}_{\mathcal{R}}$ is a real restriction for AST. Already in the non-probabilistic setting there are non-terminating TRSs $\mathcal{R}$ where all terms in $\mathtt{ANF}_{\mathcal{R}}$ are terminating (e.g., the well-known example of [46] with the rules $\mathsf{f}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{f}(x, x, x)$, $\mathsf{h}(x, y) \to x$, and $\mathsf{h}(x, y) \to y$).

For a rule $\ell \to \mu = \{p_1 : r_1, \ldots, p_k : r_k\}$*, its* canonical annotated dependency pair *is* $\mathcal{DP}(\ell \to \mu) = \ell \to \{p_1 : \#_{\mathrm{Pos}_{\mathcal{D}}(r_1)}(r_1), \ldots, p_k : \#_{\mathrm{Pos}_{\mathcal{D}}(r_k)}(r_k)\}^{\mathsf{true}}$*. 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 6. We obtain $\mathcal{DP}(\mathcal{R}_2) = \{(7), (8)\}$ and $\mathcal{DP}(\mathcal{R}_3) = \{(7), (9), (10)\}$ with

$$\mathsf{g} \to \{3/4 : \mathsf{D}(\mathsf{G}), 1/4 : \mathsf{0}\}^{\mathsf{true}} \quad (7) \qquad \mathsf{d}(x) \to \{1 : \mathsf{c}(x, x)\}^{\mathsf{true}} \quad (8)$$

$$\mathsf{d}(\mathsf{d}(x)) \to \{1 : \mathsf{c}(x, \mathsf{G})\}^{\mathsf{true}} \quad (9) \qquad \mathsf{d}(x) \to \{1 : \mathsf{0}\}^{\mathsf{true}} \quad (10)$$

Example 7. For $\mathcal{R}_{\mathsf{alg}}$, the canonical ADPs are

$$\mathsf{loop1}(y) \to \{1/2 : \mathsf{L1}(\mathsf{D}(y)),\ 1/2 : \mathsf{L2}(\mathsf{D}(y))\}^{\mathsf{true}} \quad (11)$$

$$\mathsf{loop1}(y) \to \{1/3 : \mathsf{L1}(\mathsf{T}(y)),\ 2/3 : \mathsf{L2}(\mathsf{T}(y))\}^{\mathsf{true}} \quad (12)$$

$$\mathsf{loop2}(\mathsf{s}(y)) \to \{1 : \mathsf{L2}(y)\}^{\mathsf{true}} \quad (13)$$

$$\mathsf{double}(\mathsf{s}(y)) \to \{1 : \mathsf{s}(\mathsf{s}(\mathsf{D}(y)))\}^{\mathsf{true}} \quad (14)$$

$$\mathsf{double}(\mathsf{0}) \to \{1 : \mathsf{0}\}^{\mathsf{true}} \quad (15)$$

$$\mathsf{triple}(\mathsf{s}(y)) \to \{1 : \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{T}(y))))\}^{\mathsf{true}} \quad (16)$$

$$\mathsf{triple}(\mathsf{0}) \to \{1 : \mathsf{0}\}^{\mathsf{true}} \quad (17)$$

We use the following rewrite relation in the ADP framework for `iAST`.

Definition 8 (Innermost Rewriting with ADPs, $\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}}$). *Let* $\mathcal{P}$ *be a finite set of ADPs (a so-called* ADP problem*). We define* $t \in \mathtt{ANF}_{\mathcal{P}}$ *if there are no* $t' \lhd t$*,* $\ell \to \mu^m \in \mathcal{P}$*, and substitution* σ *with* $\ell\sigma = \flat(t')$ *(i.e., no left-hand side* ℓ *matches a proper subterm* t' *of* t *when removing its annotations).*

A term $s \in \mathcal{T}^{\#}$ rewrites innermost *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 are* $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$*, 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* $1 \le j \le k$ *we have:*

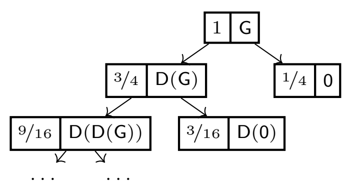
$$\begin{array}{llll} t_j = \quad s[r_j\sigma]_\pi & \text{if } \pi \in \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{true} & (\mathbf{at}) \\ t_j = \flat^{\uparrow}_{\pi}(\ s[r_j\sigma]_\pi) & \text{if } \pi \in \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{false} & (\mathbf{af}) \\ t_j = \quad s[\flat(r_j)\sigma]_\pi & \text{if } \pi \notin \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{true} & (\mathbf{nt}) \\ t_j = \flat^{\uparrow}_{\pi}(\ s[\flat(r_j)\sigma]_\pi) & \text{if } \pi \notin \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{false} & (\mathbf{nf}) \end{array}$$

Rewriting with $\mathcal{P}$ is like ordinary probabilistic term rewriting while considering and modifying annotations that indicate where a non-`iAST` evaluation may arise. A step of the form (**at**) (for **a**nnotation and **t**rue) is performed at the position of an annotation, i.e., this can potentially lead to a non-`iAST` evaluation. Hence, all annotations from the right-hand side r_j of the used ADP are kept during the rewrite step. However, annotations of subterms that correspond to variables of the ADP are removed, as these subterms are in normal form due to the innermost strategy. An example is the rewrite step $\mathsf{D}(\mathsf{G}) \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{R}_3)} \{3/4 : \mathsf{D}(\mathsf{D}(\mathsf{G})), 1/4 : \mathsf{D}(\mathsf{0})\}$ using the ADP (7). A step of the form (**af**) (for **a**nnotation and **f**alse) is similar but due to the flag $m = \mathsf{false}$ this ADP cannot be used below an annotation in a non-`iAST` evaluation. Hence, we remove all annotations above the used redex. So using an ADP of the form $\mathsf{g} \to \{3/4 : \mathsf{D}(\mathsf{G}), 1/4 : \mathsf{0}\}^{\mathsf{false}}$ on the term $\mathsf{D}(\mathsf{G})$ would yield $\mathsf{D}(\mathsf{G}) \overset{\mathrm{i}}{\hookrightarrow} \{3/4 : \mathsf{d}(\mathsf{D}(\mathsf{G})), 1/4 : \mathsf{d}(\mathsf{0})\}$, i.e., we remove the annotation of D at the root. A step of the form (**nt**) (for **n**o annotation and **t**rue) is performed at the position of a subterm without annotation. Hence, the subterm cannot lead to a non-`iAST` evaluation, but this rewrite step may be needed for an annotation at a position above. As an example, one could rewrite the non-annotated subterm g in $\mathsf{D(g)} \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{DP}(\mathcal{R}_3)} \{3/4 : \mathsf{D(d(g))}, 1/4 : \mathsf{D(0)}\}$ using the ADP (7). Finally, a step of the form (**nf**) (for **n**o annotation and **f**alse) is irrelevant for non-`iAST` evaluations, because the redex is not annotated and due to $m = \mathsf{false}$, afterwards one cannot rewrite an annotated term at a position above. For example, if one had the ADP $\mathsf{g} \to \{3/4 : \mathsf{D(G)}, 1/4 : \mathsf{0}\}^{\mathsf{false}}$, then we would obtain $\mathsf{D(g)} \overset{\mathrm{i}}{\hookrightarrow} \{3/4 : \mathsf{d(d(g))}, 1/4 : \mathsf{d(0)}\}$. The case (**nf**) is only needed to ensure that normal forms always remain the same, even if we remove or add annotations in rules.

Due to the annotations, we now consider specific RSTs, called *chain trees* [28,31]. 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 (**at**) or (**af**), as we specifically want to analyze the rewrite steps at annotated positions. We say that $\mathfrak{T} = (V, E, L, A)$ is a *$\mathcal{P}$-innermost chain tree* (iCT) 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. $L : V \to (0, 1] \times \mathcal{T}^{\#}$ 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. $A \subseteq V \setminus \mathtt{Leaf}$ (where $\mathtt{Leaf}$ are all leaves) is a subset of the inner nodes to indicate that we use (**at**) or (**af**) for the next step. $N = V \setminus (\mathtt{Leaf} \cup A)$ are all other inner nodes, i.e., where we rewrite using (**nt**) or (**nf**).
4. If $vE = \{w_1, \ldots, w_k\}$, then $t_v \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \ldots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$, where we use Case (**at**) or (**af**) if $v \in A$, and where we use Case (**nt**) or (**nf**) if $v \in N$.
5. Every infinite path in $\mathfrak{T}$ contains infinitely many nodes from A.

Let $|\mathfrak{T}|_{\mathtt{Leaf}} = \sum_{v \in \mathtt{Leaf}} p_v$. Then a PTRS $\mathcal{P}$ is `iAST` if $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$ for all $\mathcal{P}$-iCTs $\mathfrak{T}$. The corresponding $\mathcal{DP}(\mathcal{R}_1)$-chain tree for the $\mathcal{R}_1$-RST from Sect. 2.2 is shown on the right. Here, we again have $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$. With these definitions, in [31] we obtained the following result.



Theorem 9 (Chain Criterion for `iAST`). *A PTRS $\mathcal{R}$ is `iAST` iff $\mathcal{DP}(\mathcal{R})$ is `iAST`.*

So for `iAST`, one can analyze the canonical ADPs instead of the original PTRS.

3.2 ADPs and Chains - Full Rewriting

When adapting ADPs from innermost to full rewriting, the most crucial part is to define how to handle annotations if we rewrite above them. For innermost ADPs, we removed the annotations below the position of the redex, as such terms are always in normal form. However, this is not the case for full rewriting.

Example 10. Reconsider $\mathcal{R}_3$ and its canonical ADPs $\mathcal{DP}(\mathcal{R}_3) = \{(7), (9), (10)\}$ from Ex. 6. As noted in Ex. 1, $\mathcal{R}_3$ is `iAST`, but not `AST`. To adapt Def. 8 to full rewriting, clearly we have to omit the requirement that the redex is in `ANF`. However, this is not sufficient for soundness for full `AST`: Applying two rewrite steps with (7) to G would result in a chain tree with the leaves $9/16 : \mathsf{D(D(G))}$, $3/16 : \mathsf{D(0)}$

(which can be extended by the child $3/16 : \mathsf{0}$), and $1/4 : \mathsf{0}$. However, by Def. 8, every application of the ADP (9) removes the annotations of its arguments. So when applying (9) to $\mathsf{D}(\mathsf{D}(\mathsf{G}))$, we obtain $\{1 : \mathsf{c}(\mathsf{g}, \mathsf{G})\}$. But this would mean that the number of G-symbols is never increased. However, for all such chain trees $\mathfrak{T}$ we have $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$, i.e., we would falsely conclude that $\mathcal{R}_3$ is AST.

Ex. 10 shows that for full rewriting, we have to keep certain annotations below the used redex. After rewriting above a subterm like G (which starts a non-AST evaluation), it should still be possible to continue the evaluation of G if this subterm was "completely inside" the substitution of the applied rewrite step.

We use *variable reposition functions (VRFs)* to relate positions of variables in the left-hand side of an ADP to those positions of the same variables in the right-hand sides where we want to keep the annotations of the instantiated variables. So for an ADP $\ell \to \mu$ with $\ell|_\pi = x$, we indicate which occurrence of x in $r \in \mathrm{Supp}(\mu)$ should keep the annotations if one rewrites an instance of ℓ where the subterm at position π contains annotations.[4]

Definition 11 (Variable Reposition Functions). *Let $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$ be an ADP. A family of functions $\varphi_j : \mathrm{Pos}_{\mathcal{V}}(\ell) \to \mathrm{Pos}_{\mathcal{V}}(r_j) \uplus \{\bot\}$ with $1 \le j \le k$ is called a family of* variable reposition functions (VRF) *for the ADP iff for all $1 \le j \le k$ we have $\ell|_\pi = r_j|_{\varphi_j(\pi)}$ whenever $\varphi_j(\pi) \neq \bot$.*

Now we can define arbitrary (possibly non-innermost) rewriting with ADPs.

Definition 12 (Rewriting with ADPs, $\hookrightarrow_{\mathcal{P}}$). *ADPs and canonical ADPs are defined as in the innermost case. Let $\mathcal{P}$ be an ADP problem. A term $s \in \mathcal{T}^{\#}$* rewrites *with $\mathcal{P}$ to $\mu = \{p_1 : t_1, \ldots, p_k : t_k\}$ (denoted $s \hookrightarrow_{\mathcal{P}} \mu$) if there are an $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$, a VRF $(\varphi_j)_{1 \le j \le k}$ for this ADP, a substitution σ, and a $\pi \in \mathrm{Pos}_{\mathcal{D} \cup \mathcal{D}^{\#}}(s)$ such that $\flat(s|_\pi) = \ell\sigma$, and for all $1 \le j \le k$ we have:*

$$
\begin{array}{llll}
t_j = & s[\#_{\Phi_j}(r_j\sigma)]_\pi & \text{if } \pi \in \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{true} & \textbf{(at)} \\
t_j = \flat^{\uparrow}_{\pi}(& s[\#_{\Phi_j}(r_j\sigma)]_\pi) & \text{if } \pi \in \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{false} & \textbf{(af)} \\
t_j = & s[\#_{\Psi_j}(r_j\sigma)]_\pi & \text{if } \pi \notin \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \text{ and } m = \mathsf{true} & \textbf{(nt)}
\end{array}
$$

Here, $\Psi_j = \{\varphi_j(\rho).\tau \mid \rho \in \mathrm{Pos}_{\mathcal{V}}(\ell),\ \varphi_j(\rho) \neq \bot,\ \rho.\tau \in \mathrm{Pos}_{\mathcal{D}^{\#}}(s|_\pi)\}$ and $\Phi_j = \mathrm{Pos}_{\mathcal{D}^{\#}}(r_j) \cup \Psi_j$.

So Ψ_j considers all positions $\rho.\tau$ of annotated symbols in $s|_\pi$ that are below positions ρ of variables in ℓ. If φ_j maps ρ to a variable position ρ' in r_j, then the annotations below $\pi.\rho$ in s are kept in the resulting subterm at position $\pi.\rho'$ after the rewriting. As an example, consider $\mathsf{D}(\mathsf{D}(\mathsf{G})) \hookrightarrow_{\mathcal{DP}(\mathcal{R}_3)} \{1 : \mathsf{c}(\mathsf{G}, \mathsf{G})\}$. Here, we use the ADP $\mathsf{d}(\mathsf{d}(x)) \to \{1 : \mathsf{c}(x, \mathsf{G})\}^{\mathsf{true}}$ (9), with $\pi = \varepsilon$, $\sigma(x) = \mathsf{g}$, and the VRF $\varphi_1(1.1) = 1$. We get $\flat(\mathsf{D}(\mathsf{D}(\mathsf{G}))|_\varepsilon) = \mathsf{d}(\mathsf{d}(\mathsf{g})) = \ell\sigma$, $1.1 \in \mathrm{Pos}_{\mathcal{V}}(\ell)$, $1.1.\varepsilon \in \mathrm{Pos}_{\mathcal{D}^{\#}}(s|_\pi)$, and thus $\Psi_1 = \{\varphi_1(1.1).\varepsilon\} = \{1\}$ and $\Phi_1 = \mathrm{Pos}_{\mathcal{D}^{\#}}(r_1) \cup \Psi_1 = \{1, 2\}$.

The case (**nf**) from Def. 8 is missing in Def. 12, as we do not consider (argument) normal forms anymore. ADPs without annotations in the right-hand side and with

[4] VRFs were introduced in [33] when adapting ADPs to full *relative* rewriting. However, due to the probabilistic setting, our definition is slightly different.

the flag false are not needed for non-$\mathtt{AST}$ chain trees and thus, they could simply be removed from ADP problems.

Note that our VRFs in Def. 11 map a position of the left-hand side ℓ to at most one position in each right-hand side r_j of an ADP, even if the ADP is duplicating. A probabilistic rule or ADP $\ell \to \mu$ is *non-duplicating* if all rules in $\{\ell \to r \mid r \in \mathrm{Supp}(\mu)\}$ are, and a PTRS or ADP problem is non-duplicating if all of its rules are (disregarding the flag for ADPs). For example, for the duplicating ADP $\mathsf{d}(x) \to \{1 : \mathsf{c}(x, x)\}^{\mathsf{true}}$ (8), we have three different VRFs which map position 1 to either $\bot$, 1, or 2, but we cannot map it to both positions 1 and 2.

Therefore, our VRFs cannot handle duplicating rules and ADPs correctly. With VRFs as in Def. 11, $\mathcal{DP}(\mathcal{R}_2)$ would be considered to be $\mathtt{AST}$, as $\mathsf{D}(\mathsf{G})$ only rewrites to $\{1 : \mathsf{c}(\mathsf{G}, \mathsf{g})\}$ or $\{1 : \mathsf{c}(\mathsf{g}, \mathsf{G})\}$, but the annotation cannot be duplicated. Hence, the chain criterion would be unsound for duplicating PTRSs like $\mathcal{R}_2$.

To handle duplicating rules, one can adapt the direct application of orderings to prove $\mathtt{AST}$ from [28] and try to remove the duplicating rules of the PTRS before constructing the canonical ADPs.

Alternatively, one could modify the definition of the rewrite relation $\hookrightarrow_{\mathcal{P}}$ and use *generalized* VRFs (GVRFs) which can duplicate annotations instead of VRFs. This would yield a sound and complete chain criterion for full $\mathtt{AST}$ of possibly duplicating PTRSs, but then one would also have to consider this modified definition of $\hookrightarrow_{\mathcal{P}}$ for the processors of the ADP framework in Sect. 4. Unfortunately, almost all processors would become unsound when defining the rewrite relation $\hookrightarrow_{\mathcal{P}}$ via GVRFs (see Ex. 22, 35, and 37). Therefore, we use VRFs instead and restrict ourselves to non-duplicating PTRSs for the soundness of the chain criterion.[5]

Chain trees (CTs) are now defined like iCTs, where instead of $\stackrel{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}}$ we only require steps with $\hookrightarrow_{\mathcal{P}}$. Then an ADP problem $\mathcal{P}$ is $\mathtt{AST}$ if $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$ for all $\mathcal{P}$-CTs $\mathfrak{T}$. This leads to our desired chain criterion for $\mathtt{AST}$.

Theorem 13 (Chain Criterion for $\mathtt{AST}$). *A non-duplicating PTRS $\mathcal{R}$ is $\mathtt{AST}$ iff $\mathcal{DP}(\mathcal{R})$ is $\mathtt{AST}$.*

The above chain criterion allows us to analyze full $\mathtt{AST}$ for a significantly larger class of PTRSs than Thm. 3: we do not impose non-overlappingness and left-linearity anymore, and only require non-duplication instead of right-linearity.

Similar to Thm. 4, the ADP framework becomes more powerful if we restrict ourselves to basic start terms. Then it suffices if the PTRS is spare (instead of non-duplicating), since then redexes are never duplicated. In fact, *weak* spareness is sufficient, which subsumes both spareness and non-duplication. A rewrite step $\ell\sigma \to_{\mathcal{R}} \mu\sigma$ is *weakly spare* if $\sigma(x) \in \mathtt{NF}_{\mathcal{R}}$ for every $x \in \mathcal{V}$ where x occurs less often in ℓ than in some $r \in \mathrm{Supp}(\mu)$. An $\mathcal{R}$-RST is weakly spare if all rewrite steps corresponding to its edges are weakly spare. A PTRS $\mathcal{R}$ is weakly spare if each $\mathcal{R}$-RST that starts with $\{1 : t\}$ for a basic term t is weakly spare. The sufficient conditions for spareness in [16] can easily be adapted to weak spareness.

In the ADP framework for $\mathtt{bAST}$, we only have to prove that no term starting a non-$\mathtt{AST}$ evaluation can be reached from a basic start term. Here we use *basic ADP*

[5] A related restriction is needed in the setting of (non-probabilistic) relative termination due to the VRFs [33].

problems $(\mathcal{I}, \mathcal{P})$, where $\mathcal{I}$ and $\mathcal{P}$ are finite sets of ADPs. $\mathcal{P}$ are again the ADPs which we analyze for AST and the *reachability component* $\mathcal{I}$ contains so-called *initial* ADPs. A basic ADP problem $(\mathcal{I}, \mathcal{P})$ is bAST if $|\mathfrak{T}|_{\texttt{Leaf}} = 1$ holds for all those $(\mathcal{I} \cup \mathcal{P})$-CTs $\mathfrak{T}$ that start with a term $t^{\#}$ where $t \in \mathcal{T}$ is basic, and where ADPs from $\mathcal{I} \setminus \mathcal{P}$ are only used finitely often within the tree $\mathfrak{T}$. Thus, every basic ADP problem $(\mathcal{I}, \mathcal{P})$ can be replaced by $(\mathcal{I} \setminus \mathcal{P}, \mathcal{P})$. For a PTRS $\mathcal{R}$, the *canonical basic ADP problem* is $(\varnothing, \mathcal{DP}(\mathcal{R}))$.

Theorem 14 (Chain Criterion for bAST). *A weakly spare PTRS $\mathcal{R}$ is* bAST *iff $(\varnothing, \mathcal{DP}(\mathcal{R}))$ is* bAST.

Remark 15. In the chain criterion for non-probabilistic DPs, it suffices to regard only instantiations where all terms below an annotated symbol are terminating. The reason is the *minimality property* of non-probabilistic term rewriting, i.e., whenever a term starts an infinite rewrite sequence, then it also starts an infinite sequence where all proper subterms of every used redex are terminating. However, in the probabilistic setting the minimality property does not hold [30]. For $\mathcal{R}_3$, g starts a non-AST RST, but in this RST, one has to apply Rule (3) to the redex d(d(g)), although it contains the proper subterm g that starts a non-AST RST.

4 The Probabilistic ADP Framework for Full Rewriting

The idea of the DP framework for non-probabilistic TRSs is to apply *processors* repeatedly which transform a DP problem into simpler sub-problems [17,18]. Since different techniques can be applied to different sub-problems, this results in a *modular* approach for termination analysis. This idea is also used in the ADP framework. An *ADP processor* Proc has the form $\mathrm{Proc}(\mathcal{P}) = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ for ADP problems $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_n$. Let $\mathcal{Z} \in \{\mathtt{AST}, \mathtt{iAST}\}$. Proc is *sound* for $\mathcal{Z}$ if $\mathcal{P}$ is $\mathcal{Z}$ whenever $\mathcal{P}_i$ is $\mathcal{Z}$ for all $1 \leq i \leq n$. It is *complete* for $\mathcal{Z}$ if $\mathcal{P}_i$ is $\mathcal{Z}$ for all $1 \leq i \leq n$ whenever $\mathcal{P}$ is $\mathcal{Z}$. The definitions for bAST are analogous, but with basic ADP problems $(\mathcal{I}, \mathcal{P})$. Thus, one starts with the canonical (basic) ADP problem and applies sound (and preferably complete) ADP processors repeatedly until there are no more remaining ADP problems. This implies that the canonical (basic) ADP problem is $\mathcal{Z}$ and by the chain criterion, the original PTRS is $\mathcal{Z}$ as well.

An ADP problem without annotations is always AST, because then no rewrite step increases the number of annotations (recall that VRFs cannot duplicate annotations). Hence, then any term with n annotations only starts rewrite sequences with at most n steps of the form (**at**) or (**af**), i.e., all $\mathcal{P}$-CTs are finite.

In the following, we recapitulate the main processors for iAST from [31] and adapt them to our new framework for AST and bAST.

4.1 Dependency Graph Processor

The innermost $\mathcal{P}$-*dependency graph* is a control flow graph whose nodes are the ADPs from $\mathcal{P}$. It indicates 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, here we use the *non-probabilistic variant*

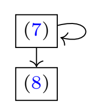
$\mathrm{np}(\mathcal{P}) = \{\ell \to \flat(r_j) \mid \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{\mathsf{true}} \in \mathcal{P}, 1 \leq j \leq k\}$, which is an ordinary TRS over the original signature Σ. For $\mathrm{np}(\mathcal{P})$ we only consider rules with the flag `true`, since only they are needed for rewriting below annotations. We define $t \trianglelefteq_{\#} s$ if there is a $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(s)$ and $t = \flat(s|_\pi)$, i.e., t results from a subterm of s with annotated root symbol by removing its annotations.

Definition 16 (Innermost Dependency Graph). *The* innermost $\mathcal{P}$-dependency graph *has the set of 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 $t^\# \sigma_1 \xrightarrow{\mathsf{i}}{}^*_{\mathrm{np}(\mathcal{P})} \ell_2^\# \sigma_2$ 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 $\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{P}}$-step of the form (**at**) or (**af**) with α at position π there may eventually come another $\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{P}}$-step of the form (**at**) or (**af**) with α' on or below π. Since every infinite path in an iCT contains infinitely many nodes from A, every such path traverses a cycle of the innermost dependency graph infinitely often. Thus, it suffices to consider its strongly connected components (SCCs)[6] separately. In our framework, this means that we remove the annotations from all ADPs except those in the SCC that we want to analyze. Since checking whether there exist σ_1, σ_2 as in Def. 16 is undecidable, to automate the following processor, the same over-approximation techniques as for the non-probabilistic dependency graph can be used, see, e.g., [2,18,23]. In the following, $\flat(\mathcal{P})$ denotes the ADP problem $\mathcal{P}$ where all annotations are removed.

Theorem 17 (Dependency Graph Processor for `iAST`). *For the SCCs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of the innermost $\mathcal{P}$-dependency graph, the processor* $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{P}) = \{\mathcal{P}_1 \cup \flat(\mathcal{P} \setminus \mathcal{P}_1), \ldots, \mathcal{P}_n \cup \flat(\mathcal{P} \setminus \mathcal{P}_n)\}$ *is sound and complete for* `iAST`.

Example 18. Consider the PTRS $\mathcal{R}_2$ and its canonical ADPs from Ex. 6. The innermost $\mathcal{DP}(\mathcal{R}_2)$-dependency graph is on the right. As the only SCC $\{(7)\}$ does not contain (8), we can remove all annotations from (8). However, (8) has no annotations. Thus, $\mathrm{Proc}_{\mathtt{DG}}$ does not change $\mathcal{DP}(\mathcal{R}_2)$.



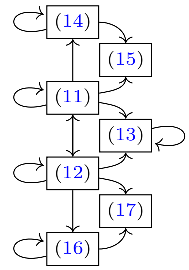
Adaption for `AST`: To handle full rewriting, we have to change the definition of the dependency graph as we can now also perform non-innermost steps.

Definition 19 (Dependency 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 $1 \leq j \leq k$ with $t^\# \sigma_1 \to^*_{\mathrm{np}(\mathcal{P})} \ell_2^\# \sigma_2$.*

Theorem 20 (Dependency Graph Processor for `AST`). *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 for* `AST`.

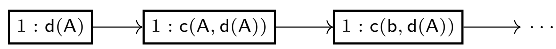
Example 21. Consider $\mathcal{R}_{\mathsf{alg}}$ and its canonical ADPs from Ex. 7. The $\mathcal{DP}(\mathcal{R}_{\mathsf{alg}})$-dependency graph is given below. Its SCCs are $\{(11), (12)\}$, $\{(13)\}$, $\{(14)\}$, $\{(16)\}$.

[6] A set $\mathcal{P}'$ of ADPs is an *SCC* if it is a maximal cycle, i.e., a maximal set where for any α, α' in $\mathcal{P}'$ there is a non-empty path from α to α' only traversing nodes from $\mathcal{P}'$.



For each SCC we create a separate ADP problem, where all annotations outside the SCC are removed. This leads to the ADP problems $\{(11),(12),\flat(13)$ - $\flat(17)\}$, $\{(13),\flat(11),\flat(12),\flat(14)$ - $\flat(17)\}$, $\{(14),\flat(11)$ - $\flat(13),\flat(15)$ - $\flat(17)\}$, and $\{(16),\flat(11)$ - $\flat(15),\flat(17)\}$.

Example 22. If we used GVRFs that can duplicate annotations, then the dependency graph processor would not be sound. The reason is that $\mathrm{Proc}_{\mathtt{DG}}$ maps ADP problems without annotations to the empty set. However, this would be unsound if we had GVRFs, because then the ADP problem with $\mathsf{a} \to \{1 : \mathsf{b}\}^{\mathsf{true}}$ and $\mathsf{d}(x) \to \{1 : \mathsf{c}(x, \mathsf{d}(x))\}^{\mathsf{true}}$ would not be AST. Here, the use of GVRFs would lead to the following CT with an infinite number of (**at**) steps that rewrite A to b.



Adaption for bAST: Here, ADPs that are not in the considered SCC $\mathcal{P}_i$ may still be necessary for the initial steps from the basic start term to the SCC. Thus, while we remove the annotations of ADPs outside the SCC $\mathcal{P}_i$ in the second component $\mathcal{P}$ of a basic ADP problem $(\mathcal{I}, \mathcal{P})$, we add (the original versions of) those ADPs to $\mathcal{I}$ that reach the SCC $\mathcal{P}_i$ in the $(\mathcal{I} \cup \mathcal{P})$-dependency graph. Let $\mathcal{P}_i\!\uparrow$ be the set of all $\mathcal{J} \subseteq (\mathcal{I} \cup \mathcal{P}) \setminus \mathcal{P}_i$ such that all ADPs of $\mathcal{J}$ reach $\mathcal{P}_i$ in the $(\mathcal{I} \cup \mathcal{P})$-dependency graph, and for all pairs of ADPs $\alpha, \beta \in \mathcal{J}$ with $\alpha \neq \beta$, α reaches β or β reaches α in the $(\mathcal{I} \cup \mathcal{P})$-dependency graph. Furthermore, $\mathcal{J}$ must be maximal w.r.t. these properties, i.e., if $\alpha \notin \mathcal{J}$ then α does not reach $\mathcal{P}_i$ or there exists a $\beta \in \mathcal{J}$ such that α does not reach β and β does not reach α.

Theorem 23 (Dependency Graph Processor for bAST). *For the SCCs* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *of the* $\mathcal{P}$*-dependency graph, the processor* $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{I}, \mathcal{P}) = \{(\mathcal{J} \cup \flat(\mathcal{I} \setminus \mathcal{J}), \mathcal{P}_i \cup \flat(\mathcal{P} \setminus \mathcal{P}_i)) \mid 1 \leq i \leq n, \mathcal{J} \in \mathcal{P}_i\!\uparrow\}$ *is sound and complete for* bAST*.*

As remarked in Sect. 3.2, every basic ADP problem $(\mathcal{I}, \mathcal{P})$ can be replaced by $(\mathcal{I} \setminus \mathcal{P}, \mathcal{P})$. Thus, this should be done after every application of a processor.

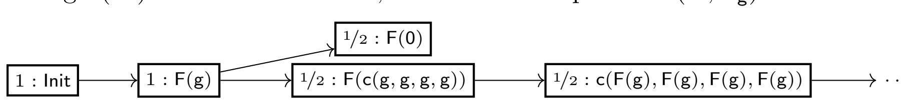
Example 24. To prove bAST of $\mathcal{R}_{\mathsf{alg}}$, we start with $(\varnothing, \mathcal{DP}(\mathcal{R}_{\mathsf{alg}}))$. The SCC $\{(16)\}$ is only reachable from (11) and (12), leading to the basic ADP problem $(\{(11),(12)\}, \{(16),\flat(11)$ - $\flat(15),\flat(17)\})$. The SCC $\{(11),(12)\}$ is not reachable from other ADPs and thus, here we obtain $(\varnothing,\ \{(11),(12),\flat(13)$ - $\flat(17)\})$, etc.

Example 25. The next ADP problem $\mathcal{P}_{\mathsf{g}}$ illustrates the reachability component.

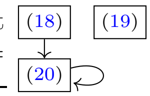
$$\mathsf{init} \to \{1 : \mathsf{F}(\mathsf{g})\}^{\mathsf{true}} \quad (18) \qquad \mathsf{g} \to \{1/2 : \mathsf{c}(\mathsf{g},\mathsf{g},\mathsf{g},\mathsf{g}), 1/2 : \mathsf{0}\}^{\mathsf{true}} \quad (19)$$

$$\mathsf{f}(\mathsf{c}(x_1, x_2, x_3, x_4)) \to \{1 : \mathsf{c}(\mathsf{F}(x_1), \mathsf{F}(x_2), \mathsf{F}(x_3), \mathsf{F}(x_4))\}^{\mathsf{true}} \quad (20)$$

Although (19) has no annotations, the basic ADP problem $(\varnothing, \mathcal{P}_{\mathsf{g}})$ is not bAST:



This is a random walk biased towards non-termination, where the number of $\mathsf{F}(\mathsf{g})$ subterms increases by 3 or decreases by 1, both with probability 1/2.



Since the only SCC of the $\mathcal{P}_{\mathsf{g}}$-dependency graph on the right is $\{(20)\}$, $\mathrm{Proc}_{\mathtt{DG}}$ replaces $\mathsf{F}(\mathsf{g})$ by $\mathsf{f}(\mathsf{g})$ in (18) and obtains $\mathcal{P}'_{\mathsf{g}} = \{\flat(18), (19), (20)\}$. However, $(\varnothing, \mathcal{P}'_{\mathsf{g}})$ would be `bAST`. So for the soundness of the dependency graph processor, we have to add the original ADP (18) to the reachability component and obtain $(\{(18)\}, \mathcal{P}'_{\mathsf{g}})$ which is again not `bAST`.

4.2 Usable Terms Processor

The dependency graph processor removes either all annotations from an ADP or none. But an ADP can still contain terms t with annotated root where no instance $t\sigma_1$ rewrites to an instance $\ell^{\#}\sigma_2$ of a left-hand side ℓ of an ADP with annotations. The *usable terms processor* removes the annotation from the root of such *non-usable* terms like $\mathsf{D}(\ldots)$ in $\mathcal{DP}(\mathcal{R}_2) = \{(7), (8)\}$. So instead of whole ADPs, here we consider the subterms in the right-hand sides of an ADP individually.

Theorem 26 (Usable Terms Processor for `iAST`). *Let $\ell_1 \in \mathcal{T}$ and $\mathcal{P}$ be an ADP problem. We call $t \in \mathcal{T}^{\#}$ with* $\mathrm{root}(t) \in \mathcal{D}^{\#}$ innermost 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 $\#_{\{\varepsilon\}}(t)\sigma_1 \xrightarrow{\mathsf{i}}{}^{*}_{\mathrm{np}(\mathcal{P})} \ell_2^{\#}\sigma_2$ and both $\ell_1\sigma_1$ and $\ell_2\sigma_2$ are in $\mathtt{ANF}_{\mathcal{P}}$. Let $\Delta_{\ell,\mathcal{P}}(s) = \{\pi \in \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \mid s|_{\pi}$ is innermost usable w.r.t. ℓ and $\mathcal{P}\}$. The transformation that removes the annotations from the roots of all non-usable terms in the right-hand sides is $\mathcal{T}_{\mathtt{UT}}(\mathcal{P}) = \{\ell \to \{p_1 : \#_{\Delta_{\ell,\mathcal{P}}(r_1)}(r_1), \ldots, p_k : \#_{\Delta_{\ell,\mathcal{P}}(r_k)}(r_k)\}^m \mid \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}\}$. Then $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{P}) = \{\mathcal{T}_{\mathtt{UT}}(\mathcal{P})\}$ is sound and complete for `iAST`.*

So for $\mathcal{DP}(\mathcal{R}_2)$, $\mathrm{Proc}_{\mathtt{UT}}$ replaces (7) by $\mathsf{g} \to \{3/4 : \mathsf{d}(\mathsf{G}), 1/4 : \mathsf{0}\}^{\mathsf{true}}$ (7′).

Adaption for `AST` and `bAST`: Similar to the dependency graph, for full rewriting, we remove the `ANF` requirement and allow non-innermost steps to reach the next ADP. To adapt the processor to `bAST`, in the reachability component we consider usability w.r.t. $\mathcal{I} \cup \mathcal{P}$, since one may use both $\mathcal{I}$ and $\mathcal{P}$ in the initial steps.

Theorem 27 (Usable Terms Processor for `AST` and `bAST`). *We call $t \in \mathcal{T}^{\#}$ with* $\mathrm{root}(t) \in \mathcal{D}^{\#}$ usable *w.r.t. an ADP problem $\mathcal{P}$ 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 $\#_{\{\varepsilon\}}(t)\sigma_1 \to^{*}_{\mathrm{np}(\mathcal{P})} \ell_2^{\#}\sigma_2$. Let $\Delta_{\mathcal{P}}(s) = \{\pi \in \mathrm{Pos}_{\mathcal{D}^{\#}}(s) \mid s|_{\pi}$ is usable w.r.t. $\mathcal{P}\}$ and $\mathcal{T}_{\mathtt{UT}}(\mathcal{P}) = \{\ell \to \{p_1 : \#_{\Delta_{\mathcal{P}}(r_1)}(r_1), \ldots, p_k : \#_{\Delta_{\mathcal{P}}(r_k)}(r_k)\}^m \mid \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}\}$. Then $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{P}) = \{\mathcal{T}_{\mathtt{UT}}(\mathcal{P})\}$ is sound and complete for `AST` and $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{I}, \mathcal{P}) = \{(\mathcal{T}_{\mathtt{UT}}(\mathcal{I} \cup \mathcal{P}), \mathcal{T}_{\mathtt{UT}}(\mathcal{P}))\}$ is sound and complete for `bAST`.*

Example 28. For `AST`, $\mathrm{Proc}_{\mathtt{UT}}$ transforms $\{(11), (12), \flat(13)$ - $\flat(17)\}$ from Ex. 21 into $\{(11'), (12'), \flat(13)$ - $\flat(17)\}$ with

$$\mathsf{loop1}(y) \to \{1/2 : \mathsf{L1}(\mathsf{double}(y)),\ 1/2 : \mathsf{loop2}(\mathsf{double}(y))\}^{\mathsf{true}} \quad (11')$$

$$\mathsf{loop1}(y) \to \{1/3 : \mathsf{L1}(\mathsf{triple}(y)),\ 2/3 : \mathsf{loop2}(\mathsf{triple}(y))\}^{\mathsf{true}} \quad (12')$$

The reason is that the left-hand sides of the only ADPs with annotations in the ADP problem have the root $\mathsf{loop1}$. Thus, $\mathsf{L2}$-, D-, or T-terms are not usable.

For bAST, applying $\mathrm{Proc}_{\mathtt{UT}}$ to $(\{(11),(12)\},\{(16),\flat(11)$ - $\flat(15),\flat(17)\})$ and afterwards removing those ADPs from the reachability component that also occur in the second component yields $(\{(11'),(12'')\},\{(16),\flat(11)$ - $\flat(15),\flat(17)\})$ with

$$\mathsf{loop1}(y) \to \{1/3 : \mathsf{L1}(\mathsf{T}(y)),\ 2/3 : \mathsf{loop2}(\mathsf{T}(y))\}^{\mathsf{true}} \qquad (12'')$$

The reason is that the left-hand sides of ADPs with annotations in their right-hand sides have the root symbols loop1 (in (11) and (12)) or triple (in (16)).

4.3 Usable Rules Processor

In an innermost rewrite step, all variables of the used rule are instantiated with normal forms. The *usable rules processor* detects rules that cannot be used below annotations in right-hand sides of ADPs when their variables are instantiated with normal forms. For these rules we can set their flag to false, indicating that the annotated subterms on their right-hand sides may still lead to a non-iAST sequence, but the context of these annotations is irrelevant.

Theorem 29 (Usable Rules Processor for iAST). *Let $\mathcal{P}$ be an ADP problem and for $f \in \Sigma^{\#}$, let $\mathrm{Rules}_{\mathcal{P}}(f) = \{\ell \to \mu^{\mathsf{true}} \in \mathcal{P} \mid \mathrm{root}(\ell) = f\}$. For $t \in \mathcal{T}^{\#}$, its* usable rules *$\mathcal{U}_{\mathcal{P}}(t)$ are the smallest set with $\mathcal{U}_{\mathcal{P}}(x) = \varnothing$ for all $x \in \mathcal{V}$ and $\mathcal{U}_{\mathcal{P}}(f(t_1,...,t_n)) = \mathrm{Rules}_{\mathcal{P}}(f) \cup \bigcup_{i=1}^{n} \mathcal{U}_{\mathcal{P}}(t_i) \cup \bigcup_{\ell \to \mu^{\mathsf{true}} \in \mathrm{Rules}_{\mathcal{P}}(f), r \in \mathrm{Supp}(\mu)} \mathcal{U}_{\mathcal{P}}(\flat(r))$, otherwise. The* usable rules *of $\mathcal{P}$ are $\mathcal{U}(\mathcal{P}) = \bigcup_{\ell \to \mu^m \in \mathcal{P}, r \in \mathrm{Supp}(\mu), t \trianglelefteq_{\#} r} \mathcal{U}_{\mathcal{P}}(t^{\#})$. 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 30. The ADP problem $\{(7'),(8)\}$ has no subterms below annotations. So both rules are not usable and we set their flags to false which leads to

$$\mathsf{g} \to \{3/4 : \mathsf{d}(\mathsf{G}),\ 1/4 : 0\}^{\mathsf{false}}\ (7'') \qquad \mathsf{d}(x) \to \{1 : \mathsf{c}(x,x)\}^{\mathsf{false}} \quad (8')$$

Adaption for AST: For full rewriting and arbitrary start terms, the usable rules processor is unsound. This is already the case for non-probabilistic rewriting, but in the classical DP framework there nevertheless exist processors for full rewriting based on usable rules which rely on taking the C_ε-rules $\mathsf{h}(x,y) \to x$ and $\mathsf{h}(x,y) \to y$ for a fresh function symbol h into account, see, e.g., [18,17,24,47]. However, the following example shows that this is not possible for AST.

Example 31. The ADP problem $\mathcal{P}'_{\mathsf{g}}$ from Ex. 25 is not AST. It has no usable rules and thus, $\mathrm{Proc}_{\mathtt{UR}}$ would transform $\mathcal{P}'_{\mathsf{g}}$ into $\mathcal{P}''_{\mathsf{g}}$ where the flag of all ADPs is false. However, then we can no longer rewrite the argument g of F(g). Similarly, if we start with F(G), rewriting G would remove the annotation of F above, i.e., $\mathsf{F}(\mathsf{G}) \hookrightarrow_{\mathcal{P}''_{\mathsf{g}}} \{1/2 : \mathsf{f}(\mathsf{c}(\mathsf{g},\mathsf{g},\mathsf{g},\mathsf{g})), 1/2 : \mathsf{f}(0)\}$. Hence, then all CTs are finite. This also holds when adding the C_ε-ADPs $\mathsf{h}(x,y) \to \{1 : x\}^{\mathsf{true}}$ and $\mathsf{h}(x,y) \to \{1 : y\}^{\mathsf{true}}$.

Thus, even integrating the C_ε-rules to represent non-determinism would not allow a usable rule processor for AST with arbitrary start terms. Moreover, the corresponding proofs in the non-probabilistic setting rely on the minimality property, which does not hold in the probabilistic setting, see Remark 15.

Adaption for bAST: For bAST, we can apply the usable rules processor as for innermost rewriting. Since the start term is basic, in the first application of an ADP all variables are instantiated with normal forms. Hence, the only rules that can be applied for rewrite steps below annotated symbols are the ones that are introduced in right-hand sides of ADPs. Therefore, we can use the same definitions as in Thm. 29 to over-approximate the set of ADPs that can be used below an annotated symbol in a CT that starts with a basic term. Here, we have to consider the reachability component as well for the usable rules, as these ADPs can also be used in the initial rewrite steps.

Theorem 32 (Usable Rules Processor for bAST). *The following processor is sound and complete for* bAST*:*

$$\begin{aligned}\mathrm{Proc}_{\mathtt{UR}}(\mathcal{I},\mathcal{P}) = \{\big(\ & \big(\mathcal{I} \cap \mathcal{U}(\mathcal{I} \cup \mathcal{P})\big) \cup \{\ell \to \mu^{\mathsf{false}} \mid \ell \to \mu^m \in \mathcal{I} \setminus \mathcal{U}(\mathcal{I} \cup \mathcal{P})\},\\ & \big(\mathcal{P} \cap \mathcal{U}(\mathcal{I} \cup \mathcal{P})\big) \cup \{\ell \to \mu^{\mathsf{false}} \mid \ell \to \mu^m \in \mathcal{P} \setminus \mathcal{U}(\mathcal{I} \cup \mathcal{P})\}\ \big)\}.\end{aligned}$$

Example 33. For the basic ADP problem $(\{(11'), (12'')\}, \{(16), \flat(11)$ - $\flat(15), \flat(17)\})$ from Ex. 28, only the double- and triple-ADPs $\flat(14), \flat(15), (16), \flat(17)$ are usable. So we can set the flag of all other ADPs in this problem to false. The same holds for the other basic ADPs resulting from the dependency graph and the usable terms processor in this example, i.e., here the usable rules processor also sets the flags of all ADPs except the double- and triple-ADPs to false.

Example 34. To see why we use $\mathcal{P} \cap \mathcal{U}(\mathcal{I} \cup \mathcal{P})$ instead of $\mathcal{U}(\mathcal{P})$ in Thm. 32 (whereas $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$ instead of $\mathcal{T}_{\mathtt{UT}}(\mathcal{I} \cup \mathcal{P})$ suffices for the second component in Thm. 27), consider the basic ADP problem $(\{(18)\}, \mathcal{P}'_{\mathsf{g}})$ from Ex. 25 which is not bAST. As noted in Ex. 31, $\mathcal{U}(\mathcal{P}'_{\mathsf{g}}) = \varnothing$, but if one sets the flags of all ADPs in $\mathcal{P}'_{\mathsf{g}}$ to false, then all CTs are finite (i.e., then $\mathrm{Proc}_{\mathtt{UR}}$ would be unsound). In contrast, for $\mathcal{I} = \{(18)\}$, we have $\mathcal{U}(\mathcal{I} \cup \mathcal{P}'_{\mathsf{g}}) = \{(19)\}$, because g occurs below the annotated symbol F in (18). Hence, $\mathrm{Proc}_{\mathtt{UR}}(\{(18)\}, \mathcal{P}'_{\mathsf{g}})$ only sets the flags of all ADPs except (19) to false and thus, the resulting basic ADP problem is still not bAST.

Example 35. Note that if one used GVRFs, then the usable rules processor would be unsound on ADP problems that are not weakly spare. For instance, it would transform the ADP problem $(\varnothing, \{(7'), (8)\})$ (which is not bAST when using GVRFs) into $(\varnothing, \{(7''), (8')\})$ (see Ex. 30). However, as $(8')$ has the flag false, it cannot be applied at the position of the non-annotated symbol d, since Def. 8 does not have a case of the form (**nf**). Hence, $(\varnothing, \{(7''), (8')\})$ is bAST.

4.4 Reduction Pair Processor

Next we adapt the reduction pair processor which lifts the direct use of orderings from PTRSs to ADP problems. This processor is the same for iAST and AST.

To handle expected values, as in [28,31] we only consider orderings based on polynomial interpretations [36]. A *polynomial interpretation* Pol is a $\Sigma^{\#}$-algebra which maps every function $f \in \Sigma^{\#}$ to a polynomial $f_{\mathrm{Pol}} \in \mathbb{N}[\mathcal{V}]$. It is *monotonic* if $x > y$ implies $f_{\mathrm{Pol}}(\ldots,x,\ldots) > f_{\mathrm{Pol}}(\ldots,y,\ldots)$ for all $f \in \Sigma^{\#}$. $\mathrm{Pol}(t)$ denotes the *interpretation* of a term $t \in \mathcal{T}^{\#}$ by Pol. An arithmetic inequation $\mathrm{Pol}(t_1) > \mathrm{Pol}(t_2)$ *holds* if it is true for all instantiations of its variables by natural numbers.

The constraints (1) - (3) in Thm. 36 are based on the conditions of a ranking function for AST as in [39]. If we prove AST by considering the rules $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}$ of a PTRS directly, then we need a monotonic polynomial interpretation Pol and require a weak decrease when comparing $\mathrm{Pol}(\ell)$ to the expected value $\sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(r_j)$ of the right-hand side, and additionally, at least one $\mathrm{Pol}(r_j)$ must be strictly smaller than $\mathrm{Pol}(\ell)$ [28]. For ADPs, we adapt these constraints by comparing the value $\mathrm{Pol}(\ell^{\#})$ of the annotated left-hand side with the $\#$-*sum* of the right-hand sides r_j, i.e., the sum of the polynomial values of their annotated subterms $\mathcal{S}um(r_j) = \sum_{t \trianglelefteq_{\#} r_j} \mathrm{Pol}(t^{\#})$. This allows us to remove the requirement of (strong) monotonicity (every polynomial f_{Pol} with natural coefficients is weakly monotonic, i.e., $x \geq y$ implies $f_{\mathrm{Pol}}(\ldots,x,\ldots) \geq f_{\mathrm{Pol}}(\ldots,y,\ldots)$).

Here, (1) we require a weak decrease when comparing the annotated left-hand side with the expected value of $\#$-*sums* in the right-hand side. The processor then removes the annotations from those ADPs where (2) in addition there is at least one right-hand side r_j whose $\#$-*sum* is strictly decreasing.[7] Finally, (3) for every rule with the flag true (which can therefore be used for steps below annotations), the expected value must be weakly decreasing when removing the annotations. As in [4,28,31], to ensure "monotonicity" w.r.t. expected values, we restrict ourselves to interpretations with multilinear polynomials, i.e., all monomials must have the form $c \cdot x_1^{e_1} \cdot \ldots \cdot x_n^{e_n}$ with $c \in \mathbb{N}$ and $e_1, \ldots, e_n \in \{0, 1\}$.

Theorem 36 (Reduction Pair Processor for iAST & AST). *Let* $\mathrm{Pol} : \mathcal{T}^{\#} \to \mathbb{N}[\mathcal{V}]$ *be a multilinear polynomial interpretation. Let* $\mathcal{P} = \mathcal{P}_{\geq} \uplus \mathcal{P}_{>}$ *with* $\mathcal{P}_{>} \neq \varnothing$ *where:*

(1) $\forall \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P} : \mathrm{Pol}(\ell^{\#}) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathcal{S}um(r_j)$.
(2) $\forall \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}_{>} : \exists j \in \{1, \ldots, k\} : \mathrm{Pol}(\ell^{\#}) > \mathcal{S}um(r_j)$.
If $m = \mathsf{true}$*, then we additionally have* $\mathrm{Pol}(\ell) \geq \mathrm{Pol}(\flat(r_j))$.
(3) $\forall \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{\mathsf{true}} \in \mathcal{P} : \mathrm{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(\flat(r_j))$.

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

Example 37. To conclude iAST for $\mathcal{R}_2$ we have to remove all remaining annotations in the ADP problem $\{(7''), (8')\}$ from Ex. 30 (then another application of the dependency graph processor yields the empty set of ADP problems). Here, we can use the reduction pair processor with the polynomial interpretation that maps G to 1, and all other symbols to 0. Then $(8')$ is weakly decreasing, and $(7'')$ is strictly

[7] In addition, the corresponding non-annotated right-hand side $\flat(r_j)$ must be at least weakly decreasing. This ensures that nested annotations behave "monotonically". So we have to ensure that $\mathrm{Pol}(A) > \mathrm{Pol}(B)$ also implies that the $\#$-*sum* of $F(A)$ is greater than $F(B)$, i.e., $\mathrm{Pol}(A) > \mathrm{Pol}(B)$ must imply that $\mathcal{S}um(F(A)) = \mathrm{Pol}(F(a)) + \mathrm{Pol}(A) > \mathrm{Pol}(F(b)) + \mathrm{Pol}(B) = \mathcal{S}um(F(B))$, which is ensured by $\mathrm{Pol}(a) \geq \mathrm{Pol}(b)$.

decreasing, since (1) $\mathrm{Pol}(\mathsf{G}) = 1 \geq {}^3\!/\!{}_4 \cdot \mathcal{S}um(\mathsf{d}(\mathsf{G})) + {}^1\!/\!{}_4 \cdot \mathcal{S}um(\mathsf{0}) = {}^3\!/\!{}_4 \cdot \mathrm{Pol}(\mathsf{G}) = {}^3\!/\!{}_4$ and (2) $\mathrm{Pol}(\mathsf{G}) = 1 > \mathcal{S}um(\mathsf{0}) = 0$. Thus, the annotation of G in (7′′) is deleted.

Note that this polynomial interpretation would also satisfy the constraints for $\mathcal{DP}(\mathcal{R}_2) = \{(7), (8)\}$ from Ex. 6, i.e., it would allow us to remove the annotations from the canonical ADP directly. Hence, if we extended our approach for `AST` to GVRFs that can duplicate annotations, then the reduction pair processor would be unsound, as it would allow us to falsely "prove" `AST` of $\mathcal{DP}(\mathcal{R}_2)$. The problem is that we compare terms with annotations via their #-sum, but for duplicating ADPs like (8), $\mathrm{Pol}(\mathsf{d}(x)) \geq \mathrm{Pol}(\mathsf{c}(x, x))$ does not imply $\mathcal{S}um(\mathsf{d}(\mathsf{G})) \geq \mathcal{S}um(\mathsf{c}(\mathsf{G}, \mathsf{G}))$ since $\mathcal{S}um(\mathsf{d}(\mathsf{G})) = \mathrm{Pol}(\mathsf{G})$ and $\mathcal{S}um(\mathsf{c}(\mathsf{G}, \mathsf{G})) = \mathrm{Pol}(\mathsf{G}) + \mathrm{Pol}(\mathsf{G})$.

Example 38. To prove `AST` for $\mathcal{R}_{\mathsf{alg}}$, we also have to remove all annotations from all remaining sub-problems. For instance, for the sub-problem $\{(11'), (12'), \flat(13)$ - $\flat(17)\}$ from Ex. 28, we can use the reduction pair processor with the polynomial interpretation that maps $\mathsf{s}(x)$ to $x + 1$, $\mathsf{double}(x)$ to $2x$, $\mathsf{triple}(x)$ to $3x$, $\mathsf{L1}(x)$ to 1, and all other symbols to 0. Then (12′) is strictly decreasing, since (1) $\mathrm{Pol}(\mathsf{L1}(y)) = 1 \geq {}^1\!/\!{}_3 \cdot \mathcal{S}um(\mathsf{L1}(\mathsf{triple}(y))) + {}^2\!/\!{}_3 \cdot \mathcal{S}um(\mathsf{loop2}(\mathsf{triple}(y))) = {}^1\!/\!{}_3$ and (2) $\mathrm{Pol}(\mathsf{L1}(y)) = 1 > \mathcal{S}um(\mathsf{loop2}(\mathsf{triple}(y))) = 0$. Similarly, (11′) is also strictly decreasing and we can remove all annotations from this ADP problem. One can find similar interpretations to delete the remaining annotations also from the other remaining sub-problems. This proves `AST` for $\mathcal{DP}(\mathcal{R}_{\mathsf{alg}})$, and hence for $\mathcal{R}_{\mathsf{alg}}$.

Adaption for `bAST`: To adapt the reduction pair processor to `bAST`, we only have to require the conditions of Thm. 36 for the second component $\mathcal{P}$ of a basic ADP problem $(\mathcal{I}, \mathcal{P})$. So the reachability component $\mathcal{I}$ is needed to determine which rules are usable in the usable rules processor, but it does not result in any additional constraints for the reduction pair processor. Thus, proving `bAST` is never harder than proving `AST`, since the second component changes in the same way for `AST` and `bAST` in all processors except for the usable rules processor, which is not applicable for `AST`. The conditions of Thm. 36 ensure that to prove `AST`, infinitely many (**at**) or (**af**) steps with ADPs from $\mathcal{P}_>$ do not have to be regarded anymore and thus, we can remove their annotations in $\mathcal{P}$. However, these ADPs may still be applied in finitely many initial (**at**) or (**af**) steps. Thus, similar to the dependency graph processor, we have to keep the original annotated ADPs from $\mathcal{P}_>$ in the reachability component $\mathcal{I}$.

Theorem 39 (Reduction Pair Processor for `bAST`). *Let* $\mathrm{Pol} : \mathcal{T}^\# \to \mathbb{N}[\mathcal{V}]$ *be a multilinear polynomial interpretation and let* $\mathcal{P} = \mathcal{P}_\geq \uplus \mathcal{P}_>$ *with* $\mathcal{P}_> \neq \varnothing$ *satisfy the conditions of Thm. 36. Then* $\mathrm{Proc}_{\mathtt{RP}}(\mathcal{I}, \mathcal{P}) = \{(\mathcal{I} \cup \mathcal{P}_>, \mathcal{P}_\geq \cup \flat(\mathcal{P}_>))\}$ *is sound and complete for* `bAST`*.*

Example 40. If we only want to prove `bAST` of $\mathcal{R}_{\mathsf{alg}}$, then the application of the reduction pair processor is easier than in Ex. 38, as we have less constraints. For instance, consider the basic ADP problem from Ex. 33 which results from $(\{(11'), (12'')\}, \{(16), \flat(11)$ - $\flat(15), \flat(17)\})$ by setting the flags of all ADPs except the double- and triple-ADPs $\flat(14), \flat(15), (16), \flat(17)$ to false. When using the polynomial interpretation $\mathrm{Pol}(\mathsf{T}(x)) = x$, $\mathrm{Pol}(\mathsf{s}(x)) = x + 1$, $\mathrm{Pol}(\mathsf{double}(x)) = 2x$, and $\mathrm{Pol}(\mathsf{triple}(x)) = 3x$, the ADP (16) is strictly decreasing and $\flat(14)$ - $\flat(17)$ are weakly decreasing. Thus, we can remove all annotations without having to regard any of the other (probabilistic) ADPs. In contrast, when proving AST instead of bAST, all ADPs in the corresponding ADP problem $\{(16), \flat(11) - \flat(15), \flat(17)\}$ have the flag true and thus, here we have to find a polynomial interpretation which also makes the ADPs $\flat(11)$ - $\flat(13)$ weakly decreasing.

4.5 Probability Removal Processor

Finally, in proofs with the ADP framework, one may obtain ADP problems $\mathcal{P}$ with a non-probabilistic structure, i.e., every ADP has the form $\ell \to \{1 : r\}^m$. Then the *probability removal processor* allows us to switch to ordinary (non-probabilistic) DPs. Ordinary DP problems for termination of TRSs have two components $(\mathcal{D}, \mathcal{R})$: a set of dependency pairs $\mathcal{D}$, i.e., rules with annotations only at the roots of both sides, and a TRS $\mathcal{R}$ containing rules that can be used below the annotations. Such a DP problem is considered to be *(innermost) non-terminating* if there exists an *infinite chain* $t_0, t_1, t_2, \ldots$ with $t_i \to_{\mathcal{D}} \circ \to_{\mathcal{R}}^* t_{i+1}$ $(t_i \xrightarrow{\mathsf{i}}_{\mathcal{D},\mathcal{R}} \circ \xrightarrow{\mathsf{i}}_{\mathcal{R}}^* t_{i+1})$ for all $i \in \mathbb{N}$. Here, "$\circ$" denotes composition and $\xrightarrow{\mathsf{i}}_{\mathcal{D},\mathcal{R}}$ is the restriction of $\to_{\mathcal{D}}$ to rewrite steps where the used redex is in $\mathtt{NF}_{\mathcal{R}}$. This definition corresponds to an infinite chain tree consisting of only a single path.

Theorem 41 (Probability Removal Processor for iAST). *Let $\mathcal{P}$ be an ADP problem where every ADP in $\mathcal{P}$ has the form $\ell \to \{1 : r\}^m$. Let $\mathrm{dp}(\mathcal{P}) = \{\ell^\# \to t^\# \mid \ell \to \{1 : r\}^m \in \mathcal{P}, t \trianglelefteq_\# r\}$. Then $\mathcal{P}$ is iAST iff the non-probabilistic DP problem $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is innermost terminating. So the processor $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{P}) = \varnothing$ is sound and complete for iAST iff $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is innermost terminating.*

Adaption for AST and bAST: $\mathrm{Proc}_{\mathtt{PR}}$ works in an analogous way for (b)AST, i.e., for both AST and bAST, we can switch to ordinary DPs for full rewriting. Of course, here the "only if" direction does not hold for bAST because the non-probabilistic DP framework considers arbitrary (possibly non-basic) start terms.

Theorem 42 (Probability Removal Processor for bAST and AST). *Let $\mathcal{P}$ be an ADP problem where every ADP in $\mathcal{P}$ has the form $\ell \to \{1 : r\}^m$. Then $\mathcal{P}$ is AST iff the non-probabilistic DP problem $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is terminating. So the processor $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{P}) = \varnothing$ is sound and complete for AST iff $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is terminating. Similarly, $(\mathcal{I}, \mathcal{P})$ is bAST if $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is terminating. So $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{I}, \mathcal{P}) = \varnothing$ is sound and complete for bAST if $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is terminating.*

4.6 Switching From Full to Innermost AST

In the non-probabilistic DP framework for analyzing termination of TRSs, there is a processor to switch from full to innermost rewriting if the DP problem satisfies certain conditions [17, Thm. 32]. This is useful as the DP framework for innermost termination is more powerful than the one for full termination and in this way, one can switch to the innermost case for certain sub-problems, even if the whole TRS does not belong to any class where innermost termination implies termination. However, the soundness of this processor relies on the minimality property, which does not hold in the probabilistic setting, see Remark 15. Indeed, the following example which corresponds to [45, Ex. 3.15] shows that a similar processor in the ADP framework would be unsound.

Example 43. The ADP problem with $\mathsf{f}(x) \to \{1 : \mathsf{F}(\mathsf{a})\}^{\mathsf{true}}$ and $\mathsf{a} \to \{1 : \mathsf{a}\}^{\mathsf{true}}$ is not AST as we can rewrite $\mathsf{F}(\mathsf{a})$ to itself with the f-ADP. However, it is iAST as in innermost evaluations, we have to rewrite the inner a, which does not terminate but does not use any annotations, i.e., any (**at**) or (**af**) steps. The ADPs are non-overlapping, and left- and right-linear. Thus, Thm. 3 to switch from full to innermost AST cannot be applied on the level of ADP problems.

Hence, for AST of PTRSs that satisfy the conditions of Thm. 3 or 4, one should apply the ADP framework for iAST [31], because its processors are more powerful. But otherwise, one has to use our novel ADP framework for full AST.

5 Conclusion and Evaluation

In this paper, we introduced the first DP framework for AST and bAST of PTRSs, which is based on the existing ADP framework from [31] for iAST. It is particularly useful when analyzing (b)AST of overlapping PTRSs, as for such PTRSs we cannot use the criteria of [30] for classes of PTRSs where iAST implies (b)AST.

Compared to the non-probabilistic DP framework for termination of TRSs [2,17,18,23,24], analyzing AST automatically is significantly more difficult due to the lack of a "minimality property" in the probabilistic setting, which would allow several further processors. Moreover, the ADP framework for PTRSs is restricted to multilinear reduction pairs. The following table compares the ADP frameworks for AST, bAST, and iAST. The parts in *italics* show the differences to the non-probabilistic DP framework. Here, "S" and "C" stand for "sound" and "complete".

Processor	ADP for AST	ADP for bAST	ADP for iAST
Chain Crit.	S & C *for non-duplicating*	S & C *for weakly spare*	S & C
Dep. Graph	S & C	S & C	S & C
Usable Terms	S & C	S & C	S & C
Usable Rules	¬ S *(even with C_ε-Rules)*	S & C	S & C
Reduction Pairs	S & C *(multilinearity)*	S & C *(multilinearity)*	S & C *(multilinearity)*
Probability Removal	S & C	S & C	S & C

For our experimental evaluation, we compared all existing approaches to prove (b)AST of PTRSs. More precisely, we compared our implementation of the novel ADP framework for (b)AST in a new version of AProVE [19] with the old version of AProVE that only implements the techniques from [30,28,31], and with the direct application of polynomial interpretations from [28].[8]

To this end, we extended the existing benchmark set of 118 PTRSs from [30] by 12 new examples including all PTRSs presented in this paper and PTRSs for typical probabilistic algorithms on lists and trees. Of these 130 examples, the direct

[8] In addition, 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 with their technique can be found in [28].

application of polynomials can find 37 (1) AST proofs, *old* AProVE shows AST for 50 (1) PTRSs, and our *new* AProVE version proves AST for 58 (6) examples. In brackets we indicate the number of AST proofs when only regarding the 12 new examples. The 118 benchmarks from [30] lack non-determinism by overlapping rules and thus, here we are only able to prove AST for three more examples than *old* AProVE. In contrast, our new 12 examples contain non-determinism and create random data objects, which are accessed or modified afterwards (see App. A). Our experiments show that our novel ADP framework can for the first time prove AST of such PTRSs. If we consider basic start terms, the numbers rise to 62 (1) for *old* AProVE and 74 (8) for *new* AProVE. 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/ADPFrameworkFullAST.

Reduction pairs were also adapted to disprove reachability [50], and thus, in the future we will also integrate reachability analysis into the ADP framework for bAST. Moreover, we aim to analyze stronger properties like PAST via DPs. Here, we will again start with innermost evaluation, which is easier to analyze. Furthermore, we want to develop methods to automatically disprove (P)AST of PTRSs.

Acknowledgments. This paper is dedicated to Joost-Pieter Katoen whose groundbreaking work on verification of probabilistic programs laid the foundations for this whole research area. His scientific excellence, his enthusiasm in developing outstanding new research results, and his energy and commitment in the establishment of new research projects (like, e.g., the DFG research training group UnRAVeL) are outstanding. While originally we only analyzed “classical” (non-probabilistic) programs, it is due to Joost-Pieter and this research training group that we extended the focus of our research towards probabilistic programs. Joost-Pieter is not only a major inspiration for our work and a fantastic chair of the research training group UnRAVeL, but he is a great and close colleague, and we look forward to many more joint years together in Aachen at the Chair i2.

References

1. Agrawal, S., Chatterjee, K., Novotný, P.: Lexicographic ranking supermartingales: An efficient approach to termination of probabilistic programs. Proc. ACM Program. Lang. **2**(POPL) (2017). https://doi.org/10.1145/3158122
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theor. Comput. Sc. **236**(1-2), 133–178 (2000). https://doi.org/10.1016/S0304-3975(99)00207-8
3. Arts, T., Giesl, J.: A collection of examples for termination of term rewriting using dependency pairs. Tech. rep., RWTH Aachen University (2001), https://verify.rwth-aachen.de/giesl/papers/examples.pdf
4. Avanzini, M., Dal Lago, U., Yamada, A.: On probabilistic term rewriting. Sci. Comput. Program. **185** (2020). https://doi.org/10.1016/j.scico.2019.102338
5. Avanzini, M., Moser, G., Schaper, M.: A modular cost analysis for probabilistic programs. Proc. ACM Program. Lang. **4**(OOPSLA) (2020). https://doi.org/10.1145/3428240
6. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998). https://doi.org/10.1017/CBO9781139172752

7. Batz, K., Kaminski, B.L., Katoen, J.P., Matheja, C., Verscht, L.: A calculus for amortized expected runtimes. Proc. ACM Program. Lang. **7**(POPL) (2023). https://doi.org/10.1145/3571260
8. Beutner, R., Ong, L.: On probabilistic termination of functional programs with continuous distributions. In: Proc. PLDI '21. pp. 1312–1326 (2021). https://doi.org/10.1145/3453483.3454111
9. Bournez, O., Kirchner, C.: Probabilistic rewrite strategies. applications to ELAN. In: Proc. RTA '02. pp. 252–266. LNCS 2378 (2002). https://doi.org/10.1007/3-540-45610-4_18
10. Bournez, O., Garnier, F.: Proving positive almost-sure termination. In: Proc. RTA '05. pp. 323–337. LNCS 3467 (2005). https://doi.org/10.1007/978-3-540-32033-3_24
11. Chatterjee, K., Fu, H., Novotný, P.: Termination analysis of probabilistic programs with martingales. In: Barthe, G., Katoen, J.P., Silva, A. (eds.) Foundations of Probabilistic Programming, p. 221–258. Cambridge University Press (2020). https://doi.org/10.1017/9781108770750.008
12. Dal Lago, U., Grellois, C.: Probabilistic termination by monadic affine sized typing. In: Proc. ESOP '17. pp. 393–419. LNCS 10201 (2017). https://doi.org/10.1007/978-3-662-54434-1_15
13. Dal Lago, U., Faggian, C., Della Rocca, S.R.: Intersection Types and (Positive) Almost-Sure Termination. Proc. ACM Program. Lang. **5**(POPL) (2021). https://doi.org/10.1145/3434313
14. Faggian, C.: Probabilistic rewriting and asymptotic behaviour: On termination and unique normal forms. Log. Methods in Comput. Sci. **18**(2) (2022). https://doi.org/10.46298/lmcs-18(2:5)2022
15. Ferrer Fioriti, L.M., Hermanns, H.: Probabilistic termination: Soundness, completeness, and compositionality. In: Proc. POPL '15. pp. 489–501 (2015). https://doi.org/10.1145/2676726.2677001
16. Frohn, F., Giesl, J.: Analyzing Runtime Complexity via Innermost Runtime Complexity. In: Proc. LPAR '17. pp. 249–228. EPiC 46 (2017). https://doi.org/10.29007/1nbh
17. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Proc. LPAR '04. pp. 301–331. LNCS 3452 (2004). https://doi.org/10.1007/978-3-540-32275-7_21
18. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. J. Autom. Reason. **37**(3), 155–203 (2006). https://doi.org/10.1007/s10817-006-9057-7
19. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with AProVE. J. Autom. Reason. **58**(1), 3–31 (2017). https://doi.org/10.1007/s10817-016-9388-y
20. Giesl, J., Giesl, P., Hark, M.: Computing expected runtimes for constant probability programs. In: Proc. CADE '19. pp. 269–286. LNCS 11716 (2019). https://doi.org/10.1007/978-3-030-29436-6_16
21. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Proc. FOSE '14. pp. 167–181 (2014). https://doi.org/10.1145/2593882.2593900
22. Gutiérrez, R., Lucas, S.: MU-TERM: Verify Termination Properties Automatically (System Description). In: Proc. IJCAR '20. pp. 436–447. LNCS 12167 (2020). https://doi.org/10.1007/978-3-030-51054-1_28
23. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. Inf. Comput. **199**(1-2), 172–199 (2005). https://doi.org/10.1016/j.ic.2004.10.004
24. Hirokawa, N., Middeldorp, A.: Tyrolean termination tool: Techniques and features. Inf. Comput. **205**(4), 474–511 (2007). https://doi.org/10.1016/J.IC.2006.08.010

25. Huang, M., Fu, H., Chatterjee, K., Goharshady, A.K.: Modular verification for almost-sure termination of probabilistic programs. Proc. ACM Program. Lang. **3**(OOPSLA) (2019). https://doi.org/10.1145/3360555
26. Kaminski, B.L., Katoen, J.P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected runtimes of randomized algorithms. J. ACM **65**, 1–68 (2018). https://doi.org/10.1145/3208102
27. Kaminski, B.L., Katoen, J.P., Matheja, C.: Expected runtime analyis by program verification. In: Barthe, G., Katoen, J.P., Silva, A. (eds.) Foundations of Probabilistic Programming, p. 185–220. Cambridge University Press (2020). https://doi.org/10.1017/9781108770750.007
28. Kassing, J.C., Giesl, J.: Proving almost-sure innermost termination of probabilistic term rewriting using dependency pairs. In: Proc. CADE '23. pp. 344–364. LNCS 14132 (2023). https://doi.org/10.1007/978-3-031-38499-8_20
29. Kassing, J.C., Giesl, J.: Proving almost-sure innermost termination of probabilistic term rewriting using dependency pairs. CoRR **abs/2305.11741** (2023). https://doi.org/10.48550/arXiv.2305.11741
30. Kassing, J.C., Frohn, F., Giesl, J.: From innermost to full almost-sure termination of probabilistic term rewriting. In: Proc. FoSSaCS '24. pp. 206–228. LNCS 14575 (2024). https://doi.org/10.1007/978-3-031-57231-9_10
31. Kassing, J.C., Dollase, S., Giesl, J.: A complete dependency pair framework for almost-sure innermost termination of probabilistic term rewriting. In: Proc. FLOPS '24. pp. 62–80. LNCS 14659 (2024). https://doi.org/10.1007/978-981-97-2300-3_4
32. Kassing, J.C., Dollase, S., Giesl, J.: A complete dependency pair framework for almost-sure innermost termination of probabilistic term rewriting. CoRR **abs/2309.00344** (2024). https://doi.org/10.48550/arXiv.2309.00344
33. Kassing, J.C., Vartanyan, G., Giesl, J.: A dependency pair framework for relative termination of term rewriting. In: Proc. IJCAR '24. pp. 360–380. LNCS 14740 (2024). https://doi.org/10.1007/978-3-031-63501-4_19
34. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean termination tool 2. In: Proc. RTA '09. pp. 295–304. LNCS 5595 (2009). https://doi.org/10.1007/978-3-642-02348-4_21
35. Kozen, D.: A probabilistic PDL. J. Comput. Syst. Sci. **30**(2), 162–178 (1985). https://doi.org/10.1016/0022-0000(85)90012-1
36. Lankford, D.S.: On proving term rewriting systems are Noetherian. Memo mtp-3, math. dept.,, Louisiana Technical University, Ruston, LA (1979), https://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford_Poly_Term.pdf
37. Leutgeb, L., Moser, G., Zuleger, F.: Automated expected amortised cost analysis of probabilistic data structures. In: Proc. CAV '22. pp. 70–91. LNCS 13372 (2022). https://doi.org/10.1007/978-3-031-13188-2_4
38. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer (2005). https://doi.org/10.1007/B138392
39. McIver, A., Morgan, C., Kaminski, B.L., Katoen, J.P.: A new proof rule for almost-sure termination. Proc. ACM Program. Lang. **2**(POPL) (2018). https://doi.org/10.1145/3158121
40. Meyer, F., Hark, M., Giesl, J.: Inferring expected runtimes of probabilistic integer programs using expected sizes. In: Proc. TACAS '21. pp. 250–269. LNCS 12651 (2021). https://doi.org/10.1007/978-3-030-72016-2_14
41. Moosbrugger, M., Bartocci, E., Katoen, J.P., Kovács, L.: Automated Termination Analysis of Polynomial Probabilistic Programs. In: Proc. ESOP '21. pp. 491–518. LNCS 12648 (2021). https://doi.org/10.1007/978-3-030-72019-3_18

42. Ngo, V.C., Carbonneaux, Q., Hoffmann, J.: Bounded expectations: Resource analysis for probabilistic programs. In: Proc. PLDI '18. pp. 496–512 (2018). https://doi.org/10.1145/3192366.3192394
43. Saheb-Djahromi, N.: Probabilistic LCF. In: Proc. MFCS '78. pp. 442–451. LNCS 64 (1978). https://doi.org/10.1007/3-540-08921-7_92
44. Schröer, P., Batz, K., Kaminski, B.L., Katoen, J., Matheja, C.: A deductive verification infrastructure for probabilistic programs. Proc. ACM Program. Lang. **7**(OOPSLA2), 2052–2082 (2023). https://doi.org/10.1145/3622870
45. Thiemann, R.: The DP Framework for Proving Termination of Term Rewriting. Ph.D. thesis, RWTH Aachen University (2007), https://verify.rwth-aachen.de/da/thiemann-diss.pdf
46. Toyama, Y.: Counterexamples to termination for the direct sum of term rewriting systems. Inf. Process. Lett. **25**(3), 141–143 (1987). https://doi.org/10.1016/0020-0190(87)90122-0
47. Urbain, X.: Modular and incremental automated termination proofs. J. Autom. Reason. **32**(4), 315–355 (2004). https://doi.org/10.1007/BF03177743
48. Wang, D., Kahn, D.M., Hoffmann, J.: Raising expectations: Automating expected cost analysis with types. Proc. ACM Program. Lang. **4**(ICFP) (2020). https://doi.org/10.1145/3408992
49. Yamada, A., Kusakari, K., Sakabe, T.: Nagoya Termination Tool. In: Proc. RTA-TLCA '14. pp. 466–475. LNCS 8560 (2014). https://doi.org/10.1007/978-3-319-08918-8_32
50. Yamada, A.: Term orderings for non-reachability of (conditional) rewriting. In: Proc. IJCAR '22. pp. 248–267 (2022). https://doi.org/10.1007/978-3-031-10769-6_15

Appendix

In App. A, we present three examples to demonstrate how our novel ADP framework can be used for full rewriting on data structures like lists or trees. App. B contains all proofs for our new contributions and observations.

A Examples

In this section, we show that in contrast to most other techniques for analyzing AST, due to probabilistic term rewriting, our approach is also suitable for the analysis of algorithms on algebraic data structures other than numbers.

A.1 Lists

We start with algorithms on lists. Similar to Alg. 1, the following algorithm first creates a random list, filled with random numbers, and afterwards uses the list for further computation. In general, algorithms that access or modify randomly generated lists can be analyzed by our new ADP framework.

The algorithm below computes the sum of all numbers in the generated list. Here, natural numbers are again represented via the constructors $\mathsf{0}$ and s, and lists are represented via nil (for the empty list) and cons, where, e.g., $\mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{cons}(\mathsf{0}, \mathsf{nil})))$ represents the list $[1, 1, 0]$. The function $\mathsf{createL}(xs)$ adds a prefix of arbitrary length filled with arbitrary natural numbers in front of the list xs. Moreover, $\mathsf{app}(xs, ys)$ concatenates the two lists xs and ys. Finally, for a non-empty list xs of numbers, $\mathsf{sum}(xs)$ computes a singleton list whose only element is the sum of all numbers in xs. So $\mathsf{sum}(\mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{cons}(\mathsf{s}(\mathsf{0}), \mathsf{cons}(\mathsf{0}, \mathsf{nil}))))$ evaluates to $\mathsf{sum}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{nil})$.

$$
\begin{aligned}
\mathsf{init} &\to \{1 : \mathsf{sum}(\mathsf{createL}(\mathsf{nil}))\}\\
\mathsf{addNum}(x, xs) &\to \{1/2 : \mathsf{cons}(x, xs), 1/2 : \mathsf{addNum}(\mathsf{s}(x), xs)\}\\
\mathsf{createL}(xs) &\to \{1/2 : \mathsf{addNum}(\mathsf{0}, xs), 1/2 : \mathsf{createL}(\mathsf{addNum}(\mathsf{0}, xs))\}\\
\mathsf{plus}(\mathsf{0}, y) &\to \{1 : y\}\\
\mathsf{plus}(\mathsf{s}(x), y) &\to \{1 : \mathsf{s}(\mathsf{plus}(x, y))\}\\
\mathsf{sum}(\mathsf{cons}(x, \mathsf{nil})) &\to \{1 : \mathsf{cons}(x, \mathsf{nil})\}\\
\mathsf{sum}(\mathsf{cons}(x, \mathsf{cons}(y, ys))) &\to \{1 : \mathsf{sum}(\mathsf{cons}(\mathsf{plus}(x, y), ys))\}\\
\mathsf{sum}(\mathsf{app}(xs, \mathsf{cons}(x, \mathsf{cons}(y, ys)))) &\to \{1 : \mathsf{sum}(\mathsf{app}(xs, \mathsf{sum}(\mathsf{cons}(x, \mathsf{cons}(y, ys)))))\}\\
\mathsf{app}(\mathsf{cons}(x, xs), ys) &\to \{1 : \mathsf{cons}(x, \mathsf{app}(xs, ys))\}\\
\mathsf{app}(\mathsf{nil}, ys) &\to \{1 : ys\}\\
\mathsf{app}(xs, \mathsf{nil}) &\to \{1 : xs\}
\end{aligned}
$$

Note that the left-hand sides of the two rules $\mathsf{app}(\mathsf{nil}, ys) \to \{1 : ys\}$ and $\mathsf{app}(xs, \mathsf{nil}) \to \{1 : xs\}$ overlap and moreover, the last sum-rule overlaps with the first app-rule. Hence, we cannot use the techniques from [30] to analyze full AST of this PTRS. Furthermore, there exists no polynomial ordering that proves AST for this example directly (i.e., without the use of DPs), because the left-hand side of the last sum-rule is embedded in its right-hand side. With our new ADP framework, AProVE can now prove AST of this example automatically.

Next, consider the following adaption of this example. Here, we only create lists of even numbers.

$$
\begin{aligned}
\mathsf{init} &\to \{1 : \mathsf{sum}(\mathsf{createL}(\mathsf{nil}))\}\\
\mathsf{addNum}(x, xs) &\to \{1/2 : \mathsf{cons}(\mathsf{plus}(x, x), xs), 1/2 : \mathsf{addNum}(\mathsf{s}(x), xs)\}\\
\mathsf{createL}(xs) &\to \{1/2 : \mathsf{addNum}(\mathsf{0}, xs), 1/2 : \mathsf{createL}(\mathsf{addNum}(\mathsf{0}, xs))\}\\
\mathsf{plus}(\mathsf{0}, y) &\to \{1 : y\}\\
\mathsf{plus}(\mathsf{s}(x), y) &\to \{1 : \mathsf{s}(\mathsf{plus}(x, y))\}\\
\mathsf{sum}(\mathsf{cons}(x, \mathsf{nil})) &\to \{1 : \mathsf{cons}(x, \mathsf{nil})\}\\
\mathsf{sum}(\mathsf{cons}(x, \mathsf{cons}(y, xs))) &\to \{1 : \mathsf{sum}(\mathsf{cons}(\mathsf{plus}(x, y), xs))\}\\
\mathsf{sum}(\mathsf{app}(xs, \mathsf{cons}(x, \mathsf{cons}(y, ys)))) &\to \{1 : \mathsf{sum}(\mathsf{app}(xs, \mathsf{sum}(\mathsf{cons}(x, \mathsf{cons}(y, ys)))))\}\\
\mathsf{app}(\mathsf{cons}(x, xs), ys) &\to \{1 : \mathsf{cons}(x, \mathsf{app}(xs, ys))\}\\
\mathsf{app}(\mathsf{nil}, ys) &\to \{1 : ys\}\\
\mathsf{app}(xs, \mathsf{nil}) &\to \{1 : xs\}
\end{aligned}
$$

Due to the subterm $\mathsf{plus}(x, x)$ in the right-hand side, the addNum-rule is duplicating. Hence, we cannot use the ADP framework for `AST`. However, the PTRS is weakly spare, as the arguments of plus cannot contain defined function symbols if we start with a basic term. Hence, AProVE can use the ADP framework for `bAST` and successfully prove `bAST` of this example.

A.2 Trees

As another example, our new ADP framework can also deal with trees. In the following algorithm (adapted from [3]), we consider binary trees represented via leaf and $\mathsf{tree}(x, y)$, where $\mathsf{concat}(x, y)$ replaces the rightmost leaf of the tree x by y. The algorithm first creates two random trees and then checks whether the first tree has less leaves than the second one.

$$
\begin{aligned}
\mathsf{init} &\to \{1 : \mathsf{lessleaves}(\mathsf{createT}(\mathsf{leaf}), \mathsf{createT}(\mathsf{leaf}))\}\\
\mathsf{concat}(\mathsf{leaf}, y) &\to \{1 : y\}\\
\mathsf{concat}(\mathsf{tree}(u, v), y) &\to \{1 : \mathsf{tree}(u, \mathsf{concat}(v, y))\}\\
\mathsf{lessleaves}(x, \mathsf{leaf}) &\to \{1 : \mathsf{false}\}\\
\mathsf{lessleaves}(\mathsf{leaf}, \mathsf{tree}(x, y)) &\to \{1 : \mathsf{true}\}\\
\mathsf{lessleaves}(\mathsf{tree}(u, v), \mathsf{tree}(x, y)) &\to \{1 : \mathsf{lessleaves}(\mathsf{concat}(u, v), \mathsf{concat}(x, y))\}\\
\mathsf{createT}(xs) &\to \{1 : xs\}\\
\mathsf{createT}(xs) &\to \{1/3 : xs, 1/3 : \mathsf{createT}(\mathsf{tree}(xs, \mathsf{leaf})), 1/3 : \mathsf{createT}(\mathsf{tree}(\mathsf{leaf}, xs))\}
\end{aligned}
$$

Note that the last two rules are overlapping. Again, our new ADP framework is able to prove `AST` for this example, while both [30] and the direct application of polynomial interpretations fail.

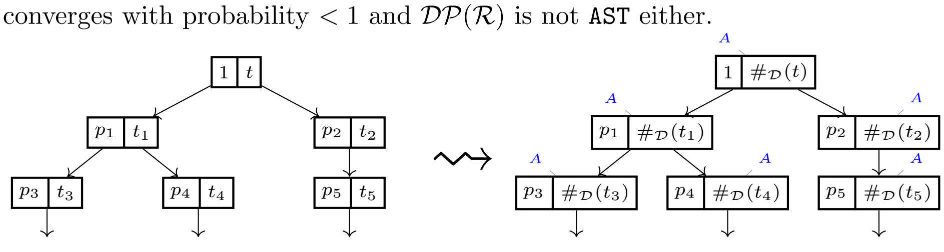
B Proofs

In this section, we give all proofs for our new results and observations. In the following, let $L(x) = (p^{\mathfrak{T}}_x, t^{\mathfrak{T}}_x)$ denote the labeling of the node x in the chain tree $\mathfrak{T}$. We say that a CT (or RST) $\mathfrak{T}$ *converges (or terminates) with probability* $p \in \mathbb{R}$ if we have $|\mathfrak{T}|_{\mathtt{Leaf}} = p$. Moreover, we often write $\#_\varepsilon(t)$ instead of $\#_{\{\varepsilon\}}(t)$ and $\#_{\mathcal{D}}(t)$ instead of $\#_{\mathrm{Pos}_{\mathcal{D}}(t)}(t)$ to annotate all defined symbols in a term t. We can now start with the proof of the sound and complete chain criterion for `AST`.

Theorem 13 (Chain Criterion for AST). *A non-duplicating PTRS $\mathcal{R}$ is* AST *iff $\mathcal{DP}(\mathcal{R})$ is* AST*.*

Proof.

<u>*Soundness:*</u> Assume that $\mathcal{R}$ is not AST. Then, there exists an $\mathcal{R}$-RST $\mathfrak{T} = (V, E, L)$ whose root is labeled with $(1 : t)$ for some term $t \in \mathcal{T}$ that converges with probability < 1. We will construct a $\mathcal{DP}(\mathcal{R})$-CT $\mathfrak{T}' = (V, E, L', V \setminus \mathtt{Leaf}^{\mathfrak{T}})$ with the same underlying tree structure and an adjusted labeling such that $p_x^{\mathfrak{T}} = p_x^{\mathfrak{T}'}$ for all $x \in V$, where all the inner nodes are in A. Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}|_{\mathtt{Leaf}} = |\mathfrak{T}'|_{\mathtt{Leaf}}$. To be precise, the set of leaves in $\mathfrak{T}$ is equal to the set of leaves in $\mathfrak{T}'$, and they have the same probabilities. Since $|\mathfrak{T}|_{\mathtt{Leaf}} < 1$, we thus have $|\mathfrak{T}'|_{\mathtt{Leaf}} < 1$. Hence, there exists a $\mathcal{DP}(\mathcal{R})$-CT $\mathfrak{T}'$ that converges with probability < 1 and $\mathcal{DP}(\mathcal{R})$ is not AST either.

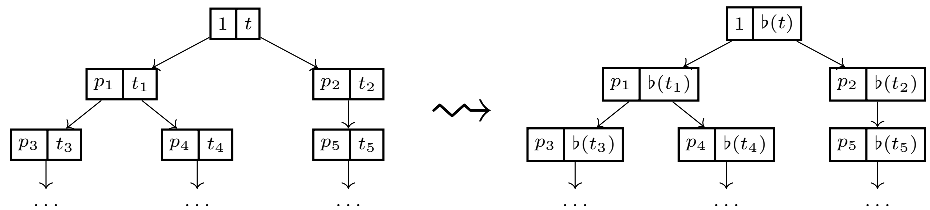


We label all nodes $x \in V$ in $\mathfrak{T}'$ with $\#_{\mathcal{D}}(t_x)$, where t_x is the term for the node x in $\mathfrak{T}$. The annotations ensure that we rewrite with Case (**at**) of Def. 12 so that the node x is contained in A. We only have to show that $\mathfrak{T}'$ is indeed a valid CT, i.e., that the edge relation represents valid rewrite steps with $\hookrightarrow_{\mathcal{DP}(\mathcal{R})}$. Let $x \in V \setminus \mathtt{Leaf}$ and $xE = \{y_1, \ldots, y_k\}$ be the set of its successors. Since x is not a leaf, we have $t_x \to_{\mathcal{R}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \ldots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$. This means that there is a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{R}$, a position π, and a substitution σ such that $t_x|_\pi = \ell\sigma$. Furthermore, we have $t_{y_j} = t_x[r_j\sigma]_\pi$ for all $1 \leq j \leq k$.

The corresponding ADP for the rule is $\ell \to \{p_1 : \#_{\mathcal{D}}(r_1), \ldots, p_k : \#_{\mathcal{D}}(r_k)\}^{\mathsf{true}}$. Furthermore, $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(\#_{\mathcal{D}}(t_x))$ as all defined symbols are annotated in $\#_{\mathcal{D}}(t_x)$. Hence, we can rewrite $\#_{\mathcal{D}}(t_x)$ with $\ell \to \{p_1 : \#_{\mathcal{D}}(r_1), \ldots, p_k : \#_{\mathcal{D}}(r_k)\}^{\mathsf{true}}$, using the position π, the substitution σ, and Case (**at**) of Def. 12 applies. Furthermore, we take some VRF $(\varphi_j)_{1 \leq j \leq k}$ that is surjective on the positions of the variables in the right-hand side, i.e., for all $1 \leq j \leq k$ and all positions $\tau \in \mathrm{Pos}_{\mathcal{V}}(r_j)$ there exists a $\tau' \in \mathrm{Pos}_{\mathcal{V}}(\ell)$ such that $\varphi_j(\tau') = \tau$. Such a VRF must exist, since $\mathcal{R}$ is non-duplicating. We have $\#_{\mathcal{D}}(t_x) \hookrightarrow_{\mathcal{DP}(\mathcal{R})} \{p_1 : \#_{\mathcal{D}}(t_{y_1}), \ldots, p_k : \#_{\mathcal{D}}(t_{y_k})\}$ since by rewriting with (**at**) we get $\#_{\mathcal{D}}(t_x)[\#_{\Phi_j}(r_j\sigma)]_\pi = \#_{\mathcal{D}}(t_{y_j})$ with Φ_j defined as in Def. 12. Note that since the used VRF is surjective on the variable positions of the right-hand side, we do not remove any annotation in the substitution. Furthermore, we annotated all defined symbols in r_j. Thus, we result in $\#_{\mathcal{D}}(t_{y_j})$ where all defined symbols are annotated again.

<u>*Completeness:*</u> Assume that $\mathcal{DP}(\mathcal{R})$ is not AST. Then, there exists a $\mathcal{DP}(\mathcal{R})$-CT $\mathfrak{T} = (V, E, L, A)$ whose root is labeled with $(1 : t)$ for some annotated term $t \in \mathcal{T}^\#$ that converges with probability < 1. We will construct an $\mathcal{R}$-RST $\mathfrak{T}' = (V, E, L')$ with the same underlying tree structure and an adjusted labeling such that $p_x^{\mathfrak{T}} = p_x^{\mathfrak{T}'}$ for all $x \in V$. Since the tree structure and the probabilities are the same, we then

get $|\mathfrak{T}'|_{\texttt{Leaf}} = |\mathfrak{T}|_{\texttt{Leaf}} < 1$. Therefore, there exists an $\mathcal{R}$-RST $\mathfrak{T}'$ that converges with probability < 1. Hence, $\mathcal{R}$ is not AST either.



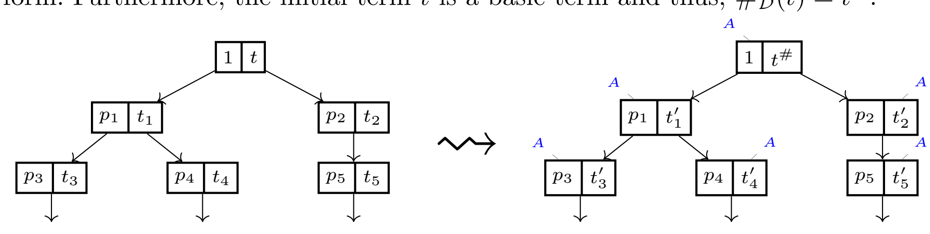
We label all nodes $x \in V$ in $\mathfrak{T}'$ with $\flat(t_x)$, where t_x is the term for the node x in $\mathfrak{T}$, i.e., we remove all annotations. We only have to show that $\mathfrak{T}'$ is indeed a valid RST, i.e., that the edge relation represents valid rewrite steps with $\to_{\mathcal{R}}$, but this follows directly from the fact that if we remove all annotations in Def. 12, then we get the ordinary probabilistic term rewriting relation again. □

Next, we prove the sound and complete chain criterion for bAST. In the following, for two (possibly annotated) terms s, t we define $s \doteq t$ if $\flat(s) = \flat(t)$.

Theorem 14 (Chain Criterion for bAST). *A weakly spare PTRS $\mathcal{R}$ is* bAST *iff* $(\varnothing, \mathcal{DP}(\mathcal{R}))$ *is* bAST.

Proof. Since the reachability component of the canonical basic ADP problem is empty, we just consider $\mathcal{DP}(\mathcal{R})$-CTs.

Soundness: We use the same construction as for AST but the definition of the terms in the new $\mathcal{DP}(\mathcal{R})$-CT is slightly different. Previously, all defined symbols were annotated. Now, we do not annotate all of them, but we may remove annotations from defined symbols if the corresponding subterm at this position is in normal form. Furthermore, the initial term t is a basic term and thus, $\#_{\mathcal{D}}(t) = t^{\#}$.



We construct the new labeling L' for the $\mathcal{DP}(\mathcal{R})$-CT inductively such that for all inner nodes $x \in V \setminus \texttt{Leaf}$ with children nodes $xE = \{y_1, \ldots, y_k\}$ we have $t'_x \hookrightarrow_{\mathcal{DP}(\mathcal{R})} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \ldots, \frac{p_{y_k}}{p_x} : t'_{y_k}\}$. Let $X \subseteq V$ be the set of nodes x where we have already defined the labeling $L'(x)$. Furthermore, for any term $t \in \mathcal{T}$ let $\text{Pos}_{\text{Poss}}(t, \mathcal{R}) = \{\pi \mid \pi \in \text{Pos}_{\mathcal{D}}(t), t|_\pi \notin \texttt{NF}_{\mathcal{R}}\}$. During our construction, we ensure that the following property holds:

$$\text{For every node } x \in X \text{ we have } t_x \doteq t'_x \text{ and } \text{Pos}_{\text{Poss}}(t_x, \mathcal{R}) \subseteq \text{Pos}_{\mathcal{D}^\#}(t'_x). \quad (21)$$

This means that the corresponding term t_x for the node x in $\mathfrak{T}$ has the same structure as the term t'_x in $\mathfrak{T}'$, and additionally, at least all possible redexes in t_x are annotated in t'_x. The annotations ensure that we rewrite with Case (**at**) of Def. 12 so that the node x is contained in A. We label the root of $\mathfrak{T}'$ with $t^{\#}$. Here, we have $t \doteq t^{\#}$ and $\mathrm{Pos}_{\mathrm{Poss}}(t, \mathcal{R}) = \{\varepsilon\} = \mathrm{Pos}_{\mathcal{D}^{\#}}(t^{\#})$, since t is a basic term. As long as there is still an inner node $x \in X$ such that its successors are not contained in X, we do the following. Let $xE = \{y_1, \ldots, y_k\}$ be the set of its successors. We need to define the corresponding terms $t'_{y_1}, \ldots, t'_{y_k}$ for the nodes $y_1, \ldots, y_k$. Since x is not a leaf, we have $t_x \to_{\mathcal{R}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \ldots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$, i.e., there is a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{R}$, a position π, and a substitution σ such that $t_x|_\pi = \ell\sigma$. Furthermore, we have $t_{y_j} = t_x[r_j\sigma]_\pi$ for all $1 \leq j \leq k$.

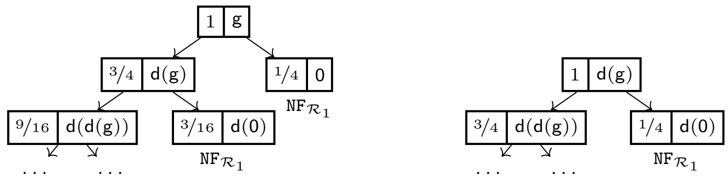
The corresponding ADP for the rule is $\ell \to \{p_1 : \#_{\mathcal{D}}(r_1), \ldots, p_k : \#_{\mathcal{D}}(r_k)\}^{\mathsf{true}}$. Furthermore, $\pi \in \mathrm{Pos}_{\mathrm{Poss}}(t_x, \mathcal{R}) \subseteq_{(IH)} \mathrm{Pos}_{\mathcal{D}^{\#}}(t'_x)$ and $t_x \doteq_{(IH)} t'_x$. Hence, we can rewrite t'_x with $\ell \to \{p_1 : \#_{\mathcal{D}}(r_1), \ldots, p_k : \#_{\mathcal{D}}(r_k)\}^{\mathsf{true}}$, using the position π and the substitution σ, and Case (**at**) of Def. 12 applies. Additionally, we use a VRF $(\varphi_j)_{1 \leq j \leq k}$ that is surjective on the positions of those variables that occur at least as often in the left-hand side as in the right-hand sides. The positions of all other variables (i.e., all variables that are duplicated) are mapped to $\perp$. Note that $\mathcal{R}$ is weakly spare, hence such variables can only be instantiated by normal forms. We get $t'_x \hookrightarrow_{\mathcal{DP}(\mathcal{R})} \{p_1 : t'_{y_1}, \ldots, p_k : t'_{y_k}\}$ with $t'_{y_j} = t'_x[\#_{\Phi_j}(r_j\sigma)]_\pi$ by (**at**) with Φ_j defined as in Def. 12. This means that we have $t_{y_j} \doteq t'_{y_j}$. It remains to prove $\mathrm{Pos}_{\mathrm{Poss}}(t_{y_j}, \mathcal{R}) \subseteq \mathrm{Pos}_{\mathcal{D}^{\#}}(t'_{y_j})$ for all $1 \leq j \leq k$. For all positions $\tau \in \mathrm{Pos}_{\mathrm{Poss}}(t_{y_j}, \mathcal{R}) = \mathrm{Pos}_{\mathrm{Poss}}(t_x[r_j\sigma]_\pi, \mathcal{R})$ that are orthogonal or above π, we have $\tau \in \mathrm{Pos}_{\mathrm{Poss}}(t_x, \mathcal{R}) \subseteq_{(IH)} \mathrm{Pos}_{\mathcal{D}^{\#}}(t'_x)$, and all annotations orthogonal or above π remain in t'_{y_j} as they were in t'_x. For all positions $\tau \in \mathrm{Pos}_{\mathrm{Poss}}(t_{y_j}, \mathcal{R}) = \mathrm{Pos}_{\mathrm{Poss}}(t_x[r_j\sigma]_\pi, \mathcal{R})$ that are below π, we have two possibilities: (1) at least the defined root symbol of $t_{y_j}|_\tau$ is inside r_j, and thus $\tau \in \mathrm{Pos}_{\mathcal{D}^{\#}}(t'_{y_j})$, as all defined symbols of r_j are annotated in $t'_{y_j} = t'_x[\#_{\Phi_j}(r_j)\sigma]_\pi$, or (2) τ is below a non-duplicated variable (otherwise the subterm $t_{y_j}|_\tau$ has to be a normal form and thus, $\tau \notin \mathrm{Pos}_{\mathrm{Poss}}(t_{y_j}, \mathcal{R})$), and hence, it is still annotated in t'_{y_j} due to the used VRF. This ends the induction proof for this direction.

Completeness: Completely the same as in the proof of Thm. 13. □

Next, we consider the theorems regarding the processors that we adapted from [31] to our new framework for AST and bAST. We first recapitulate the notion of a *sub-chain tree* from [29].

Definition 44 (Subtree, Sub-CT). *Let $\mathcal{P}$ be an ADP problem and let $\mathfrak{T} = (V, E, L, A)$ be a tree that satisfies Conditions (1)-(5) of a $\mathcal{P}$-CT. Let $W \subseteq V$ be non-empty, weakly connected, and for all $x \in W$ we have $xE \cap W = \varnothing$ or $xE \cap W = xE$. Then, we define the* subtree *(or* sub-CT *if it satisfies Condition (6) as well) $\mathfrak{T}[W]$ by $\mathfrak{T}[W] = (W, E \cap (W \times W), L^W, A \cap (W \setminus W_{\mathtt{Leaf}}))$. Here, $W_{\mathtt{Leaf}}$ denotes the leaves of the tree $G^{\mathfrak{T}[W]} = (W, E \cap (W \times W))$ so that the new set $A \cap (W \setminus W_{\mathtt{Leaf}})$ only contains inner nodes. Let $w \in W$ be the root of $G^{\mathfrak{T}[W]}$. To ensure that the root of our subtree has the probability* 1 *again, we use the labeling $L^W(x) = (\frac{p_x^{\mathfrak{T}}}{p_w^{\mathfrak{T}}} : t_x^{\mathfrak{T}})$ for all nodes $x \in W$. If W contains the root of (V, E), then we call the sub-chain tree* grounded.

Example 45. Reconsider the PTRS $\mathcal{R}_1$ containing the only rule $\mathsf{g} \to \{3/4 : \mathsf{d}(\mathsf{g}), 1/4 : \mathsf{0}\}$. Below one can see the $\mathcal{R}_1$-RST from Sect. 2 (on the left), and the subtree that starts at the node of the term $\mathsf{d}(\mathsf{g})$ (on the right). Note that the probabilities are normalized such that the root has the probability 1 again.



The property of being non-empty and weakly connected ensures that the resulting graph $G^{\mathfrak{T}[W]}$ is a tree again. The property that we either have $xE \cap W = \varnothing$ or $xE \cap W = xE$ ensures that the sum of probabilities for the successors of a node x is equal to the probability for the node x itself.

Next, we recapitulate a lemma and adapt another important lemma from [29]. Afterwards, we prove the theorems on the processors. We start with the *A-partition lemma*. This lemma was proven in [29] (where it was called "P-partition lemma") and still applies to our new ADP problems, since the structure of our CTs is the same as in [28,29].

Lemma 46 (A-Partition Lemma). *Let $\mathcal{P}$ be an ADP problem and let $\mathfrak{T} = (V, E, L, A)$ be a $\mathcal{P}$-CT that converges with probability < 1. Assume that we can partition $A = A_1 \uplus A_2$ such that every sub-CT that only contains A-nodes from A_1 converges with probability 1. Then there is a grounded sub-CT $\mathfrak{T}'$ that converges with probability < 1 such that every infinite path has an infinite number of nodes from A_2.*

Proof. See [29], as this proof does not depend on the used rewrite strategy but just on the structure of a chain tree. □

Next, we adapt the proof of the *starting lemma* from [29,32]. It shows that w.l.o.g., we can assume that we label the root of our CT with $(1 : t)$ for an annotated term t such that $\mathrm{Pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$, i.e., only the root is annotated. This obviously holds for `bAST`, but the starting lemma shows that it can also be assumed for `AST`. For the following proofs, we extend Def. 12 by the missing $(\mathbf{nf})$ case. As discussed in Sect. 3, this does not change the definition of `AST` for chain trees, but it allows us to still perform the same rewrite steps in a CT if annotations are removed.

Definition 47 (Rewriting with ADPs Including (nf)). *Let $\mathcal{P}$ be an ADP problem. A term $s \in \mathcal{T}^\#$* rewrites *with $\mathcal{P}$ to $\mu = \{p_1 : t_1, \ldots, p_k : t_k\}$ (denoted $s \hookrightarrow_{\mathcal{P}} \mu$) if there are an ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$, a VRF $(\varphi_j)_{1 \le j \le k}$ for this ADP, a substitution σ, and a position $\pi \in \mathrm{Pos}_{\mathcal{D} \cup \mathcal{D}^\#}(s)$ such that $\flat(s|_\pi) = \ell\sigma$, and for all $1 \le j \le k$ we have*

$$
\begin{array}{llr}
t_j = s[\#_{\Phi_j}(r_j\sigma)]_\pi & \text{if } \pi \in \mathrm{Pos}_{\mathcal{D}^\#}(s) \text{ and } m = \mathsf{true} & (\mathbf{at})\\
t_j = \flat^\uparrow_\pi(\, s[\#_{\Phi_j}(r_j\sigma)]_\pi) & \text{if } \pi \in \mathrm{Pos}_{\mathcal{D}^\#}(s) \text{ and } m = \mathsf{false} & (\mathbf{af})\\
t_j = s[\#_{\Psi_j}(r_j\sigma)]_\pi & \text{if } \pi \notin \mathrm{Pos}_{\mathcal{D}^\#}(s) \text{ and } m = \mathsf{true} & (\mathbf{nt})\\
t_j = \flat^\uparrow_\pi(\, s[\#_{\Psi_j}(r_j\sigma)]_\pi) & \text{if } \pi \notin \mathrm{Pos}_{\mathcal{D}^\#}(s) \text{ and } m = \mathsf{false} & (\mathbf{nf})
\end{array}
$$

Here, $\Psi_j = \{\varphi_j(\rho).\tau \mid \rho \in \mathrm{Pos}_{\mathcal{V}}(\ell),\ \varphi_j(\rho) \neq \bot,\ \rho.\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(s|_\pi)\}$ and $\Phi_j = \mathrm{Pos}_{\mathcal{D}^\#}(r_j) \cup \Psi_j$.

Lemma 48 (Starting Lemma). *If an ADP problem $\mathcal{P}$ is not* AST*, then there exists a $\mathcal{P}$-CT $\mathfrak{T}$ with $|\mathfrak{T}|_{\mathtt{Leaf}} < 1$ that starts with $(1 : t)$ where $\mathrm{Pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$.*

Proof. We prove the contraposition. Assume that every $\mathcal{P}$-CT $\mathfrak{T}$ converges with probability 1 if it starts with $(1 : t)$ and $\mathrm{Pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$. We now prove that then also every $\mathcal{P}$-CT $\mathfrak{T}$ that starts with $(1 : t)$ for some arbitrary term t converges with probability 1, and thus $\mathcal{P}$ is AST. We prove the claim by induction on the number of annotations in the initial term t.

If t contains no annotation, then the CT starting with $(1 : t)$ is trivially finite (it cannot contain an infinite path, since there are no nodes in A) and hence, it converges with probability 1. Next, if t contains exactly one annotation at position π, then we can ignore everything above the annotation, as we will never use an A-step above the annotated position, and we cannot duplicate or change annotations by rewriting above them, since we use VRFs and not GVRFs. However, for $t|_\pi$ with $\mathrm{Pos}_{\mathcal{D}^\#}(t|_\pi) = \{\varepsilon\}$, we know by our assumption that such a CT converges with probability 1.

Now we regard the induction step, and assume for a contradiction that for a term t with $n > 1$ annotations, there is a CT $\mathfrak{T}$ that converges with probability < 1. Here, our induction hypothesis is that every $\mathcal{P}$-CT $\mathfrak{T}$ that starts with $(1 : t')$, where t' contains m annotations for some $1 \leq m < n$ converges with probability 1. Let $\mathit{\Pi}_1 = \{\tau\}$ and $\mathit{\Pi}_2 = \{\chi \in \mathrm{Pos}_{\mathcal{D}^\#}(t) \mid \chi \neq \tau\}$ for some $\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(t)$ and consider the two terms $\#_{\mathit{\Pi}_1}(t)$ and $\#_{\mathit{\Pi}_2}(t)$, which contain both strictly less than n annotations. By our induction hypothesis, we know that every $\mathcal{P}$-CT that starts with $(1 : \#_{\mathit{\Pi}_1}(t))$ or $(1 : \#_{\mathit{\Pi}_2}(t))$ converges with probability 1. Let $\mathfrak{T}_1 = (V, E, L_1, A_1)$ be the tree that starts with $(1 : \#_{\mathit{\Pi}_1}(t))$ and uses the same rules as we did in $\mathfrak{T}$. (Here, the new definition of the rewrite relation $\hookrightarrow_{\mathcal{P}}$ including the case (**nf**) from Def. 47 is needed in order to ensure that one can still use the same rules as in $\mathfrak{T}$ although we now may have less annotations.)

We can partition A into the sets A_1 and $A_2 = A \setminus A_1$. Note that $\mathfrak{T}_1$ itself may not be a $\mathcal{P}$-CT again, since there might exist paths without an infinite number of A_1-nodes, but obviously every subtree $\mathfrak{T}_1'$ of $\mathfrak{T}_1$ such that every infinite path has an infinite number of A_1-nodes is a $\mathcal{P}$-CT again. Moreover, by extending such a subtree to be grounded, i.e., adding the initial path from the root of $\mathfrak{T}_1$ to $\mathfrak{T}_1'$, we created a $\mathcal{P}$-CT that starts with $\#_{\mathit{\Pi}_1}(t)$, and hence by our induction hypothesis, converges with probability 1. Thus, this also holds for $\mathfrak{T}_1'$.

We want to use the A-partition lemma (Lemma 46) for the tree $\mathfrak{T}$. For this, we have to show that every sub-CT $\mathfrak{T}_1'$ of $\mathfrak{T}$ that only contains A-nodes from A_1 converges with probability 1. But since $\mathfrak{T}_1'$ only contains A-nodes from A_1 it must either contain infinitely many A_1-nodes, and by the previous paragraph it converges with probability 1, or it contains only finitely many A_1-nodes, hence must be finite itself, and also converges with probability 1.

Now, we have shown that the conditions for the A-partition lemma (Lemma 46) are satisfied. Thus, we can apply the A-partition lemma to obtain a grounded sub-CT $\mathfrak{T}'$ of $\mathfrak{T}$ with $|\mathfrak{T}'|_{\mathtt{Leaf}} < 1$ such that on every infinite path, we have an infinite number of A_2 nodes. Let $\mathfrak{T}_2$ be the tree that starts with $\#_{\mathit{\Pi}_2}(t)$ and uses the same rules as we did in $\mathfrak{T}'$. Again, all local properties for a $\mathcal{P}$-CT are satisfied. Additionally, this time we know that every infinite path has an infinite number of A_2-nodes in $\mathfrak{T}'$, hence we also know that the global property for $\mathfrak{T}_2$ is satisfied. This means that $\mathfrak{T}_2$ is a $\mathcal{P}$-CT that starts with $\#_{\Pi_2}(t)$ and with $|\mathfrak{T}_2|_{\mathtt{Leaf}} < 1$. This is our desired contradiction, which proves the induction step. □

As only the root position of the term at the root of the CT is annotated, we can assume that the step from the root of the CT to its children corresponds to a rewrite step at the root of this term. The reason is that we have to eventually rewrite at this root position of the term in a CT that converges with probability < 1. Hence, w.l.o.g. we can start with this root rewrite step.

Next, we adapt the soundness and completeness proofs for the processors from [32] from innermost to full rewriting.

Theorem 20 (Dependency Graph Processor for AST). *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 for* AST.

Proof. Let $\overline{X} = X \cup \flat(\mathcal{P} \setminus X)$ for $X \subseteq \mathcal{P}$.

Completeness: Every $\overline{\mathcal{P}_i}$-CT is also a $\mathcal{P}$-CT with fewer annotations in the terms. So if some $\overline{\mathcal{P}_i}$ is not AST, then there exists a $\overline{\mathcal{P}_i}$-CT $\mathfrak{T}$ that converges with probability < 1. By adding annotations to the terms of the tree, we result in a $\mathcal{P}$-CT that converges with probability < 1 as well. Hence, if $\overline{\mathcal{P}_i}$ is not AST, then $\mathcal{P}$ is not AST either.

Soundness: Let $\mathfrak{G}$ be the $\mathcal{P}$-dependency graph. Suppose that every $\overline{\mathcal{P}_i}$-CT converges with probability 1 for all $1 \leq i \leq n$. We prove that then also every $\mathcal{P}$-CT converges with probability 1. Let $W = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \cup \{\{v\} \subseteq \mathcal{P} \mid v \text{ is not in an SCC of } \mathfrak{G}\}$ be the set of all SCCs and all singleton sets of nodes that do not belong to any SCC. The core steps of this proof are the following:

1. We show that every ADP problem $\overline{X}$ with $X \in W$ is AST.
2. We show that composing SCCs maintains the AST property.
3. We show that for every $X \in W$, the ADP problem $\overline{\bigcup_{X >_{\mathfrak{G}}^* Y} Y}$ is AST by induction on $>_{\mathfrak{G}}$.
4. We conclude that $\mathcal{P}$ must be AST.

Here, for two $X_1, X_2 \in W$ we say that X_2 is a *direct successor* of X_1 (denoted $X_1 >_{\mathfrak{G}} X_2$) if there exist nodes $v \in X_1$ and $w \in X_2$ such that there is an edge from v to w in $\mathfrak{G}$.

1. Every ADP problem $\overline{X}$ with $X \in W$ is AST.
We start by proving the following:

$$\text{Every ADP problem } \overline{X} \text{ with } X \in W \text{ is AST.} \tag{22}$$

To prove (22), note that if X is an SCC, then it follows from our assumption that $\overline{X}$ is AST. If X is a singleton set of a node that does not belong to any SCC, then assume for a contradiction that $\overline{X}$ is not AST. By Lemma 48 there exists an $\overline{X}$-CT $\mathfrak{T} = (V, E, L, A)$ that converges with probability < 1 and starts with $(1 : t)$ where $\mathrm{Pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$ and $\flat(t) = s\theta$ for a substitution θ and some ADP $s \to \{p_1 : t_1, \ldots, p_k : r_k\}^m \in \overline{X}$. If $s \to \ldots \notin X$, then the resulting terms after the first rewrite step contain no annotations anymore and this cannot start a CT that converges with probability < 1. Hence, we have $s \to \ldots \in X$ and thus, $X = \{s \to \ldots\}$, since X is a singleton set. Assume for a contradiction that there exists a node $x \in A$ in $\mathfrak{T}$ that is not the root and introduces new annotations. W.l.o.G., let x be reachable from the root without traversing any other node that introduces new annotations. This means that for the corresponding term t_x for node x there is a $t' \trianglelefteq_\# t_x$ at position τ such that $t' = s\sigma'$ for some substitution σ' and the only ADP $s \to \ldots \in X$ (since this is the only ADP in $\overline{X}$ that contains any annotations in the right-hand side). Let $(z_0, \ldots, z_m)$ with $z_m = x$ be the path from the root to x in $\mathfrak{T}$. The first rewrite step at the root must be $s\theta \hookrightarrow_{\overline{X}} \{p_1 : r_1\theta, \ldots, p_k : r_k\theta\}$. After that, we only use ADPs with the flag `true` below the annotated position that will be used for the rewrite step at node x, as otherwise, the position τ would not be annotated in t_x. Therefore, we must have an $1 \leq j \leq k$ and a $t'' \trianglelefteq_\# r_j$ such that $t''^\# \theta \to^*_{\mathrm{np}(\mathcal{P})} s^\# \sigma'$, which means that there must be a self-loop for the only ADP in X, which is a contradiction to our assumption that X is a singleton consisting of an ADP that is not in any SCC of $\mathfrak{G}$.

Now, we have proven that the $\overline{X}$-CT $\mathfrak{T}$ does not introduce new annotations. By definition of a $\mathcal{P}$-CT, every infinite path must contain an infinite number of nodes in A, i.e., nodes where we rewrite at an annotation. Thus, every path in $\mathfrak{T}$ must be finite, which means that $\mathfrak{T}$ is finite itself, as the tree is finitely branching. But every finite CT converges with probability 1, which is a contradiction to our assumption that $\mathfrak{T}$ converges with probability < 1.

2. Composing SCCs maintains the `AST` property.

Next, we show that composing SCCs maintains the `AST` property. More precisely, we prove the following:

> Let $\hat{X} \subseteq W$ and $\hat{Y} \subseteq W$ such that there are no $X_1, X_2 \in \hat{X}$ and $Y \in \hat{Y}$ which satisfy both $X_1 >^*_{\mathfrak{G}} Y >^*_{\mathfrak{G}} X_2$ and $Y \notin \hat{X}$, and such that there are no $Y_1, Y_2 \in \hat{Y}$ and $X \in \hat{X}$ which satisfy both $Y_1 >^*_{\mathfrak{G}} X >^*_{\mathfrak{G}} Y_2$ and $X \notin \hat{Y}$. If both $\overline{\bigcup_{X \in \hat{X}} X}$ and $\overline{\bigcup_{Y \in \hat{Y}} Y}$ are `AST`, then $\overline{\bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y}$ is `AST`. (23)

To show (23), we assume that both $\overline{\bigcup_{X \in \hat{X}} X}$ and $\overline{\bigcup_{Y \in \hat{Y}} Y}$ are `AST`. Let $\overline{Z} = \overline{\bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y}$. The property in (23) for $\hat{X}$ and $\hat{Y}$ says that a path between two nodes from $\bigcup_{X \in \hat{X}} X$ that only traverses nodes from Z must also be a path that only traverses nodes from $\bigcup_{X \in \hat{X}} X$, so that $\bigcup_{Y \in \hat{Y}} Y$ cannot be used to "create" new paths between two nodes from $\bigcup_{X \in \hat{X}} X$, and vice versa. Assume for a contradiction that $\overline{Z}$ is not `AST`. By Lemma 48 there exists a $\overline{Z}$-CT $\mathfrak{T} = (V, E, L, A)$ that converges with probability < 1 and starts with $(1 : t)$ where $\mathrm{Pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$ and $\flat(t) = s\theta$ for a substitution θ and an ADP $s \to \ldots \in \overline{Z}$.

If $s \to \ldots \notin \bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y$, then the resulting terms contain no annotations anymore and this cannot start a CT that converges with probability < 1. W.l.o.g., we may assume that the ADP that is used for the rewrite step at the root is in $\bigcup_{X \in \hat{X}} X$. Otherwise, we simply swap $\bigcup_{X \in \hat{X}} X$ with $\bigcup_{Y \in \hat{Y}} Y$ in the following.

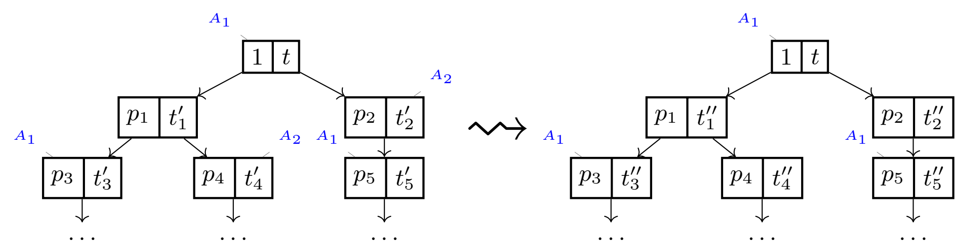
We can partition the set A of our $\overline{Z}$-CT $\mathfrak{T}$ into the sets

- $A_1 := \{x \in A \mid x$ together with the labeling and its successors represents a step with an ADP from $\bigcup_{X \in \hat{X}} X\}$

- $A_2 := A \setminus A_1$

Note that in the case of $x \in A_2$, we know that x together with its successors and the labeling represents a step with an ADP from $\mathcal{P} \setminus \bigcup_{X \in \hat{X}} X$. We know that every $\overline{\bigcup_{Y \in \hat{Y}} Y}$-CT converges with probability 1, since $\overline{\bigcup_{Y \in \hat{Y}} Y}$ is $\mathtt{AST}$. Thus, also every $\overline{\bigcup_{Y \in \hat{Y}} Y \setminus \bigcup_{X \in \hat{X}} X}$-CT converges with probability 1 (as it contains fewer annotations than $\overline{\bigcup_{Y \in \hat{Y}} Y}$). Furthermore, we have $|\mathfrak{T}|_{\mathtt{Leaf}} < 1$ by our assumption. By the A-partition lemma (Lemma 46) we can find a grounded sub $\overline{Z}$-CT $\mathfrak{T}' = (V', E', L', A')$ with $|\mathfrak{T}'|_{\mathtt{Leaf}} < 1$ such that every infinite path has an infinite number of A_1-edges. Since $\mathfrak{T}'$ is a grounded sub-CT of $\mathfrak{T}$ it must also start with $(1 : t)$.

We now construct a $\overline{\bigcup_{X \in \hat{X}} X}$-CT $\mathfrak{T}'' = (V', E', L'', A'')$ with $A_1 \cap A' \subseteq A''$ that has the same underlying tree structure and adjusted labeling such that all nodes get the same probabilities as in $\mathfrak{T}'$. Since the tree structure and the probabilities are the same, we then obtain $|\mathfrak{T}'|_{\mathtt{Leaf}} = |\mathfrak{T}''|_{\mathtt{Leaf}}$. To be precise, the set of leaves in $\mathfrak{T}'$ is equal to the set of leaves in $\mathfrak{T}''$, and every leaf has the same probability. Since $|\mathfrak{T}'|_{\mathtt{Leaf}} < 1$ we thus have $|\mathfrak{T}''|_{\mathtt{Leaf}} < 1$, which is a contradiction to our assumption that $\overline{\bigcup_{X \in \hat{X}} X}$ is $\mathtt{AST}$.



The core idea of this construction is that annotations introduced by rewrite steps at a node $x \in A_2$ are not important for our computation. The reason is that if annotations are introduced using an ADP from $\bigcup_{Y \in \hat{Y}} Y$ that is not contained in $\bigcup_{X \in \hat{X}} X$, then by the prerequisite of (23), we know that such an ADP has no path in the dependency graph to an ADP in $\bigcup_{X \in \hat{X}} X$. Hence, by definition of the dependency graph, we are never able to use these terms for a rewrite step with an ADP from $\bigcup_{X \in \hat{X}} X$ to introduce new annotations. We can therefore apply the non-annotated ADP from $\bigcup_{Y \in \hat{Y}} Y$ to perform the rewrite step.

We now construct the new labeling L'' for the $\overline{\bigcup_{X \in \hat{X}} X}$-CT $\mathfrak{T}''$ recursively. Let $Q \subseteq V$ be the set of nodes where we have already defined the labeling L''. Furthermore, for any term t'_x, let $\mathrm{Junk}_{\hat{X}}(t'_x)$ denote the positions of all annotated subterms $s \trianglelefteq_\# t'_x$ that can never be used for a rewrite step with an ADP from $\hat{X}$, as indicated by the dependency graph. To be precise, we define $\pi \in \mathrm{Junk}_{\hat{X}}(t'_x) :\Leftrightarrow$ there is no $A \in W$ with $A >^*_{\mathfrak{G}} X$ for some $X \in \hat{X}$ such that there is an ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in A$, and a substitution σ with $\#_\varepsilon(t'_x|_\pi) \to^*_{\mathrm{np}(\mathcal{P})} \ell^\# \sigma$. During our construction, we ensure that the following property holds:

$$\text{For every } x \in Q \text{ we have } t'_x \doteq t''_x \text{ and } \mathrm{Pos}_{\mathcal{D}^\#}(t'_x) \setminus \mathrm{Junk}_{\hat{X}}(t'_x) \subseteq \mathrm{Pos}_{\mathcal{D}^\#}(t''_x). \quad (24)$$

We start by setting $t''_v = t'_v$ for the root v of $\mathfrak{T}'$. Here, our property (24) is clearly satisfied. As long as there is still an inner node $x \in Q$ such that its successors are not contained in Q, we do the following. Let $xE = \{y_1, \dots, y_k\}$ be the set of its successors. We need to define the corresponding terms for the nodes $y_1, \dots, y_k$ in $\mathfrak{T}''$. Since x is not a leaf and $\mathfrak{T}'$ is a $\overline{Z}$-CT, we have $t'_x \hookrightarrow_{\overline{Z}} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t'_{y_k}\}$, and hence, we have to deal with the following two cases:

1. If we use an ADP from $\bigcup_{X \in \hat{X}} X$ in $\mathfrak{T}'$, then we perform the rewrite step with the same ADP, the same VRF $(\varphi_j)_{1 \leq j \leq k}$, the same position π, and the same substitution in $\mathfrak{T}''$. Since we have $t'_x \doteq_{(IH)} t''_x$, we also get $t'_{y_j} \doteq t''_{y_j}$ for all $1 \leq j \leq k$. Furthermore, since we rewrite at position π it cannot be in $\mathrm{Junk}_{\hat{X}}(t'_x)$, and hence, if $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(t'_x)$, then also $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(t''_x)$ by (24). Thus, whenever we create annotations in the rewrite step in $\mathfrak{T}'$ (a step with (**af**) or (**at**)), then we do the same in $\mathfrak{T}''$ (the step is also an (**af**) or (**at**) step, respectively), and whenever we remove annotations in the rewrite step in $\mathfrak{T}''$ (a step with (**af**) or (**nf**)), then the same happened in $\mathfrak{T}'$ (the step is also an (**af**) or (**nf**) step). Therefore, we also get $\mathrm{Pos}_{\mathcal{D}^\#}(t'_{y_j}) \setminus \mathrm{Junk}_{\hat{X}}(t'_{y_j}) \subseteq \mathrm{Pos}_{\mathcal{D}^\#}(t''_{y_j})$ for all $1 \leq j \leq k$ and (24) is again satisfied.
2. If we use an ADP from $\mathcal{P} \setminus \bigcup_{X \in \hat{X}} X$ in $\mathfrak{T}'$, and we use the ADP $\ell \to \{p_1 : r_1, \dots, p_k : r_k\}^m$, then we can use $\ell \to \{p_1 : \flat(r_1), \dots, p_k : \flat(r_k)\}^m$ instead, with the same VRF $(\varphi_j)_{1 \leq j \leq k}$, the same position π, and the same substitution. Note that if $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(t'_x)$, then all the annotations introduced by the ADP are in $\mathrm{Junk}_{\hat{X}}(t'_{y_j})$ for all $1 \leq j \leq k$, since the used ADP is not in $\bigcup_{X \in \hat{X}} X$ and by (23) we cannot use another ADP to create a path in the dependency graph to a node in $\bigcup_{X \in \hat{X}} X$ again. Otherwise, we remove the annotations during the application of the rule anyway. Again, (24) is satisfied.

We have now shown that (23) holds.

3. For every $X \in W$, the ADP problem $\overline{\bigcup_{X >^*_{\mathfrak{G}} Y} Y}$ is AST.

Using (22) and (23), by induction on $>_{\mathfrak{G}}$ we now prove that

$$\text{for every } X \in W, \text{ the ADP problem } \overline{\bigcup_{X >^*_{\mathfrak{G}} Y} Y} \text{ is AST.} \tag{25}$$

Note that $>_{\mathfrak{G}}$ is well founded, since $\mathfrak{G}$ is finite.

For the base case, we consider an $X \in W$ that is minimal w.r.t. $>_{\mathfrak{G}}$. Hence, we have $\bigcup_{X >^*_{\mathfrak{G}} Y} Y = X$. By (22), $\overline{X}$ is AST.

For the induction step, we consider an $X \in W$ and assume that $\overline{\bigcup_{Y >^*_{\mathfrak{G}} Z} Z}$ is AST for every $Y \in W$ with $X >^+_{\mathfrak{G}} Y$. Let $\mathtt{Succ}(X) = \{Y \in W \mid X >_{\mathfrak{G}} Y\} = \{Y_1, \dots Y_m\}$ be the set of all direct successors of X. The induction hypothesis states that $\overline{\bigcup_{Y_u >^*_{\mathfrak{G}} Z} Z}$ is AST for all $1 \leq u \leq m$. We first prove by induction that for all $1 \leq u \leq m$, $\overline{\bigcup_{1 \leq i \leq u} \bigcup_{Y_i >^*_{\mathfrak{G}} Z} Z}$ is AST.

In the inner induction base, we have $u = 1$ and hence $\overline{\bigcup_{1 \leq i \leq u} \bigcup_{Y_i >^*_{\mathfrak{G}} Z} Z} = \overline{\bigcup_{Y_1 >^*_{\mathfrak{G}} Z} Z}$. By our outer induction hypothesis we know that $\overline{\bigcup_{Y_1 >^*_{\mathfrak{G}} Z} Z}$ is AST.

In the inner induction step, assume that the claim holds for some $1 \leq u < m$. Then $\overline{\bigcup_{Y_{u+1} >^*_{\mathfrak{G}} Z} Z}$ is AST by our outer induction hypothesis and $\overline{\bigcup_{1 \leq i \leq u} \bigcup_{Y_i >^*_{\mathfrak{G}} Z} Z}$ is AST by our inner induction hypothesis. By (23), we know that then $\overline{\bigcup_{1 \leq i \leq u+1} \bigcup_{Y_i >^*_{\mathfrak{G}} Z} Z}$ is AST as well. The conditions for (23) are clearly satisfied, as we use the reflexive, transitive closure $>_{\mathfrak{G}}^*$ of the direct successor relation in both $\bigcup_{1 \leq i \leq u} \bigcup_{Y_i >_{\mathfrak{G}}^* Z} Z$ and $\bigcup_{Y_{u+1} >_{\mathfrak{G}}^* Z} Z$.

Now we have shown that $\overline{\bigcup_{1 \leq i \leq m} \bigcup_{Y_i >_{\mathfrak{G}}^* Z} Z}$ is AST. We know that $\overline{X}$ is AST by our assumption and that $\overline{\bigcup_{1 \leq i \leq m} \bigcup_{Y_i >_{\mathfrak{G}}^* Z} Z}$ is AST. Hence, by (23) we obtain that $\overline{\bigcup_{X >_{\mathfrak{G}}^* Y} Y}$ AST. Again, the conditions of (23) are satisfied, since X is strictly greater w.r.t. $>_{\mathfrak{G}}^+$ than all Z with $Y_i >_{\mathfrak{G}}^* Z$ for some $1 \leq i \leq m$.

4. $\mathcal{P}$ is AST.
In (25) we have shown that $\overline{\bigcup_{X >_{\mathfrak{G}}^* Y} Y}$ for every $X \in W$ is AST. Let $X_1, \ldots, X_m \in W$ be the maximal elements of W w.r.t. $>_{\mathfrak{G}}$. By induction, one can prove that $\overline{\bigcup_{1 \leq i \leq u} \bigcup_{X_i >_{\mathfrak{G}}^* Y} Y}$ is AST for all $1 \leq u \leq m$ by (23), analogous to the previous induction. Again, the conditions of (23) are satisfied as we use the reflexive, transitive closure of $>_{\mathfrak{G}}$. In the end, we know that $\overline{\bigcup_{1 \leq i \leq m} \bigcup_{X_i >_{\mathfrak{G}}^* Y} Y} = \mathcal{P}$ is AST and this ends the proof. □

Theorem 23 (Dependency Graph Processor for bAST). *For the SCCs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of the $\mathcal{P}$-dependency graph, the processor* $\mathrm{Proc}_{\mathtt{DG}}(\mathcal{I}, \mathcal{P}) = \{(\mathcal{J} \cup \flat(\mathcal{I} \setminus \mathcal{J}), \mathcal{P}_i \cup \flat(\mathcal{P} \setminus \mathcal{P}_i)) \mid 1 \leq i \leq n,\ \mathcal{J} \in \mathcal{P}_i\!\uparrow\}$ *is sound and complete for* bAST.

Proof. Let $\overline{X}^{\mathcal{P}} = X \cup \flat(\mathcal{P} \setminus X)$ for $X \subseteq \mathcal{P}$ and $\overline{X}^{\mathcal{I}} = X \cup \flat(\mathcal{I} \setminus X)$ for $X \subseteq \mathcal{I} \cup \mathcal{P}$.
Completeness: Every $(\overline{\mathcal{J}}^{\mathcal{I}} \cup \overline{\mathcal{P}_i}^{\mathcal{P}})$-CT is also a $(\mathcal{I} \cup \mathcal{P})$-CT with fewer annotations in the terms. So if some $(\overline{\mathcal{J}}^{\mathcal{I}}, \overline{\mathcal{P}_i}^{\mathcal{P}})$ is not bAST, then there exists a $(\overline{\mathcal{J}}^{\mathcal{I}} \cup \overline{\mathcal{P}_i}^{\mathcal{P}})$-CT $\mathfrak{T}$ that converges with probability < 1 and uses $\overline{\mathcal{J}}^{\mathcal{I}} \setminus \overline{\mathcal{P}_i}^{\mathcal{P}}$ only finitely often. By adding annotations to the terms of the tree, we result in an $(\mathcal{I} \cup \mathcal{P})$-CT that converges with probability < 1 as well and uses ADPs from $\mathcal{I} \setminus \mathcal{P}$ only finitely often. Hence, if $(\overline{\mathcal{J}}^{\mathcal{I}}, \overline{\mathcal{P}_i}^{\mathcal{P}})$ is not bAST, then $(\mathcal{I}, \mathcal{P})$ is not bAST either.

Soundness: Let $(\mathcal{I}, \mathcal{P})$ be not bAST. Then there exists an $(\mathcal{I}, \mathcal{P})$-CT $\mathfrak{T}$ that converges with probability < 1, whose root is labeled with $(1 : t^{\#})$ for a basic term t, and $\mathcal{I} \setminus \mathcal{P}$ is used only finitely often. So there exists a depth $H \in \mathbb{N}$, such that we only use ADPs from $\mathcal{P}$ and no ADPs from $\mathcal{I} \setminus \mathcal{P}$ anymore. All the subtrees that start at depth H are $\mathcal{P}$-CTs and one of them needs to converge with probability < 1, since $\mathfrak{T}$ converges with probability < 1. W.l.o.G., let x be the root node of such a subtree that converges with probability < 1 (with x at depth H). We can use the previous proof of Thm. 20 to show that then there exists an SCC $\mathcal{P}_i \subseteq \mathcal{P}$ and a CT $\mathfrak{T}'$ such that $\mathfrak{T}'$ starts with $(1 : t_x)$, where t_x is the term of the node x in $\mathfrak{T}$, $\mathfrak{T}'$ converges with probability < 1, and $\mathfrak{T}'$ is a $\overline{\mathcal{P}_i}^{\mathcal{P}}$-CT. It remains to show that we can reach the root term t_x of $\mathfrak{T}'$ from a basic term in a $(\overline{\mathcal{J}}^{\mathcal{I}} \cup \overline{\mathcal{P}_i}^{\mathcal{P}})$-CT with $\mathcal{J} \in \mathcal{P}_i\!\uparrow$.

Let $\ell \to \mu$ be the ADP used at the root of $\mathfrak{T}$. Since the root of $\mathfrak{T}$ is $t^{\#}$ for a basic term t, the ADP from $\mathcal{P}_i$ applied to t_x must be reachable from $\ell \to \mu$ in the $(\mathcal{I} \cup \mathcal{P})$-dependency graph. Hence, there exists a $\mathcal{J} \in \mathcal{P}_i\!\uparrow$ such that one can reach t_x from $t^{\#}$ by applying only ADPs from $\overline{\mathcal{J}}^{\mathcal{I}} \cup \overline{\mathcal{P}_i}^{\mathcal{P}}$. Therefore, $\mathfrak{T}''$ together with the prefix tree that includes the path from $t^{\#}$ to t_x is a $(\overline{\mathcal{J}}^{\mathcal{I}}, \overline{\mathcal{P}_i}^{\mathcal{P}})$-CT that converges with probability < 1. Hence, $(\overline{\mathcal{J}}^{\mathcal{I}}, \overline{\mathcal{P}_i}^{\mathcal{P}})$ is not bAST. □

Theorem 27 (Usable Terms Processor for AST and bAST). *We call $t \in \mathcal{T}^{\#}$ with* $\mathrm{root}(t) \in \mathcal{D}^{\#}$ usable *w.r.t. an ADP problem $\mathcal{P}$ 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 $\#_{\{\varepsilon\}}(t)\sigma_1 \to^*_{\mathrm{np}(\mathcal{P})} \ell_2^{\#}\sigma_2$. Let $\Delta_{\mathcal{P}}(s) = \{\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(s) \mid s|_\pi$ is usable w.r.t. $\mathcal{P}\}$ and $\mathcal{T}_{\mathtt{UT}}(\mathcal{P}) = \{\ell \to \{p_1 : \#_{\Delta_{\mathcal{P}}(r_1)}(r_1), \ldots, p_k : \#_{\Delta_{\mathcal{P}}(r_k)}(r_k)\}^m \mid \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}\}$. Then $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{P}) = \{\mathcal{T}_{\mathtt{UT}}(\mathcal{P})\}$ is sound and complete for* AST *and* $\mathrm{Proc}_{\mathtt{UT}}(\mathcal{I}, \mathcal{P}) = \{(\mathcal{T}_{\mathtt{UT}}(\mathcal{I} \cup \mathcal{P}), \mathcal{T}_{\mathtt{UT}}(\mathcal{P}))\}$ *is sound and complete for* bAST.

Proof.
Completeness: We only prove this direction for AST. The proof for bAST is completely analogous. Every $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$-CT is also a $\mathcal{P}$-CT with fewer annotations in the terms. So if $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$ is not AST, then there exists a $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$-CT $\mathfrak{T}$ that converges with probability < 1. By adding annotations to the terms of the tree, we result in a $\mathcal{P}$-CT that converges with probability < 1 as well. Hence, if $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$ is not AST, then $\mathcal{P}$ is not AST either.

Soundness: Here, the proofs for AST and bAST differ slightly, similar to the proofs of Thm. 20 and Thm. 23. We start with the proof of AST.

Let $\mathcal{P}$ be not AST. Then by Lemma 48 there exists a $\mathcal{P}$-CT $\mathfrak{T} = (V, E, L, A)$ that converges with probability < 1 whose root is labeled with $(1 : t)$ and $\mathrm{Pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$. We will now create a $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$-CT $\mathfrak{T}' = (V, E, L', A)$, with the same underlying tree structure, and an adjusted labeling such that $p_x^{\mathfrak{T}} = p_x^{\mathfrak{T}'}$ for all $x \in V$. Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}'|_{\mathtt{Leaf}} = |\mathfrak{T}|_{\mathtt{Leaf}} < 1$, and hence $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$ is not AST either.

We construct the new labeling L' for the $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$-CT $\mathfrak{T}'$ recursively. Let $X \subseteq V$ be the set of nodes where we have already defined the labeling L'. During our construction, we ensure that the following property holds for every node $x \in X$:

$$\text{For every } x \in X \text{ we have } t_x \doteq t'_x \text{ and } \mathrm{Pos}_{\mathcal{D}^\#}(t_x) \setminus \mathrm{Junk}(t_x) \subseteq \mathrm{Pos}_{\mathcal{D}^\#}(t'_x). \quad (26)$$

Here, for any term t_x, let $\mathrm{Junk}(t_x)$ be the set of positions that can never be used for a rewrite step with an ADP that contains annotations. To be precise, we define $\pi \in \mathrm{Junk}(t_x) :\Leftrightarrow$ there is no ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$ with annotations and no substitution σ such that $\#_\varepsilon(t_x|_\pi) \to^*_{\mathrm{np}(\mathcal{P})} \ell^\# \sigma$.

We start with the same term t at the root. Here, our property (26) is clearly satisfied. As long as there is still an inner node $x \in X$ such that its successors are not contained in X, we do the following. Let $xE = \{y_1, \ldots, y_k\}$ be the set of its successors. We need to define the terms for the nodes $y_1, \ldots, y_k$ in $\mathfrak{T}'$. Since x is not a leaf and $\mathfrak{T}$ is a $\mathcal{P}$-CT, we have $t_x \hookrightarrow_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \ldots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$. If we performed a step with $\hookrightarrow_{\mathcal{P}}$ using the ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$, the VRF $(\varphi_j)_{1 \leq j \leq k}$, the position π, and the substitution σ in $\mathfrak{T}$, then we can use the ADP $\ell \to \{p_1 : \#_{\Delta_{\mathcal{P}}(r_1)}(r_1), \ldots, p_k : \#_{\Delta_{\mathcal{P}}(r_k)}(r_k)\}^m$ with the same VRF $(\varphi_j)_{1 \leq j \leq k}$, the same position π, and the same substitution σ. Now, we directly get $t_{y_j} \doteq t'_{y_j}$ for all $1 \leq j \leq k$. To prove $\mathrm{Pos}_{\mathcal{D}^\#}(t_{y_j}) \setminus \mathrm{Junk}(t_{y_j}) \subseteq \mathrm{Pos}_{\mathcal{D}^\#}(t'_{y_j})$, note that if $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(t_x) \cap \mathrm{Junk}(t_x)$, then $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$ contains no annotations by definition of $\mathrm{Junk}(t_x)$. Therefore, it does not matter whether we rewrite with case **(at)** or **(nt)** (**(af)** or **(nf)**). Otherwise, if $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(t_x) \setminus \mathrm{Junk}(t_x)$, then the original rule contains the same terms with possibly more annotations, but all missing annotations are in $\mathrm{Junk}(t_x)$ by definition of $\#_{\Delta_{\mathcal{P}}(r_j)}(r_j)$. Thus, we get $\mathrm{Pos}_{\mathcal{D}^\#}(t_{y_j}) \setminus \mathrm{Junk}(t_{y_j}) \subseteq \mathrm{Pos}_{\mathcal{D}^\#}(t'_{y_j})$ for all $1 \leq j \leq k$.

Next, we consider `bAST`. Let $(\mathcal{I}, \mathcal{P})$ be not `bAST`. Then there exists an $(\mathcal{I}, \mathcal{P})$-CT $\mathfrak{T}$ that converges with probability < 1, whose root is labeled with $(1 : t^\#)$ for a basic term t, and $\mathcal{I} \setminus \mathcal{P}$ is used only finitely often. So there exists a depth $H \in \mathbb{N}$, such that we only use ADPs from $\mathcal{P}$ and no ADPs from $\mathcal{I} \setminus \mathcal{P}$ anymore. All the subtrees that start at depth H are $\mathcal{P}$-CTs and one of them needs to converge with probability < 1, since $\mathfrak{T}$ converges with probability < 1. W.l.o.G., let x be the root node of such a subtree that converges with probability < 1 (with x at depth H). We can use the previous proof for `AST` to show that then there exists a $\mathcal{T}_{\mathtt{UT}}(\mathcal{P})$-CT $\mathfrak{T}'$ such that $\mathfrak{T}'$ starts with $(1 : t_x)$, where t_x is the term of the node x in $\mathfrak{T}$, and $\mathfrak{T}'$ converges with probability < 1. It remains to show that we can reach the root term t_x of $\mathfrak{T}'$ from a basic term in a $(\mathcal{T}_{\mathtt{UT}}(\mathcal{I} \cup \mathcal{P}), \mathcal{T}_{\mathtt{UT}}(\mathcal{P}))$-CT. But since we consider both $\mathcal{I}$ and $\mathcal{P}$ in $\mathcal{T}_{\mathtt{UT}}(\mathcal{I} \cup \mathcal{P})$, we can use the same construction as before to show that we can reach a term t'_x with $t_x \doteq t'_x$ and $\mathrm{Pos}_{\mathcal{D}^\#}(t_x) \setminus \mathrm{Junk}(t_x) \subseteq \mathrm{Pos}_{\mathcal{D}^\#}(t'_x)$, which is sufficient. □

Theorem 32 (Usable Rules Processor for `bAST`). *The following processor is sound and complete for* `bAST`*:*

$$\begin{aligned}\mathrm{Proc}_{\mathtt{UR}}(\mathcal{I}, \mathcal{P}) = \{ (\; & \big(\mathcal{I} \cap \mathcal{U}(\mathcal{I} \cup \mathcal{P})\big) \cup \{\ell \to \mu^{\mathsf{false}} \mid \ell \to \mu^m \in \mathcal{I} \setminus \mathcal{U}(\mathcal{I} \cup \mathcal{P})\}, \\ & \big(\mathcal{P} \cap \mathcal{U}(\mathcal{I} \cup \mathcal{P})\big) \cup \{\ell \to \mu^{\mathsf{false}} \mid \ell \to \mu^m \in \mathcal{P} \setminus \mathcal{U}(\mathcal{I} \cup \mathcal{P})\} \;)\}.\end{aligned}$$

Proof. Let $\overline{X} = (X \cap \mathcal{U}(\mathcal{I} \cup \mathcal{P})) \cup \{\ell \to \mu^{\mathsf{false}} \mid \ell \to \mu^m \in X \setminus \mathcal{U}(\mathcal{I} \cup \mathcal{P})\}$.

Completeness: Every $(\overline{\mathcal{I}} \cup \overline{\mathcal{P}})$-CT is also an $(\mathcal{I} \cup \mathcal{P})$-CT with fewer annotations in the terms. If $(\overline{\mathcal{I}}, \overline{\mathcal{P}})$ is not `bAST`, then there exists an $(\overline{\mathcal{I}} \cup \overline{\mathcal{P}})$-CT $\mathfrak{T}$ that converges with probability < 1 and uses ADPs from $\overline{\mathcal{I}} \setminus \overline{\mathcal{P}}$ only finitely often. By adding annotations to the terms of the tree, we result in an $(\mathcal{I} \cup \mathcal{P})$-CT that converges with probability < 1 as well and uses ADPs from $\mathcal{I} \setminus \mathcal{P}$ only finitely often. Hence, if $(\overline{\mathcal{I}}, \overline{\mathcal{P}})$ is not `bAST`, then $(\mathcal{I}, \mathcal{P})$ is not `bAST` either.

Soundness: Assume that $(\mathcal{I}, \mathcal{P})$ is not `bAST`. Then there exists an $(\mathcal{I} \cup \mathcal{P})$-CT that converges with probability < 1, whose root is labeled with $(1 : t^\#)$ for a basic term t, and $\mathcal{I} \setminus \mathcal{P}$ is used only finitely often. As t is basic, in the first rewrite step at the root of the tree, the substitution only instantiates variables of the ADP by normal forms.

By the definition of usable rules, as in the non-probabilistic case, rules $\ell \to \mu^m \in \mathcal{I} \cup \mathcal{P}$ that are not usable (i.e., $\ell \to \mu^m \notin \mathcal{U}(\mathcal{I} \cup \mathcal{P})$) will never be used below an annotated symbol in such an $(\mathcal{I} \cup \mathcal{P})$-CT. Hence, we can also view $\mathfrak{T}$ as an $(\overline{\mathcal{I}} \cup \overline{\mathcal{P}})$-CT that converges with probability < 1 and thus $(\overline{\mathcal{I}}, \overline{\mathcal{P}})$ is not `bAST`. □

In the following, we use the *prefix ordering* ($\pi \leq \tau \Leftrightarrow$ there exists $\chi \in \mathbb{N}^*$ such that $\pi.\chi = \tau$) to compare positions.

Theorem 36 (Reduction Pair Processor for `iAST` & `AST`). *Let* $\mathrm{Pol} : \mathcal{T}^\# \to \mathbb{N}[\mathcal{V}]$ *be a multilinear polynomial interpretation. Let* $\mathcal{P} = \mathcal{P}_\geq \uplus \mathcal{P}_>$ *with* $\mathcal{P}_> \neq \varnothing$ *where:*

(1) $\forall \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P} : \mathrm{Pol}(\ell^\#) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathcal{S}um(r_j)$.

(2) $\forall \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}_> : \exists j \in \{1, \ldots, k\} : \text{Pol}(\ell^\#) > \mathcal{S}um(r_j)$. *If* $m = \mathsf{true}$*, then we additionally have* $\text{Pol}(\ell) \geq \text{Pol}(\flat(r_j))$.

(3) $\forall \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{\mathsf{true}} \in \mathcal{P} : \text{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(\flat(r_j))$.

Then $\text{Proc}_{\mathtt{RP}}(\mathcal{P}) = \{\mathcal{P}_{\geq} \cup \flat(\mathcal{P}_>)\}$ *is sound and complete for* iAST *and* AST*.*

Proof. For the proof for iAST, see [32]. Let $\overline{\mathcal{P}} = \mathcal{P}_{\geq} \cup \flat(\mathcal{P}_>)$.

Completeness: Every $\overline{\mathcal{P}}$-CT is also a $\mathcal{P}$-CT with fewer annotations in the terms. So if $\overline{\mathcal{P}}$ is not AST, then there exists a $\overline{\mathcal{P}}$-CT $\mathfrak{T}$ that converges with probability < 1. By adding annotations to the terms of the tree, we result in a $\mathcal{P}$-CT that converges with probability < 1 as well. Hence, if $\overline{\mathcal{P}}$ is not AST, then $\mathcal{P}$ is not AST either.

Soundness: This proof uses the proof idea for AST from [39]. The core steps of the proof are the following:

(I) We extend the conditions (1), (2), and (3) to rewrite steps instead of just rules (and thus, to edges of a CT).
(II) We create a CT $\mathfrak{T}^{\leq N}$ for any $N \in \mathbb{N}$.
(III) We prove that $|\mathfrak{T}^{\leq N}|_{\mathtt{Leaf}} \geq p_{min}^N$ for any $N \in \mathbb{N}$.
(IV) We prove that $|\mathfrak{T}^{\leq N}|_{\mathtt{Leaf}} = 1$ for any $N \in \mathbb{N}$.
(V) Finally, we prove that $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$.

Here, p_{min} is the minimal probability occurring in $\mathcal{P}$. Parts (II) to (V) remain completely the same as in [29]. We only show that we can adjust part (I) to our new rewrite relation for AST.

(I) We extend the conditions to rewrite steps instead of just rules

We show that the conditions (1), (2), and (3) of the lemma extend to rewrite steps instead of just rules:

(a) If $s \to \{p_1 : t_1, \ldots, p_k : t_k\}$ using a rewrite rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}$ with $\text{Pol}(\ell) \geq \text{Pol}(r_j)$ for some $1 \leq j \leq k$, then we have $Pol(s) \geq Pol(t_j)$.
(b) If $a \hookrightarrow_{\mathcal{P}} \{p_1 : b_1, \ldots, p_k : b_k\}$ using the ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}_>$ at a position $\pi \in \text{Pos}_{\mathcal{D}^\#}(a)$, then $\mathcal{S}um(a) > \mathcal{S}um(b_j)$ for some $1 \leq j \leq k$.
(c) If $s \to \{p_1 : t_1, \ldots, p_k : t_k\}$ using a rewrite rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}$ with $Pol(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(r_j)$, then $Pol(s) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(t_j)$.
(d) If $a \hookrightarrow_{\mathcal{P}} \{p_1 : b_1, \ldots, p_k : b_k\}$ using the ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$, then $\mathcal{S}um(a) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathcal{S}um(b_j)$.

(a) In this case, there exist a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}$ with $\text{Pol}(\ell) \geq \text{Pol}(r_j)$ for some $1 \leq j \leq k$, a substitution σ, and a position π of s such that $s|_\pi = \ell\sigma$ and $t_h = s[r_h\sigma]_\pi$ for all $1 \leq h \leq k$.
We perform structural induction on π. So in the induction base, let $\pi = \varepsilon$. Hence, we have $s = \ell\sigma \to \{p_1 : r_1\sigma, \ldots, p_k : r_k\sigma\}$. By assumption, we have $\text{Pol}(\ell) \geq \text{Pol}(r_j)$ for some $1 \leq j \leq k$. As these inequations hold for all instantiations of the occurring variables, for $t_j = r_j\sigma$ we have

$$\text{Pol}(s) = \text{Pol}(\ell\sigma) \geq \text{Pol}(r_j\sigma) = \text{Pol}(t_j).$$

In the induction step, we have $\pi = i.\pi'$, $s = f(s_1, \ldots, s_i, \ldots, s_n)$, $f \in \Sigma$, $s_i \to \{p_1 : t_{i,1}, \ldots, p_k : t_{i,k}\}$, and $t_j = f(s_1, \ldots, t_{i,j}, \ldots, s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \leq j \leq k$. Then by the induction hypothesis we have $Pol(s_i) \geq Pol(t_{i,j})$. For $t_j = f(s_1, \ldots, t_{i,j}, \ldots, s_n)$ we obtain

$$\begin{aligned}
\mathrm{Pol}(s) &= \mathrm{Pol}(f(s_1, \ldots, s_i, \ldots, s_n)) \\
&= f_{\mathrm{Pol}}(\mathrm{Pol}(s_1), \ldots, \mathrm{Pol}(s_i), \ldots, \mathrm{Pol}(s_n)) \\
&\geq f_{\mathrm{Pol}}(\mathrm{Pol}(s_1), \ldots, \mathrm{Pol}(t_{i,j}), \ldots, \mathrm{Pol}(s_n)) \\
&\qquad \text{(by weak monotonicity of } f_{\mathrm{Pol}} \text{ and } Pol(s_i) \geq Pol(t_{i,j})\text{)} \\
&= \mathrm{Pol}(f(s_1, \ldots, t_{i,j}, \ldots, s_n)) \\
&= \mathrm{Pol}(t_j).
\end{aligned}$$

(b) In this case, there exist an ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}_>$, a VRF $(\varphi_j)_{1 \leq j \leq k}$, a substitution σ, and position $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(a)$ with $\flat(a|_\pi) = \ell\sigma$ and $b_j \doteq a[r_j\sigma]_\pi$. First, assume that $m = \mathsf{true}$. Let $I_1 = \{\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(a) \mid \tau < \pi\}$ be the set of positions of all annotations strictly above π, $I_2 = \{\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(a) \mid \gamma \in \mathrm{Pos}_{\mathcal{V}}(\ell), \pi < \tau \leq \pi.\gamma\}$ be the set of positions of all annotations inside the left-hand side ℓ of the used redex $\ell\sigma$ (but not on the root of ℓ), $I_3 = \{\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(a) \mid \gamma \in \mathrm{Pos}_{\mathcal{V}}(\ell), \pi.\gamma < \tau\}$ be the set of positions of all annotations inside the substitution, and let $I_4 = \{\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(a) \mid \tau \perp \pi\}$ be the set of positions of all annotations orthogonal to π. Furthermore, for each $\tau \in I_1$ let κ_τ be the position such that $\tau.\kappa_\tau = \pi$, and for each $\tau \in I_3$ let γ_τ and ρ_τ be the positions such that $\gamma_\tau \in \mathrm{Pos}_{\mathcal{V}}(\ell)$ and $\pi.\gamma_\tau.\rho_\tau = \tau$. By Requirement (2), there exists a $1 \leq j \leq k$ with $\mathrm{Pol}(\ell^\#) > \mathcal{S}um(r_j) = \sum_{s \trianglelefteq_\# r_j} \mathrm{Pol}(s^\#)$ and, additionally, $\mathrm{Pol}(\ell) \geq \mathrm{Pol}(\flat(r_j))$ since $m = \mathsf{true}$. As these inequations hold for all instantiations of the occurring variables, we have

$$\begin{aligned}
\mathcal{S}um(a) &= \textstyle\sum_{s \trianglelefteq_\# a} \mathrm{Pol}(s^\#) \\
&= \mathrm{Pol}(\#_\varepsilon(s|_\pi)) + \textstyle\sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_2} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\quad + \textstyle\sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\geq \mathrm{Pol}(\#_\varepsilon(s|_\pi)) + \textstyle\sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\quad + \textstyle\sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\qquad \text{(removed } I_2\text{)} \\
&= \mathrm{Pol}(\#_\varepsilon(\ell)\sigma) + \textstyle\sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\quad + \textstyle\sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\qquad \text{(as } \#_\varepsilon(s|_\pi) = \#_\varepsilon(\ell)\sigma\text{)} \\
&> \textstyle\sum_{s \trianglelefteq_\# r_j} \mathrm{Pol}(\#_\varepsilon(s)\sigma) + \sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\quad + \textstyle\sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\qquad \text{(as } \mathrm{Pol}(\#_\varepsilon(\ell)) > \textstyle\sum_{s \trianglelefteq_\# r_j} \mathrm{Pol}(\#_\varepsilon(s))\text{, hence } \mathrm{Pol}(\#_\varepsilon(\ell)\sigma) > \sum_{s \trianglelefteq_\# r_j} \mathrm{Pol}(\#_\varepsilon(s)\sigma)\text{)} \\
&\geq \textstyle\sum_{s \trianglelefteq_\# r_j\sigma} \mathrm{Pol}(\#_\varepsilon(s)) + \sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau})) \\
&\quad + \textstyle\sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\qquad \text{(by } \mathrm{Pol}(\ell) \geq \mathrm{Pol}(r_j) \text{ and (a))} \\
&\geq \textstyle\sum_{s \trianglelefteq_\# r_j\sigma} \mathrm{Pol}(\#_\varepsilon(s)) + \sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau})) \\
&\quad + \textstyle\sum_{\tau \in I_3, \varphi_j(\gamma_\tau) \neq \bot} \mathrm{Pol}(\#_\varepsilon(b_j|_{\pi.\varphi_j(\gamma_\tau).\rho_\tau})) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&\qquad \text{(moving } \tau = \pi.\gamma_\tau.\rho_\tau \in I_3 \text{ via the VRF)} \\
&= \textstyle\sum_{s \trianglelefteq_\# b_j} \mathrm{Pol}(s^\#) \\
&= \mathcal{S}um(b_j)
\end{aligned}$$

In the case $m = \mathsf{false}$, we remove $\sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau}))$, so that the inequation remains correct.

(c) In this case, there exist a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}$ with $\mathrm{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(r_j)$, a substitution σ, and a position π of s such that $s|_\pi = \ell\sigma$, and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$.

We perform structural induction on π. So in the induction base $\pi = \varepsilon$ we have $s = \ell\sigma \to \{p_1 : r_1\sigma, \ldots, p_k : r_k\sigma\}$. As $\mathrm{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(r_j)$ holds for all instantiations of the occurring variables, for $t_j = r_j\sigma$ we obtain

$$\mathrm{Pol}(s) \;=\; \mathrm{Pol}(\ell\sigma) \;\geq\; \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(r_j\sigma) \;=\; \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(t_j).$$

In the induction step, we have $\pi = i.\pi'$, $s = f(s_1, \ldots, s_i, \ldots, s_n)$, $s_i \to \{p_1 : t_{i,1}, \ldots, p_k : t_{i,k}\}$, and $t_j = f(s_1, \ldots, t_{i,j}, \ldots, s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \leq j \leq k$. Then by the induction hypothesis we have $\mathrm{Pol}(s_i) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(t_{i,j})$. Thus, we have

$$\begin{aligned}
\mathrm{Pol}(s) &= \mathrm{Pol}(f(s_1, \ldots, s_i, \ldots, s_n))\\
&= f_{\mathrm{Pol}}(\mathrm{Pol}(s_1), \ldots, \mathrm{Pol}(s_i), \ldots, \mathrm{Pol}(s_n))\\
&\geq f_{\mathrm{Pol}}(\mathrm{Pol}(s_1), \ldots, \textstyle\sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(t_{i,j}), \ldots, \mathrm{Pol}(s_n))\\
&\qquad \text{(by weak monotonicity of } f_{\mathrm{Pol}} \text{ and } \mathrm{Pol}(s_i) \geq \textstyle\sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(t_{i,j}))\\
&= \textstyle\sum_{1 \leq j \leq k} p_j \cdot f_{\mathrm{Pol}}(\mathrm{Pol}(s_1), \ldots, \mathrm{Pol}(t_{i,j}), \ldots, \mathrm{Pol}(s_n))\\
&\qquad \text{(as } f_{\mathrm{Pol}} \text{ is multilinear)}\\
&= \textstyle\sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(f(s_1, \ldots, t_{i,j}, \ldots, s_n))\\
&= \textstyle\sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(t_j).
\end{aligned}$$

(d) In this case, there exist an ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$, a substitution σ, and position π with $\flat(a|_\pi) = \ell\sigma$ and $b_j \doteq a[r_j\sigma]_\pi$. First, assume that $m = \mathsf{true}$ and $\pi \in \mathrm{Pos}_{\mathcal{D}^\#}(a)$. Let $I_1 = \{\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(a) \mid \tau < \pi\}$ be the set of positions of all annotations strictly above π, $I_2 = \{\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(a) \mid \gamma \in \mathrm{Pos}_{\mathcal{V}}(\ell), \pi < \tau \leq \pi.\gamma\}$ be the set of positions of all annotations inside the left-hand side ℓ of the used redex $\ell\sigma$ (but not on the root of ℓ), $I_3 = \{\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(a) \mid \gamma \in \mathrm{Pos}_{\mathcal{V}}(\ell), \pi.\gamma < \tau\}$ be the set of positions of all annotations inside the substitution, and let $I_4 = \{\tau \in \mathrm{Pos}_{\mathcal{D}^\#}(a) \mid \tau \bot \pi\}$ be the set of positions of all annotations orthogonal to π. Furthermore, for each $\tau \in I_1$ let κ_τ be the position such that $\tau.\kappa_\tau = \pi$, and for each $\tau \in I_3$ let γ_τ and ρ_τ be the positions such that $\gamma_\tau \in \mathrm{Pos}_{\mathcal{V}}(\ell)$ and $\pi.\gamma_\tau.\rho_\tau = \tau$. By Requirement (1), we have $\mathrm{Pol}(\#_\varepsilon(\ell)) \geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_\# r_j} \mathrm{Pol}(\#_\varepsilon(t))$ and by (3) we have $\mathrm{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(\flat(r_j))$. As these inequations hold for all instantiations of the occurring variables, we have

$$\begin{aligned}
\mathcal{Sum}(a) &= \textstyle\sum_{t \trianglelefteq_\# a} \mathrm{Pol}(t^\#)\\
&= \mathrm{Pol}(\#_\varepsilon(a|_\pi)) + \textstyle\sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_2} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\\
&\quad + \textstyle\sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\\
&\geq \mathrm{Pol}(\#_\varepsilon(a|_\pi)) + \textstyle\sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\\
&\qquad \text{(removed } I_2)\\
&= \mathrm{Pol}(\#_\varepsilon(\ell)\sigma) + \textstyle\sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\\
&\qquad \text{(as } a|_\pi = \#_\varepsilon(\ell)\sigma)\\
&\geq \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_\# r_j\sigma} \mathrm{Pol}(\#_\varepsilon(t)) + \sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\\
&\quad + \textstyle\sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\\
&\qquad \text{(by } \mathrm{Pol}(\#_\varepsilon(\ell)) \geq \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_\# r_j} \mathrm{Pol}(\#_\varepsilon(t)),\\
&\qquad\qquad \text{hence } \mathrm{Pol}(\#_\varepsilon(\ell)\sigma) \geq \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_\# r_j\sigma} \mathrm{Pol}(\#_\varepsilon(t)))\\
&\geq \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_\# r_j\sigma} \mathrm{Pol}(\#_\varepsilon(t)) + \sum_{\tau \in I_1} \sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau}))\\
&\quad + \textstyle\sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\\
&\qquad \text{(by } \mathrm{Pol}(\ell) \geq \textstyle\sum_{1 \leq j \leq k} p_j \cdot \mathrm{Pol}(r_j) \text{ and (c))}
\end{aligned}$$

$$
\begin{aligned}
&= \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_{\#} r_j \sigma} \mathrm{Pol}(\#_\varepsilon(t)) + \sum_{1 \leq j \leq k} \sum_{\tau \in I_1} p_j \cdot \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau})) \\
&\quad + \textstyle\sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&= \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_{\#} r_j \sigma} \mathrm{Pol}(\#_\varepsilon(t)) + \sum_{1 \leq j \leq k} p_j \cdot \sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau})) \\
&\quad + \textstyle\sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&= \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_{\#} r_j \sigma} \mathrm{Pol}(\#_\varepsilon(t)) + \sum_{1 \leq j \leq k} p_j \cdot \sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau})) \\
&\quad + \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{1 \leq j \leq k} p_j \cdot \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) \\
&= \textstyle\sum_{1 \leq j \leq k} p_j \cdot \big(\sum_{t \trianglelefteq_{\#} r_j \sigma} \mathrm{Pol}(\#_\varepsilon(t)) + \sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau})) \\
&\quad + \textstyle\sum_{\tau \in I_3} \mathrm{Pol}(\#_\varepsilon(a|_\tau)) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\big) \\
&\geq \textstyle\sum_{1 \leq j \leq k} p_j \cdot \big(\sum_{t \trianglelefteq_{\#} r_j \sigma} \mathrm{Pol}(\#_\varepsilon(t)) + \sum_{\tau \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau})) \\
&\quad + \textstyle\sum_{\tau \in I_3, \varphi_j(\gamma_\tau) \neq \bot} \mathrm{Pol}(\#_\varepsilon(b_j|_{\pi.\varphi_j(\gamma_\tau).\rho_\tau})) + \sum_{\tau \in I_4} \mathrm{Pol}(\#_\varepsilon(a|_\tau))\big) \\
&\qquad\qquad\qquad \text{(moving } \tau = \pi.\gamma_\tau.\rho_\tau \in I_3 \text{ via the VRF)} \\
&= \textstyle\sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_{\#} b_j} \mathrm{Pol}(t^{\#}) \\
&= \mathcal{S}\mathit{um}(b_j)
\end{aligned}
$$

In the case $\pi \notin \mathrm{Pos}_{\mathcal{D}^\#}(a)$, we need to remove $\mathrm{Pol}(\#_\varepsilon(\ell)\sigma)$ as this annotated subterm does not exist in a, and therefore also $\sum_{t \trianglelefteq_{\#} r_j \sigma} \mathrm{Pol}(t^{\#})$ in the end, leading to the same result. In the case $m = \mathsf{false}$, we additionally remove $\sum_{i \in I_1} \mathrm{Pol}(\#_\varepsilon(a|_\tau[r_j\sigma]_{\kappa_\tau}))$ in the end.

The rest is completely analogous to the proof in [28]. □

Theorem 39 (Reduction Pair Processor for `bAST`). *Let* $\mathrm{Pol} : \mathcal{T}^{\#} \to \mathbb{N}[\mathcal{V}]$ *be a multilinear polynomial interpretation and let* $\mathcal{P} = \mathcal{P}_{\geq} \uplus \mathcal{P}_{>}$ *with* $\mathcal{P}_{>} \neq \varnothing$ *satisfy the conditions of Thm. 36. Then* $\mathrm{Proc}_{\mathtt{RP}}(\mathcal{I}, \mathcal{P}) = \{(\mathcal{I} \cup \mathcal{P}_{>}, \mathcal{P}_{\geq} \cup \flat(\mathcal{P}_{>}))\}$ *is sound and complete for* `bAST`*.*

Proof. Analogous to the proof of Thm. 36. We can only use the ADPs from $\mathcal{I} \setminus \mathcal{P}$ finitely often, hence we can ignore them for the constraints in the theorem. □

Theorem 42 (Probability Removal Processor for `bAST` and `AST`). *Let* $\mathcal{P}$ *be an ADP problem where every ADP in* $\mathcal{P}$ *has the form* $\ell \to \{1 : r\}^m$*. Then* $\mathcal{P}$ *is* `AST` *iff the non-probabilistic DP problem* $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ *is terminating. So the processor* $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{P}) = \varnothing$ *is sound and complete for* `AST` *iff* $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ *is terminating. Similarly,* $(\mathcal{I}, \mathcal{P})$ *is* `bAST` *if* $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ *is terminating. So* $\mathrm{Proc}_{\mathtt{PR}}(\mathcal{I}, \mathcal{P}) = \varnothing$ *is sound and complete for* `bAST` *if* $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ *is terminating.*

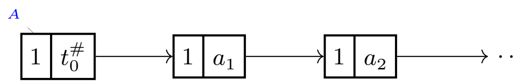
Proof. Let $\mathcal{P}$ be an ADP problem such that every ADP in $\mathcal{P}$ has the form $\ell \to \{1 : r\}^m$. Note that every $\mathcal{P}$-chain tree is a single (not necessarily finite) path. For such a chain tree $\mathfrak{T}$ that is only a single path, we have only two possibilities for $|\mathfrak{T}|_{\mathtt{Leaf}}$. If the path is finite, then $|\mathfrak{T}|_{\mathtt{Leaf}} = 1$, since we have a single leaf in this tree with probability 1. Otherwise, we have an infinite path, which means that there is no leaf at all and hence $|\mathfrak{T}|_{\mathtt{Leaf}} = 0$.

"only if"

This direction only works for `AST`. Assume that $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is not terminating. Then there exists an infinite $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$-chain

$$t_0^{\#} \to_{\mathrm{dp}(\mathcal{P})} \circ \to_{\mathrm{np}(\mathcal{P})}^{*} t_1^{\#} \to_{\mathrm{dp}(\mathcal{P})} \circ \to_{\mathrm{np}(\mathcal{P})}^{*} t_2^{\#} \to_{\mathrm{dp}(\mathcal{P})} \circ \to_{\mathrm{np}(\mathcal{P})}^{*} \cdots$$

such that for all $i \in \mathbb{N}$ we have $t_i^{\#} = \ell_i^{\#} \sigma_i$ for some dependency pair $\ell_i^{\#} \to r_i^{\#} \in \mathrm{dp}(\mathcal{P})$ and some substitution σ_i. From this infinite $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$-chain, we will now construct an infinite $\mathcal{P}$-chain tree $\mathfrak{T} = (V, E, L, A)$. As explained above, we then know that this infinite $\mathcal{P}$-chain tree must be an infinite path, and thus $|\mathfrak{T}|_{\texttt{Leaf}} = 0$, which means that $\mathcal{P}$ is not AST, and thus, a processor with $\mathrm{Proc}_{\texttt{PR}}(\mathcal{P}) = \varnothing$ would be unsound.



We start our chain tree with $(1 : t_0^\#)$. In the non-probabilistic rewrite sequence, we have $t_0^\# \to_{\mathrm{dp}(\mathcal{P})} \circ \to_{\mathrm{np}(\mathcal{P})}^* t_1^\#$, so there exists a natural number $k \geq 1$ such that

$$t_0^\# = \ell_0^\# \sigma_0 \to_{\mathrm{dp}(\mathcal{P})} r_0^\# \sigma_0 = v_1^\# \to_{\mathrm{np}(\mathcal{P})} v_2^\# \to_{\mathrm{np}(\mathcal{P})} \ldots \to_{\mathrm{np}(\mathcal{P})} v_k^\# = t_1^\# = \ell_1^\# \sigma_1$$

Performing the same rewrite steps with $\mathcal{P}$ yields terms $a_1, \ldots, a_k$ such that $v_i \trianglelefteq_\# a_i$ for all $1 \leq i \leq k$. Here, one needs that all ADPs that yield the rules in $\mathrm{np}(\mathcal{P})$ have the flag true and thus, the annotations above the redex are not removed. For all $1 \leq i \leq k$, we now construct a_i inductively.

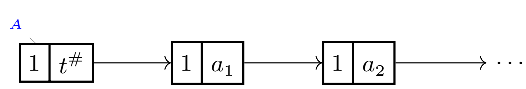
In the induction base ($i = 1$), let $\ell_0 \to \{1 : r_0'\}^m \in \mathcal{P}$ be the ADP that was used to create the dependency pair $\ell_0^\# \to r_0^\#$ in $\mathrm{dp}(\mathcal{P})$. This means that we have $r_0 \trianglelefteq_\# r_0'$. Since we have $t_0^\# = \ell_0^\# \sigma_0$, we can also rewrite $t_0^\#$ with the ADP $\ell_0 \to \{1 : r_0'\}^m \in \mathcal{P}$ and the substitution σ_0. We result in $a_1 = r_0' \sigma_0$ and thus we have $v_1 = r_0 \sigma_0 \trianglelefteq_\# r_0' \sigma_0 = a_1$.

In the induction step, we assume that we have $v_i \trianglelefteq_\# a_i$ for some $1 \leq i < k$. Let π be the annotated position of a_i where $v_i = \flat(a_i)|_\pi$. In our non-probabilistic rewrite sequence we have $v_i^\# \to_{\mathrm{np}(\mathcal{P})} v_{i+1}^\#$ using a rule $\ell' \to \flat(r') \in \mathrm{np}(\mathcal{P})$ and substitution δ_i at a position $\tau \in \mathbb{N}^+$ such that $v_i^\#|_\tau = \ell' \delta_i$ and $v_{i+1}^\# = v_i^\#[\flat(r')\delta_i]_\tau$. We can mirror this rewrite step with the ADP $\ell' \to \{1 : r'\}^{\mathsf{true}} \in \mathcal{P}$, since by construction we have $v_i \trianglelefteq_\# a_i$ and $v_i = \flat(a_i)|_\pi$. We obtain $a_i \hookrightarrow_{\mathcal{P}} a_{i+1} = a_i[\#_X(r'\delta_i)]_{\pi.\tau}$ with $X = \Phi_1$ (step with (**at**)) or $X = \Psi_1$ (step with (**nt**)) by rewriting the subterm of a_i at position $\pi.\tau$, which implies $v_{i+1} = v_i[\flat(r')\delta_i]_\tau \trianglelefteq_\# a_i[\#_X(r'\delta_i)]_{\pi.\tau} = a_{i+1}$.

At the end of this induction, we result in a_k. Next, we can mirror the step $t_1^\# \to_{\mathrm{dp}(\mathcal{P})} \circ \to_{\mathrm{np}(\mathcal{P})}^* t_2^\#$ from our non-probabilistic rewrite sequence with the same construction, etc. This results in an infinite $\mathcal{P}$-chain tree. To see that this is indeed a $\mathcal{P}$-chain tree, note that all the local properties are satisfied since every edge represents a rewrite step with $\hookrightarrow_{\mathcal{P}}$. The global property is also satisfied since, in an infinite $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$-chain, we use an infinite number of steps with $\to_{\mathrm{dp}(\mathcal{P})}$ so that our resulting chain tree has an infinite number of nodes in A.

"if"

This direction works analogously for both AST and bAST, and we prove it only for AST. For bAST we simply ignore the reachability component $\mathcal{I}$. Assume that $\mathcal{P}$ is not AST, i.e., that the processor $\mathrm{Proc}_{\texttt{PR}}(\mathcal{P}) = \varnothing$ is unsound. By Lemma 48, there exists a $\mathcal{P}$-chain tree $\mathfrak{T} = (V, E, L, A)$ that converges with probability < 1 and starts with $(1 : t^\#)$ such that $t^\# = \ell^\# \sigma_0$ for some substitution σ_0 and an ADP $\ell \to \{1 : r\}^m \in \mathcal{P}$. As explained above, this tree must be an infinite path.



From $\mathfrak{T}$, we will now construct an infinite $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$-chain, which shows that $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is not terminating. We start our infinite chain with the term $t_0^{\#} = t^{\#}$. We have $t^{\#} \hookrightarrow_{\mathcal{P}} \{1 : a_1\}$, where $a_1 = \#_{\Phi_1}(r\sigma_0) = r\sigma_0$.

There must be a term $r_0 \trianglelefteq_{\#} r$ (i.e., $r_0^{\#}\sigma_0 \trianglelefteq r\sigma_0$) such that if we replace $a_1 = r\sigma_0$ with $r_0^{\#}\sigma_0$ and obtain the same $\mathcal{P}$-chain tree except when we would rewrite terms that do not exist anymore (i.e., we ignore these rewrite steps), then we still end up in an infinite number of nodes in A (otherwise, $\mathfrak{T}$ would not have an infinite number of nodes in A).

Let π be the annotated position of $r\sigma_0$ where $r_0\sigma_0 = \flat(r\sigma_0)|_{\pi}$. We can rewrite the term $t_0^{\#}$ with the dependency pair $\ell^{\#} \to r_0^{\#} \in \mathrm{dp}(\mathcal{P})$, using the substitution σ_0 since $t_0 = t = \ell\sigma_0$. Hence, we result in $t_0^{\#} \to_{\mathrm{dp}(\mathcal{P})} r_0^{\#}\sigma_0 = v_1^{\#}$. Next, we mirror the rewrite steps from the $\mathcal{P}$-chain tree that are performed strictly below the root of $r_0^{\#}\sigma_0$ with $\mathrm{np}(\mathcal{P})$ until we would rewrite at the root of $r_0^{\#}\sigma_0$. With this construction, we ensure that each the term $v_i^{\#}$ in our $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$-chain satisfies $v_i \trianglelefteq_{\#} a_i$ and $v_i = \flat(a_i)|_{\pi}$. A rewrite step at position $\pi.\tau$ in a_i with $\mathcal{P}$ is then mirrored with $\mathrm{np}(\mathcal{P})$ in $v_i^{\#}$ at position τ. Note that we only use a finite number of $\mathrm{np}(\mathcal{P})$ steps until we rewrite at the root.

So eventually, we result in a term $t_1^{\#} = v_k^{\#}$ with $v_k = \flat(a_k)|_{\pi}$, and we rewrite at position π in the $\mathcal{P}$-chain tree. We mirror the step $a_k \hookrightarrow_{\mathcal{P}} a_{k+1}$ with $\mathrm{dp}(\mathcal{P})$ and then use the same construction again until we reach term $t_2^{\#}$, etc. This construction creates a sequence $t_0^{\#}, t_1^{\#}, \ldots$ of terms such that

$$t_0^{\#} \to_{\mathrm{dp}(\mathcal{P})} r_0^{\#}\sigma_0 \to_{\mathrm{np}(\mathcal{P})}^{*} t_1^{\#} \to_{\mathrm{dp}(\mathcal{P})} r_1^{\#}\sigma_1 \to_{\mathrm{np}(\mathcal{P})}^{*} \ldots$$

Therefore, $(\mathrm{dp}(\mathcal{P}), \mathrm{np}(\mathcal{P}))$ is not terminating. □

Example 49. A counterexample for the "only if" part of Thm. 41 for `bAST` is the basic ADP problem $(\varnothing, \mathcal{P})$ with $\mathcal{P}$ containing the ADPs $\mathsf{f}(\mathsf{a}, \mathsf{b}, x) \to \{1 : \mathsf{F}(x, x, x)\}^{\mathsf{true}}$, $\mathsf{h}(x, y) \to \{1 : x\}^{\mathsf{true}}$, and $\mathsf{h}(x, y) \to \{1 : y\}^{\mathsf{true}}$, which is based on the well-known TRS from [46]. It is `bAST`, but the DP problem $(\mathcal{D}, \mathcal{R})$ with $\mathcal{D} = \{\mathsf{F}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{F}(x, x, x)\}$ and $\mathcal{R} = \{\mathsf{h}(x, y) \to x, \mathsf{h}(x, y) \to y\}$ considers termination w.r.t. arbitrary start terms again, hence it is non-terminating.