

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

Jan-Christoph Kassing[✉], Stefan Dollase[✉], and Jürgen Giesl[✉]

LuFG Informatik 2, RWTH Aachen University, Aachen, Germany

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

1 Introduction

Termination of term rewrite systems (TRSs) has been studied for decades and TRSs are used for automated termination analysis of many programming languages. One of the most powerful techniques integrated in essentially all current termination tools for TRSs is the *dependency pair* (DP) framework [2, 17, 18, 24] which allows modular proofs that apply different techniques in different sub-proofs.

In [8, 9], term rewriting was extended to the probabilistic setting. Probabilistic programs describe randomized algorithms and probability distributions, with applications in many areas. In the probabilistic setting, there are several notions of “termination”. A program is *almost-surely terminating* (AST) if the probability of termination is 1. A strictly stronger notion is *positive AST* (PAST), which requires that the expected runtime is finite. While numerous techniques exist to prove (P)AST of imperative programs on numbers (e.g., [1, 4, 10, 14, 21, 25–27, 33–36]), there are only few automatic approaches for programs with complex non-tail recursive structure [7, 11, 12]. The approaches that are also suitable for algorithms on recursive data structures [6, 32, 40] are mostly specialized for specific data structures and cannot easily be adjusted to other (possibly user-defined) ones, or are not yet fully automated. In contrast, our goal is a fully automatic termination analysis for (arbitrary) probabilistic TRSs (PTRSs).

Up to now, only two approaches for automatic termination analysis of PTRSs were developed [3, 28]. In [3], orderings based on interpretations were adapted to prove PAST. However, already for non-probabilistic TRSs such a direct application

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

of orderings is limited in power. To obtain a powerful approach, one should combine such orderings in a modular way, as in the DP framework.

Indeed, in [28], we adapted the DP framework to the probabilistic setting in order to prove innermost AST (iAST), i.e., AST for rewrite sequences which follow the innermost evaluation strategy. However, in contrast to the DP framework for ordinary TRSs, the probabilistic *dependency tuple* (DT) framework in [28] is *incomplete*, i.e., there are PTRSs which are iAST but where this cannot be proved with DTs. In this paper, we introduce a new concept of probabilistic DPs and a corresponding new rewrite relation. In this way, we obtain a novel *complete* criterion for iAST via DPs while maintaining soundness for all processors that were developed in the probabilistic DT framework of [28]. Moreover, our improvement allows us to introduce additional more powerful “transformational” probabilistic DP processors which were not possible in the framework of [28].

We recapitulate the DP framework for non-probabilistic TRSs in Sect. 2. Then, we present our novel ADPs (*annotated dependency pairs*) for probabilistic TRSs in Sect. 3. In Sect. 4, we show how to adapt the processors from the framework of [28] to our probabilistic ADP framework. In addition, our framework allows for the definition of new processors which *transform* ADPs. As an example, in Sect. 5 we adapt the *rewriting processor* to the probabilistic setting, which benefits from our new, more precise rewrite relation. The implementation of our approach in the tool AProVE is evaluated in Sect. 6. We refer to App. A for all proofs. In App. B we show how the other transformational processors of the DP framework can also be adapted to the probabilistic setting. Finally, in App. C we present selected examples from our new set of benchmarks.

2 The DP Framework

We assume familiarity with term rewriting [5] and recapitulate the DP framework with its core processors (see e.g., [2, 17, 18, 24] for details). We regard finite TRSs \mathcal{R} over a finite signature Σ and let $\mathcal{T}(\Sigma, \mathcal{V})$ denote the set of terms over Σ and a set of variables \mathcal{V} . We decompose $\Sigma = \mathcal{D} \uplus \mathcal{C}$ such that $f \in \mathcal{D}$ if $f = \text{root}(\ell)$ for some $\ell \rightarrow r \in \mathcal{R}$. The symbols in \mathcal{D} are called *defined symbols*. For every $f \in \mathcal{D}$, we introduce a fresh *annotated symbol* $f^\#$ of the same arity.¹ Let $\mathcal{D}^\#$ be the set of all annotated symbols and $\Sigma^\# = \mathcal{D}^\# \uplus \Sigma$. For any $t = f(t_1, \dots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ with $f \in \mathcal{D}$, let $t^\# = f^\#(t_1, \dots, t_n)$. For every rule $\ell \rightarrow r$ and every (not necessarily proper) subterm t of r with defined root symbol, one obtains a *dependency pair* (DP) $\ell^\# \rightarrow t^\#$. $\mathcal{DP}(\mathcal{R})$ denotes the set of all dependency pairs of \mathcal{R} . As an example, consider $\mathcal{R}_{\text{ex}} = \{(1), (2)\}$ with its dependency pairs $\mathcal{DP}(\mathcal{R}_{\text{ex}}) = \{(3), (4)\}$. To ease readability, we often write F instead of $f^\#$, etc.

$$\begin{array}{ll} f(\mathbf{s}(x)) \rightarrow c(f(\mathbf{g}(x))) & (1) \quad \mathbf{F}(\mathbf{s}(x)) \rightarrow \mathbf{F}(\mathbf{g}(x)) \quad (3) \\ \mathbf{g}(x) \rightarrow \mathbf{s}(x) & (2) \quad \mathbf{F}(\mathbf{s}(x)) \rightarrow \mathbf{G}(x) \quad (4) \end{array}$$

The DP framework uses *DP problems* $(\mathcal{P}, \mathcal{R})$ where \mathcal{P} is a (finite) set of DPs and \mathcal{R} is a TRS. A (possibly infinite) sequence t_0, t_1, t_2, \dots with $t_i \xrightarrow{i}_{\mathcal{P}, \mathcal{R}} t_{i+1}$

¹ The symbols $f^\#$ were called *tuple symbols* in the original DP framework [18] and also in [28], as they represent the tuple of arguments of the original defined symbol f .

for all i is an (innermost) $(\mathcal{P}, \mathcal{R})$ -chain which represents subsequent “function calls” in evaluations. Here, “ \circ ” denotes composition and steps with $\xrightarrow{i}_{\mathcal{P}, \mathcal{R}}$ are called **p-steps**, where $\xrightarrow{i}_{\mathcal{P}, \mathcal{R}}$ is the restriction of $\rightarrow_{\mathcal{P}}$ to rewrite steps where the used redex is in $\text{NF}_{\mathcal{R}}$ (the set of normal forms w.r.t. \mathcal{R}). Steps with $\xrightarrow{i}_{\mathcal{R}}$ are called **r-steps** and are used to evaluate the arguments of an annotated function symbol. So an infinite chain consists of an infinite number of **p-steps** with a finite number of **r-steps** between consecutive **p-steps**. For example, $F(s(x)), F(s(x)), \dots$ is an infinite $(\mathcal{DP}(\mathcal{R}_{\text{ex}}), \mathcal{R}_{\text{ex}})$ -chain, as $F(s(x)) \xrightarrow{i}_{\mathcal{DP}(\mathcal{R}_{\text{ex}}), \mathcal{R}_{\text{ex}}} F(g(x)) \xrightarrow{i}_{\mathcal{R}_{\text{ex}}} F(s(x))$. Throughout the paper, we restrict ourselves to innermost rewriting (“ $\xrightarrow{i}_{\mathcal{R}}$ ”), because our adaption of DPs to the probabilistic setting relies on this evaluation strategy.²

A DP problem $(\mathcal{P}, \mathcal{R})$ is called *innermost terminating* (iTerm) if there is no infinite innermost $(\mathcal{P}, \mathcal{R})$ -chain. The main result on DPs is the *chain criterion* which states that there is no infinite sequence $t_1 \xrightarrow{i}_{\mathcal{R}} t_2 \xrightarrow{i}_{\mathcal{R}} \dots$, i.e., \mathcal{R} is iTerm, iff $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$ is iTerm. The DP framework is a *divide-and-conquer* approach, which applies *DP processors* to transform DP problems into simpler sub-problems. A *DP processor* Proc has the form $\text{Proc}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}_1, \mathcal{R}_1), \dots, (\mathcal{P}_n, \mathcal{R}_n)\}$, where $\mathcal{P}, \mathcal{P}_1, \dots, \mathcal{P}_n$ are sets of DPs and $\mathcal{R}, \mathcal{R}_1, \dots, \mathcal{R}_n$ are TRSs. A processor Proc is *sound* if $(\mathcal{P}, \mathcal{R})$ is iTerm whenever $(\mathcal{P}_i, \mathcal{R}_i)$ is iTerm for all $1 \leq i \leq n$. It is *complete* if $(\mathcal{P}_i, \mathcal{R}_i)$ is iTerm for all $1 \leq i \leq n$ whenever $(\mathcal{P}, \mathcal{R})$ is iTerm.

So given a TRS \mathcal{R} , one starts with the initial DP problem $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$ and applies sound (and preferably complete) DP processors repeatedly until all sub-problems are “solved” (i.e., sound processors transform them to the empty set). This yields a modular framework for termination proofs, as different techniques can be used for different sub-problems $(\mathcal{P}_i, \mathcal{R}_i)$. The following three theorems recapitulate the three most important processors of the DP framework.

The (innermost) $(\mathcal{P}, \mathcal{R})$ -*dependency graph* is a control flow graph that indicates which DPs can be used after each other in a chain. Its set of nodes is \mathcal{P} and there is an edge from $\ell_1^{\#} \rightarrow t_1^{\#}$ to $\ell_2^{\#} \rightarrow t_2^{\#}$ if there exist substitutions σ_1, σ_2 such that $t_1^{\#} \sigma_1 \xrightarrow{i}_{\mathcal{R}} \ell_2^{\#} \sigma_2$ and $\ell_1^{\#} \sigma_1, \ell_2^{\#} \sigma_2 \in \text{NF}_{\mathcal{R}}$. Any infinite $(\mathcal{P}, \mathcal{R})$ -chain corresponds to an infinite path in the dependency graph, and since the graph is finite, this infinite path must end in some strongly connected component (SCC).³ Hence, it suffices to consider the SCCs of this graph independently.

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

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

² Moreover, already in the non-probabilistic setting, the restriction to innermost rewriting makes termination analysis with DPs substantially more powerful, e.g., by allowing the application of additional techniques like *usable rules* and *rewriting* of DPs [17, 18]. Indeed, we also adapt these techniques in our novel ADP framework for probabilistic rewriting. Nevertheless, we conjecture that ADPs are also suitable for an adaption to analyze full instead of innermost AST, and we will investigate that in future work.

³ Here, a set \mathcal{P}' of DPs is an *SCC* if it is a maximal cycle, i.e., it is a maximal set such that for any $\ell_1^{\#} \rightarrow t_1^{\#}$ and $\ell_2^{\#} \rightarrow t_2^{\#}$ in \mathcal{P}' there is a non-empty path from $\ell_1^{\#} \rightarrow t_1^{\#}$ to $\ell_2^{\#} \rightarrow t_2^{\#}$ which only traverses nodes from \mathcal{P}' .

$$\begin{array}{rcl}
& & F(f(g(x))) \rightarrow F(g(f(g(f(x)))))) \quad (6) \\
f(f(g(x))) \rightarrow f(g(f(g(f(x)))))) \quad (5) & F(f(g(x))) \rightarrow F(g(f(x))) \quad (7) & \begin{array}{c} \boxed{6} \leftarrow \boxed{8} \\ \uparrow \cup \\ \boxed{7} \end{array} \\
& & F(f(g(x))) \rightarrow F(x) \quad (8)
\end{array}$$

While the exact dependency graph is not computable in general, there exist several techniques to over-approximate it automatically, see, e.g., [2, 18, 24]. In our example, $\text{Proc}_{\text{DG}}(\mathcal{DP}(\mathcal{R}_{\text{ffg}}), \mathcal{R}_{\text{ffg}})$ yields the DP problem $(\{(8)\}, \mathcal{R}_{\text{ffg}})$.

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

Theorem 3 (Usable Rules Processor). *Let \mathcal{R} be a TRS. For every $f \in \Sigma^\#$ let $\text{Rules}_{\mathcal{R}}(f) = \{\ell \rightarrow r \in \mathcal{R} \mid \text{root}(\ell) = f\}$. For any $t \in \mathcal{T}(\Sigma^\#, \mathcal{V})$, its usable rules $\mathcal{U}_{\mathcal{R}}(t)$ are the smallest set such that $\mathcal{U}_{\mathcal{R}}(x) = \emptyset$ for all $x \in \mathcal{V}$ and $\mathcal{U}_{\mathcal{R}}(f(t_1, \dots, t_n)) = \text{Rules}_{\mathcal{R}}(f) \cup \bigcup_{i=1}^n \mathcal{U}_{\mathcal{R}}(t_i) \cup \bigcup_{\ell \rightarrow r \in \text{Rules}_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}}(r)$. The usable rules for the DP problem $(\mathcal{P}, \mathcal{R})$ are $\mathcal{U}(\mathcal{P}, \mathcal{R}) = \bigcup_{\ell^\# \rightarrow t^\# \in \mathcal{P}} \mathcal{U}_{\mathcal{R}}(t^\#)$. Then $\text{Proc}_{\text{UR}}(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}, \mathcal{U}(\mathcal{P}, \mathcal{R}))\}$ is sound but not complete.⁴*

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

A *polynomial interpretation* Pol is a Σ -algebra which maps every function symbol $f \in \Sigma$ to a polynomial $f_{\text{Pol}} \in \mathbb{N}[\mathcal{V}]$ over the variables \mathcal{V} with coefficients from \mathbb{N} , see [31]. $\text{Pol}(t)$ denotes the *interpretation* of a term t by the Σ -algebra Pol . An arithmetic inequation like $\text{Pol}(t_1) > \text{Pol}(t_2)$ *holds* if it is true for all instantiations of its variables by natural numbers. The reduction pair processor⁵ allows us to use *weakly monotonic* polynomial interpretations that do not have to depend on all of their arguments, i.e., $x \geq y$ implies $f_{\text{Pol}}(\dots, x, \dots) \geq f_{\text{Pol}}(\dots, y, \dots)$ for all $f \in \Sigma^\#$. The processor requires that all rules and DPs are weakly decreasing and it removes those DPs that are strictly decreasing.

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

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

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

For $(\{(8)\}, \emptyset)$, one can use the reduction pair processor with the polynomial interpretation that maps $f(x)$ to $x + 1$ and both $F(x)$ and $g(x)$ to x . Then, $\text{Proc}_{\text{RP}}(\{(8)\}, \emptyset) = \{(\emptyset, \emptyset)\}$. As $\text{Proc}_{\text{DG}}(\emptyset, \dots) = \emptyset$ and all processors used are sound, this means that there is no infinite innermost chain for the initial DP problem $(\mathcal{DP}(\mathcal{R}_{\text{ffg}}), \mathcal{R}_{\text{ffg}})$ and thus, \mathcal{R}_{ffg} is innermost terminating.

⁴ See [17] for a complete version of this processor. It extends DP problems by an additional set to store the left-hand sides of all rules (including the non-usable ones) to determine whether a rewrite step is innermost. We omit this here for readability.

⁵ In this paper, we only regard the reduction pair processor with polynomial interpretations, because for most other classical orderings it is not clear how to extend them to probabilistic TRSs, where one has to consider “expected values of terms”.

3 Probabilistic Annotated Dependency Pairs

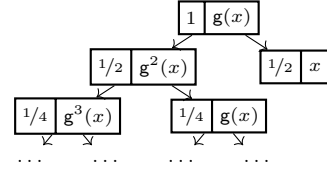
In this section we present our novel adaption of DPs to the probabilistic setting. As in [3, 9, 13, 28], the rules of a probabilistic TRS have finite multi-distributions on the right-hand sides. A finite *multi-distribution* μ on a set $A \neq \emptyset$ is a finite multiset of pairs $(p : a)$, where $0 < p \leq 1$ is a probability and $a \in A$, with $\sum_{(p:a) \in \mu} p = 1$. $\text{FDist}(A)$ is the set of all finite multi-distributions on A . For $\mu \in \text{FDist}(A)$, its *support* is the multiset $\text{Supp}(\mu) = \{a \mid (p:a) \in \mu \text{ for some } p\}$.

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

A PTRS \mathcal{R} induces a *rewrite relation* $\rightarrow_{\mathcal{R}} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \text{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ where $s \rightarrow_{\mathcal{R}} \{p_1 : t_1, \dots, p_k : t_k\}$ if there is a position π of s , a rule $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{R}$, and a substitution σ such that $s|_{\pi} = \ell\sigma$ and $t_j = s[r_j\sigma]_{\pi}$ for all $1 \leq j \leq k$. We call $s \rightarrow_{\mathcal{R}} \mu$ an *innermost* rewrite step (denoted $s \xrightarrow{i}_{\mathcal{R}} \mu$) if $\ell\sigma \in \text{ANF}_{\mathcal{R}}$, where $\text{ANF}_{\mathcal{R}}$ is the set of all *terms in argument normal form w.r.t. \mathcal{R}* , i.e., $t \in \text{ANF}_{\mathcal{R}}$ iff $t' \in \text{NF}_{\mathcal{R}}$ for all proper subterms t' of t .

To track all possible rewrite sequences (up to non-determinism) with their probabilities, we *lift* $\xrightarrow{i}_{\mathcal{R}}$ to (*innermost*) *rewrite sequence trees* (RSTs). An (innermost) \mathcal{R} -RST is a tree whose nodes v are labeled by pairs (p_v, t_v) of a probability p_v and a term t_v such that the edge relation represents a probabilistic innermost rewrite step. More precisely, $\mathfrak{T} = (V, E, L)$ is an (innermost) \mathcal{R} -RST if (1) (V, E) is a (possibly infinite) directed tree with nodes $V \neq \emptyset$ and directed edges $E \subseteq V \times V$ where $vE = \{w \mid (v, w) \in E\}$ is finite for every $v \in V$, (2) $L : V \rightarrow (0, 1] \times \mathcal{T}(\Sigma, \mathcal{V})$ labels every node v by a probability p_v and a term t_v where $p_v = 1$ for the root $v \in V$ of the tree, and (3) for all $v \in V$: if $vE = \{w_1, \dots, w_k\} \neq \emptyset$, then $t_v \xrightarrow{i}_{\mathcal{R}} \{ \frac{p_{w_1}}{p_v} : t_{w_1}, \dots, \frac{p_{w_k}}{p_v} : t_{w_k} \}$. For any innermost \mathcal{R} -RST \mathfrak{T} we define $|\mathfrak{T}|_{\text{Leaf}} = \sum_{v \in \text{Leaf}} p_v$, where Leaf is the set of \mathfrak{T} 's leaves. An RST \mathfrak{T} is *innermost almost-surely terminating* (iAST) if $|\mathfrak{T}|_{\text{Leaf}} = 1$. Similarly, a PTRS \mathcal{R} is *iAST* if all innermost \mathcal{R} -RSTs are iAST. While $|\mathfrak{T}|_{\text{Leaf}} = 1$ holds for every finite RST \mathfrak{T} , for infinite RSTs \mathfrak{T} we may have $|\mathfrak{T}|_{\text{Leaf}} < 1$, and even $|\mathfrak{T}|_{\text{Leaf}} = 0$ if \mathfrak{T} has no leaf at all. This notion is equivalent to the notions of AST in [3, 28], where one uses a lifting to multisets instead of trees. For example, the infinite \mathcal{R}_{rw} -RST \mathfrak{T} on the side has $|\mathfrak{T}|_{\text{Leaf}} = 1$. In fact, \mathcal{R}_{rw} is iAST, because $|\mathfrak{T}|_{\text{Leaf}} = 1$ holds for all innermost \mathcal{R}_{rw} -RSTs \mathfrak{T} .

As shown in [28], to adapt the DP framework in order to prove iAST of PTRSs, one has to regard all DPs resulting from the same rule *at*



once. Otherwise, one would not be able to distinguish between the DPs of the TRS with the rule $\mathbf{a} \rightarrow \{1/2 : \mathbf{b}, 1/2 : \mathbf{c}(\mathbf{a}, \mathbf{a})\}$ which is iAST and the rule $\mathbf{a} \rightarrow \{1/2 : \mathbf{b}, 1/2 : \mathbf{c}(\mathbf{a}, \mathbf{a}, \mathbf{a})\}$, which is not iAST. For that reason, in the adaption of the DP framework to PTRSs in [28], one constructs *dependency tuples* (DTs) whose right-hand sides combine the right-hand sides of all dependency pairs resulting from one rule. However, a drawback of this approach is that the resulting chain

criterion is not complete, i.e., it allows for chains that do not correspond to any rewrite sequence of the original PTRS \mathcal{R} .

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

$$a \rightarrow \{1 : f(h(g), g)\} \quad (9) \quad h(b_1) \rightarrow \{1 : a\} \quad (11)$$

$$g \rightarrow \{1/2 : b_1, 1/2 : b_2\} \quad (10) \quad f(x, b_2) \rightarrow \{1 : a\} \quad (12)$$

and the $\mathcal{R}_{\text{incpl}}$ -RST below. So a can be rewritten to the normal form $f(h(b_2), b_1)$

with probability $1/4$

and to the terms

$f(a, b_1)$ and a that

contain the redex

with a probability of

$1/4 + 1/4 = 1/2$. In the

term $f(a, b_2)$, one can

rewrite the subterm

a , and if that ends in a

normal form, one can still

rewrite the outer f which

will yield a again. So to

over-approximate the probability

of non-termination, one could

consider the term $f(a, b_2)$

as if one had two occurrences

of a . Then this would

correspond to a random walk

where the number of a symbols

is decreased by 1 with

probability $1/4$, increased by 1

with probability $1/4$, and kept

the same with probability

$1/2$. Such a random walk is

AST, and since a similar

observation holds for all

$\mathcal{R}_{\text{incpl}}$ -RSTs, $\mathcal{R}_{\text{incpl}}$ is

iAST (we will prove iAST of

$\mathcal{R}_{\text{incpl}}$ with our new ADP

framework in Sect. 4 and 5).

In contrast, the DT framework

from [28] fails on this example.

As mentioned, the right-hand

sides of DTs combine the right-

hand sides of all dependency

pairs resulting from one rule.

So the right-hand side of the

DT for (9) contains the term

$\text{com}_4(F(h(g), g), H(g), G, G)$,

where com_4 is a special compound

symbol of arity 4. However,

here it is no longer clear which

occurrence of the annotated

symbol G corresponds to which

occurrences of g . Therefore,

when rewriting an occurrence

of G , in the “chains” of [28]

one may also rewrite arbitrary

occurrences of g simultane-

ously. (For that reason, in [28]

one also couples the DT together

with its original rule.) In particular,

[28] also allows a simultaneous

rewrite step of all underlined

symbols in $\text{com}(F(h(g), g), H(g), \underline{G}, G)$

even though the underlined G

cannot correspond to both

underlined g symbols. As shown

in Lemma 28 in App. A, this

leads to a chain that is not

iAST and that does not correspond

to any $\mathcal{R}_{\text{incpl}}$ -rewrite sequence.

To avoid this problem, one would

have to keep track of the

connections between annotated

symbols and the corresponding

original subterms. However,

such an improvement would

become very complicated in the

formalization of [28].

Therefore, in contrast to [28],

in our new notion of DPs, we

annotate defined symbols directly

in the original rewrite rule

instead of extracting annotated

subterms from its right-hand

side. This makes the definition

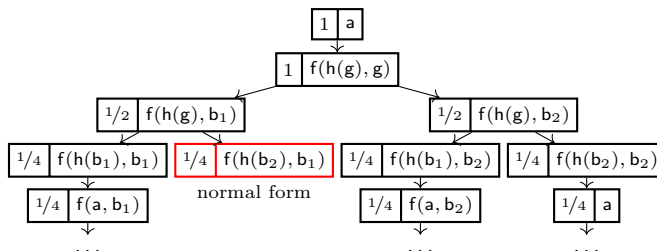
easier, more elegant, and more

readable, and allows us to solve

the incompleteness problem of

[28].

Definition 7 (Annotations). Let $t \in \mathcal{T}(\Sigma^\#, \mathcal{V})$ be an annotated term and for $\Sigma' \subseteq \Sigma^\#$, let $\text{pos}_{\Sigma'}(t)$ be all positions of t with symbols from Σ' . For a set of positions $\Phi \subseteq \text{pos}_{\mathcal{D} \cup \mathcal{D}^\#}(t)$, 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, $\text{pos}_{\mathcal{D}^\#}(\#_\Phi(t)) = \Phi$, and $\#_\emptyset(t)$ removes all annotations from t , where we often



write $\flat(t)$ instead of $\#_{\emptyset}(t)$. We extend \flat to multi-distributions, rules, and sets of rules by removing the annotations of all occurring terms. We write $\#_{\mathcal{D}}(t)$ instead of $\#_{\text{pos}_{\mathcal{D}}(t)}(t)$ to annotate all defined symbols in t , and $\#_{\varepsilon}(t)$ instead of $\#_{\{\varepsilon\}}(t)$ to annotate just the root symbol of t . Moreover, let $\flat_{\pi}^{\uparrow}(t)$ result from removing all annotations from t that are strictly above the position π . Finally, we write $t \trianglelefteq_{\#} s$ if there is a $\pi \in \text{pos}_{\mathcal{D}\#}(s)$ and $t = \flat(s|_{\pi})$, i.e., t results from a subterm of s with annotated root symbol by removing its annotation.

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

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

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

$$\mathcal{DP}(\ell \rightarrow \mu) = \ell \rightarrow \{p_1 : \#_{\mathcal{D}}(r_1), \dots, p_k : \#_{\mathcal{D}}(r_k)\}^{\text{true}}$$

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

Example 10. For \mathcal{R}_{rw} , the canonical ADP for $\mathbf{g}(x) \rightarrow \{1/2 : \mathbf{g}(\mathbf{g}(x)), 1/2 : x\}$ is $\mathbf{g}(x) \rightarrow \{1/2 : \mathbf{G}(\mathbf{G}(x)), 1/2 : x\}^{\text{true}}$ instead of the (complicated) DT from [28]:

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

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

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

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

$$\begin{aligned} t_j &= s[r_j\sigma]_{\pi} && \text{if } \pi \in \text{pos}_{\mathcal{D}\#}(s) \text{ and } m = \text{true} && \text{(pr)} \\ t_j &= \flat_{\pi}^{\uparrow}(s[r_j\sigma]_{\pi}) && \text{if } \pi \in \text{pos}_{\mathcal{D}\#}(s) \text{ and } m = \text{false} && \text{(p)} \\ t_j &= s[\flat(r_j)\sigma]_{\pi} && \text{if } \pi \notin \text{pos}_{\mathcal{D}\#}(s) \text{ and } m = \text{true} && \text{(r)} \\ t_j &= \flat_{\pi}^{\uparrow}(s[\flat(r_j)\sigma]_{\pi}) && \text{if } \pi \notin \text{pos}_{\mathcal{D}\#}(s) \text{ and } m = \text{false} && \text{(irr)} \end{aligned}$$

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

Rewriting with \mathcal{P} can be seen as ordinary term rewriting while considering and modifying annotations. In the ADP framework, we represent all DPs resulting

from a rule as well as the original rule by just one ADP. So for example, the ADP $\mathbf{g}(x) \rightarrow \{1/2 : \mathbf{G}(\mathbf{G}(x)), 1/2 : x\}^{\text{true}}$ for the rule $\mathbf{g}(x) \rightarrow \{1/2 : \mathbf{g}(\mathbf{g}(x)), 1/2 : x\}$ represents both DPs resulting from the two occurrences of \mathbf{g} on the right-hand side, and the rule itself (by simply disregarding all annotations of the ADP).

As in the classical DP framework, our goal is to track specific reduction sequences where (1) there are **p**-steps where the root symbol of the redex is annotated and a DP is applied, and (2) between two **p**-steps there can be several **r**-steps where rules are applied below the position of the next **p**-step.

A step of the form **(pr)** in Def. 11 can represent both **p**- and **r**-steps. All annotations are kept during this step except for annotations of the subterms that correspond to variables of the applied rule. These subterms are always in normal form due to the innermost evaluation strategy and we erase their annotations in order to handle rewriting with non-left-linear rules correctly. A **(pr)**-step at position π plays the role of an **r**-step for terms in multi-distributions where one later rewrites an annotated symbol at a position above π , and for all other terms it plays the role of a **p**-step. As an example, for a PTRS \mathcal{R}_{ex2} with the rules $\mathbf{g}(x, x) \rightarrow \{1 : \mathbf{f}(x)\}$ and $\mathbf{f}(a) \rightarrow \{1 : \mathbf{f}(b)\}$, we have the canonical ADPs $\mathbf{g}(x, x) \rightarrow \{1 : \mathbf{F}(x)\}^{\text{true}}$ and $\mathbf{f}(a) \rightarrow \{1 : \mathbf{F}(b)\}^{\text{true}}$, and we can rewrite $\mathbf{G}(\mathbf{F}(b), \mathbf{f}(b)) \xrightarrow{\text{c}}_{\mathcal{DP}(\mathcal{R}_{\text{ex2}})} \{1 : \mathbf{F}(\mathbf{f}(b))\}$ using the first ADP. Here, we have $\pi = \varepsilon$, $\mathfrak{b}(s|_{\varepsilon}) = \mathbf{g}(\mathbf{f}(b), \mathbf{f}(b)) = \ell\sigma$ where σ instantiates x with the normal form $\mathbf{f}(b)$, and $r_1 = \mathbf{F}(x)$.

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

A step of the form **(p)** represents a **p**-step. Thus, we remove all annotations above the position π , because no **p**-steps are possible above π . So if \mathcal{P} contains $\mathbf{f}(a) \rightarrow \{1 : \mathbf{F}(b)\}^{\text{false}}$, then $\mathbf{G}(\mathbf{F}(b), \mathbf{F}(a)) \xrightarrow{\text{c}}_{\mathcal{P}} \{1 : \mathbf{g}(\mathbf{F}(b), \mathbf{F}(b))\}$.

Finally, a step of the form **(irr)** is an **r**-step that is irrelevant for proving iAST, because due to $m = \text{false}$, afterwards there cannot be a **p**-step at a position above. For example, if \mathcal{P} again contains $\mathbf{f}(a) \rightarrow \{1 : \mathbf{F}(b)\}^{\text{false}}$, then $\mathbf{G}(\mathbf{F}(b), \mathbf{f}(a)) \xrightarrow{\text{c}}_{\mathcal{P}} \{1 : \mathbf{g}(\mathbf{F}(b), \mathbf{f}(b))\}$. Such **(irr)**-steps are needed to ensure that all rewrite steps with \mathcal{R} are also possible with the ADP problems \mathcal{P} that result from $\mathcal{DP}(\mathcal{R})$ when applying ADP processors. These processors only modify annotations, but keep the rest of the rules unchanged. So for all these ADP problems \mathcal{P} , we have $\mathcal{R} = \mathfrak{b}(\mathcal{P})$ and $\mathfrak{b}(t) \in \text{ANF}_{\mathcal{R}}$ iff $t \in \text{ANF}_{\mathcal{P}}$ for all $t \in \mathcal{T}(\Sigma^{\#}, \mathcal{V})$, i.e., the innermost evaluation strategy is not affected by the application of ADP processors. This is different from the classical DP framework, where the usable rules processor reduces the number of rules. This may result in new redexes that are allowed for innermost rewriting. Thus, the usable rules processor in our new ADP framework is *complete*, whereas in [17], one has to extend DP problems by an additional component to achieve completeness of this processor (see Footnote 4).

Now, $s \xrightarrow{\text{c}}_{\mathcal{R}} \{p_1 : t_1, \dots, p_k : t_k\}$ essentially⁶ implies $\#_{\mathcal{D}}(s) \xrightarrow{\text{c}}_{\mathcal{DP}(\mathcal{R})} \{p_1 : \#_{\mathcal{D}}(t_1), \dots, p_k : \#_{\mathcal{D}}(t_k)\}$, and we got rid of any ambiguities in the rewrite relation

⁶ We have $\#_{\mathcal{D}}(s) \xrightarrow{\text{c}}_{\mathcal{DP}(\mathcal{R})} \{p_1 : t'_1, \dots, p_k : t'_k\}$ where t'_j and $\#_{\mathcal{D}}(t_j)$ are the same up to some annotations of subterms that are $\mathcal{DP}(\mathcal{R})$ -normal forms. The reason is that as mentioned above, annotations of the subterms (in normal form) that correspond to

that led to incompleteness in [28]. While our ADPs are much simpler than the DTs of [28], due to their annotations they still contain all information that is needed to define the required DP processors.

Instead of chains of DPs, in the probabilistic setting one works with *chain trees* [28], where **p**- and **r**-steps are indicated by *P*- and *R*-nodes in the tree. Chain trees are defined analogously to RSTs, but the crucial requirement is that every infinite path of the tree must contain infinitely many steps of the forms (**pr**) or (**p**). Thus, in our setting $\mathfrak{T} = (V, E, L, P)$ is a \mathcal{P} -chain tree (CT) if

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

Let $|\mathfrak{T}|_{\mathbf{Leaf}} = \sum_{v \in \mathbf{Leaf}} p_v$. We define that \mathcal{P} is iAST if $|\mathfrak{T}|_{\mathbf{Leaf}} = 1$ for all \mathcal{P} -CTs \mathfrak{T} . So Conditions 1–5 ensure that the chain tree corresponds to an RST and Condition 6 requires that one may only use finitely many **r**-steps before the next **p**-step. This yields a chain criterion as in the non-probabilistic setting, where (in contrast to the chain criterion of [28]) we again have “iff” instead of “if”.

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

Our chain criterion is complete (“only if”), because ADPs only add annotations to rules. Hence, every $\mathcal{DP}(\mathcal{R})$ -CT can be turned into an \mathcal{R} -RST by omitting all annotations. So in contrast to [28], the step from the original PTRS to ADPs does not affect the “potential power” of the approach. Moreover, in the future this may also allow the development of techniques to *disprove* iAST within the ADP framework. To prove soundness (“if”), one has to show that every \mathcal{R} -RST can be simulated by a $\mathcal{DP}(\mathcal{R})$ -CT. As mentioned, all proofs can be found in App. A.

4 The ADP Framework

Our new (probabilistic) ADP framework again applies processors to transform an ADP problem into simpler sub-problems. An *ADP processor* Proc has the form $\text{Proc}(\mathcal{P}) = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, where $\mathcal{P}, \mathcal{P}_1, \dots, \mathcal{P}_n$ are ADP problems. Proc is *sound* if \mathcal{P} is iAST whenever \mathcal{P}_i is iAST for all $1 \leq i \leq n$. It is *complete* if \mathcal{P}_i is iAST for all $1 \leq i \leq n$ whenever \mathcal{P} is iAST. For a PTRS \mathcal{R} , one starts with the canonical

variables of the rule are erased. So for example, rewriting $G(F(\mathbf{b}), F(\mathbf{b}))$ with $\mathcal{DP}(\mathcal{R}_{\text{ex2}})$ yields $\{1 : F(\mathbf{f}(\mathbf{b}))\}$ and not $\{1 : F(F(\mathbf{b}))\}$.

ADP problem $\mathcal{DP}(\mathcal{R})$ and applies sound (and preferably complete) ADP processors repeatedly until the ADPs contain no annotations anymore. Such an ADP problem is trivially iAST. The framework again allows for modular termination proofs, since different techniques can be applied on each sub-problem \mathcal{P}_i .

We now adapt the processors from [28] to our new framework. The (innermost) \mathcal{P} -dependency graph is a control flow graph between ADPs from \mathcal{P} , indicating whether an ADP α may lead to an application of another ADP α' on an annotated subterm introduced by α . This possibility is not related to the probabilities. Hence, we can use the *non-probabilistic variant* $\text{np}(\mathcal{P}) = \{\ell \rightarrow \text{b}(r_j) \mid \ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^{\text{true}} \in \mathcal{P}, 1 \leq j \leq k\}$, which is an ordinary TRS over the signature Σ . Note that for $\text{np}(\mathcal{P})$ we only need to consider rules with the flag `true`, since only such rules can be used at a position below the next `p`-step.

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

So there is an edge from an ADP α to an ADP α' if after a step of the form `(pr)` or `(p)` with α at position π there may eventually come another step of the form `(pr)` or `(p)` with α' on or below π . Hence, for every path in a \mathcal{P} -CT from a P -node where an annotated subterm $f^{\#}(\dots)$ is introduced to the next P -node where the subterm $f^{\#}(\dots)$ at this position is rewritten, there is a corresponding edge in the \mathcal{P} -dependency graph. Since every infinite path in a CT contains infinitely many nodes from P , every such path traverses a cycle of the dependency graph infinitely often. Thus, it suffices to consider the SCCs of the dependency graph separately. In our framework, this means that we remove the annotations from all rules except those that are in the SCC that we want to analyze. As in [28], to automate the following two processors, the same over-approximation techniques as for the non-probabilistic dependency graph can be used.

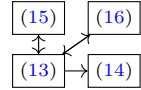
Theorem 14 (Probabilistic Dependency Graph Processor). *For the SCCs $\mathcal{P}_1, \dots, \mathcal{P}_n$ of the \mathcal{P} -dependency graph, $\text{Proc}_{\text{DG}}(\mathcal{P}) = \{\mathcal{P}_1 \cup \text{b}(\mathcal{P} \setminus \mathcal{P}_1), \dots, \mathcal{P}_n \cup \text{b}(\mathcal{P} \setminus \mathcal{P}_n)\}$ is sound and complete.*

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

$$\mathbf{a} \rightarrow \{1 : \text{F}(\text{H}(\mathbf{G}), \mathbf{G})\}^{\text{true}} \quad (13) \qquad \mathbf{h}(\mathbf{b}_1) \rightarrow \{1 : \mathbf{A}\}^{\text{true}} \quad (15)$$

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

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



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

Theorem 16 (Usable Terms Processor). *Let $\ell_1 \in \mathcal{T}(\Sigma, \mathcal{V})$ and \mathcal{P} be an ADP problem. We call $t \in \mathcal{T}(\Sigma^{\#}, \mathcal{V})$ with $\text{root}(t) \in \mathcal{D}^{\#}$ usable w.r.t. ℓ_1 and \mathcal{P} if there*

are substitutions σ_1, σ_2 and an $\ell_2 \rightarrow \mu_2 \in \mathcal{P}$ where μ_2 contains an annotated symbol, such that $\#_\varepsilon(t)\sigma_1 \xrightarrow{i}_{\text{np}(\mathcal{P})}^* \ell_2^\# \sigma_2$ and both $\ell_1\sigma_1$ and $\ell_2\sigma_2$ are in $\text{ANF}_{\mathcal{P}}$. Let $\flat_{\ell, \mathcal{P}}(s)$ result from s by removing the annotations from the roots of all its subterms that are not usable w.r.t. ℓ and \mathcal{P} , i.e., $\text{pos}_{\mathcal{D}^\#}(\flat_{\ell, \mathcal{P}}(s)) = \{\pi \in \text{pos}_{\mathcal{D}^\#}(s) \mid s|_\pi \text{ is usable w.r.t. } \ell_1 \text{ and } \mathcal{P}\}$. The transformation that removes the annotations from the roots of all non-usable terms in the right-hand sides of ADPs is $\mathcal{T}_{\text{UT}}(\mathcal{P}) = \{\ell \rightarrow \{p_1 : \flat_{\ell, \mathcal{P}}(r_1), \dots, p_k : \flat_{\ell, \mathcal{P}}(r_k)\}^m \mid \ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in \mathcal{P}\}$. Then $\text{Proc}_{\text{UT}}(\mathcal{P}) = \{\mathcal{T}_{\text{UT}}(\mathcal{P})\}$ is sound and complete.

So for $\mathcal{DP}(\mathcal{R}_{\text{incpl}})$, Proc_{UT} replaces (13) by $\mathbf{a} \rightarrow \{1 : \mathbf{F}(\mathbf{H}(\mathbf{g}), \mathbf{g})\}^{\text{true}}$ (13').

As in Thm. 3 of the ordinary DP framework, the idea of the *usable rules processor* remains to find rules that cannot be used below steps at annotations in right-hand sides of ADPs when their variables are instantiated with normal forms.

Theorem 17 (Prob. Usable Rules Processor). *For an ADP problem \mathcal{P} and $f \in \Sigma^\#$, let $\text{Rules}_{\mathcal{P}}(f) = \{\ell \rightarrow \mu^{\text{true}} \in \mathcal{P} \mid \text{root}(\ell) = f\}$. For any $t \in \mathcal{T}(\Sigma^\#, \mathcal{V})$, its usable rules $\mathcal{U}_{\mathcal{P}}(t)$ are the smallest set with $\mathcal{U}_{\mathcal{P}}(x) = \emptyset$ for all $x \in \mathcal{V}$ and $\mathcal{U}_{\mathcal{P}}(f(t_1, \dots, t_n)) = \text{Rules}_{\mathcal{P}}(f) \cup \bigcup_{i=1}^n \mathcal{U}_{\mathcal{P}}(t_i) \cup \bigcup_{\ell \rightarrow \mu^{\text{true}} \in \text{Rules}_{\mathcal{P}}(f), r \in \text{Supp}(\mu)} \mathcal{U}_{\mathcal{P}}(\flat(r))$, otherwise. The usable rules for \mathcal{P} are $\mathcal{U}(\mathcal{P}) = \bigcup_{\ell \rightarrow \mu^m \in \mathcal{P}, r \in \text{Supp}(\mu), t \triangleleft_{\#} r} \mathcal{U}_{\mathcal{P}}(t^\#)$. Then $\text{Proc}_{\text{UR}}(\mathcal{P}) = \{\mathcal{U}(\mathcal{P}) \cup \{\ell \rightarrow \mu^{\text{false}} \mid \ell \rightarrow \mu^m \in \mathcal{P} \setminus \mathcal{U}(\mathcal{P})\}\}$ is sound and complete, i.e., we turn the flag of all non-usable rules to false.*

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

Finally, we adapt the reduction pair processor. Here, (1) for every rule with the flag **true** (which can therefore be used for **r**-steps), the expected value must be weakly decreasing when removing the annotations. Since rules can also be used for **p**-steps, (2) we also require a weak decrease when comparing the annotated left-hand side with the expected value of all annotated subterms in the right-hand side. Since we sum up the values of the annotated subterms of each right-hand side, we can again use *weakly monotonic* interpretations. As in [3, 28], to ensure “monotonicity” w.r.t. expected values we have to restrict ourselves to interpretations with multilinear polynomials, where all monomials have the form $c \cdot x_1^{e_1} \cdot \dots \cdot x_n^{e_n}$ with $c \in \mathbb{N}$ and $e_1, \dots, e_n \in \{0, 1\}$. The processor then removes the annotations from those ADPs where (3) in addition there is at least one right-hand side r_j whose annotated subterms are strictly decreasing.⁷

⁷ In addition, the corresponding non-annotated right-hand side $\flat(r_j)$ must be at least weakly decreasing. The reason is that in contrast to the original DP framework, we may now have nested annotated symbols and thus, we have to ensure that they behave “monotonically”. So we have to ensure that $\text{Pol}(A) > \text{Pol}(B)$ also implies that the measure of $F(A)$ is greater than $F(B)$. Every term r is “measured” as $\sum_{t \triangleleft_{\#} r} \text{Pol}(t^\#)$, i.e., $F(A)$ is measured as $\text{Pol}(F(a)) + \text{Pol}(A)$. Hence, in this example we must ensure that $\text{Pol}(A) > \text{Pol}(B)$ implies $\text{Pol}(F(a)) + \text{Pol}(A) > \text{Pol}(F(b)) + \text{Pol}(B)$. For that reason, we also have to require $\text{Pol}(a) \geq \text{Pol}(b)$.

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

- (1) *For every $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^{\text{true}} \in \mathcal{P}$, we have $\text{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(b(r_j))$.*
- (2) *For every $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in \mathcal{P}$, we have $\text{Pol}(\ell^\#) \geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \trianglelefteq_{\#} r_j} \text{Pol}(t^\#)$.*
- (3) *For every $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in \mathcal{P}_{>}$, there exists a $1 \leq j \leq k$ with $\text{Pol}(\ell^\#) > \sum_{t \trianglelefteq_{\#} r_j} \text{Pol}(t^\#)$.*
If $m = \text{true}$, then we additionally have $\text{Pol}(\ell) \geq \text{Pol}(b(r_j))$.

Then $\text{Proc}_{\text{RP}}(\mathcal{P}) = \{\mathcal{P}_{\geq} \cup b(\mathcal{P}_{>})\}$ is sound and complete.

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

$$a \rightarrow \{1/4 : f(\mathbf{H}(\mathbf{b}_1), \mathbf{b}_1), 1/4 : f(\mathbf{h}(\mathbf{b}_2), \mathbf{b}_1), 1/4 : F(\mathbf{H}(\mathbf{b}_1), \mathbf{b}_2), 1/4 : F(\mathbf{h}(\mathbf{b}_2), \mathbf{b}_2)\}^{\text{true}} \quad (13'')$$

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

$$g \rightarrow \{1/2 : \mathbf{b}_1, 1/2 : \mathbf{b}_2\}^{\text{true}} \quad (14) \quad \mathbf{h}(\mathbf{b}_1) \rightarrow \{1 : \mathbf{A}\}^{\text{true}} \quad (15) \quad f(x, \mathbf{b}_2) \rightarrow \{1 : \mathbf{A}\}^{\text{false}} \quad (16')$$

we use the reduction pair processor with the polynomial interpretation that maps \mathbf{A} , \mathbf{F} , and \mathbf{H} to 1 and all other symbols to 0, to remove all annotations from the a-ADP (13''), because it contains the right-hand side $f(\mathbf{h}(\mathbf{b}_2), \mathbf{b}_1)$ without annotations and thus, $\text{Pol}(\mathbf{A}) = 1 > \sum_{t \trianglelefteq_{\#} f(\mathbf{h}(\mathbf{b}_2), \mathbf{b}_1)} \text{Pol}(t^\#) = 0$. Another application of the usable terms processor removes the remaining \mathbf{A} -annotations from (15) and (16'). Since there are no more annotations left, this proves iAST of $\mathcal{R}_{\text{incompl}}$.

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

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

5 Transforming ADPs

Compared to the DT framework for PTRSs in [28], our new ADP framework is not only easier, more elegant, and yields a complete chain criterion, but it also has important practical advantages, because every processor that performs a rewrite step benefits from our novel definition of rewriting with ADPs (whereas the rewrite relation with DTs in [28] was an “incomplete over-approximation” of the rewrite relation of the original TRS). To illustrate this, we adapt the *rewriting processor* from the original DP framework [18] to the probabilistic setting, which allows us to prove iAST of $\mathcal{R}_{\text{incompl}}$ from Ex. 5. Such transformational processors had not been

adapted in the probabilistic DT framework of [28]. While one could also adapt the rewriting processor to the setting of [28], then it would be substantially weaker, and we would fail in proving iAST of $\mathcal{R}_{\text{incompl}}$. We refer to App. B for our adaptation of the remaining transformational processors from [18] (based on *instantiation*, *forward instantiation*, and *narrowing*) to the probabilistic setting.

In the non-probabilistic setting, the rewriting processor may rewrite a redex in the right-hand side of a DP if this does not affect the construction of chains. To ensure that, the usable rules for this redex must be non-overlapping (NO). If the DP occurs in a chain, then this redex is weakly innermost terminating, hence by NO also terminating and confluent, and thus, it has a unique normal form [22].

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

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

$$\text{Proc}_{\mathbf{r}}(\mathcal{P}) = \left\{ \mathcal{P}' \cup \left\{ \begin{array}{l} \ell \rightarrow \{p_1 : b(r_1), \dots, p_k : b(r_k)\}^m, \\ \ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\} \setminus \{p_j : r_j\} \\ \cup \{p_j \cdot q_1 : e_1, \dots, p_j \cdot q_h : e_h\}^m \end{array} \right\} \right\}$$

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

Theorem 23 (Soundness⁸ of the Rewriting Processor). *Proc_r as in Def. 22 is sound if one of the following cases holds:*

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

We refer to App. A for a discussion on the requirements L and NE in the first case (see the counterexamples in Ex. 35 and 36). The second case corresponds to

⁸ For completeness in the non-probabilistic setting [18], one uses a different definition of “non-terminating” (or “infinite”) DP problems. In future work, we will examine if such a definition would also yield completeness of $\text{Proc}_{\mathbf{r}}$ in the probabilistic case.

the original rewrite processor where all usable rules of $r_j|_\tau$ are non-probabilistic. In the last case, for any instantiation only a single innermost rewrite step is possible for $r_j|_\tau$. The restriction to innermost rewrite steps is only useful if $r_j|_\tau$ is ground. Otherwise, an innermost step on $r_j|_\tau$ might become a non-innermost step when instantiating $r_j|_\tau$'s variables.

The rewriting processor benefits from our ADP framework, because it applies the rewrite relation $\hookrightarrow_{\mathcal{P}}$. In contrast, a rewriting processor in the DT framework of [28] would have to replace a DT by *multiple* new DTs, due to the ambiguities in their rewrite relation. Such a rewriting processor would fail for $\mathcal{R}_{\text{incpl}}$ whereas with the processor of [Thm. 23](#) we can now prove that $\mathcal{R}_{\text{incpl}}$ is iAST.

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

$$\begin{aligned} \mathbf{a} &\rightarrow \{1 : \mathbf{F}(\mathbf{H}(\mathbf{g}), \mathbf{g})\}^{\text{true}} & (13') & \quad \mathbf{h}(\mathbf{b}_1) &\rightarrow \{1 : \mathbf{A}\}^{\text{true}} & (15) \\ \mathbf{g} &\rightarrow \{1/2 : \mathbf{b}_1, 1/2 : \mathbf{b}_2\}^{\text{true}} & (14) & \quad \mathbf{f}(x, \mathbf{b}_2) &\rightarrow \{1 : \mathbf{A}\}^{\text{false}} & (16') \end{aligned}$$

Now we can apply the rewriting processor on (13') repeatedly until all \mathbf{g} s are rewritten and replace it by the ADP $\mathbf{a} \rightarrow \{1/4 : \mathbf{F}(\mathbf{H}(\mathbf{b}_1), \mathbf{b}_1), 1/4 : \mathbf{F}(\mathbf{H}(\mathbf{b}_2), \mathbf{b}_1), 1/4 : \mathbf{F}(\mathbf{H}(\mathbf{b}_1), \mathbf{b}_2), 1/4 : \mathbf{F}(\mathbf{H}(\mathbf{b}_2), \mathbf{b}_2)\}^{\text{true}}$ as well as several resulting ADPs $\mathbf{a} \rightarrow \dots$ without annotations. Now in the subterms $\mathbf{F}(\dots, \mathbf{b}_1)$ and $\mathbf{H}(\mathbf{b}_2)$, the annotations are removed from the roots by the usable terms processor, as these subterms cannot rewrite to annotated instances of left-hand sides of ADPs. So the \mathbf{a} -ADP is changed to $\mathbf{a} \rightarrow \{1/4 : \mathbf{f}(\mathbf{H}(\mathbf{b}_1), \mathbf{b}_1), 1/4 : \mathbf{f}(\mathbf{h}(\mathbf{b}_2), \mathbf{b}_1), 1/4 : \mathbf{F}(\mathbf{H}(\mathbf{b}_1), \mathbf{b}_2), 1/4 : \mathbf{F}(\mathbf{h}(\mathbf{b}_2), \mathbf{b}_2)\}^{\text{true}}$ (13''). This ADP corresponds to the observations that explain why $\mathcal{R}_{\text{incpl}}$ is iAST in Ex. 5: We have two terms in the right-hand side that correspond to one \mathbf{A} each, both with probability $1/4$, one term that corresponds to a normal form with probability $1/4$, and one that corresponds to two \mathbf{A} s with probability $1/4$. So again this corresponds to a random walk where the number of \mathbf{A} s is decreased by 1 with probability $1/4$, increased by 1 with probability $1/4$, and kept the same with probability $1/2$. Then we use the reduction pair processor as in [Ex. 20](#) to prove iAST for $\mathcal{R}_{\text{incpl}}$.

6 Conclusion and Evaluation

We developed a new ADP framework, which advances our work in [28] into a *complete* criterion for almost-sure innermost termination by using annotated DPs instead of dependency tuples, which also simplifies the framework substantially. Moreover, we adapted the *rewriting* processor of the classic DP framework to the probabilistic setting. We also adapted the other transformational processors of the non-probabilistic DP framework, see [App. B](#). The soundness proofs for the adapted processors are much more involved than in the non-probabilistic setting, due to the more complex structure of chain trees. However, the processors themselves are analogous to their non-probabilistic counterparts, and thus, existing implementations of the processors can easily be adapted to their probabilistic versions.

We implemented our new contributions in our termination prover AProVE [19] and compared the new probabilistic ADP framework with transformational processors (ADP) to the DT framework from [28] (DT) and to AProVE's techniques

for ordinary non-probabilistic TRSs (AProVE-NP), which include many additional processors and which benefit from using separate dependency pairs instead of ADPs or DTs. For the processors in Sect. 4, we could re-use the existing implementation of [28] for our ADP framework. The main goal for probabilistic termination analysis is to become as powerful as termination analysis in the non-probabilistic setting. Therefore, in our first experiment, we considered the non-probabilistic TRSs of the *TPDB* [38] (the benchmark set used in the annual *Termination and Complexity Competition (TermComp)* [20]) and compared ADP and DT with AProVE-NP, because at the current *TermComp*, AProVE-NP was the most powerful tool for termination of ordinary non-probabilistic TRSs. Clearly, a TRS can be represented as a PTRS with trivial probabilities, and then (innermost) AST is the same as (innermost) termination. While both ADP and DT have a probability removal processor to switch to the classical DP framework for such problems, we disabled that processor in this experiment. Since ADP and DT can only deal with innermost evaluation, we used the benchmarks from the “TRS Innermost” and “TRS Standard” categories of the *TPDB*, but only considered innermost evaluation for all examples. We used a timeout of 300 seconds for each example. The “TRS Innermost” category contains 366 benchmarks, where AProVE-NP proves innermost termination for 293, DT is able to prove it for 133 (45% of AProVE-NP), and for ADP this number rises to 159 (54%). For the 1512 benchmarks from the “TRS Standard” category, AProVE-NP can prove innermost termination for 1114, DT for 611 (55% of AProVE-NP), and ADP for 723 (65%). This shows that the transformations are very important for automatic termination proofs as we get around 10% closer to AProVE-NP’s results in both categories.

As a second experiment, we extended the PTRS benchmark set from [28] by 33 new PTRSs for typical probabilistic programs, including some examples with complicated probabilistic structure. For instance, we added the following PTRS $\mathcal{R}_{\text{qsrt}}$ for probabilistic quicksort. Here, we write r instead of $\{1 : r\}$ for readability.

$$\begin{aligned} \text{rotate}(\text{cons}(x, xs)) &\rightarrow \{1/2 : \text{cons}(x, xs), 1/2 : \text{rotate}(\text{app}(xs, \text{cons}(x, \text{nil})))\} \\ \text{qsrt}(xs) &\rightarrow \text{if}(\text{empty}(xs), \text{low}(\text{hd}(xs), \text{tl}(xs)), \text{hd}(xs), \text{high}(\text{hd}(xs), \text{tl}(xs))) \\ \text{if}(\text{true}, xs, x, ys) &\rightarrow \text{nil} \quad \text{empty}(\text{nil}) \rightarrow \text{true} \quad \text{empty}(\text{cons}(x, xs)) \rightarrow \text{false} \\ \text{if}(\text{false}, xs, x, ys) &\rightarrow \text{app}(\text{qsrt}(\text{rotate}(xs)), \text{cons}(x, \text{qsrt}(\text{rotate}(ys)))) \\ \text{hd}(\text{cons}(x, xs)) &\rightarrow x \quad \text{tl}(\text{cons}(x, xs)) \rightarrow xs \end{aligned}$$

The *rotate*-rules rotate a list randomly often (they are AST, but not terminating). Thus, by choosing the first element of the resulting list, one obtains random pivot elements for the recursive calls of *qsrt* in the second if-rule. In addition to the rules above, $\mathcal{R}_{\text{qsrt}}$ contains rules for list concatenation (*app*), and rules such that *low*(x, xs) (*high*(x, xs)) returns all elements of the list xs that are smaller (greater or equal) than x , see App. C.1. In contrast to the quicksort example in [28], proving iAST of the above rules requires transformational processors to instantiate and rewrite the *empty*-, *hd*-, and *tl*-subterms in the right-hand side of the *qsrt*-rule. So while DT fails for this example, ADP can prove iAST of $\mathcal{R}_{\text{qsrt}}$.

90 of the 100 PTRSs in our set are iAST, and DT succeeds for 54 of them (60 %) with the technique of [28] that does not use transformational processors. Adding the new processors in ADP increases this number to 77 (86 %), which demonstrates their power for PTRSs with non-trivial probabilities. For details on

our experiments and for instructions on how to run our implementation in AProVE via its *web interface* or locally, see:

<https://aprove-developers.github.io/ProbabilisticADPs/>

On this website, we also performed experiments where we disabled individual transformational processors of the ADP framework, which shows the usefulness of each new processor. In addition to the ADP and DT framework, an alternative technique to analyze PTRSs via a direct application of interpretations was presented in [3]. However, [3] analyzes PAST (or rather *strong* AST), and a comparison between the DT framework and their technique can be found in [28]. In future work, we will adapt more processors of the DP framework to the probabilistic setting. Moreover, we work on analyzing AST also for full instead of innermost rewriting and already developed criteria when iAST implies full AST [30].

References

- [1] S. Agrawal, K. Chatterjee, and P. Novotný. “Lexicographic Ranking Supermartingales: An Efficient Approach to Termination of Probabilistic Programs”. In: *Proc. ACM Program. Lang.* 2.POPL (2017). DOI: [10.1145/3158122](https://doi.org/10.1145/3158122).
- [2] T. Arts and J. Giesl. “Termination of Term Rewriting Using Dependency Pairs”. In: *Theor. Comput. Sc.* 236.1-2 (2000), pp. 133–178. DOI: [10.1016/S0304-3975\(99\)00207-8](https://doi.org/10.1016/S0304-3975(99)00207-8).
- [3] M. Avanzini, U. Dal Lago, and A. Yamada. “On Probabilistic Term Rewriting”. In: *Sci. Comput. Program.* 185 (2020). DOI: [10.1016/j.scico.2019.102338](https://doi.org/10.1016/j.scico.2019.102338).
- [4] M. Avanzini, G. Moser, and M. Schaper. “A Modular Cost Analysis for Probabilistic Programs”. In: *Proc. ACM Program. Lang.* 4.OOPSLA (2020). DOI: [10.1145/3428240](https://doi.org/10.1145/3428240).
- [5] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. DOI: [10.1017/CBO9781139172752](https://doi.org/10.1017/CBO9781139172752).
- [6] K. Batz, B. L. Kaminski, J.-P. Katoen, C. Matheja, and L. Verscht. “A Calculus for Amortized Expected Runtimes”. In: *Proc. ACM Program. Lang.* 7.POPL (2023). DOI: [10.1145/3571260](https://doi.org/10.1145/3571260).
- [7] R. Beutner and L. Ong. “On Probabilistic Termination of Functional Programs with Continuous Distributions”. In: *Proc. PLDI '21*. 2021, pp. 1312–1326. DOI: [10.1145/3453483.3454111](https://doi.org/10.1145/3453483.3454111).
- [8] O. Bournez and C. Kirchner. “Probabilistic Rewrite Strategies. Applications to ELAN”. In: *Proc. RTA '02*. LNCS 2378. 2002, pp. 252–266. DOI: [10.1007/3-540-45610-4_18](https://doi.org/10.1007/3-540-45610-4_18).
- [9] O. Bournez and F. Garnier. “Proving Positive Almost-Sure Termination”. In: *Proc. RTA '05*. LNCS 3467. 2005, pp. 323–337. DOI: [10.1007/978-3-540-32033-3_24](https://doi.org/10.1007/978-3-540-32033-3_24).
- [10] K. Chatterjee, H. Fu, and P. Novotný. “Termination Analysis of Probabilistic Programs with Martingales”. In: *Foundations of Probabilistic Programming*. Ed. by G. Barthe, J.-P. Katoen, and A. Silva. Cambridge University Press, 2020, 221–258. DOI: [10.1017/9781108770750.008](https://doi.org/10.1017/9781108770750.008).

- [11] U. Dal Lago and C. Grellois. “Probabilistic Termination by Monadic Affine Sized Typing”. In: *Proc. ESOP '17*. LNCS 10201. 2017, pp. 393–419. DOI: [10.1007/978-3-662-54434-1_15](https://doi.org/10.1007/978-3-662-54434-1_15).
- [12] U. Dal Lago, C. Faggian, and S. R. Della Rocca. “Intersection Types and (Positive) Almost-Sure Termination”. In: *Proc. ACM Program. Lang.* 5:POPL (2021). DOI: [10.1145/3434313](https://doi.org/10.1145/3434313).
- [13] C. Faggian. “Probabilistic Rewriting and Asymptotic Behaviour: On Termination and Unique Normal Forms”. In: *Log. Methods in Comput. Sci.* 18.2 (2022). DOI: [10.46298/lmcs-18\(2:5\)2022](https://doi.org/10.46298/lmcs-18(2:5)2022).
- [14] L. M. Ferrer Fioriti and H. Hermanns. “Probabilistic Termination: Soundness, Completeness, and Compositionality”. In: *Proc. POPL '15*. 2015, pp. 489–501. DOI: [10.1145/2676726.2677001](https://doi.org/10.1145/2676726.2677001).
- [15] C. Fuhs, J. Giesl, M. Parting, P. Schneider-Kamp, and S. Swiderski. “Proving Termination by Dependency Pairs and Inductive Theorem Proving”. In: *J. Autom. Reason.* 47.2 (2011), pp. 133–160. DOI: [10.1007/s10817-010-9215-9](https://doi.org/10.1007/s10817-010-9215-9).
- [16] J. Giesl and T. Arts. “Verification of Erlang Processes by Dependency Pairs”. In: *Appl. Algebra Eng. Commun. Comput.* 12.1/2 (2001), pp. 39–72. DOI: [10.1007/s002000100063](https://doi.org/10.1007/s002000100063).
- [17] J. Giesl, R. Thiemann, and P. Schneider-Kamp. “The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs”. In: *Proc. LPAR '04*. LNCS 3452. 2004, pp. 301–331. DOI: [10.1007/978-3-540-32275-7_21](https://doi.org/10.1007/978-3-540-32275-7_21).
- [18] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. “Mechanizing and Improving Dependency Pairs”. In: *J. Autom. Reason.* 37.3 (2006), pp. 155–203. DOI: [10.1007/s10817-006-9057-7](https://doi.org/10.1007/s10817-006-9057-7).
- [19] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. “Analyzing Program Termination and Complexity Automatically with AProVE”. In: *J. Autom. Reason.* 58.1 (2017), pp. 3–31. DOI: [10.1007/s10817-016-9388-y](https://doi.org/10.1007/s10817-016-9388-y).
- [20] J. Giesl, A. Rubio, C. Sternagel, J. Waldmann, and A. Yamada. “The Termination and Complexity Competition”. In: *Proc. TACAS '19*. LNCS 11429. 2019, pp. 156–166. DOI: [10.1007/978-3-030-17502-3_10](https://doi.org/10.1007/978-3-030-17502-3_10).
- [21] J. Giesl, P. Giesl, and M. Hark. “Computing Expected Runtimes for Constant Probability Programs”. In: *Proc. CADE '19*. LNCS 11716. 2019, pp. 269–286. DOI: [10.1007/978-3-030-29436-6_16](https://doi.org/10.1007/978-3-030-29436-6_16).
- [22] B. Gramlich. “Abstract Relations between Restricted Termination and Confluence Properties of Rewrite Systems”. In: *Fundam. Informaticae* 24 (1995), pp. 2–23.
- [23] B. Gramlich. “Termination and confluence: properties of structured rewrite systems”. PhD thesis. University of Kaiserslautern, 1996. URL: <https://www.logic.at/staff/gramlich/papers/thesis96.pdf>.
- [24] N. Hirokawa and A. Middeldorp. “Automating the Dependency Pair Method”. In: *Inf. Comput.* 199.1-2 (2005), pp. 172–199. DOI: [10.1016/j.ic.2004.10.004](https://doi.org/10.1016/j.ic.2004.10.004).
- [25] M. Huang, H. Fu, K. Chatterjee, and A. K. Goharshady. “Modular Verification for Almost-Sure Termination of Probabilistic Programs”. In: *Proc. ACM Program. Lang.* 3:OOPSLA (2019). DOI: [10.1145/3360555](https://doi.org/10.1145/3360555).

- [26] B. L. Kaminski, J.-P. Katoen, C. Matheja, and F. Olmedo. “Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms”. In: *J. ACM* 65 (2018), pp. 1–68. DOI: [10.1145/3208102](https://doi.org/10.1145/3208102).
- [27] B. L. Kaminski, J.-P. Katoen, and C. Matheja. “Expected Runtime Analysis by Program Verification”. In: *Foundations of Probabilistic Programming*. Ed. by G. Barthe, J.-P. Katoen, and A. Silva. Cambridge University Press, 2020, 185–220. DOI: [10.1017/9781108770750.007](https://doi.org/10.1017/9781108770750.007).
- [28] J.-C. Kassing and J. Giesl. “Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs”. In: *Proc. CADE ’23*. LNCS 14132. 2023, pp. 344–364. DOI: [10.1007/978-3-031-38499-8_20](https://doi.org/10.1007/978-3-031-38499-8_20).
- [29] J.-C. Kassing and J. Giesl. “Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs”. In: *CoRR* abs/2305.11741 (2023). Full version of [28] with all proofs. DOI: [10.48550/arXiv.2305.11741](https://doi.org/10.48550/arXiv.2305.11741).
- [30] J.-C. Kassing, F. Frohn, and J. Giesl. “From Innermost to Full Almost-Sure Termination of Probabilistic Term Rewriting”. In: *Proc. FoSSaCS ’24*. LNCS. To appear. Long version available at *CoRR* abs/2310.06121. URL: <https://doi.org/10.48550/arXiv.2310.06121>. 2024.
- [31] D. S. Lankford. *On Proving Term Rewriting Systems are Noetherian*. Memo MTP-3, Math. Dept., Louisiana Technical University, Ruston, LA, 1979. URL: http://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford_Poly_Term.pdf.
- [32] L. Leutgeb, G. Moser, and F. Zuleger. “Automated Expected Amortised Cost Analysis of Probabilistic Data Structures”. In: *Proc. CAV ’22*. LNCS 13372. 2022, pp. 70–91. DOI: [10.1007/978-3-031-13188-2_4](https://doi.org/10.1007/978-3-031-13188-2_4).
- [33] A. McIver, C. Morgan, B. L. Kaminski, and J.-P. Katoen. “A New Proof Rule for Almost-Sure Termination”. In: *Proc. ACM Program. Lang.* 2.POPL (2018). DOI: [10.1145/3158121](https://doi.org/10.1145/3158121).
- [34] F. Meyer, M. Hark, and J. Giesl. “Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes”. In: *Proc. TACAS ’21*. LNCS 12651. 2021, pp. 250–269. DOI: [10.1007/978-3-030-72016-2_14](https://doi.org/10.1007/978-3-030-72016-2_14).
- [35] M. Moosbrugger, E. Bartocci, J.-P. Katoen, and L. Kovács. “Automated Termination Analysis of Polynomial Probabilistic Programs”. In: *Proc. ESOP ’21*. LNCS 12648. 2021, pp. 491–518. DOI: [10.1007/978-3-030-72019-3_18](https://doi.org/10.1007/978-3-030-72019-3_18).
- [36] V. C. Ngo, Q. Carbonneaux, and J. Hoffmann. “Bounded Expectations: Resource Analysis for Probabilistic Programs”. In: *Proc. PLDI ’18*. 2018, pp. 496–512. DOI: [10.1145/3192366.3192394](https://doi.org/10.1145/3192366.3192394).
- [37] L. Noschinski, F. Emmes, and J. Giesl. “Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs”. In: *J. Autom. Reason.* 51 (2013), pp. 27–56. DOI: [10.1007/978-3-642-22438-6_32](https://doi.org/10.1007/978-3-642-22438-6_32).
- [38] *Termination Problem Data Base*. <https://github.com/TermCOMP/TPDB>.
- [39] R. Thiemann. “The DP Framework for Proving Termination of Term Rewriting”. PhD thesis. RWTH Aachen University, 2007. URL: <https://verify.rwth-aachen.de/da/thiemann-diss.pdf>.
- [40] D. Wang, D. M. Kahn, and J. Hoffmann. “Raising Expectations: Automating Expected Cost Analysis with Types”. In: *Proc. ACM Program. Lang.* 4.ICFP (2020). DOI: [10.1145/3408992](https://doi.org/10.1145/3408992).

Appendix

In [App. A](#) we give all proofs for our new results and observations. Then in [App. B](#) we adapt further transformational DP processors to the probabilistic setting. More precisely, we consider the *instantiation*, *forward instantiation*, and the *narrowing* processor. Because the original *narrowing* processor turns out to be unsound in the probabilistic setting, the probabilistic narrowing processor only instantiates variables according to the narrowing substitutions. Hence, we call it the *rule overlap instantiation* processor. Finally, in [App. C](#), we present some examples from our collection of benchmarks that demonstrate certain aspects of our new contributions.

A Proofs

We start by showing that $\mathcal{R}_{\text{incompl}}$ is a counterexample for completeness of the DT framework from [28]. First, we introduce the framework and all needed notation.

For any term $t \in \mathcal{T}(\Sigma, \mathcal{V})$ with $f \in \mathcal{D}$, we say that t is the *flattened copy* of $t^\#$. In the probabilistic adaption of the DP framework from [28], for any term r in the right-hand side of a rule, one has to consider all subterms of r with defined root symbol *at once*. In order to deal just with terms instead of multisets, one defines r 's *dt transformation* $dt(r) = \text{com}_n(t_1^\#, \dots, t_n^\#)$, if $\{t_1, \dots, t_n\}$ are all subterms of r with defined root symbol. Here, \mathcal{C} is extended by fresh *compound* constructor symbols com_n of arity n for every $n \in \mathbb{N}$. To make $dt(r)$ unique, one uses the lexicographic ordering $<$ on positions where $t_i = r|_{\pi_i}$ and $\pi_1 < \dots < \pi_n$. As an example, $dt(\mathbf{g}^2(x)) = \text{com}_2(\mathbf{G}(\mathbf{g}(x)), \mathbf{G}(x))$.

To abstract from nested compound symbols and from the order of their arguments, the following normalization is introduced. For any term t , its *content* $\text{cont}(t)$ is the multiset defined by $\text{cont}(\text{com}_n(t_1, \dots, t_n)) = \text{cont}(t_1) \cup \dots \cup \text{cont}(t_n)$ and $\text{cont}(t) = \{t\}$ for other terms t . For any term t with $\text{cont}(t) = \{t_1, \dots, t_n\}$, the term $\text{com}_n(t_1, \dots, t_n)$ is a *normalization* of t . For two terms t, t' , one defines $t \approx t'$ if $\text{cont}(t) = \text{cont}(t')$. So for example, $\text{com}_3(x, x, y)$ is a normalization of $\text{com}_2(\text{com}_1(x), \text{com}_2(y, x))$. One does not distinguish between terms that are equal w.r.t. \approx and writes $\text{com}_n(t_1, \dots, t_n)$ for any term t with a compound root symbol where $\text{cont}(t) = \{t_1, \dots, t_n\}$, i.e., all such t are considered to be normalized.

For every rule $\ell \rightarrow \mu = \{p_1 : r_1, \dots, p_k : r_k\}$, the corresponding *dependency tuple* (DT) relates $\ell^\#$ with $\{p_1 : dt(r_1), \dots, p_k : dt(r_k)\}$. However, in addition, DTs also store the original rule $\ell \rightarrow \mu$. As shown in [28], this is needed for the soundness of the approach, because otherwise, one cannot simulate every possible rewrite sequence with dependency tuples. So one defines $\mathcal{DT}(\ell \rightarrow \mu) = \langle \ell^\#, \ell \rangle \rightarrow \{p_1 : \langle dt(r_1), r_1 \rangle, \dots, p_k : \langle dt(r_k), r_k \rangle\}$. $\mathcal{DT}(\mathcal{R})$ denotes the set of all dependency tuples of a PTRS \mathcal{R} . For example, $\mathcal{DT}(\mathcal{R}_{\text{rw}}) = \{\langle \mathbf{G}(x), \mathbf{g}(x) \rangle \rightarrow \{1/2 : \langle \text{com}_2(\mathbf{G}(\mathbf{g}(x)), \mathbf{G}(x)), \mathbf{g}^2(x) \rangle, 1/2 : \langle \text{com}_0, x \rangle\}\}$. This type of rewrite system is called a *probabilistic pair term rewrite system (PPTRS)*.

Definition 25 (PPTRS, $\xrightarrow{i}_{\mathcal{P}, \mathcal{R}}$). Let \mathcal{P} be a finite set of rules of the form $\langle \ell^\#, \ell \rangle \rightarrow \{p_1 : \langle d_1, r_1 \rangle, \dots, p_k : \langle d_k, r_k \rangle\}$. For every such rule, let $\text{proj}_1(\mathcal{P})$ contain $\ell^\# \rightarrow \{p_1 : d_1, \dots, p_k : d_k\}$ and $\text{proj}_2(\mathcal{P})$ contain $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}$. If $\text{proj}_2(\mathcal{P})$

is a PTRS and $\text{cont}(d_j) \subseteq \text{cont}(dt(r_j))$ for all $1 \leq j \leq k$, then \mathcal{P} is a PPTRS.

A normalized term $\text{com}_n(s_1, \dots, s_n)$ rewrites with the PPTRS \mathcal{P} to $\{p_1 : b_1, \dots, p_k : b_k\}$ w.r.t. a PTRS \mathcal{R} (denoted $\xrightarrow{i} \mathcal{P}, \mathcal{R}$) if there are an $\langle \ell^\#, \ell \rangle \rightarrow \{p_1 : \langle d_1, r_1 \rangle, \dots, p_k : \langle d_k, r_k \rangle\} \in \mathcal{P}$, a substitution σ , and an $1 \leq i \leq n$ with $s_i = \ell^\# \sigma \in \text{ANF}_{\mathcal{R}}$, and for all $1 \leq j \leq k$ we have $b_j = \text{com}_n(t_1^j, \dots, t_n^j)$ where:

(a) $t_i^j = d_j \sigma$ for all $1 \leq j \leq k$, i.e., we rewrite the term s_i using $\text{proj}_1(\mathcal{P})$.

(b) For every $1 \leq i' \leq n$ with $i' \neq i$ we have

(i) $t_{i'}^j = s_{i'}$ for all $1 \leq j \leq k$ or

(ii) $t_{i'}^j = s_{i'}[r_j \sigma]_\tau$ for all $1 \leq j \leq k$,

if $s_{i'}|_\tau = \ell \sigma$ for some position τ and if $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{R}$.

So $s_{i'}$ stays the same in all b_j or one can apply the rule from $\text{proj}_2(\mathcal{P})$ to rewrite $s_{i'}$ in all b_j , provided that this rule is also contained in \mathcal{R} . Note that even if the rule is applicable, the term $s_{i'}$ can still stay the same in all b_j .

Now one can simulate the rewrite step $\mathbf{g}^2(x) \xrightarrow{i} \mathcal{R}_{\text{rw}} \{1/2 : \mathbf{g}^3(x), 1/2 : \mathbf{g}(x)\}$ by $\text{com}_2(\mathbf{G}(\mathbf{g}(x)), \mathbf{G}(x)) \xrightarrow{i} \mathcal{DT}(\mathcal{R}_{\text{rw}}), \mathcal{R}_{\text{rw}} \{1/2 : \text{com}_3(\mathbf{G}(\mathbf{g}^2(x)), \mathbf{G}(\mathbf{g}(x)), \mathbf{G}(x)), 1/2 : \text{com}_1(\mathbf{G}(x))\}$ using $\mathcal{DT}(\mathcal{R}_{\text{rw}})$. In $\text{com}_2(\mathbf{G}(\underline{\mathbf{g}(x)}), \underline{\mathbf{G}(x)})$, due to (a), the (underlined) second argument $s_i = s_2 = \underline{\mathbf{G}(x)}$ is rewritten with $\text{proj}_1(\mathcal{DT}(\mathcal{R}_{\text{rw}}))$ to $\text{com}_2(\mathbf{G}(\mathbf{g}(x)), \mathbf{G}(x))$ or com_0 , both with probability $1/2$. At the same time, due to (b)(ii), the (twice underlined) subterm $\underline{\underline{\mathbf{g}(x)}}$ of the first argument $s_{i'} = s_1 = \underline{\underline{\mathbf{G}(\underline{\mathbf{g}(x)})}}$ is rewritten using the original rule $\mathbf{g}(x) \rightarrow \{1/2 : \mathbf{g}^2(x), 1/2 : x\}$ to $\mathbf{g}^2(x)$ or x , both with probability $1/2$. So when rewriting $s_i = s_2 = \underline{\mathbf{G}(x)}$ one can also perform the corresponding rewrite step on its flattened copy inside $s_{i'} = s_1 = \underline{\underline{\mathbf{G}(\underline{\mathbf{g}(x)})}}$. But the ambiguity in Def. 25 also allows the step $\text{com}_2(\mathbf{G}(\mathbf{g}(x)), \underline{\mathbf{G}(x)}) \xrightarrow{i} \mathcal{DT}(\mathcal{R}_{\text{rw}}), \mathcal{R}_{\text{rw}} \{1/2 : \text{com}_3(\mathbf{G}(\mathbf{g}(x)), \mathbf{G}(\mathbf{g}(x)), \mathbf{G}(x)), 1/2 : \text{com}_1(\mathbf{G}(\mathbf{g}(x)))\}$ that does not simulate any original rewrite step with \mathcal{R}_{rw} . Therefore, the approach of [28] is not complete in the probabilistic setting.

In [28], there is also an analogous rewrite relation for PTRSs, where one can apply the same rule simultaneously to the same subterms in a single rewrite step.

Definition 26 ($\xrightarrow{i} \mathcal{R}$). For a normalized term $\text{com}_n(s_1, \dots, s_n)$ and a PTRS \mathcal{R} , let $\text{com}_n(s_1, \dots, s_n) \xrightarrow{i} \mathcal{R} \{p_1 : b_1, \dots, p_k : b_k\}$ if there are $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{R}$, a position π , a substitution σ , and an $1 \leq i \leq n$ with $s_i|_\pi = \ell \sigma \in \text{ANF}_{\mathcal{R}}$, and for all $1 \leq j \leq k$ we have $b_j = \text{com}_n(t_1^j, \dots, t_n^j)$ where

(a) $t_i^j = s_i[r_j \sigma]_\pi$ for all $1 \leq j \leq k$, i.e., we rewrite the term s_i using \mathcal{R} .

(b) For every $1 \leq i' \leq n$ with $i' \neq i$ we have

(i) $t_{i'}^j = s_{i'}$ for all $1 \leq j \leq k$ or

(ii) $t_{i'}^j = s_{i'}[r_j \sigma]_\tau$ for all $1 \leq j \leq k$, if $s_{i'}|_\tau = \ell \sigma$ for some position τ .

The DT framework of [28] works on (probabilistic) DT problems $(\mathcal{P}, \mathcal{R})$, where \mathcal{P} is a PPTRS and \mathcal{R} is a PTRS. To analyze a PTRS \mathcal{R} , one starts with the DT problem $(\mathcal{DT}(\mathcal{R}), \mathcal{R})$. For the chain criterion for DTs, one uses a notion of chain trees that is defined as follows. $\mathfrak{T} = (V, E, L, P)$ is a $(\mathcal{P}, \mathcal{R})$ -chain tree (CT) if

1. (V, E) is a (possibly infinite) directed tree with nodes $V \neq \emptyset$ and directed edges $E \subseteq V \times V$ where $vE = \{w \mid (v, w) \in E\}$ is finite for every $v \in V$.

2. $L : V \rightarrow (0, 1] \times \mathcal{T}(\Sigma^\#, \mathcal{V})$ labels every node v by a probability p_v and a term t_v . For the root $v \in V$ of the tree, we have $p_v = 1$.
3. $P \subseteq V \setminus \text{Leaf}$ (where Leaf are all leaves) is a subset of the inner nodes to indicate whether we use \mathcal{P} or \mathcal{R} for the next rewrite step. $R = V \setminus (\text{Leaf} \cup P)$ are all inner nodes that are not in P . Thus, $V = P \uplus R \uplus \text{Leaf}$.
4. For all $v \in P$: If $vE = \{w_1, \dots, w_k\}$, then $t_v \xrightarrow{i} \mathcal{P}, \mathcal{R} \left\{ \frac{p_{w_1}}{p_v} : t_{w_1}, \dots, \frac{p_{w_k}}{p_v} : t_{w_k} \right\}$.
5. For all $v \in R$: If $vE = \{w_1, \dots, w_k\}$, then $t_v \xrightarrow{i} \mathcal{R} \left\{ \frac{p_{w_1}}{p_v} : t_{w_1}, \dots, \frac{p_{w_k}}{p_v} : t_{w_k} \right\}$.
6. Every infinite path in \mathfrak{T} contains infinitely many nodes from P .

Again, one defines $|\mathfrak{T}|_{\text{Leaf}} = \sum_{v \in \text{Leaf}} p_v$ and says that $(\mathcal{P}, \mathcal{R})$ is iAST if $|\mathfrak{T}|_{\text{Leaf}} = 1$ for all $(\mathcal{P}, \mathcal{R})$ -CTs \mathfrak{T} . This yields a chain criterion which is only *incomplete*, i.e., here we only have “if” instead of “iff”.

Theorem 27 (Chain Crit. of [28]). *A PTRS \mathcal{R} is iAST if $(\mathcal{DT}(\mathcal{R}), \mathcal{R})$ is iAST.*

Lemma 28 (Incompleteness of Chain Criterion for $\mathcal{R}_{\text{incpl}}$). *The DT Problem $(\mathcal{DT}(\mathcal{R}_{\text{incpl}}), \mathcal{R}_{\text{incpl}})$ is not iAST.*

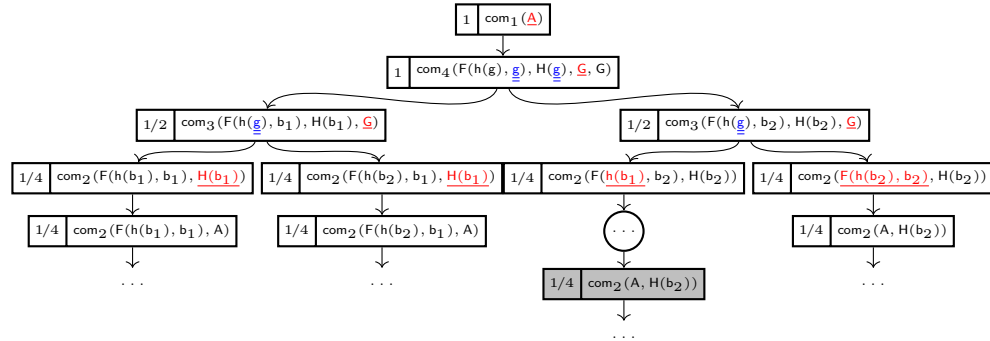
Proof. Consider $\mathcal{R}_{\text{incpl}}$ from Ex. 5 with the rules

$$\begin{array}{ll} a \rightarrow \{1 : f(h(g), g)\} & h(b_1) \rightarrow \{1 : a\} \\ g \rightarrow \{1/2 : b_1, 1/2 : b_2\} & f(x, b_2) \rightarrow \{1 : a\} \end{array}$$

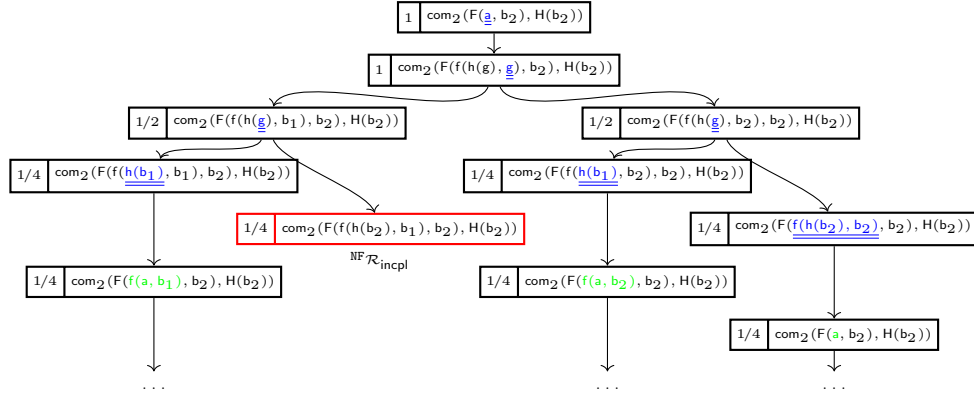
The set $\mathcal{DT}(\mathcal{R}_{\text{incpl}})$ consists of the dependency tuples

$$\begin{array}{l} \langle A, a \rangle \rightarrow \{1 : \langle \text{com}_4(F(h(g), g), H(g), G, G), f(h(g), g)) \rangle\} \\ \langle G, g \rangle \rightarrow \{1/2 : \langle \text{com}_0, b_1 \rangle, 1/2 : \langle \text{com}_0, b_2 \rangle\} \\ \langle H(b_1), h(b_1) \rangle \rightarrow \{1 : \langle \text{com}_1(A), a \rangle\} \\ \langle F(x, b_2), f(x, b_2) \rangle \rightarrow \{1 : \langle \text{com}_1(A), a \rangle\} \end{array}$$

Now we obtain the following $(\mathcal{DT}(\mathcal{R}_{\text{incpl}}), \mathcal{R}_{\text{incpl}})$ -chain tree \mathfrak{T} with $|\mathfrak{T}|_{\text{Leaf}} = 0$ that uses both $\xrightarrow{i} \mathcal{DT}(\mathcal{R}_{\text{incpl}}), \mathcal{R}_{\text{incpl}}$ and $\xrightarrow{i} \mathcal{R}_{\text{incpl}}$ for the edge relation:



The part marked with a circle has the following form:



This tree has the form of the RST from Ex. 5 with the additional context $\text{com}_2(F(\square, b_2), H(b_2))$ around it. Again, the part inside this context will reach a normal form with probability 1 and afterwards, e.g., at the node labeled with $\text{com}_2(F(f(h(b_2), b_1), b_2), H(b_2))$, we can rewrite $F(\dots)$ with the DT to A , which yields $\text{com}_2(A, H(b_2))$ in the gray node. In this way, one obtains infinitely many nodes labeled with $\text{com}_2(A, H(b_2))$ whose probabilities add up to $1/4$ (in the tree we depicted this by a gray node instead).

Note that this tree satisfies the conditions (1)-(5) of a CT, but it does not satisfy condition (6) as the part corresponding to the circle contains infinite paths without P nodes. We have to additionally cut these infinite paths to create a valid CT \mathfrak{T} and show that we still have $|\mathfrak{T}|_{\text{Leaf}} < 1$. Since these infinite subtrees of the parts that do not use any DTs are all iAST, we can use the same idea as in the proof of the P-Partition Lemma of [29, Lemma 50] (see also Lemma 33). The variant of the P-Partition Lemma that we apply here is called *Cutting Lemma* (Lemma 30) below. Then we obtain the desired $(DT(\mathcal{R}_{\text{incpl}}, \mathcal{R}_{\text{incpl}})$ -CT \mathfrak{T} with $|\mathfrak{T}|_{\text{Leaf}} < 1$.

The problem with this CT is that when applying a DT to rewrite the underlined \underline{G} in the child of the root, then one should not rewrite both flattened copies \underline{g} in $\text{com}_4(F(h(\underline{g}), \underline{g}), H(\underline{g}), \underline{G}, G)$. The reason is that \underline{G} either corresponds to the two flattened copies \underline{g} in the arguments of h and H or to the flattened copy \underline{g} in the second argument of F . This “wrong” rewriting yields terms like $\text{com}_2(F(h(b_2), b_1), H(\underline{b}_1))$ which do not correspond to any terms in the original RST since the arguments of h and H are different. Indeed, while the corresponding term $f(h(b_2), b_1)$ in the RST is a normal form, $\text{com}_2(F(h(b_2), b_1), H(\underline{b}_1))$ contains the subterm $H(\underline{b}_1)$ which is not a normal form. \square

We first recapitulate the notion of a *sub-chain tree* from [29].

Definition 29 (Subtree, Sub-CT). Let $(\mathcal{P}, \mathcal{R})$ be a DT problem and let $\mathfrak{T} = (V, E, L, P)$ be a tree that satisfies Conditions (1)-(5) of a $(\mathcal{P}, \mathcal{R})$ -CT. Let $W \subseteq V$ be non-empty, weakly connected, and for all $x \in W$ we have $xE \cap W = \emptyset$ or $xE \cap W = xE$. Then, we define the subtree (or sub-CT if it satisfies Condition (6) as well) $\mathfrak{T}[W]$ by $\mathfrak{T}[W] = (W, E \cap (W \times W), L^W, P \cap (W \setminus W_{\text{Leaf}}))$. Here, W_{Leaf} denotes the leaves of the tree $G^{\mathfrak{T}[W]} = (W, E \cap (W \times W))$ so that the new set

$P \cap (W \setminus W_{\text{Leaf}})$ 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.

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 $xE \cap W = \emptyset$ 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.

We say that a CT (or RST) \mathfrak{T} converges (or terminates) with probability $p \in \mathbb{R}$ if we have $|\mathfrak{T}|_{\text{Leaf}} = p$. Now we can prove the cutting lemma that is needed for the proof of [Lemma 28](#).

Lemma 30 (Cutting Lemma). *Let $(\mathcal{P}, \mathcal{R})$ be a DT problem, let $\mathfrak{T} = (V, E, L, P)$ be a tree that only satisfies Conditions (1)-(5) of a $(\mathcal{P}, \mathcal{R})$ -CT, and let \mathfrak{T} converge with probability < 1 . Assume that every subtree that only contains nodes from R converges with probability 1. Then there exists a subtree \mathfrak{T}' that converges with probability < 1 such that every infinite path has an infinite number of nodes from P , i.e., \mathfrak{T}' is a valid $(\mathcal{P}, \mathcal{R})$ -CT as it now also satisfies Condition (6).*

Proof. Let $\mathfrak{T} = (V, E, L, P)$ be a tree that only satisfies Conditions (1)-(5) of a $(\mathcal{P}, \mathcal{R})$ -CT with $|\mathfrak{T}|_{\text{Leaf}} = c < 1$ for some $c \in \mathbb{R}$. Since we have $0 \leq c < 1$, there is an $\varepsilon > 0$ such that $c + \varepsilon < 1$. Remember that the formula for the geometric series is:

$$\sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n = \frac{1}{d-1}, \text{ for all } d \in \mathbb{R} \text{ such that } \frac{1}{|d|} < 1$$

Let $d = \frac{1}{\varepsilon} + 2$. Now, we have $\frac{1}{d} = \frac{1}{\frac{1}{\varepsilon} + 2} < 1$ and:

$$\frac{1}{\varepsilon} + 1 < \frac{1}{\varepsilon} + 2 \Leftrightarrow \frac{1}{\varepsilon} + 1 < d \Leftrightarrow \frac{1}{\varepsilon} < d - 1 \Leftrightarrow \frac{1}{d-1} < \varepsilon \Leftrightarrow \sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n < \varepsilon \quad (17)$$

We will now construct a subtree $\mathfrak{T}' = (V', E', L', P')$ such that every infinite path has an infinite number of P nodes and such that

$$|\mathfrak{T}'|_{\text{Leaf}} \leq |\mathfrak{T}|_{\text{Leaf}} + \sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n \quad (18)$$

and then, we finally have

$$|\mathfrak{T}'|_{\text{Leaf}} \stackrel{(18)}{\leq} |\mathfrak{T}|_{\text{Leaf}} + \sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n = c + \sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n \stackrel{(17)}{<} c + \varepsilon < 1$$

The idea of this construction is that we cut infinite subtrees of pure R nodes as soon as the probability for normal forms is high enough. In this way, one obtains paths where after finitely many R nodes, there is a P node, or we reach a leaf.

The construction works as follows. For any node $x \in V$, let $\mathcal{L}(x)$ be the number of P nodes in the path from the root to x . Furthermore, for any set $W \subseteq V$ and $k \in \mathbb{N}$, let $\mathfrak{L}(W, k) = \{x \in W \mid \mathcal{L}(x) \leq k \vee (x \in P \wedge \mathcal{L}(x) \leq k + 1)\}$ be the set of all nodes in W that have at most k nodes from P in the path from the root to its predecessor. So if $x \in \mathfrak{L}(W, k)$ is not in P , then we have at most k nodes from P

For the induction step, assume that we have already defined a subset $U_i \subseteq \mathfrak{L}(V, i)$. Let $H_i = \{x \in U_i \mid x \in P, \mathcal{L}(x) = i + 1\}$ be the set of leaves in $\mathfrak{T}[U_i]$ that are in P . For each $x \in H_i$, we consider the subtree that starts at x until we reach the next node from P , including the node itself. Everything below such a node will be cut. To be precise, we regard the tree $\mathfrak{T}_x = (V_x, E_x, L_x, P_x) = \mathfrak{T}[\mathfrak{L}(xE^*, i + 1)]$. Here, xE^* is the set of all nodes that are reachable from x by arbitrary many steps.

First, we show that $|\mathfrak{T}_x|_{\text{Leaf}} = 1$. For every direct successor y of x , the subtree $\mathfrak{T}_y = \mathfrak{T}_x[yE_x^*]$ of \mathfrak{T}_x that starts at y does not contain any nodes from P . Hence, we have $|\mathfrak{T}_y|_{\text{Leaf}} = 1$ by the prerequisite of the lemma, and hence

$$|\mathfrak{T}_x|_{\text{Leaf}} = \sum_{y \in xE} p_y \cdot |\mathfrak{T}_y|_{\text{Leaf}} = \sum_{y \in xE} p_y \cdot 1 = \sum_{y \in xE} p_y = 1.$$

For the construction of U_{i+1} , we have the same cases as before, see Fig. 1. Either \mathfrak{T}_x is finite or \mathfrak{T}_x is infinite. Let Z_x be the set of nodes that we want to add to our node set U_{i+1} from the tree \mathfrak{T}_x . In the first case we can add all the nodes again and set $Z_x = V_x$. In the second case, we once again cut the tree at a specific depth once the probability for leaves is high enough. Let $d_x(z)$ be the depth of the node z in the tree \mathfrak{T}_x . Moreover, let $D_x(k) = \{x \in V_x \mid d_x(z) \leq k\}$ be the set of nodes in \mathfrak{T}_x that have a depth of at most k . Since $|\mathfrak{T}_x|_{\text{Leaf}} = 1$ and $|\cdot|_{\text{Leaf}}$ is monotonic w.r.t. the depth of the tree \mathfrak{T}_x , we can find an $N_x \in \mathbb{N}$ such that

$$\sum_{y \in \text{Leaf}^{\mathfrak{T}_x}, d_x(y) \leq N_x} p_y^{\mathfrak{T}_x} \geq 1 - \left(\frac{1}{d}\right)^{i+1} \cdot \frac{1}{|H_i|}$$

We include all nodes from $D_x(N_x)$ in U_{i+1} and delete every other node of \mathfrak{T}_x . In other words, we cut the tree after depth N_x . We now know that this cut may increase the probability of leaves by at most $\left(\frac{1}{d}\right)^{i+1} \cdot \frac{1}{|H_i|}$. Therefore, we set $Z_x = D_x(N_x)$.

We do this for each $x \in H_i$ and in the end, we set $U_{i+1} = U_i \cup \bigcup_{x \in H_i} Z_x$.

It is straightforward to see that $\bigcup_{k \in \mathbb{N}} U_k$ satisfies the conditions of Def. 29, as we only cut after certain nodes in our construction. Hence, $\bigcup_{k \in \mathbb{N}} U_k$ is non-empty and weakly connected, and for each of its nodes, it either contains no or all successors. Furthermore, $\mathfrak{T}' = \mathfrak{T}[\bigcup_{k \in \mathbb{N}} U_k]$ is a sub-chain tree which does not contain an infinite path of pure R nodes as we cut every such path after a finite depth.

It remains to prove that $|\mathfrak{T}'|_{\text{Leaf}} \leq |\mathfrak{T}|_{\text{Leaf}} + \sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n$ holds. During the i -th iteration of the construction, we may increase the value of $|\mathfrak{T}|_{\text{Leaf}}$ by the sum of all probabilities corresponding to the new leaves resulting from the cuts. As we cut at most $|H_i|$ trees in the i -th iteration and for each such tree, we added at most a total probability of $\left(\frac{1}{d}\right)^{i+1} \cdot \frac{1}{|H_i|}$ for the new leaves, the value of $|\mathfrak{T}|_{\text{Leaf}}$ might increase by

$$|H_i| \cdot \left(\frac{1}{d}\right)^{i+1} \cdot \frac{1}{|H_i|} = \left(\frac{1}{d}\right)^{i+1}$$

in the i -th iteration, and hence in total, we then get

$$|\mathfrak{T}'|_{\text{Leaf}} \leq |\mathfrak{T}|_{\text{Leaf}} + \sum_{n=1}^{\infty} \left(\frac{1}{d}\right)^n,$$

as desired (see (18)). \square

Next, we want to prove our new chain criterion for ADPs (Thm. 12), which is both sound and complete. Due to our new definition of ADPs, this proof is easier than for the probabilistic chain criterion that is only sound from [28] (Thm. 27). We start by defining an important set of positions that we will use throughout the proof.

Definition 31 (pos_{Poss}). *Let \mathcal{R} be a PTRS. For a term $t \in \mathcal{T}(\Sigma, \mathcal{V})$ we define $\text{pos}_{\text{Poss}}(t, \mathcal{R}) = \{\pi \mid \pi \in \text{pos}_{\mathcal{D}}(t), t|_{\pi} \notin \text{NF}_{\mathcal{R}}\}$. Here, $\text{NF}_{\mathcal{R}}$ again denotes the set of all normal forms w.r.t. \mathcal{R} .*

So $\text{pos}_{\text{Poss}}(t, \mathcal{R})$ contains all positions of subterms of t that may be used as a redex now or in future rewrite steps, because the subterm has a defined root symbol and is not in $\text{NF}_{\mathcal{R}}$.

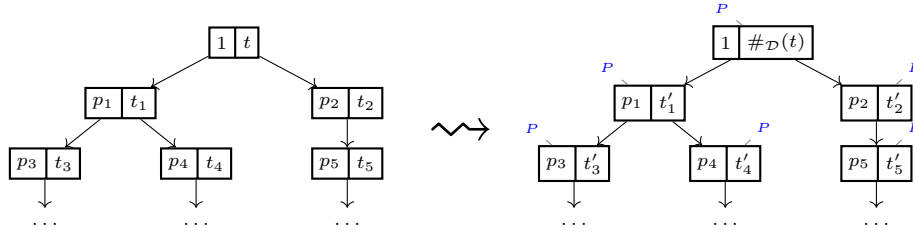
Example 32. Consider the following PTRS \mathcal{R} over a signature with $\mathcal{D} = \{f, g\}$ and $\mathcal{C} = \{a, s\}$ with the rules $f(a, a) \rightarrow \{1 : s(f(g, g))\}$ and $g \rightarrow \{1 : a\}$. For the term $t = s(f(g, g))$ we have $\text{pos}_{\text{Poss}}(t, \mathcal{R}) = \{1, 1.1, 1.2\}$.

Finally, for two (possibly annotated) terms s, t we define $s \doteq t$ if $b(s) = b(t)$.

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

Proof. In the following, we will often implicitly use that for an annotated term $t \in \mathcal{T}(\Sigma^{\#}, \mathcal{V})$, we have $b(t) \in \text{ANF}_{\mathcal{R}}$ iff $t \in \text{ANF}_{\mathcal{DP}(\mathcal{R})}$ since a rewrite rule and its corresponding canonical annotated dependency pair have the same left-hand side.

Soundness: Assume that \mathcal{R} is not iAST. Then, there exists an innermost \mathcal{R} -RST $\mathfrak{T} = (V, E, L)$ whose root is labeled with $(1 : t)$ for some term $t \in \mathcal{T}(\Sigma, \mathcal{V})$ that converges with probability < 1 . We will construct a $\mathcal{DP}(\mathcal{R})$ -CT $\mathfrak{T}' = (V, E, L', V \setminus \text{Leaf}^{\mathfrak{T}'})$ 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 P . Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}|_{\text{Leaf}} = |\mathfrak{T}'|_{\text{Leaf}}$. 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 $|\mathfrak{T}|_{\text{Leaf}} < 1$, we thus have $|\mathfrak{T}'|_{\text{Leaf}} < 1$. Hence, there exists a $\mathcal{DP}(\mathcal{R})$ -CT \mathfrak{T}' that converges with probability < 1 and $\mathcal{DP}(\mathcal{R})$ is not iAST either.



We construct the new labeling L' for the $\mathcal{DP}(\mathcal{R})$ -CT inductively such that for all inner nodes $x \in V \setminus \text{Leaf}$ with children nodes $xE = \{y_1, \dots, y_k\}$ we have $t'_x \xrightarrow{\mathcal{DP}(\mathcal{R})} \left\{ \frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t'_{y_k} \right\}$. Let $X \subseteq V$ be the set of nodes x where we

have already defined the labeling $L'(x)$. During our construction, we ensure that the following property holds:

$$\text{For every node } x \in X \text{ we have } t_x \doteq t'_x \text{ and } \text{pos}_{\text{Poss}}(t_x, \mathcal{R}) \subseteq \text{pos}_{\mathcal{D}\#}(t'_x). \quad (19)$$

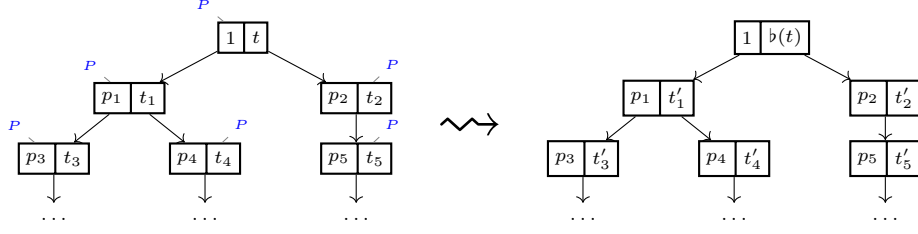
This means that the corresponding term t_x for the node x in \mathfrak{T} has the same structure as the term t'_x in \mathfrak{T}' , and additionally, all the possible redexes in t_x are annotated in t'_x . The annotations ensure that we rewrite with Case (PR) of Def. 11 so that the node x is contained in P . We label the root of \mathfrak{T}' with $\#_{\mathcal{D}}(t)$. Here, we have $t \doteq \#_{\mathcal{D}}(t)$ and $\text{pos}_{\text{Poss}}(t, \mathcal{R}) \subseteq \text{pos}_{\mathcal{D}}(t) = \text{pos}_{\mathcal{D}\#}(\#_{\mathcal{D}}(t))$. 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 $xE = \{y_1, \dots, y_k\}$ be the set of its successors. We need to define the corresponding terms $t'_{y_1}, \dots, t'_{y_k}$ for the nodes y_1, \dots, y_k . Since x is not a leaf, we have $t_x \xrightarrow{i} \mathcal{R} \left\{ \frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t_{y_k} \right\}$. This means that there is a rule $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\} \in \mathcal{R}$, a position π , and a substitution σ such that $t_x|_{\pi} = \ell\sigma \in \text{ANF}_{\mathcal{R}}$. Furthermore, we have $t_{y_j} = t_x[r_j\sigma]_{\pi}$ for all $1 \leq j \leq k$. So the labeling of the successor y_j in \mathfrak{T} is $L(y_j) = (p_x \cdot p_j : t_x[r_j\sigma]_{\pi})$ for all $1 \leq j \leq k$.

The corresponding ADP for the rule is $\ell \rightarrow \{p_1 : \#_{\mathcal{D}}(r_1), \dots, p_k : \#_{\mathcal{D}}(r_k)\}^{\text{true}}$. Furthermore, $\pi \in \text{pos}_{\text{Poss}}(t_x, \mathcal{R}) \subseteq_{(IH)} \text{pos}_{\mathcal{D}\#}(t'_x)$ and $t_x \doteq_{(IH)} t'_x$. Hence, we can rewrite t'_x with $\ell \rightarrow \{p_1 : \#_{\mathcal{D}}(r_1), \dots, p_k : \#_{\mathcal{D}}(r_k)\}^{\text{true}}$, using the position π and the substitution σ , and Case (PR) of Def. 11 applies. We get $t'_x \xrightarrow{i} \mathcal{DP}(\mathcal{R}) \{p_1 : t'_{y_1}, \dots, p_k : t'_{y_k}\}$ with $t'_{y_j} = t'_x[\#_{\mathcal{D}}(r_j)\sigma]_{\pi}$ by (PR). This means that we have $t_{y_j} \doteq t'_{y_j}$. It remains to prove $\text{pos}_{\text{Poss}}(t_{y_j}, \mathcal{R}) \subseteq \text{pos}_{\mathcal{D}\#}(t'_{y_j})$ for all $1 \leq j \leq k$. For all positions $\tau \in \text{pos}_{\text{Poss}}(t_{y_j}, \mathcal{R}) = \text{pos}_{\text{Poss}}(t_x[r_j\sigma]_{\pi}, \mathcal{R})$ that are orthogonal or above π , we have $\tau \in \text{pos}_{\text{Poss}}(t_x, \mathcal{R}) \subseteq_{(IH)} \text{pos}_{\mathcal{D}\#}(t'_x)$, and all annotations orthogonal or above π remain in t'_{y_j} as they were in t'_x . For all positions $\tau \in \text{pos}_{\text{Poss}}(t_{y_j}, \mathcal{R}) = \text{pos}_{\text{Poss}}(t_x[r_j\sigma]_{\pi}, \mathcal{R})$ that are below π , we know that, due to innermost evaluation, at least the defined root symbol of a term that is not in normal form must be inside r_j , and thus $\tau \in \text{pos}_{\mathcal{D}\#}(t'_{y_j})$, as all defined symbols of r_j are annotated in $t'_{y_j} = t'_x[\#_{\mathcal{D}}(r_j)\sigma]_{\pi}$. This ends the induction proof for this direction.

Completeness: Assume that $\mathcal{DP}(\mathcal{R})$ is not iAST. Then, there exists a $\mathcal{DP}(\mathcal{R})$ -CT $\mathfrak{T} = (V, E, L, P)$ whose root is labeled with $(1 : t)$ for some annotated term $t \in \mathcal{T}(\Sigma^{\#}, \mathcal{V})$ 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 $|\mathfrak{T}'|_{\text{Leaf}} = |\mathfrak{T}|_{\text{Leaf}} < 1$. Therefore, there exists an \mathcal{R} -RST \mathfrak{T}' that converges with probability < 1 . Hence, \mathcal{R} is not iAST either.

We construct the new labeling L' for the \mathcal{R} -RST inductively such that for all inner nodes $x \in V \setminus \text{Leaf}$ with children nodes $xE = \{y_1, \dots, y_k\}$ we have $t'_x \xrightarrow{i} \mathcal{R} \left\{ \frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t'_{y_k} \right\}$. Let $X \subseteq V$ be the set of nodes x where we have already defined the labeling $L'(x)$. During our construction, we ensure the following property:

$$\text{For every node } x \in X \text{ we have } t_x \doteq t'_x \text{ and } \text{pos}_{\mathcal{D}\#}(t'_x) = \emptyset. \quad (20)$$



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, it contains no annotations. We label the root of \mathfrak{T}' with $\mathfrak{b}(t)$. Here, we have $t \doteq \mathfrak{b}(t)$ and $\text{pos}_{\mathcal{D}\#}(\mathfrak{b}(t)) = \emptyset$. 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 $xE = \{y_1, \dots, y_k\}$ be the set of its successors. Since x is not a leaf, we have $t_x \xrightarrow{i}_{\mathcal{DP}(\mathcal{R})} \{ \frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t_{y_k} \}$. This means that there is an ADP $\ell \rightarrow \{p_1 : \#_{\mathcal{D}}(r_1), \dots, p_k : \#_{\mathcal{D}}(r_k)\}^{\text{true}} \in \mathcal{DP}(\mathcal{R})$, a position π , and a substitution σ such that $\mathfrak{b}(t_x|_{\pi}) = \ell\sigma \in \text{ANF}_{\mathcal{R}}$. Furthermore, we have $t_{y_j} = t_x[r_j\sigma]_{\pi}$ or $t_{y_j} = t_x[\#_{\mathcal{D}}(r_j)\sigma]_{\pi}$ for all $1 \leq j \leq k$.

The original rule for the ADP is $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}$. Furthermore, we have $t_x \doteq_{(IH)} t'_x$. Hence, we can rewrite t'_x with $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}$, using the position π and the substitution σ , since $t'_x|_{\pi} = \ell\sigma \in \text{ANF}_{\mathcal{R}}$ (as $\text{pos}_{\mathcal{D}\#}(t'_x) = \emptyset$). We get $t'_x \xrightarrow{i}_{\mathcal{R}} \{p_1 : t'_{y_1}, \dots, p_k : t'_{y_k}\}$ with $t_{y_j} = t'_x[r_j\sigma]_{\pi}$. This means that we have $t_{y_j} \doteq_{(IH)} t'_{y_j}$ and $\text{pos}_{\mathcal{D}\#}(t'_{y_j}) = \emptyset$ for all $1 \leq j \leq k$, which ends the induction proof for this direction. \square

Next, we prove the theorems regarding the processors that we adapted from [28] to our new ADP framework. We will see that the proofs become way more readable compared to [28] (even though they are still more complicated than in the non-probabilistic setting). First, we repeat two lemmas from [28] and prove the theorems on the processors afterwards. We start with the *P-partition lemma*. This lemma was proven in [28] and still applies to our new ADP problems, since the structure of CTs are the same as in [28].

Lemma 33 (P-Partition Lemma, [28]). *Let \mathcal{P} be an ADP problem and let $\mathfrak{T} = (V, E, L, P)$ be a \mathcal{P} -CT that converges with probability < 1 . Assume that we can partition $P = P_1 \uplus P_2$ such that every sub-CT that only contains P -nodes from P_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 P_2 .*

Proof. Analogous to the proof for Lemma 30. \square

Next, we recapitulate the *starting lemma*. W.l.o.g., we will often assume that we label the root of our CT with $(1 : t)$ for an annotated term t such that $\mathfrak{b}(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \mathcal{P}$, and $\text{pos}_{\mathcal{D}\#}(t) = \{\varepsilon\}$.

Lemma 34 (Starting Lemma, [28]). *If an ADP problem \mathcal{P} is not iAST, then there exists a \mathcal{P} -CT \mathfrak{T} with $|\mathfrak{T}|_{\text{Leaf}} < 1$ that starts with $(1 : t)$ with $\mathfrak{b}(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \mathcal{P}$, and $\text{pos}_{\mathcal{D}\#}(t) = \{\varepsilon\}$.*

Proof. We prove the contraposition. Assume that every \mathcal{P} -CT \mathfrak{T} converges with probability 1 if it starts with $(1 : t)$ and $\flat(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \mathcal{P}$, and $\text{pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$. 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 iAST. 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 P) 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 a P -step above the annotated position. If $t|_\pi$ is in normal form, then the claim is again trivial. If we have $t|_\pi = s\sigma$ for some ADP $s \rightarrow \dots \in \mathcal{P}$ and some substitution σ such that $t|_\pi = s\theta \in \text{ANF}_{\mathcal{P}}$ and $\text{pos}_{\mathcal{D}^\#}(t|_\pi) = \{\varepsilon\}$, then we know by our assumption that such a CT converges with probability 1. Otherwise, we can rewrite below π . However, in this case, we have to eventually rewrite at position π in every infinite path of the tree (as this is the only annotation in the beginning), hence again, such a CT converges with probability 1 by assumption.

Now we regard the induction step, and assume 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 $\Phi_1 = \{\tau\}$ and $\Phi_2 = \{\chi \in \text{pos}_{\mathcal{D}^\#}(t) \mid \chi \neq \tau\}$ for some $\tau \in \text{pos}_{\mathcal{D}^\#}(t)$ and consider the two terms $\#_{\Phi_1}(t)$ and $\#_{\Phi_2}(t)$, which contain both strictly less than n annotations. By our induction hypothesis, we know that every \mathcal{P} -CT that starts with $(1 : \#_{\Phi_1}(t))$ or $(1 : \#_{\Phi_2}(t))$ converges with probability 1. Let $\mathfrak{T}_1 = (V, E, L_1, P_1)$ be the tree that starts with $(1 : \#_{\Phi_1}(t))$ and uses the same rules as we did in \mathfrak{T} . We can partition P into the sets P_1 and $P_2 = P \setminus P_1$. Note that every sub-CT of \mathfrak{T} such that every infinite path has an infinite number of P_1 -nodes is a \mathcal{P} -CT again. In order to use the P-Partition Lemma (Lemma 33) for the tree \mathfrak{T} , we have to show that every sub-CT \mathfrak{T}'_1 of \mathfrak{T} that only contains P -nodes from P_1 converges with probability 1. Let $\mathfrak{T}'_1 = (V', E', L', P')$ be a sub-CT of \mathfrak{T} that does not contain nodes from P_2 . There exists a set W satisfying the conditions of Def. 29 such that $\mathfrak{T}'_1 = \mathfrak{T}[W]$. Since \mathfrak{T} and \mathfrak{T}_1 have the same tree structure, $\mathfrak{T}_1[W]$ is a sub-CT of \mathfrak{T}_1 . Moreover, $\mathfrak{T}_1[W]$ is a \mathcal{P} -CT, since the set W does not contain any inner nodes from P_2 . Finally, since $\mathfrak{T}_1[W]$ is a sub-CT of a \mathcal{P} -CT that converges with probability 1, we know that $\mathfrak{T}_1[W]$ must be converging with probability 1 as well.

Now, we have shown that the conditions for the P-Partition Lemma (Lemma 33) are satisfied. We can now apply the P-Partition Lemma to get a grounded sub-CT \mathfrak{T}' of \mathfrak{T} with $|\mathfrak{T}'|_{\text{Leaf}} < 1$ such that on every infinite path, we have an infinite number of P_2 nodes. Let \mathfrak{T}_2 be the tree that starts with $\#_{\Phi_2}(t)$ 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 P_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 $\#_{\Phi_2}(t)$ and with $|\mathfrak{T}_2|_{\text{Leaf}} < 1$. This is our desired contradiction, which proves the induction step. \square

Now we show soundness and completeness for all processors.

Theorem 14 (Probabilistic Dependency Graph Processor). *For the SCCs $\mathcal{P}_1, \dots, \mathcal{P}_n$ of the \mathcal{P} -dependency graph, $\text{Proc}_{\text{DG}}(\mathcal{P}) = \{\mathcal{P}_1 \cup \text{b}(\mathcal{P} \setminus \mathcal{P}_1), \dots, \mathcal{P}_n \cup \text{b}(\mathcal{P} \setminus \mathcal{P}_n)\}$ is sound and complete.*

Proof. Let $\overline{X} = X \cup \text{b}(\mathcal{P} \setminus X)$ 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 iAST, 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 iAST, then \mathcal{P} is not iAST 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 $W = \{\mathcal{P}_1, \dots, \mathcal{P}_n\} \cup \{\{v\} \subseteq \mathcal{P} \mid v \text{ is not in an SCC of } \mathfrak{G}\}$ 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 iAST.
2. We show that composing SCCs maintains the iAST property.
3. We show that for every $X \in W$, the ADP problem $\bigcup_{X >_{\mathfrak{G}}^* Y} Y$ is iAST by induction on $>_{\mathfrak{G}}$.
4. We conclude that \mathcal{P} must be iAST.

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 iAST.

We start by proving the following:

$$\text{Every ADP problem } \overline{X} \text{ with } X \in W \text{ is iAST.} \quad (21)$$

To prove (21), note that if X is an SCC, then it follows from our assumption that \overline{X} is iAST. 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 iAST. By Lemma 34 there exists an \overline{X} -CT $\mathfrak{T} = (V, E, L, P)$ that converges with probability < 1 and starts with $(1 : t)$ where $\text{b}(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and some ADP $s \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in \overline{X}$, and $\text{pos}_{\mathcal{D}^{\#}}(t) = \{\varepsilon\}$. If $s \rightarrow \dots \notin X$, 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 $s \rightarrow \dots \in X$. Assume for a contradiction that there exists a node $x \in P$ 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 from P . This means that for the corresponding term t_x for node x there is a $t' \sqsubseteq_{\#} t_x$ such that $t' = s\sigma' \in \text{ANF}_{\mathcal{P}}$ for some substitution σ' and the only ADP $s \rightarrow \dots \in X$ (which is the only ADP that contains any annotations in the right-hand side). Let (z_0, \dots, 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 $s\theta \xrightarrow{i}_{\overline{X}} \{p_1 : r_1\theta, \dots, p_k : r_k\theta\}$. After

that, we only use steps at non-annotated positions in the path since all the nodes z_1, \dots, z_{m-1} are contained in R . Therefore, we must have an $1 \leq j \leq k$ and a $t'' \trianglelefteq_{\#} r_j$ such that $t''\#\theta \xrightarrow{i}_{\text{np}(P)}^* s\#\sigma'$, 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 \bar{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 P , 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 iAST property.

Next, we show that composing SCCs maintains the iAST 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 (22)
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 iAST, then $\overline{\bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y}$ is iAST.

To show (22), we assume that both $\overline{\bigcup_{X \in \hat{X}} X}$ and $\overline{\bigcup_{Y \in \hat{Y}} Y}$ are iAST. Let $\bar{Z} = \overline{\bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y}$. The property in (22) 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 \bar{Z} is not iAST. By Lemma 34 there exists a \bar{Z} -CT $\mathfrak{T} = (V, E, L, P)$ that converges with probability < 1 and starts with $(1 : t)$ where $\mathfrak{b}(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \bar{Z}$, and $\text{pos}_{\mathcal{D}\#}(t) = \{\varepsilon\}$.

If $s \rightarrow \dots \notin \bigcup_{X \in \hat{X}} X \cup \bigcup_{Y \in \hat{Y}} Y$, 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 P of our \bar{Z} -CT \mathfrak{T} into the sets

- $P_1 := \{x \in P \mid x \text{ together with the labeling and its successors represents a step with an ADP from } \bigcup_{X \in \hat{X}} X\}$
- $P_2 := P \setminus P_1$

Note that in the case of $x \in P_2$, 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 iAST. 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 $|\mathfrak{T}|_{\text{Leaf}} < 1$ by our assumption. By the P-Partition Lemma (Lemma 33) we can find a grounded sub \bar{Z} -CT $\mathfrak{T}' = (V', E', L', P')$ with $|\mathfrak{T}'|_{\text{Leaf}} < 1$ such that every infinite path has an infinite number of P_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 $\mathfrak{T}'' = (V', E', L'', P'')$ with $P_1 \cap P' \subseteq P''$ 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 get $|\mathfrak{T}'|_{\text{Leaf}} = |\mathfrak{T}''|_{\text{Leaf}}$. 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 $|\mathfrak{T}'|_{\text{Leaf}} < 1$ we thus have $|\mathfrak{T}''|_{\text{Leaf}} < 1$, which is a contradiction to our assumption that $\overline{\bigcup_{X \in \hat{X}} X}$ is iAST.

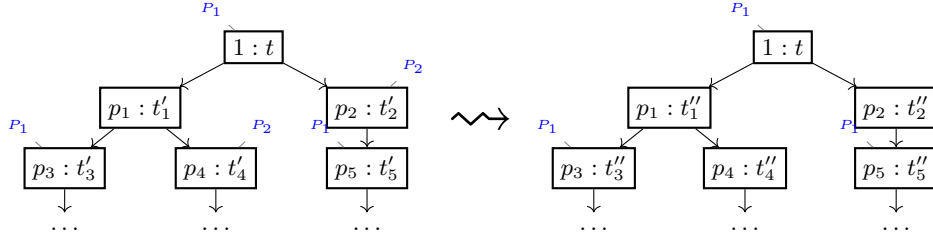


Fig. 2: Construction for this proof. Some nodes $x \in P_2$ in \mathfrak{T}' are removed from P , which yields \mathfrak{T}'' .

The core idea of this construction is that annotations introduced by rewrite steps at a node $x \in P_2$ 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 (22), 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'' . During our construction, we ensure that the following property holds:

$$\text{For every } x \in Q \text{ we have } t'_x \doteq t''_x \text{ and } \text{pos}_{\mathcal{D}^\#}(t'_x) \setminus \text{Junk}(t'_x, \hat{X}) \subseteq \text{pos}_{\mathcal{D}^\#}(t''_x). \quad (23)$$

Here, for any term t'_x , let $\text{Junk}(t'_x, \hat{X})$ denote the positions of all annotated subterms $s \leq_{\#} t'_x$ 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 $\pi \in \text{Junk}(t'_x, \hat{X}) : \Leftrightarrow$ there is no $A \in W$ with $A >_{\mathfrak{G}}^* X$ for some $X \in \hat{X}$ such that there is an ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in A$, and a substitution σ with $\#_{\varepsilon}(t'_x | \pi) \xrightarrow{i}_{\text{np}(\mathcal{P})}^* \ell \# \sigma \in \text{ANF}_{\mathcal{P}}$.

We start by setting $t''_v = t'_v$ for the root v of \mathfrak{T}' . Here, our property (23) 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 \bar{Z} -CT, we have $t'_x \xrightarrow{i}_{\bar{Z}} \{ \frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t'_{y_k} \}$, 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 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 $\text{Junk}(t'_x, \hat{X})$, and hence, if $\pi \in \text{pos}_{\mathcal{D}\#}(t'_x)$, then also $\pi \in \text{pos}_{\mathcal{D}\#}(t''_x)$ by (23). Thus, whenever we create annotations in the rewrite step in \mathfrak{T}' (a step with **(p)** or **(pr)**), then we do the same in \mathfrak{T}'' (the step is also a **(p)** or **(pr)** step, respectively), and whenever we remove annotations in the rewrite step in \mathfrak{T}'' (a step with **(p)** or **(irr)**), then we do the same in \mathfrak{T}' (the step is also either a **(p)** or **(irr)** step). Therefore, we also get $\text{pos}_{\mathcal{D}\#}(t'_{y_j}) \setminus \text{Junk}(t'_{y_j}, \hat{X}) \subseteq \text{pos}_{\mathcal{D}\#}(t''_{y_j})$ for all $1 \leq j \leq k$ and (23) 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 $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$, then we can use $\ell \rightarrow \{p_1 : b(r_1), \dots, p_k : b(r_k)\}^m$ instead, with the same position π , and the same substitution. Note that if $\pi \in \text{pos}_{\mathcal{D}\#}(t'_x)$, then all the annotations introduced by the ADP are in $\text{Junk}(t'_{y_j}, \hat{X})$ for all $1 \leq j \leq k$, since the used ADP is not in $\bigcup_{X \in \hat{X}} X$ and by (22) 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, (23) is satisfied.

We have now shown that (22) holds.

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

Using (21) and (22), by induction on $>_{\mathfrak{G}}$ we now prove that

$$\text{for every } X \in W, \text{ the ADP problem } \overline{\bigcup_{X >_{\mathfrak{G}}^* Y} Y} \text{ is iAST.} \quad (24)$$

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 (21), \bar{X} is iAST.

For the induction step, we consider an $X \in W$ and assume that $\overline{\bigcup_{Y >_{\mathfrak{G}}^* Z} Z}$ is iAST for every $Y \in W$ with $X >_{\mathfrak{G}}^+ Y$. Let $\text{Succ}(X) = \{Y \in W \mid X >_{\mathfrak{G}} Y\} = \{Y_1, \dots, Y_m\}$ be the set of all direct successors of X . The induction hypothesis states that $\overline{\bigcup_{Y_u >_{\mathfrak{G}}^* Z} Z}$ is iAST 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 iAST.

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 iAST.

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 iAST by our outer induction hypothesis and $\overline{\bigcup_{1 \leq i \leq u} \bigcup_{Y_i >_{\mathfrak{G}}^* Z} Z}$ is iAST by our inner induction hypothesis. By (22), we know that then $\overline{\bigcup_{1 \leq i \leq u+1} \bigcup_{Y_i >_{\mathfrak{G}}^* Z} Z}$ is iAST as well. The conditions for (22) 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 iAST. We know that \bar{X} is iAST by our assumption and that $\overline{\bigcup_{1 \leq i \leq m} \bigcup_{Y_i >_{\mathfrak{G}}^* Z} Z}$ is iAST. Hence, by (22) we obtain

that $\overline{\bigcup_{X >_{\mathfrak{G}}^* Y} Y}$ iAST. Again, the conditions of (22) are satisfied, since X is strictly greater w.r.t. $>_{\mathfrak{G}}^+$ than all Z with $Y_i >_{\mathfrak{G}}^* Z$.

4. \mathcal{P} is iAST.

In (24) we have shown that $\overline{\bigcup_{X >_{\mathfrak{G}}^* Y} Y}$ for every $X \in W$ is iAST. Let $X_1, \dots, X_m \in W$ 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 iAST for all $1 \leq u \leq m$ by (22), analogous to the previous induction. Again, the conditions of (22) 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 iAST and this ends the proof. \square

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

Proof.

Completeness: Every $\mathcal{T}_{\text{UT}}(\mathcal{P})$ -CT is also a \mathcal{P} -CT with fewer annotations in the terms. So if $\overline{\mathcal{T}_{\text{UT}}(\mathcal{P})}$ is not iAST, then there exists a $\mathcal{T}_{\text{UT}}(\mathcal{P})$ -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 $\mathcal{T}_{\text{UT}}(\mathcal{P})$ is not iAST, then \mathcal{P} is not iAST either.

Soundness: Let \mathcal{P} be not iAST. Then by Lemma 34 there exists a \mathcal{P} -CT $\mathfrak{T} = (V, E, L, P)$ that converges with probability < 1 whose root is labeled with $(1 : t)$ and $\flat(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \mathcal{P}$, and $\text{pos}_{\mathcal{D}^{\#}}(t) = \{\varepsilon\}$. We will now create a $\mathcal{T}_{\text{UT}}(\mathcal{P})$ -CT $\mathfrak{T}' = (V, E, L', P)$, 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 $|\mathfrak{T}'|_{\text{Leaf}} = |\mathfrak{T}|_{\text{Leaf}} < 1$, and hence $\overline{\mathcal{T}_{\text{UT}}(\mathcal{P})}$ is not iAST either.

We now construct the new labeling L' for the $\mathcal{T}_{\text{UT}}(\mathcal{P})$ -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 $x \in X$:

$$\text{For every } x \in X \text{ we have } t_x \dot{=} t'_x \text{ and } \text{pos}_{\mathcal{D}^{\#}}(t_x) \setminus \text{Junk}(t_x) \subseteq \text{pos}_{\mathcal{D}^{\#}}(t'_x). \quad (25)$$

Here, for any annotated term t_x , let $\text{Junk}(t_x)$ denote the set of all positions of annotations in t_x that will never be used for a rewrite step in \mathfrak{T} . To be precise, we define $\text{Junk}(t_x)$ recursively: For the term t at the root, we define $\text{Junk}(t) = \emptyset$. For a node y_j for some $1 \leq j \leq h$ with predecessor x such that $t_x \xrightarrow{i}_{\mathcal{P}} \left\{ \frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_h}}{p_x} : t_{y_h} \right\}$ at a position π , we define $\text{Junk}(t_{y_j}) = \{\rho \mid \rho \in \text{Junk}(t_x), \pi \not\prec \rho\}$

if $\pi \notin \text{pos}_{\mathcal{D}^\#}(t_x)$, and otherwise, we define $\text{Junk}(t_{y_j}) = \{\rho \mid \rho \in \text{Junk}(t_x), \pi \not\prec \rho\} \cup \text{Junk}((t_x)_+^j, \mathfrak{T})$. Here, we have $\rho \in \text{Junk}((t_x)_+^j, \mathfrak{T})$, if $\rho = \pi.\tau$ and $\tau \in \text{pos}_{\mathcal{D}^\#}(\hat{r}_j)$, for the used ADP $\hat{\ell} \rightarrow \{\hat{p}_1 : \hat{r}_1, \dots, \hat{p}_h : \hat{r}_h\}^{m'}$, and there is no (not necessarily direct) successor node in \mathfrak{T} that rewrites at position ρ without rewriting above position ρ before.

We start with the same term t at the root. Here, our property (25) 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 $xE = \{y_1, \dots, y_k\}$ be the set of its successors. We need to define the terms for the nodes y_1, \dots, y_k in \mathfrak{T}' . Since x is not a leaf and \mathfrak{T} is a \mathcal{P} -CT, we have $t_x \xrightarrow{i}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$. If we performed a step with $\xrightarrow{i}_{\mathcal{P}}$ using the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$, the position π , and the substitution σ in \mathfrak{T} , then we can use the ADP $\ell \rightarrow \{p_1 : b_{\ell, \mathcal{P}}(r_1), \dots, p_k : b_{\ell, \mathcal{P}}(r_k)\}^m$ with the same position π and the same substitution σ . Now, we directly get $t_{y_j} \doteq t'_{y_j}$ and $\text{pos}_{\mathcal{D}^\#}(t_{y_j}) \setminus \text{Junk}(t_{y_j}) \subseteq \text{pos}_{\mathcal{D}^\#}(t'_{y_j})$ for all $1 \leq j \leq k$ since the original rule contains the same terms with more annotations, but all missing annotations are in $\text{Junk}(t_x)$ by definition of $b_{\ell, \mathcal{P}}(r_j)$ for each $1 \leq j \leq k$. \square

Theorem 17 (Prob. Usable Rules Processor). *For an ADP problem \mathcal{P} and $f \in \Sigma^\#$, let $\text{Rules}_{\mathcal{P}}(f) = \{\ell \rightarrow \mu^{\text{true}} \in \mathcal{P} \mid \text{root}(\ell) = f\}$. For any $t \in \mathcal{T}(\Sigma^\#, \mathcal{V})$, its usable rules $\mathcal{U}_{\mathcal{P}}(t)$ are the smallest set with $\mathcal{U}_{\mathcal{P}}(x) = \emptyset$ for all $x \in \mathcal{V}$ and $\mathcal{U}_{\mathcal{P}}(f(t_1, \dots, t_n)) = \text{Rules}_{\mathcal{P}}(f) \cup \bigcup_{i=1}^n \mathcal{U}_{\mathcal{P}}(t_i) \cup \bigcup_{\ell \rightarrow \mu^{\text{true}} \in \text{Rules}_{\mathcal{P}}(f), r \in \text{Supp}(\mu)} \mathcal{U}_{\mathcal{P}}(b(r))$, otherwise. The usable rules for \mathcal{P} are $\mathcal{U}(\mathcal{P}) = \bigcup_{\ell \rightarrow \mu^m \in \mathcal{P}, r \in \text{Supp}(\mu), t \triangleleft_{\#} r} \mathcal{U}_{\mathcal{P}}(t^\#)$. Then $\text{ProcUR}(\mathcal{P}) = \{\mathcal{U}(\mathcal{P}) \cup \{\ell \rightarrow \mu^{\text{false}} \mid \ell \rightarrow \mu^m \in \mathcal{P} \setminus \mathcal{U}(\mathcal{P})\}\}$ is sound and complete, i.e., we turn the flag of all non-usable rules to false.*

Proof. Let $\bar{\mathcal{P}} = \mathcal{U}(\mathcal{P}) \cup \{\ell \rightarrow \mu^{\text{false}} \mid \ell \rightarrow \mu^m \in \mathcal{P} \setminus \mathcal{U}(\mathcal{P})\}$.

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

Soundness: Assume that \mathcal{P} is not iAST. Then by Lemma 34 there exists a \mathcal{P} -CT $\mathfrak{T} = (V, E, L, P)$ that converges with probability < 1 whose root is labeled with $(1 : t)$ and $b(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \mathcal{P}$, and $\text{pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$.

By the definition of usable rules, as in the non-probabilistic case, rules $\ell \rightarrow \mu \in \mathcal{P}$ that are not usable (i.e., $\ell \rightarrow \mu \notin \bar{\mathcal{P}}$) will never be used below an annotated symbol in such a \mathcal{P} -CT. Hence, we can also view \mathfrak{T} as a $\bar{\mathcal{P}}$ -CT that converges with probability < 1 and thus \mathcal{P} is not iAST. \square

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

- (1) *For every $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^{\text{true}} \in \mathcal{P}$, we have $\text{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(b(r_j))$.*

- (2) For every $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in \mathcal{P}$, we have $\text{Pol}(\ell^\#) \geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \triangleleft_{\#} r_j} \text{Pol}(t^\#)$.
- (3) For every $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in \mathcal{P}_>$, there exists a $1 \leq j \leq k$ with $\text{Pol}(\ell^\#) > \sum_{t \triangleleft_{\#} r_j} \text{Pol}(t^\#)$.
- If $m = \text{true}$, then we additionally have $\text{Pol}(\ell) \geq \text{Pol}(b(r_j))$.

Then $\text{Proc}_{\text{RP}}(\mathcal{P}) = \{\mathcal{P}_{\geq} \cup b(\mathcal{P}_{>})\}$ is sound and complete.

Proof. Let $\bar{\mathcal{P}} = \mathcal{P}_{\geq} \cup b(\mathcal{P}_{>})$.

Completeness: Every $\bar{\mathcal{P}}$ -CT is also a \mathcal{P} -CT with fewer annotations in the terms. So if $\bar{\mathcal{P}}$ is not iAST, then there exists a $\bar{\mathcal{P}}$ -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 $\bar{\mathcal{P}}$ is not iAST, then \mathcal{P} is not iAST either.

Soundness: This proof uses the proof idea for AST from [33]. 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 $|\mathfrak{T}^{\leq N}|_{\text{Leaf}} \geq p_{\min}^N$ for any $N \in \mathbb{N}$.
- (IV) We prove that $|\mathfrak{T}^{\leq N}|_{\text{Leaf}} = 1$ for any $N \in \mathbb{N}$.
- (V) Finally, we prove that $|\mathfrak{T}|_{\text{Leaf}} = 1$.

Parts (II) to (V) remain completely the same as in [28]. We only show that we can adjust part (I) to our new rewrite relation and new annotated dependency pairs.

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

We set $V(s) = \sum_{t \triangleleft_{\#} s} \text{Pol}(t^\#)$, and show that the conditions (1), (2), and (3) of the lemma extend to rewrite steps instead of just rules:

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

- (a) In this case, there exist a rule $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}$ 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 $s|_{\pi} = \ell\sigma \in \text{ANF}_{\mathcal{P}}$, 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, let $\pi = \varepsilon$. Hence, we have $s = \ell\sigma \xrightarrow{i} \{p_1 : r_1\sigma, \dots, p_k : r_k\sigma\}$. 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 $\pi = i.\pi'$, $s = f(s_1, \dots, s_i, \dots, s_n)$, $f \in \Sigma$, $s_i \xrightarrow{i} \{p_1 : t_{i,1}, \dots, p_k : t_{i,k}\}$, and $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \leq j \leq k$. Then by the induction hypothesis we have $\text{Pol}(s_i) \geq \text{Pol}(t_{i,j})$. For $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ we obtain

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

- (b) In this case, there exist an ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in \mathcal{P}_{>}$, a substitution σ , and position $\pi \in \text{pos}_{\mathcal{D}^\#}(a)$ with $b(a|_\pi) = \ell\sigma \in \text{ANF}_{\mathcal{P}}$ and $b_j \doteq a[r_j\sigma]_\pi$. First, assume that $m = \text{true}$. Let $I_1 = \{\tau \in \text{pos}_{\mathcal{D}^\#}(a) \mid \tau < \pi\}$ be the set of positions of all annotations strictly above π , $I_2 = \{\tau \in \text{pos}_{\mathcal{D}^\#}(a) \mid \tau > \pi\}$ be the set of positions of all annotations strictly below π , and let $I_3 = \{\tau \in \text{pos}_{\mathcal{D}^\#}(a) \mid \tau \perp \pi\}$ be the set of positions of all annotations orthogonal to π . Furthermore, for each $i \in I_1$ let τ_i be the positions such that $i.\tau_i = \pi$. By Requirement (3), there exists a $1 \leq j \leq k$ with $\text{Pol}(\ell^\#) > \sum_{t \leq_\# r_j} \text{Pol}(t^\#)$ and, additionally, $\text{Pol}(\ell) \geq \text{Pol}(b(r_j))$ since $m = \text{true}$. As these inequations hold for all instantiations of the occurring variables, we have

$$\begin{aligned} V(a) &= \sum_{s \leq_\# a} \text{Pol}(s^\#) \\ &= \text{Pol}(\#_\varepsilon(s|_\pi)) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i)) + \sum_{i' \in I_2} \text{Pol}(\#_\varepsilon(a|i')) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\ &\geq \text{Pol}(\#_\varepsilon(s|_\pi)) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i)) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\ &= \text{Pol}(\#_\varepsilon(\ell)\sigma) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i)) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\ &\quad \text{(as } \#_\varepsilon(s|_\pi) = \#_\varepsilon(\ell)\sigma) \\ &> \sum_{s \leq_\# r_j} \text{Pol}(\#_\varepsilon(s)\sigma) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i)) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\ &\quad \text{(as } \text{Pol}(\#_\varepsilon(\ell)) > \sum_{s \leq_\# r_j} \text{Pol}(\#_\varepsilon(s)), \text{ hence } \text{Pol}(\#_\varepsilon(\ell)\sigma) > \sum_{s \leq_\# r_j} \text{Pol}(\#_\varepsilon(s)\sigma)) \\ &\geq \sum_{s \leq_\# b_j|_\pi} \text{Pol}(\#_\varepsilon(s)) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i[r_j\sigma]_{\tau_i})) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\ &\quad \text{(by } \text{Pol}(\ell) \geq \text{Pol}(r_j) \text{ and (a))} \\ &= \sum_{s \leq_\# b_j} \text{Pol}(s^\#) \\ &= V(b_j) \end{aligned}$$

In case of $m = \text{false}$ we additionally remove $\sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i[r_j\sigma]_{\tau_i}))$, so that the inequation remains correct.

- (c) In this case, there exists a rule $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}$ with $\text{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(r_j)$, a substitution σ , and a position π of s such that $s|_\pi = \ell\sigma \in \text{ANF}_{\mathcal{P}}$, 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 $\pi = \varepsilon$ we have $s = \ell\sigma \xrightarrow{i} \{p_1 : r_1\sigma, \dots, p_k : r_k\sigma\}$. As $\text{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(r_j)$ holds for all instantiations of the occurring variables, for $t_j = r_j\sigma$ we obtain

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

In the induction step, we have $\pi = i.\pi'$, $s = f(s_1, \dots, s_i, \dots, s_n)$, $s_i \xrightarrow{i} \{p_1 : t_{i,1}, \dots, p_k : t_{i,k}\}$, and $t_j = f(s_1, \dots, t_{i,j}, \dots, s_n)$ with $t_{i,j} = s_i[r_j\sigma]_{\pi'}$ for all $1 \leq$

$j \leq k$. Then by the induction hypothesis we have $\text{Pol}(s_i) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(t_{i,j})$. Thus, we have

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

- (d) In this case, there exist an ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m \in \mathcal{P}$, a substitution σ , and position π with $b(a|_\pi) = \ell\sigma \in \text{ANF}_{\mathcal{P}}$ and $b_j \doteq a[r_j\sigma]_\pi$. First, assume that $m = \text{true}$ and $\pi \in \text{pos}_{\mathcal{D}^\#}(a)$. Let $I_1 = \{\tau \in \text{pos}_{\mathcal{D}^\#}(a) \mid \tau < \pi\}$ be the set of positions of all annotations strictly above π , $I_2 = \{\tau \in \text{pos}_{\mathcal{D}^\#}(a) \mid \tau > \pi\}$ be the set of positions of all annotations strictly below π , and let $I_3 = \{\tau \in \text{pos}_{\mathcal{D}^\#}(a) \mid \tau \perp \pi\}$ be the set of positions of all annotations orthogonal to π . Furthermore, for each $i \in I_1$ let τ_i be the position such that $i.\tau_i = \pi$. By Requirement (2), we have $\text{Pol}(\#_\varepsilon(\ell)) \geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \sqsubseteq_{\#} r_j} \text{Pol}(\#_\varepsilon(t))$ and by (1) we have $\text{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(b(r_j))$. As these inequations hold for all instantiations of the occurring variables, we have

$$\begin{aligned}
V(a) &= \sum_{t \sqsubseteq_{\#} a} \text{Pol}(t^\#) \\
&= \text{Pol}(\#_\varepsilon(a|_\pi)) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i)) + \sum_{i' \in I_2} \text{Pol}(\#_\varepsilon(a|i')) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\
&\geq \text{Pol}(\#_\varepsilon(a|_\pi\sigma)) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i)) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\
&= \text{Pol}(\#_\varepsilon(\ell)\sigma) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i)) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\
&\quad \text{(as } a|_\pi = \#_\varepsilon(\ell)\sigma) \\
&\geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \sqsubseteq_{\#} r_j\sigma} \text{Pol}(\#_\varepsilon(t)) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i)) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\
&\quad \text{(by } \text{Pol}(\#_\varepsilon(\ell)) \geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \sqsubseteq_{\#} r_j} \text{Pol}(\#_\varepsilon(t)), \\
&\quad \text{hence } \text{Pol}(\#_\varepsilon(\ell)\sigma) \geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \sqsubseteq_{\#} r_j\sigma} \text{Pol}(\#_\varepsilon(t))) \\
&\geq \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \sqsubseteq_{\#} r_j\sigma} \text{Pol}(\#_\varepsilon(t)) + \sum_{i \in I_1} \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(\#_\varepsilon(a|i[r_j\sigma]_{\tau_i})) \\
&\quad + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\
&\quad \text{(by } \text{Pol}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \text{Pol}(r_j) \text{ and (c))} \\
&= \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \sqsubseteq_{\#} r_j\sigma} \text{Pol}(\#_\varepsilon(t)) + \sum_{1 \leq j \leq k} \sum_{i \in I_1} p_j \cdot \text{Pol}(\#_\varepsilon(a|i[r_j\sigma]_{\tau_i})) \\
&\quad + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\
&= \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \sqsubseteq_{\#} r_j\sigma} \text{Pol}(\#_\varepsilon(t)) + \sum_{1 \leq j \leq k} p_j \cdot \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i[r_j\sigma]_{\tau_i})) \\
&\quad + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i')) \\
&= \sum_{1 \leq j \leq k} p_j \cdot (\sum_{t \sqsubseteq_{\#} r_j\sigma} \text{Pol}(\#_\varepsilon(t)) + \sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i[r_j\sigma]_{\tau_i})) + \sum_{i' \in I_3} \text{Pol}(\#_\varepsilon(a|i'))) \\
&= \sum_{1 \leq j \leq k} p_j \cdot \sum_{t \sqsubseteq_{\#} b_j} \text{Pol}(t^\#) \\
&= V(b_j)
\end{aligned}$$

In case of $\pi \notin \text{pos}_{\mathcal{D}^\#}(a)$, we remove $\sum_{t \sqsubseteq_{\#} r_j\sigma} \text{Pol}(t^\#)$ in the end, and in case of $m = \text{false}$ we remove $\sum_{i \in I_1} \text{Pol}(\#_\varepsilon(a|i[r_j\sigma]_{\tau_i}))$.

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

Finally, we prove soundness of the new rewriting processor.

Theorem 23 (Soundness of the Rewriting Processor). $\text{Proc}_{\mathcal{R}}$ as in Def. 22 is sound if one of the following cases holds:

1. $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is NO, and the rule used for rewriting $r_j|_{\tau}$ is L and NE.
2. $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is NO, and all its rules have the form $\ell' \rightarrow \{1 : r'\}$ for some ℓ', r' .
3. $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is NO, $r_j|_{\tau}$ is a ground term, and $r_j \xrightarrow{i}_{\tau} \{q_1 : e_1, \dots, q_h : e_h\}$ is an innermost step.

Proof. Let $\overline{\mathcal{P}'} = \mathcal{P}' \cup N \cup \{\ell \rightarrow \{p_1 : b(r_1), \dots, p_k : b(r_k)\}^m\}$ and $\overline{\mathcal{P}} = \overline{\mathcal{P}'} \cup \{\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m\}$. We call $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ the *old ADP*, we call $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\} \setminus \{p_j : r_j\} \cup \{p_j \cdot q_1 : e_1, \dots, p_j \cdot q_h : e_h\}^m$ the *new ADP*, and $\ell \rightarrow \{p_1 : b(r_1), \dots, p_k : b(r_k)\}^m$ is called the *non-annotated old ADP*.

First Case

We start with the case where $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is NO and the used rule $\hat{\ell} \rightarrow \{\hat{p}_1 : \hat{r}_1, \dots, \hat{p}_h : \hat{r}_h\}$ is L and NE. First, note that the rule used for rewriting $r_j|_{\tau}$ in the rewrite processor is contained in $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$.⁹ Since $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is NO, we can therefore be sure that there is only a single rule applicable at position τ .

Let \mathcal{P} be not iAST. Then there exists a \mathcal{P} -CT $\mathfrak{T} = (V, E, L, P)$ that converges with probability $c < 1$. We will now create a $\overline{\mathcal{P}'}$ -CT $\mathfrak{T}' = (V', E', L', P')$ such that $|\mathfrak{T}'|_{\text{Leaf}} \leq |\mathfrak{T}|_{\text{Leaf}} < 1$, and hence $\overline{\mathcal{P}'}$ is not iAST either.

The core steps of the proof are the following:

1. We iteratively remove usages of the old ADP using a construction $\Phi(\circ)$. The limit of this iteration, namely $\mathfrak{T}^{(\infty)}$ is a $\overline{\mathcal{P}'}$ -CT that converges with probability at most $c < 1$, hence, $\overline{\mathcal{P}'}$ is not iAST.
 - 1.1. For a \mathcal{P} -CT \mathfrak{T}_x that uses the old ADP at the root x at a position $\pi \in \text{pos}_{\mathcal{D}\#}(t_x)$, we create a new $\overline{\mathcal{P}'}$ -CT $\Phi(\mathfrak{T}_x)$ that uses the new ADP at the root.
 - 1.2. For a \mathcal{P} -CT \mathfrak{T}_x that uses the old ADP at the root x at a position $\pi \notin \text{pos}_{\mathcal{D}\#}(t_x)$, we create a new $\overline{\mathcal{P}'}$ -CT $\Phi(\mathfrak{T}_x)$ that uses the non-annotated old ADP at the root.

This proof structure will also be used in the other cases for soundness.

1. We iteratively remove usages of the old ADP

W.l.o.g., in \mathfrak{T} there exists at least one rewrite step performed at some node x with the old ADP (otherwise, \mathfrak{T} would already be a $\overline{\mathcal{P}'}$ -CT). Furthermore, we can assume that this is the first such rewrite step in the path from the root to the node x and that x is a node of minimum depth with this property. We will now replace this rewrite step with a rewrite step using the new ADP such that we result in a CT $\mathfrak{T}^{(1)}$ with the following connections between \mathfrak{T} and $\mathfrak{T}^{(1)}$. Here, for an infinite path $p = v_1, v_2, \dots$ in a CT $\mathfrak{T} = (V, E, L, P)$, by $\mathcal{C}(p) = w_1, w_2, \dots$ we denote the sequence of $\{P, R\}$ -labels for these nodes, i.e., we have $w_i = P$ if $v_i \in P$ or $w_i = R$ otherwise for all i .

- (a) $|\mathfrak{T}^{(1)}|_{\text{Leaf}} \leq |\mathfrak{T}|_{\text{Leaf}} = c$, and
- (b) for every infinite path p in $\mathfrak{T}^{(1)}$ we can find an infinite path p' in \mathfrak{T} such that $\mathcal{C}(p)$ and $\mathcal{C}(p')$ only differ in a finite number of R -nodes, i.e., we can remove and add a finite number of R -nodes from $\mathcal{C}(p')$ to get $\mathcal{C}(p)$.

⁹ The rules that are applicable at position τ do not have to be applicable in an innermost RST, see [16, Sect. 6] and [39, Ex. 5.14].

This construction only works because $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is NO and there exists no annotation below τ , and the resulting CT has only at most the same probability of termination as the original one due to the fact that $\hat{\ell} \rightarrow \{\hat{p}_1 : \hat{r}_1, \dots, \hat{p}_h : \hat{r}_h\}^{m'}$ is also L and NE. Let \mathfrak{T}_x be the induced sub-CT that starts at node x , i.e. $\mathfrak{T}_x = \mathfrak{T}[xE^*]$. The construction defines a new tree $\Phi(\mathfrak{T}_x)$ such that (a) and (b) w.r.t. \mathfrak{T}_x and $\Phi(\mathfrak{T}_x)$ holds, and where we use the new ADP at the root node x instead of the old one (i.e., we pushed the first use of the old ADP deeper into the tree). Then, by replacing the subtree \mathfrak{T}_x with the new tree $\Phi(\mathfrak{T}_x)$ in \mathfrak{T} , we get a \mathcal{P} -CT $\mathfrak{T}^{(1)}$, with (a) and (b) w.r.t. \mathfrak{T} and $\mathfrak{T}^{(1)}$, and where we use the new ADP at node x instead of the old one. We can then do this replacement iteratively for every use of the old ADP, i.e., we again replace the first use of the old ADP in $\mathfrak{T}^{(1)}$ to get $\mathfrak{T}^{(2)}$ with (a) and (b) w.r.t. $\mathfrak{T}^{(1)}$ and $\mathfrak{T}^{(2)}$, and so on. In the end, the limit of all these CTs $\lim_{i \rightarrow \infty} \mathfrak{T}^{(i)}$ is a $\overline{\mathcal{P}}$ -CT, that we denote by $\mathfrak{T}^{(\infty)}$ and that converges with probability at most $c < 1$, and hence, $\overline{\mathcal{P}}$ is not iAST.

To see that $\mathfrak{T}^{(\infty)}$ is indeed a valid $\overline{\mathcal{P}}$ -CT, note that in every iteration of the construction we turn a use of the old ADP at minimum depth into a use of the new one. Hence, for every depth H of the tree, we eventually turned every use of the old ADP up to depth H into a use of the new one so that the construction will not change the tree above depth H anymore, i.e., there exists an m_H such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(m_H)}$ are the same trees up to depth H . This means that the sequence $\lim_{i \rightarrow \infty} \mathfrak{T}^{(i)}$ really converges into a tree that satisfies the first five conditions of a $\overline{\mathcal{P}}$ -CT. We only have to show that the last condition of a CT, namely that every infinite path in $\lim_{i \rightarrow \infty} \mathfrak{T}^{(i)}$ contains infinitely many nodes from P , holds as well. First, by induction on n one can prove that all trees $\mathfrak{T}^{(i)}$ for all $1 \leq i \leq n$ satisfy Condition (6), because due to (b) we can find for each infinite path $p \in \mathfrak{T}^{(i)}$ an infinite path p' in \mathfrak{T} such that $\mathcal{C}(p)$ and $\mathcal{C}(p')$ only differ in a finite number of R -labels. Hence, if p contains no nodes from P , then p' contains no nodes from P , which is a contradiction to \mathfrak{T} satisfying Condition (6). Moreover, we only replace subtrees after a node in P , the node itself remains in P , and after we replaced the subtree at a node v , we will never replace a predecessor of v anymore. This means that the subsequence between the i -th and $(i+1)$ -th occurrence of P in p and the subsequence between the i -th and $(i+1)$ -th occurrence of P in p' only differ in a finite number of R nodes again, for every i . Now, let $p = v_1, v_2, \dots$ be an infinite path that starts at the root in $\mathfrak{T}^{(\infty)}$ and only contains finitely many nodes from P , i.e., there exists an $1 \leq i$ such that v_i, v_{i+1}, \dots contains no node from P . Again, let $m_H \in \mathbb{N}$ such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(m_H)}$ are the same trees up to depth H . The path v_1, \dots, v_H must be a path in $\mathfrak{T}^{(m_H)}$ as well. For two different H_1, H_2 with $i < H_1 < H_2$ we know that since the path v_i, \dots, v_{H_2} contains no node from P , the path must also exist in $\mathfrak{T}^{(m_{H_1})}$. We can now construct an infinite path in $\mathfrak{T}^{(m_i)}$ that contains no nodes from P , which is a contradiction.

Next, we want to prove that we really have $|\mathfrak{T}^{(\infty)}|_{\text{Leaf}} \leq c$. Again, by induction on n one can prove that $|\mathfrak{T}^{(i)}|_{\text{Leaf}} \leq c$ for all $1 \leq i \leq n$. Assume for a contradiction that $\mathfrak{T}^{(\infty)}$ converges with probability greater than c , i.e. $|\mathfrak{T}^{(\infty)}|_{\text{Leaf}} > c$. Then there exists an $H \in \mathbb{N}$ for the depth such that $\sum_{x \in \text{Leaf } \mathfrak{T}^{(\infty)}, d(x) \leq H} p_x > c$. Here, $d(x)$ denotes the depth of node x . Again, let $m_H \in \mathbb{N}$ such that $\mathfrak{T}^{(\infty)}$ and $\mathfrak{T}^{(m_H)}$ are the same trees up to depth H . But this would mean that $|\mathfrak{T}^{(m_H)}|_{\text{Leaf}} \geq$

$\sum_{x \in \text{Leaf}^{\mathfrak{T}(m_H)}, d(x) \leq H} p_x = \sum_{x \in \text{Leaf}^{\mathfrak{T}(\infty)}, d(x) \leq H} p_x > c$, which is a contradiction to $|\mathfrak{T}(m_H)|_{\text{Leaf}} \leq c$.

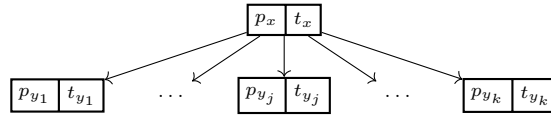
It remains to show the mentioned construction $\Phi(\circ)$.

1.1. Construction of $\Phi(\circ)$ if $\pi \in \text{pos}_{\mathcal{D}\#}(t_x)$

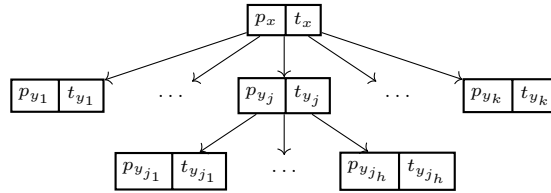
Let \mathfrak{T}_x be a \mathcal{P} -CT that uses the old ADP at the root node x , i.e. $t_x \xrightarrow{\mathcal{P}} \{p_{y_1} : t_{y_1}, \dots, p_{y_k} : t_{y_k}\}$ using the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$, the position π with $\pi \in \text{pos}_{\mathcal{D}\#}(t_x)$, and a substitution σ such that $\mathfrak{b}(t_x|_{\pi}) = \ell\sigma \in \text{ANF}_{\mathcal{P}}$. Then $t_{y_j} = t_x[r_j\sigma]_{\pi}$ if $m = \text{true}$, or $t_{y_j} = \mathfrak{b}_{\pi}^{\uparrow}(t_x[r_j\sigma]_{\pi})$, otherwise.

1.1.1 General construction of $\Phi(\mathfrak{T}_x)$

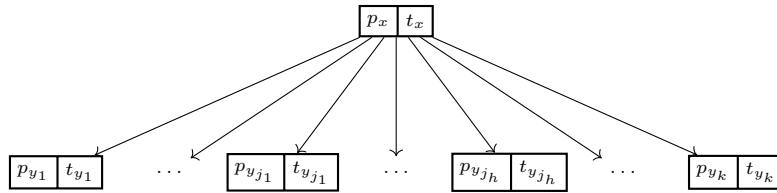
Instead of applying only the old ADP at the root x



where we use an arbitrary rewrite step at node y_j afterwards, we want to directly apply the rewrite rule at position $\pi.\tau$ of the term t_{y_j} in our CT, which we performed on r_j at position τ to transform the old into the new ADP, to get

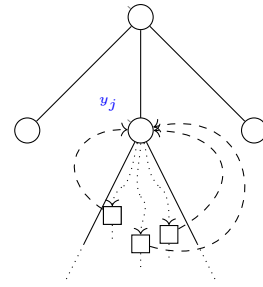


Then, we can contract the edge (x, y_j) to get



and this is equivalent to applying the new ADP. Note that the rewrite step at position $\pi.\tau$ may not be an innermost rewrite step in the CT.

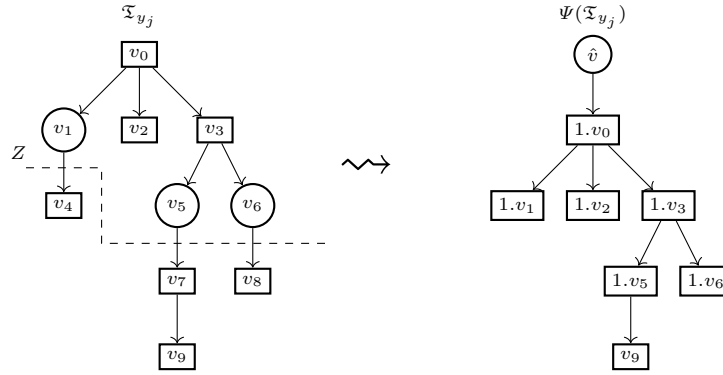
The subtrees that start at the nodes $y_1, \dots, y_{j-1}, y_{j+1}, \dots, y_k$ remain completely the same. We only have to construct a new subtree for node y_j , i.e., the node that really changed when applying the rewriting processor. To be precise, let $\mathfrak{T}_{y_j} = \mathfrak{T}_x[y_j E^*]$ be the subtree starting at node y_j . The construction first creates a new subtree $\Psi(\mathfrak{T}_{y_j})$ such that (a) and (b) hold w.r.t. \mathfrak{T}_{y_j} and $\Psi(\mathfrak{T}_{y_j})$, and that directly performs the first rewrite step at position $\pi.\tau$ at the root of the tree, by pushing it from the original position



in the tree \mathfrak{T}_{y_j} to the root. This can be seen in the diagram above. Again, this push only results in the exact same termination probability due to our restriction that $\hat{\ell} \rightarrow \{\hat{p}_1 : \hat{r}_1, \dots, \hat{p}_h : \hat{r}_h\}$ is L and NE. Then, by replacing \mathfrak{T}_{y_j} by $\Psi(\mathfrak{T}_{y_j})$ in \mathfrak{T}_x we result in a tree \mathfrak{T}'_x such that (a) and (b) hold w.r.t. \mathfrak{T}_x and \mathfrak{T}'_x , and such that we perform the desired rewrite step at node y_j . Finally, we contract the edge (x, y_j) in \mathfrak{T}'_x , in order to get $\Phi(\mathfrak{T}_x)$. Again, (a) and (b) hold w.r.t. \mathfrak{T}_x and $\Phi(\mathfrak{T}_x)$, and we use the new ADP at the root x in $\Phi(\mathfrak{T}_x)$. It only remains to explain the construction of $\Psi(\mathfrak{T}_{y_j})$.

1.1.2. Construction of $\Psi(\mathfrak{T}_{y_j})$

We will move the first rewrite step that takes place at position $\pi.\tau$ from the original tree \mathfrak{T}_{y_j} (example on the left below) to the top of the new tree $\Psi(\mathfrak{T}_{y_j})$ (example on the right below) and show that (a) and (b) both hold after this construction. Below, the circled nodes represent the nodes where we perform a rewrite step at position $\pi.\tau$.



We will define the \mathcal{P} -CT $\Psi(\mathfrak{T}_{y_j})$ that satisfies the properties (a) and (b) w.r.t. \mathfrak{T}_{y_j} and $\Psi(\mathfrak{T}_{y_j})$, and that directly performs the rewrite step $t_{y_j} \xrightarrow{i \rightarrow \mathcal{P}, \pi.\tau} \{\hat{p}_1 : t_{y_{j_1}}, \dots, \hat{p}_h : t_{y_{j_h}}\}$, with the rule $\hat{\ell} \rightarrow \{\hat{p}_1 : \hat{r}_1, \dots, \hat{p}_h : \hat{r}_h\}^{\hat{m}} \in \mathcal{P}$, a substitution $\hat{\sigma}$, and the position $\pi.\tau$, at the new root \hat{v} . Here, we have $\flat(t_{y_j} |_{\pi.\tau}) = \flat(r_j \sigma |_{\tau}) = \flat(r_j |_{\tau} \sigma) = \hat{\ell} \hat{\sigma}$. Let Z be the set of all nodes v of \mathfrak{T}_{y_j} where we did not perform a rewrite step at position $\pi.\tau$ in the path from the root x to the node v , or v is the first node in the path that performs a rewrite step at position $\pi.\tau$. In the example we have $Z = \{v_0, \dots, v_6\} \setminus \{v_4\}$. For each of these nodes $z \in Z$ and each $1 \leq e \leq h$, we create a new node $e.z \in V'$ with edges as in \mathfrak{T}_{y_j} for the nodes in Z , e.g., for the node $1.v_3$ we create an edge to $1.v_5$ and $1.v_6$. Furthermore, we add the edges from the new root \hat{v} to the nodes $e.y_j$ for all $1 \leq e \leq h$. Remember that y_j was the root in the tree \mathfrak{T}_{y_j} and has to be contained in Z . For example, for the node \hat{v} we create an edge to $1.v_0$. For all these new nodes in Z , we show that the following holds:

- (T1) $t_z[\hat{r}_e \gamma]_{\pi.\tau} \doteq t'_{e.z}$ for the substitution γ such that $\flat(t_z |_{\pi.\tau}) = \hat{\ell} \gamma$
- (T2) $\text{pos}_{\mathcal{D}^\#}(t_z[\hat{r}_e \gamma]_{\pi.\tau}) \subseteq \text{pos}_{\mathcal{D}^\#}(t'_{e.z})$.
- (T3) $p_{e.z}^{\Psi(\mathfrak{T}_{y_j})} = p_z^{\mathfrak{T}_{y_j}} \cdot \hat{p}_e$

Note that we only regard the subtree of the j -th child of the root. Because of the prerequisite in the definition of the rewrite processor (there is no annotation below or at position τ), there are no annotations on or below $\pi.\tau$ for all nodes in Z .

Now, for a leaf $e.z \in V'$ either $z \in V$ is also a leaf (e.g., node v_2) or we rewrite the position $\pi.\tau$ at node z in \mathfrak{T}_{y_j} (e.g., node v_1). If we rewrite $t_z \xrightarrow{i} \mathcal{P}, \pi.\tau \{\hat{p}_1 : t_{w_1}, \dots, \hat{p}_h : t_{w_h}\}$, then we have $t_{w_e} = t_x[b(\hat{r}_e)\gamma]_{\pi.\tau}$ if $m = \text{true}$ or $t_{w_e} = b_{\pi.\tau}^\uparrow(t_x[b(\hat{r}_e)\gamma]_{\pi.\tau})$, otherwise. In both cases, we get $t_{w_e} \doteq t_x[b(\hat{r}_e)\gamma]_{\pi.\tau} \doteq_{(T1)} t'_{e.z}$, $\text{pos}_{\mathcal{D}\#}(t_{w_e}) \subseteq \text{pos}_{\mathcal{D}\#}(t_x[b(\hat{r}_e)\gamma]_{\pi.\tau}) \subseteq_{(T2)} \text{pos}_{\mathcal{D}\#}(t'_{e.z})$ and $p_{e.z}^{\Psi(\mathfrak{T}_{y_j})} = p_z^{\mathfrak{T}_{y_j}} \cdot \hat{p}_e \doteq_{(T3)} p_{w_e}^{\mathfrak{T}_{y_j}}$, and we can again copy the rest of this subtree of \mathfrak{T}_{y_j} in our newly generated tree $\Psi(\mathfrak{T}_{y_j})$. In our example, v_1 has the only successor v_4 , hence we can copy the subtree starting at node v_4 , which is only the node itself, to the node $1.v_1$ in $\Psi(\mathfrak{T}_{y_j})$. For v_5 we have the only successor v_7 , hence we can copy the subtree starting at node v_7 , which is the node itself together with its successor v_9 , to the node $1.v_5$ in $\Psi(\mathfrak{T}_{y_j})$. So essentially, we just have to define the part of the tree before we reach the rewrite step in \mathfrak{T}_{y_j} , and then, we have to show that (a) and (b) for \mathfrak{T}_{y_j} and $\Psi(\mathfrak{T}_{y_j})$ are satisfied. We show the latter first, and then explain the proof that this label gives us indeed a valid \mathcal{P} -CT.

We start by showing (a) for \mathfrak{T}_{y_j} and $\Psi(\mathfrak{T}_{y_j})$. Let u be a leaf in $\Psi(\mathfrak{T}_{y_j})$. If $u = e.v$ for some node $v \in Z$ that is a leaf in \mathfrak{T}_{y_j} (e.g., node $1.v_2$), then also $e.v$ must be a leaf in $\Psi(\mathfrak{T}_{y_j})$ for every $1 \leq e \leq h$. Here, we get $\sum_{1 \leq e \leq h} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} \stackrel{(T3)}{=} \sum_{1 \leq e \leq h} p_v^{\mathfrak{T}_{y_j}} \cdot \hat{p}_e = p_v^{\mathfrak{T}_{y_j}} \cdot \sum_{1 \leq e \leq h} \hat{p}_e = p_v^{\mathfrak{T}_{y_j}} \cdot 1 = p_v^{\mathfrak{T}_{y_j}}$. If $u = e.v$ for some node $v \in Z$ that is not a leaf in \mathfrak{T}_{y_j} (e.g., node $1.v_1$), then we know by construction that all successors of v in \mathfrak{T}_{y_j} are not contained in Z and are leaves. Here, we get $p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} \stackrel{(T3)}{=} p_v^{\mathfrak{T}_{y_j}} \cdot \hat{p}_e = p_w^{\mathfrak{T}_{y_j}}$ for the (unique) e -th successor w of v . Finally, if u does not have the form $u = e.v$, then u is also a leaf in \mathfrak{T}_{y_j} with $p_u^{\Psi(\mathfrak{T}_{y_j})} = p_u^{\mathfrak{T}_{y_j}}$. Note that these cases cover no leaf of \mathfrak{T}_{y_j} twice. This implies that we have

$$\begin{aligned}
& |\Psi(\mathfrak{T}_{y_j})|_{\text{Leaf}} \\
&= \sum_{v \in \text{Leaf}^{\Psi(\mathfrak{T}_{y_j})}} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} \\
&= \sum_{\substack{e.v \in \text{Leaf}^{\Psi(\mathfrak{T}_{y_j})} \\ v \in Z}} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} + \sum_{\substack{e.v \in \text{Leaf}^{\Psi(\mathfrak{T}_{y_j})} \\ v \in wE, v \notin Z, w \in Z}} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} + \sum_{\substack{v \in \text{Leaf}^{\Psi(\mathfrak{T}_{y_j})} \\ v \in \text{Leaf}^{\mathfrak{T}_{y_j}}, v \in wE, w \notin Z}} p_v^{\Psi(\mathfrak{T}_{y_j})} \\
&\leq \sum_{\substack{v \in \text{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in Z}} \left(\sum_{1 \leq e \leq h} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} \right) + \sum_{\substack{v \in \text{Leaf}^{\mathfrak{T}_{y_j}}, 1 \leq e \leq h \\ v \in wE, v \notin Z, w \in Z}} p_{e.v}^{\Psi(\mathfrak{T}_{y_j})} + \sum_{\substack{v \in \text{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in wE, w \notin Z}} p_v^{\Psi(\mathfrak{T}_{y_j})} \\
&\leq \sum_{\substack{v \in \text{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in Z}} p_v^{\mathfrak{T}_{y_j}} + \sum_{\substack{v \in \text{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in wE, v \notin Z, w \in Z}} p_v^{\mathfrak{T}_{y_j}} + \sum_{\substack{v \in \text{Leaf}^{\mathfrak{T}_{y_j}} \\ v \in wE, w \notin Z}} p_v^{\mathfrak{T}_{y_j}} \\
&= \sum_{v \in \text{Leaf}^{\mathfrak{T}_{y_j}}} p_v^{\mathfrak{T}_{y_j}} \\
&= |\mathfrak{T}_{y_j}|_{\text{Leaf}}
\end{aligned}$$

Next, we show (b) for \mathfrak{T}_{y_j} and $\Psi(\mathfrak{T}_{y_j})$. Let $p = u_0, u_1, \dots$ be an infinite path in $\Psi(\mathfrak{T}_{y_j})$ that starts at the root \hat{v} . If for all $1 \leq i$ we have $u_i = e.v_i$ for some node $v_i \in Z$ and $1 \leq e \leq h$, then $p = v_1, \dots$ is our desired path in \mathfrak{T}_{y_j} . Otherwise, there is a maximal $1 \leq o$ such that for all $1 \leq i \leq o$ we have $u_i = e.v_i$ for some node $v_i \in Z$ and $1 \leq e \leq h$. Then our desired path is $v_1, \dots, v_o, w, u_{o+1}, \dots$. Here, w is the e -th successor of v_o in \mathfrak{T}_{y_j} . Note that in the first case, we remove one R -label at the start of our path, while in the other case, we just move an R -label from the first position to a later one in the path. This shows that (b) is satisfied.

Finally, we prove that $\Psi(\mathfrak{T}_{y_j})$ is a valid \mathcal{P} -CT. We only need to prove the construction that satisfies all of our conditions for the nodes in Z . As the rest of the tree is copied, we can be sure that all Conditions (1)-(5) of a \mathcal{P} -CT are satisfied. Additionally, due to (b) w.r.t. \mathfrak{T}_{y_j} and $\Psi(\mathfrak{T}_{y_j})$, we get (6) as well, because \mathfrak{T}_{y_j} satisfies (6).

At the root, after applying the old ADP, we directly perform the rewrite step $t_{y_j} \xrightarrow{\hat{c}}_{\mathcal{P}} \{\hat{p}_1 : t'_{1.y_j}, \dots, \hat{p}_h : t'_{h.y_j}\}$, with the rule $\hat{\ell} \rightarrow \{\hat{p}_1 : \hat{r}_1, \dots, \hat{p}_h : \hat{r}_h\}^m \in \mathcal{P}$, a substitution $\hat{\sigma}$, and the position τ . Then, $t_{e.y_j} = t_{y_j}[\hat{b}(\hat{r}_e)\hat{\sigma}]_{\pi.\tau}$ if $m = \text{true}$ or $t_{w_e} = \hat{b}_{\pi.\tau}^\uparrow(t_{y_j}[\hat{b}(\hat{r}_e)\hat{\sigma}]_{\pi.\tau})$, otherwise. Here, the conditions (T1)-(T3) are clearly satisfied. We now construct the rest of this subtree by mirroring the rewrite steps in the original tree (which is always possible due to the fact that $\mathcal{U}_{\mathcal{P}}(r_j|\tau)$ is NO), and once we encounter the rewrite step that we moved to the top, i.e., once we use a rewrite step at position $\pi.\tau$, we skip this rewrite step and directly go on with the e -th successor if we are in a path that went to the e -th successor in the initial rewrite step, as described above. In the following, we distinguish between two different cases for a rewrite step at a node u :

- (A) We use a step with $\hat{c}_{\mathcal{P}}$ in \mathfrak{T}_{y_j} at a position orthogonal to $\pi.\tau$.
- (B) We use a step with $\hat{c}_{\mathcal{P}}$ in \mathfrak{T}_{y_j} at a position below $\pi.\tau$. Note that this is the more interesting case, where we need to use the properties L and NE.

Note that we cannot rewrite above $\pi.\tau$ before rewriting at position $\pi.\tau$ due to the innermost restriction.

(A) If we have $t_u \xrightarrow{\hat{c}}_{\mathcal{P}} \{\frac{p_{g_1}}{p_u} : t_{g_1}, \dots, \frac{p_{g_h}}{p_u} : t_{g_h}\}$, then there is a rule $\bar{\ell} \rightarrow \{\bar{p}_1 : \bar{r}_1, \dots, \bar{p}_h : \bar{r}_h\}^m \in \mathcal{P}$, a substitution δ , and a position $\zeta \in \mathbb{N}^+$ with $\hat{b}(t_u|\zeta) = \bar{\ell}\delta \in \text{ANF}_{\mathcal{P}}$. Furthermore, let $\zeta \perp \pi.\tau$. Then, we have $t'_{e.u}|\zeta \doteq_{(T1)} t_u[\hat{r}_e\gamma]_{\pi.\tau|\zeta} = t_u|\zeta$ for the substitution γ such that $t_u|\pi.\tau = \hat{\ell}\gamma$, and we can rewrite $t'_{e.u}$ using the same rule, same substitution, and same position. Then (T3) is again satisfied. Furthermore, we have $t'_{e.g_j} = t'_{e.u}[\bar{r}_e\delta]_{\zeta}$ if $\zeta \in \text{pos}_{\mathcal{D}\#}(t'_{e.u})$ and $m = \text{true}$, $t'_{e.g_j} = \hat{b}_{\zeta}^\uparrow(t'_{e.u}[\bar{r}_e\delta]_{\zeta})$ if $\zeta \in \text{pos}_{\mathcal{D}\#}(t'_{e.u})$ and $m = \text{false}$, $t'_{e.g_j} = t'_{e.u}[\hat{b}(\bar{r}_e)\delta]_{\zeta}$ if $\zeta \notin \text{pos}_{\mathcal{D}\#}(t'_{e.u})$ and $m = \text{true}$, or $t'_{e.g_j} = \hat{b}_{\zeta}^\uparrow(t'_{e.u}[\hat{b}(\bar{r}_e)\delta]_{\zeta})$ if $\zeta \notin \text{pos}_{\mathcal{D}\#}(t'_{e.u})$ and $m = \text{false}$. In all four cases we get (T1) (using the same substitution γ) and (T2) as well.

(B) If we have $t_u \xrightarrow{\hat{c}}_{\mathcal{P}} \{\frac{p_{g_1}}{p_u} : t_{g_1}, \dots, \frac{p_{g_h}}{p_u} : t_{g_h}\}$, then there is a rule $\bar{\ell} \rightarrow \{\bar{p}_1 : \bar{r}_1, \dots, \bar{p}_h : \bar{r}_h\}^m \in \mathcal{P}$, a substitution δ , and a position $\zeta \in \mathbb{N}^+$ with $\hat{b}(t_u|\zeta) = \bar{\ell}\delta \in \text{ANF}_{\mathcal{P}}$. Furthermore, let $\zeta > \pi.\tau$. Since $\hat{\ell} \rightarrow \{\hat{p}_1 : \hat{r}_1, \dots, \hat{p}_h : \hat{r}_h\}$ is L and NE, we know that $\hat{\ell}$ contains exactly the same variables as \hat{r}_e and all of them exactly once. Furthermore, since $\mathcal{U}_{\mathcal{P}}(r_j|\tau)$ is non-overlapping, we know that the rewriting must be completely inside the substitution γ for the substitution γ

such that $t_u|_{\pi.\tau} = \hat{\ell}\gamma$, i.e., there is a position α_c of a variable c in $\hat{\ell}$ and another position β_c with $\pi.\tau.\alpha_c.\beta_c = \zeta$. Let $\varphi_e(c)$ be the (unique) variable position of c in \hat{r}_e . Then, we have $t'_{e,u}|_{\pi.\tau.\varphi_e(c).\beta_c} \stackrel{(T1)}{=} t_u[\hat{r}_e\gamma]|_{\pi.\tau.\varphi_e(c).\beta_c} = \hat{r}_e\gamma|_{\varphi_e(c).\beta_c} = \gamma(c)|_{\beta_c} = \hat{\ell}\hat{\sigma}|_{\alpha_c.\beta_c} = t_u|_{\pi.\tau.\alpha_c.\beta_c}$, and we can rewrite $t'_{e,u}$ using the same rule, same substitution, and position $\pi.\tau.\varphi_e(c).\beta_c$. Again, in all four cases (T1)-(T3) are satisfied.

1.2. $\pi \notin \text{pos}_{\mathcal{D}^\#}(t_x)$

If we have $\pi \notin \text{pos}_{\mathcal{D}^\#}(t_x)$, then we can simply use the non-annotated old ADP instead of the old one. Since we would remove all annotations in the right-hand side of the rule anyway, due to $\pi \notin \text{pos}_{\mathcal{D}^\#}(t_x)$, this leads to the same labels in the resulting $\overline{\mathcal{P}}$ -CT.

Second Case

Next, we prove the theorem in the case where $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is NO and all rules in $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ have the form $\ell' \rightarrow \{1 : r'\}$ for some terms ℓ' and r' .

The proof uses the same idea as in the first case but the construction of $\Phi(\circ)$ is easier since $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is non-probabilistic. First assume that $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is not weakly innermost terminating. This means that after using the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ as we did in the soundness proof for the first case, every subterm at a position above $\pi.\tau$ will never be in $\text{NF}_{\mathcal{P}}$ in the subtree starting at the j -th successor y_j . Furthermore, since all rules in $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ have the form $\ell' \rightarrow \{1 : r'\}$, we can simply remove all nodes that perform a rewrite step below $\pi.\tau$. To be precise, if there is a node v that performs a rewrite step below position $\pi.\tau$, then we have $t_v \xrightarrow{i}_{\mathcal{P}} \{1 : t_w\}$ for the only successor w of v . Here, we have $t_w \stackrel{\zeta}{=} t_v[r'\sigma]$ for the used substitution σ and position ζ below $\pi.\tau$. The construction $\Psi(\mathfrak{Y}_{y_j})$ contracts all edges (x, y) where we use a rewrite step at a position below $\pi.\tau$. This only removes R -nodes, as there is no annotation below or at position $\pi.\tau$. Furthermore, we adjust the labeling such that the subterm at position $\pi.\tau$ remains the same for the whole CT. Finally, we exchange the rewrite step at the root x from using the old ADP to using the new ADP. Since all subterms at a position above $\pi.\tau$ will never be reduced to normal forms in the original CT, it does not matter which subterms really occur. It is easy to see that $\Phi(\mathfrak{Y}_x)$ is a valid \mathcal{P} -CT and that (a) and (b) hold w.r.t. \mathfrak{Y}_x and $\Phi(\mathfrak{Y}_x)$, and this ends the proof if $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is not weakly innermost terminating.

If $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is weakly innermost terminating, then it follows directly that it is also confluent and terminating [23, Thm. 3.2.11]. We can use the construction $\Psi(\circ)$ from the first case to iteratively push the next innermost rewrite step that is performed below position $\pi.\tau$ to a higher position in the tree, until we reach the node that performs the rewrite step at position $\pi.\tau$. Note that we do not need the conditions L or NE for the used rule here, because only Case (A) of the construction can happen. There is no rewrite step below possible (Case (B)), since we move an innermost rewrite step further up and there is no rewrite step above possible, since before doing this construction it was a valid innermost \mathcal{P} -CT. Since $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is terminating, this construction ends after a finite number of $\Psi(\circ)$ -applications in a tree $\mathfrak{Y}_{y_j}^{(\infty)}$. Now $\mathfrak{Y}_{y_j}^{(\infty)}$ first rewrites below or at position $\pi.\tau$ until it is a normal form. Since all rules in $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ have the form $\ell' \rightarrow \{1 : r'\}$, these rewrite steps are

a single path in $\mathfrak{T}_{y_j}^{(\infty)}$, and it does not matter how long the path is. Furthermore, it does not matter which rewrite strategy we use, since $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is confluent. We will always reach the same normal form at the end of this path. Hence, we can replace the steps with the old ADP and the corresponding innermost rewrite steps in the CT by a step with the new ADP (where the rewriting does not necessarily correspond to the innermost strategy), i.e., we move this non-innermost step directly to the point where the ADP is applied. We will reach the same normal form and can copy the rest of the tree again.

Third Case

Finally, we prove the theorem in the case where $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is NO, $r_j|_{\tau}$ is a ground term, and we have $D_j \xrightarrow{\mathcal{P}, \pi, \tau} \{q_1 : E_1, \dots, q_h : E_h\}$, i.e., it is an innermost step.

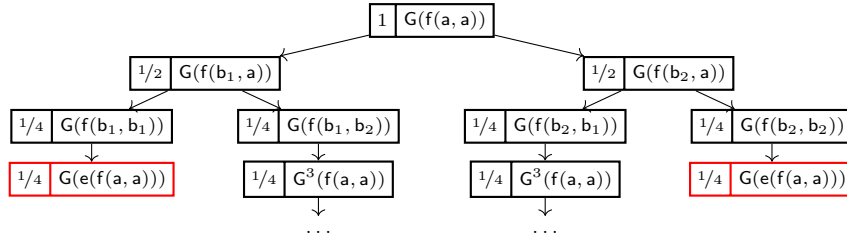
Once again, we use the same idea as in the proof for the second case but the construction of $\Phi(\circ)$ is, again, easier. Note that if $\mathcal{U}_{\mathcal{P}}(r_j|_{\tau})$ is non-overlapping, r_j contains no variable below position τ , and we perform an innermost rewrite step, then this is always an innermost rewrite step in every possible CT and this is the only possible rewrite step at this position. Hence, we can move this innermost step directly after the use of the ADP using the construction $\Psi(\circ)$. Again, here only Case (A) can happen. The reason is that we have to perform this rewrite step eventually if we want to rewrite above position τ , and all other rewrite steps that we can perform in such a situation would be at orthogonal positions. So we get the same normal forms in the leaves with the same probability. \square

Next, we show why the rewriting processor needs the new requirement L that was not imposed in the non-probabilistic setting. More precisely, we give counterexamples for soundness if the used rule is not left-linear, i.e., a variable occurs more than once in the left-hand side, and if the used rule is not right-linear, i.e., a variable occurs more than once in a term on the right-hand side. The other new requirement, namely NE, is currently used in the soundness proof, but we were unable to find a counterexample to soundness if the used rule is not NE. In fact, we conjecture that one can omit this requirement, but then one needs a much more complicated construction and estimation of the resulting termination probability in the soundness proof. The reason is that with only L we can guarantee that performing this rewrite step at a (possibly) non-innermost redex can only increase the probability of innermost termination for the rewritten subterm but not decrease it. Increasing the probability of termination for a proper subterm without any annotations means that we have a higher probability to apply a rewrite step at the position of an annotated symbol. Remember that we have to rewrite redexes with annotated root symbol on each path of the CT infinitely often (the nodes that are labeled P), hence a higher probability of termination of the proper subterm leads to a lower probability of the leaves in the actual CT as the probability to rewrite redexes with annotated root is higher. However, proving this requires a much more involved approximation of the probability for termination than our current proof, where we additionally require NE.

Example 35 (Left-Linearity is Required for Soundness). To see why left-linearity is required for soundness in the probabilistic setting, consider the ADP problem¹⁰ with

$$g(f(x, y)) \rightarrow \{1 : d(G(f(a, a)), G(f(a, a)), G(f(a, a)))\}^{\text{false}} \quad \begin{array}{l} f(x, x) \rightarrow \{1 : e(f(a, a))\}^{\text{true}} \\ a \rightarrow \{1/2 : b_1, 1/2 : b_2\}^{\text{true}} \end{array}$$

This example could also be made non-erasing by adding x as an additional argument to e and by instantiating x, y by all possible values from $\{b_1, b_2\}$ in the g -rule. This ADP problem is not iAST, as it allows for the following CT whose leaves have a probability < 1 .

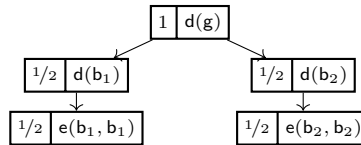


Here, $G^3(f(a, a))$ is an abbreviation for the term $d(G(f(a, a)), G(f(a, a)), G(f(a, a)))$. The paths starting in $G(e(f(a, a)))$ can never use a rewrite step with the g -ADP anymore and therefore, they converge with probability 1 in our CT. So we can rewrite a single G -term to a leaf with a probability of $1/2$ and to three copies of itself with a probability of $1/2$. Hence, this CT corresponds to a random walk that terminates with probability < 1 . But without the restriction to left-linearity, we could apply the rewriting processor and replace the g -ADP by $g(f(x, y)) \rightarrow \{1 : d(G(e(f(a, a))), G(e(f(a, a))), G(e(f(a, a))))\}^{\text{false}}$. Now in every path of the CT this ADP can be used at most once and hence, the resulting ADP problem is iAST, which shows unsoundness of the rewriting processor without left-linearity.

Example 36 (Right-Linearity is Required for Soundness). For right-linearity consider the ADP problem¹¹ \mathcal{P} with

$$f(e(b_1, b_1)) \rightarrow \{1 : h(F(d(g)), F(d(g)), F(d(g)), F(d(g)))\}^{\text{false}} \quad \begin{array}{l} d(x) \rightarrow \{1 : e(x, x)\}^{\text{true}} \\ g \rightarrow \{1/2 : b_1, 1/2 : b_2\}^{\text{true}} \end{array}$$

Note that for the term $d(g)$ we have the following innermost \mathcal{P} -rewrite sequence tree:



¹⁰ Here, we have already applied the usable rules processor to turn the flag of the g -ADP to **false**, and we have moved the f -ADP with F on its right-hand side to another ADP problem via the dependency graph processor.

¹¹ Again, this ADP problem results from an actual PTRS, where we already applied the usable rules processor and the usable terms processor before.

Hence, the term $F(d(g))$ can rewrite to $F(e(b_1, b_1))$ and then to four occurrences of $F(d(g))$ again with a chance of $1/2$, or it reaches a leaf with a chance of $1/2$ as it can never rewrite an annotated redex again. This is once again a random walk that terminates with probability < 1 , so that our ADP problem is not iAST. But without the restriction to right-linearity of the used rule, it would be possible to apply the rewriting processor and replace the f-ADP by $f(e(b_1, b_1)) \rightarrow \{1 : h(F(e(g, g)), F(e(g, g)), F(e(g, g)), F(e(g, g)))\}^{\text{false}}$. The term $e(g, g)$ can now be rewritten to the term $e(b_i, b_j)$ with a probability of $1/4$, whereas one obtains a term of the form $e(b_i, b_j)$ with $i \neq 1$ or $j \neq 1$ with a probability of $3/4$. Hence, now $F(e(g, g))$ can rewrite to a term with four subterms $F(e(g, g))$ only with a probability of $1/4$, or it reaches a leaf with a chance of $3/4$. This is now a random walk that terminates with probability 1 and the same happens for all possible CTs. Hence, the resulting ADP problem is iAST, which shows unsoundness of the rewriting processor without right-linearity.

As mentioned in [Footnote 8](#), the rewriting processor is complete in the non-probabilistic setting. The reason is that there one used a different definition to determine when a DP problem $(\mathcal{P}, \mathcal{R})$ is “not terminating” (in [\[18\]](#), this was called “infinite”). There, a problem is not only considered to be non-terminating if there is an infinite $(\mathcal{P}, \mathcal{R})$ -chain, but also if \mathcal{R} is not terminating. In the future, we will examine whether such a modified definition is also useful in the probabilistic setting. However, as in [\[18\]](#), this modified definition would mean that there can be ADP problems that are both “iAST” and “non iAST”.

B Further ADP Transformation Processors

Now we present three further transformational ADP processors which we adapted to the probabilistic setting, viz., the *instantiation*, *forward instantiation*, and *rule overlap instantiation* processors. The latter is a weaker version of the *narrowing* processor, which is unsound in the probabilistic setting in general, as we will see in [Ex. 42](#).

B.1 Instantiation

In the non-probabilistic setting [\[18\]](#), the idea of the instantiation processor is to consider all possible *predecessors* $s \rightarrow t$ of a dependency pair $u \rightarrow v$ in a chain and to compute the skeleton $\text{Cap}(t)$ of t that remains unchanged when we reduce $t\sigma$ to $u\sigma$, i.e., when going from one DP in a chain to the next. Then $\text{Cap}(t)$ and u must unify with some mgu δ , and $t\sigma \xrightarrow{i}_{\mathcal{R}}^* u\sigma$ implies that σ is an instance of δ . Hence, the instantiation processor replaces the DP $u \rightarrow v$ by $u\delta \rightarrow v\delta$.

As in [\[18\]](#), $\text{Cap}_{\mathcal{P}}(t)$ results from replacing all those subterms of t by different fresh variables whose root is a defined symbol of \mathcal{P} . Here, multiple occurrences of the same subterm are also replaced by pairwise different variables. So if $g \in \mathcal{D}$ and $c \in \mathcal{C}$, then $\text{Cap}_{\mathcal{P}}(c(g(x), g(x))) = c(x_1, x_2)$. There exist several improvements in order to replace this definition of $\text{Cap}_{\mathcal{P}}$ by more precise approximations of the “skeleton”, see, e.g., [\[18, 24, 39\]](#). Moreover, one could also improve $\text{Cap}_{\mathcal{P}}$ by only

regarding the root symbols of left-hand sides of ADPs with the flag $m = \text{true}$ as “defined”.

To adapt the instantiation processor to ADPs, we consider all terms in the distributions on the right-hand sides of all predecessors. Note that in the ordinary DP framework, one only instantiates the DPs, but the rules are left unchanged. Since our ADPs represent both DPs and rules, when instantiating an ADP, we add a copy of the original ADP without any annotations (i.e., this corresponds to the original non-instantiated rule which can now only be used for R -steps). In the following, $\text{vr}(\mathcal{P})$ is a variable-renamed copy of \mathcal{P} where all variables are replaced by fresh ones.

Theorem 37 (Instantiation Processor). *Let \mathcal{P} be an ADP problem with $\mathcal{P} = \mathcal{P}' \uplus \{\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m\}$. Then Proc_i is sound and complete, where $\text{Proc}_i(\mathcal{P}) = \{\mathcal{P}' \cup N \cup \{\ell \rightarrow \{p_1 : b(r_1), \dots, p_k : b(r_k)\}^m\}\}$ with*

$$N = \{\ell\delta \rightarrow \{p_1 : r_1\delta, \dots, p_k : r_k\delta\}^m \mid \begin{array}{l} \ell' \rightarrow \{p'_1 : r'_1, \dots, p'_h : r'_h\}^{m'} \in \text{vr}(\mathcal{P}), 1 \leq j \leq h, \\ t \trianglelefteq_{\#} r'_j, \delta = \text{mgu}(\text{Cap}_{\mathcal{P}}(t^{\#}), \ell^{\#}), \{\ell'\delta, \ell\delta\} \subseteq \text{ANF}_{\mathcal{P}} \end{array} \}$$

Example 38. Consider the PTRS \mathcal{R} with the rules $f(x, y, z) \rightarrow \{1 : g(x, y, z)\}$ and $g(\mathbf{a}, \mathbf{b}, u) \rightarrow \{1/2 : f(u, u, u), 1/2 : g(\mathbf{a}, \mathbf{b}, u)\}$. Its ADPs are $f(x, y, z) \rightarrow \{1 : G(x, y, z)\}^{\text{true}}$ and $g(\mathbf{a}, \mathbf{b}, u) \rightarrow \{1/2 : F(u, u, u), 1/2 : G(\mathbf{a}, \mathbf{b}, u)\}^{\text{true}}$. Using only the processors of Sect. 4, we cannot prove that \mathcal{R} is iAST. However, we can apply the instantiation processor on the f-ADP.

There is a term $t = F(u, u, u)$ in the right-hand side of the g-ADP. As it does not contain defined symbols, we have $\text{Cap}_{\mathcal{R}}(\#_{\varepsilon}(t)) = \text{Cap}_{\mathcal{R}}(t) = t$. For the left-hand side $\ell = f(x, y, z)$, $\delta = \text{mgu}(t, \#_{\varepsilon}(\ell)) = \text{mgu}(t, F(x, y, z))$ instantiates x , y , and z by u . So the instantiation processor replace the f-ADP by $f(u, u, u) \rightarrow \{1 : G(u, u, u)\}^{\text{true}}$ (and moreover, we add $f(x, y, z) \rightarrow \{1 : g(x, y, z)\}^{\text{true}}$). We can now remove the annotation in the transformed f-ADP by the dependency graph processor, and afterwards remove the annotation in the g-ADP by applying the reduction pair processor with the polynomial interpretation that maps every function symbol to the constant 0 except for G that is mapped to 1. As we removed all annotations, $\mathcal{DP}(\mathcal{R})$, and hence \mathcal{R} must be iAST.

Before we can prove soundness and completeness of the new instantiation processor, we start with a corollary that expresses the essential property of the $\text{Cap}_{\mathcal{P}}$ -function.

Corollary 39 (Property of $\text{Cap}_{\mathcal{P}}$). *Let $t, u \in \mathcal{T}(\Sigma^{\#}, \mathcal{V})$. If $t\sigma \xrightarrow{i}_{\text{np}(\mathcal{P})}^* u$ for some substitution σ , then $u = \text{Cap}_{\mathcal{P}}(t)\delta$ for some substitution δ which only differs from σ on the fresh variables that are introduced by $\text{Cap}_{\mathcal{P}}$.*

Proof. Let $t\sigma \xrightarrow{i}_{\text{np}(\mathcal{P}), \rho_1} u_1 \xrightarrow{i}_{\text{np}(\mathcal{P}), \rho_2} \dots \xrightarrow{i}_{\text{np}(\mathcal{P}), \rho_n} u_n = u$. Let $\{\pi_1, \dots, \pi_m\}$ be the set of positions where $\text{Cap}_{\mathcal{P}}$ replaces the subterms of t by corresponding fresh variables x_1, \dots, x_m . By the definition of $\text{Cap}_{\mathcal{P}}$ for each ρ_i there is a higher position $\pi_j \leq \rho_i$. Hence, u can at most differ from $t\sigma$ on positions below a π_j . We define δ to be like σ but on the fresh variables x_1, \dots, x_m we define $\delta(x_j) = u|_{\pi_j}$. Then by construction $\text{Cap}_{\mathcal{P}}(t)\delta = u$ and δ and σ differ only on the fresh variables. \square

We now prove [Thm. 37](#), i.e., we prove soundness and completeness of the instantiation processor.

Proof. We will use the following two observations. As in the proof of [Thm. 23](#), let $\overline{\mathcal{P}}' = \mathcal{P}' \cup N \cup \{\ell \rightarrow \{p_1 : \flat(r_1), \dots, p_k : \flat(r_k)\}^m\}$ and $\overline{\mathcal{P}} = \overline{\mathcal{P}}' \cup \{\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m\}$. First, note that $\text{ANF}_{\mathcal{P}} = \text{ANF}_{\overline{\mathcal{P}}} = \text{ANF}_{\overline{\mathcal{P}}'}$, since the left-hand sides in $\overline{\mathcal{P}}$ and $\overline{\mathcal{P}}'$ are either already from rules in \mathcal{P} or instantiated left-hand sides from \mathcal{P} . So it suffices to consider only $\text{ANF}_{\mathcal{P}}$. Second, assume that there exists a $\overline{\mathcal{P}}$ -CT \mathfrak{T} that converges with probability < 1 whose root is labeled with $(1 : t)$ and $\flat(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \overline{\mathcal{P}}$, and $\text{pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$. If in addition, the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ may only be used at the root, then we know that not only $\overline{\mathcal{P}}$ is not iAST but also that $\overline{\mathcal{P}}'$ is not iAST. The reason is that if only the root x uses the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ then all the subtrees starting at one of its direct successors $xE = \{y_1, \dots, y_k\}$ are $\overline{\mathcal{P}}'$ -CTs. Furthermore, since \mathfrak{T} converges with probability < 1 , there must be at least one subtree $\mathfrak{T}[y_i E^*]$ that starts at the node y_i for some $1 \leq i \leq k$ and also converges with probability < 1 .

Soundness: Let \mathcal{P} be not iAST. Then by [Lemma 34](#) there exists a \mathcal{P} -CT $\mathfrak{T} = (V, E, L, P)$ that converges with probability < 1 whose root is labeled with $(1 : t)$ and $\flat(t) = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \mathcal{P}$, and $\text{pos}_{\mathcal{D}^\#}(t) = \{\varepsilon\}$. We will now create a $\overline{\mathcal{P}}$ -CT $\mathfrak{T}' = (V, E, L', P)$, with the same underlying tree structure, and an adjusted labeling such that $p_x^{\mathfrak{T}} = p_x^{\mathfrak{T}'}$ for all $x \in V$. Furthermore, we will at most use the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ at the root. Since the tree structure and the probabilities are the same, we then get $|\mathfrak{T}'|_{\text{Leaf}} = |\mathfrak{T}|_{\text{Leaf}} < 1$, and hence, using our previous discussion, $\overline{\mathcal{P}}'$ is not iAST either.

The core idea is that every rewrite step with $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ at a node v that is not the root can also be done with a rule from N , or we can use $\ell \rightarrow \{p_1 : \flat(r_1), \dots, p_k : \flat(r_k)\}^m$ if the annotations do not matter, e.g., we rewrite at a position that is not annotated. We construct the new labeling L' for the $\overline{\mathcal{P}}$ -CT \mathfrak{T}' inductively such that for all nodes $x \in V \setminus \text{Leaf}$ with children nodes $xE = \{y_1, \dots, y_h\}$ we have $t'_x \xrightarrow{i}_{\overline{\mathcal{P}}} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_h}}{p_x} : t'_{y_h}\}$ and for all non-root nodes in P we even have $t'_x \xrightarrow{i}_{\overline{\mathcal{P}}'} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_h}}{p_x} : t'_{y_h}\}$. Let $X \subseteq V$ be the set of nodes x where we have already defined the labeling $L'(x)$. During our construction, we ensure that the following property holds:

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

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, every annotation in t_x also exists in t'_x . The second condition ensures that if we rewrite using Case **(p)** or **(pr)** of [Def. 11](#) in \mathfrak{T} , we do the same in \mathfrak{T}' , i.e., the corresponding node x remains in P in \mathfrak{T}' . We label the root of \mathfrak{T}' by $(1 : t)$. Here, [\(26\)](#) obviously holds. 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 $xE = \{y_1, \dots, y_h\}$ be the set of its successors. We need to define the corresponding terms $t'_{y_1}, \dots, t'_{y_m}$ for the nodes y_1, \dots, y_h . Since x is not a leaf and \mathfrak{T} is a \mathcal{P} -CT, we have $t_x \xrightarrow{i}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_h}}{p_x} : t_{y_h}\}$. We have the following three different cases:

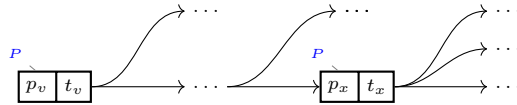
(A) We have $t_x \xrightarrow{i}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_h}}{p_x} : t_{y_h}\}$ with an ADP $\ell' \rightarrow \{p_1 : r'_1, \dots, p_h : r'_h\}^{m'}$ that is either different to $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ or the node x is the root, using the position π , and a substitution σ such that $b(t_x|_{\pi}) = \ell\sigma \in \text{ANF}_{\mathcal{P}}$. Then, $t_x \stackrel{\dagger}{(IH)} t'_x$, and hence also $b(t'_x|_{\pi}) = \ell\sigma \in \text{ANF}_{\mathcal{P}}$. Thus, we can rewrite the term t'_x using the same ADP, the same position and the same substitution. This means that we have $t'_x \xrightarrow{i}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_h}}{p_x} : t'_{y_h}\}$. Let $1 \leq j \leq h$. If $\pi \in \text{pos}_{\mathcal{D}\#}(t_x)$, then also $\pi \in \text{pos}_{\mathcal{D}\#}(t'_x)$ by (26). Whenever we create annotations in the rewrite step in \mathfrak{T} (a step with **(p)** or **(pr)**), then we do the same in \mathfrak{T}' (the step is also a **(p)**- or **(pr)**-step, respectively), and whenever we remove annotations in the rewrite step in \mathfrak{T} (a step with **(r)** or **(irr)**), then we do the same in \mathfrak{T}' (the step is also either a **(r)**- or **(irr)**-step). Therefore, we also get $\text{pos}_{\mathcal{D}\#}(t_{y_j}) \subseteq \text{pos}_{\mathcal{D}\#}(t'_{y_j})$ for all $1 \leq j \leq k$ and (26) is again satisfied.

(B) We have $t_x \xrightarrow{i}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$ using the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$, the position π , and a substitution σ such that $b(t_x|_{\pi}) = \ell\sigma \in \text{ANF}_{\mathcal{P}}$, and $\pi \notin \text{pos}_{\mathcal{D}\#}(t_x)$. Since $t_x \stackrel{\dagger}{(IH)} t'_x$, we can rewrite the term t'_x using the ADP $\ell \rightarrow \{p_1 : b(r_1), \dots, p_k : b(r_k)\}^m$, the same position, and the same substitution. This means that we have $t'_x \xrightarrow{i}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t'_{y_k}\}$ and (26) is again satisfied. In order to prove this, one can do a similar analysis as above. Note that only cases **(irr)** and **(r)** can be applied in \mathfrak{T} (since $\pi \notin \text{pos}_{\mathcal{D}\#}(t_x)$), and we would remove the annotations of the terms r_j anyway in those cases.

(C) Finally, we have $t_x \xrightarrow{i}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$ using the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$, the position π , and a substitution σ such that $b(t_x|_{\pi}) = \ell\sigma \in \text{ANF}_{\mathcal{P}}$, $\pi \in \text{pos}_{\mathcal{D}\#}(t_x)$, and x is not the root. Then $t_{y_j} = b_{\pi}^{\uparrow}(t_x[r_j\sigma]_{\pi})$ if $m = \text{false}$ and $t_{y_j} = t_x[r_j\sigma]_{\pi}$, otherwise.

We now look at the (not necessarily direct) predecessor v of x that is in P , where an ADP is applied on a position above or equal to π , and where in the path from v to x , no ADP is applied on a position on or above π . There is always such a node v . The reason is that x is not the first node in P and by the Starting Lemma (Lemma 34) we can assume that a step at the root of the term takes place at the root of the CT, i.e., t_x results from right-hand sides of \mathcal{P} . Furthermore, we only use rules with the flag $m = \text{true}$ as otherwise, the position π would not be annotated in t_x . We show that the path from v to x can also be taken when using one of the new instantiations of $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ instead.

Let this predecessor $v \in P$ use the ADP $\ell' \rightarrow \{p'_1 : r'_1, \dots, p'_h : r'_h\}^{m'}$, the position π' , and the substitution σ' . Furthermore, let τ be the position such that $\pi'.\tau = \pi$. Because we never rewrite at a position above π before reaching node x , we have $\#_{\varepsilon}(r'_j|_{\tau})\sigma' = \#_{\varepsilon}(t_v|_{\pi}) \xrightarrow{i}_{\text{np}(\mathcal{P})} \#_{\varepsilon}(t_x|_{\pi}) = \#_{\varepsilon}(\ell)\sigma$.



From $\#_{\varepsilon}(r'_j|_{\tau})\sigma' \xrightarrow{i}_{\text{np}(\mathcal{P})} \#_{\varepsilon}(\ell)\sigma$, Cor. 39 implies $\#_{\varepsilon}(\ell)\sigma = \text{Cap}_{\mathcal{P}}(\#_{\varepsilon}(r'_j|_{\tau}))\delta$ for some substitution δ that differs from σ' at most on the variables that are introduced by $\text{Cap}_{\mathcal{P}}$. W.l.o.g. we can assume that σ' is equal to δ on all these fresh variables, and since the ADPs are variable-renamed, we can also assume that σ is equal to δ on all

the fresh variables and all the variables from ℓ' . Hence, $\#_\varepsilon(\ell)\delta = \text{Cap}_{\mathcal{P}}(\#_\varepsilon(r'_j|\tau))\delta$ shows there is an mgu γ of $\#_\varepsilon(\ell)$ and $\text{Cap}_{\mathcal{P}}(\#_\varepsilon(r'_j|\tau))$ with $\sigma = \gamma\zeta$ for some substitution ζ . Moreover, the property $\{\ell'\sigma', \ell\sigma\} \subseteq \text{ANF}_{\mathcal{P}}$ must remain true when replacing σ and σ' by the more general substitution γ , i.e., $\{\ell'\gamma, \ell\gamma\} \subseteq \text{ANF}_{\mathcal{P}}$. Hence, we can apply the new ADP $\ell\gamma \rightarrow \{p_1 : r_1\gamma, \dots, p_k : r_k\gamma\}^m \in N$ with the position π and the substitution ζ . This means that we have $t'_x \xrightarrow{\zeta}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t'_{y_k}\}$ with $t'_{y_j} = t'_x[r_j\gamma\zeta]_\pi$ if $m = \text{true}$, or $t'_{y_j} = b_\pi^\dagger(t'_x[r_j\gamma\zeta]_\pi)$, otherwise. Since, $\sigma = \gamma\zeta$ we directly get $t_{y_j} \doteq t'_{y_j}$ and $\text{pos}_{\mathcal{D}\#}(t_{y_j}) \subseteq \text{pos}_{\mathcal{D}\#}(t'_{y_j})$ so that (26) is satisfied again, and this ends the proof.

Completeness: Let $\overline{\mathcal{P}'}$ be not iAST. Then there exists a $\overline{\mathcal{P}'}$ -CT $\mathfrak{T} = (V, E, L, P)$ that converges with probability < 1 . We will now create a \mathcal{P} -CT $\mathfrak{T}' = (V, E, L', P)$, 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 $|\mathfrak{T}'|_{\text{Leaf}} = |\mathfrak{T}|_{\text{Leaf}} < 1$, and thus \mathcal{P} is not iAST either.

The core idea of this construction is that every rewrite step with an ADP from N or the ADP $\ell \rightarrow \{p_1 : b(r_1), \dots, p_k : b(r_k)\}^m$ is also possible with the more general ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ that may also contain more annotations. We construct the new labeling L' for the \mathcal{P} -CT \mathfrak{T}' inductively such that for all inner nodes $x \in V \setminus \text{Leaf}$ with children nodes $xE = \{y_1, \dots, y_m\}$ we have $t'_x \xrightarrow{\zeta}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_m}}{p_x} : t'_{y_m}\}$. Let $X \subseteq V$ be the set of nodes x where we have already defined the labeling $L'(x)$. During our construction, we ensure that the following property holds (analogous to the soundness construction):

$$\text{For every node } x \in X \text{ we have } t_x \doteq t'_x \text{ and } \text{pos}_{\mathcal{D}\#}(t_x) \subseteq \text{pos}_{\mathcal{D}\#}(t'_x). \quad (27)$$

We label the root of \mathfrak{T}' exactly as the root of \mathfrak{T} . Here, (27) obviously holds. 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 $xE = \{y_1, \dots, y_h\}$ be the set of its successors. We need to define the corresponding terms $t'_{y_1}, \dots, t'_{y_h}$ for the nodes y_1, \dots, y_h . Since x is not a leaf and \mathfrak{T} is a $\overline{\mathcal{P}'}$ -CT, we have $t_x \xrightarrow{\zeta}_{\overline{\mathcal{P}'}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_h}}{p_x} : t_{y_h}\}$. We have the following three different cases:

- (A) If it is a step with $\xrightarrow{\zeta}_{\mathcal{P}}$ using an ADP from \mathcal{P}' in \mathfrak{T} , then we perform a rewrite step with the same ADP, the same position, and the same substitution in \mathfrak{T}' . This is analogous to Case (A) of the soundness proof.
- (B) If it is a step with $\xrightarrow{\zeta}_{\mathcal{P}}$ using the ADP $\ell \rightarrow \{p_1 : b(r_1), \dots, p_k : b(r_k)\}^m$ in \mathfrak{T} , then we use the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ that contain more annotations in \mathfrak{T}' . Since we use the same rule but with more annotations, we end up with $t_{y_j} \doteq t'_{y_j}$ and $\text{pos}_{\mathcal{D}\#}(t_{y_j}) \subseteq \text{pos}_{\mathcal{D}\#}(t'_{y_j})$ again.
- (C) If it is a step with $\xrightarrow{\zeta}_{\mathcal{P}}$ using an ADP from N in \mathfrak{T} , then we use the more general ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ in \mathfrak{T}' . For this rewrite step, we use the substitution δ such that $\gamma\delta = \sigma$.

□

B.2 Forward Instantiation

Next we adapt the *forward instantiation* processor and prove its soundness and completeness. In the non-probabilistic setting, the idea of the forward instantiation

processor [18] is to consider all possible *successors* $u \rightarrow v$ of a DP $s \rightarrow t$ in a chain, again, in order to find the skeleton that remains unchanged when rewriting $t\sigma$ to $u\sigma$. To find this skeleton, we reverse the rules of the TRS and then proceed as for the instantiation processor. Moreover, we can restrict ourselves to the (reversed) usable rules of t . Note that these reversed rules might violate the variable conditions of TRSs, i.e., the right-hand side of a rule may contain variables that do not occur in the left-hand side or the left-hand side may be a variable.

Theorem 40 (Forward Instantiation Processor). *Let \mathcal{P} be an ADP problem with $\mathcal{P} = \mathcal{P}' \uplus \{\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m\}$. Then $\text{Proc}_{\mathbf{f}}$ is sound and complete, where $\text{Proc}_{\mathbf{f}} = \{\mathcal{P}' \cup N \cup \{\ell \rightarrow \{p_1 : \mathfrak{b}(r_1), \dots, p_k : \mathfrak{b}(r_k)\}^m\}\}$. Here,*

$$N = \left\{ \begin{array}{l} \ell\delta \rightarrow \{p_1 : r_1\delta, \dots, p_k : r_k\delta\}^m \\ \left. \begin{array}{l} \ell' \rightarrow \{p'_1 : r'_1, \dots, p'_h : r'_h\}^{m'} \in \text{vr}(\mathcal{P}), 1 \leq j \leq k, t \triangleleft_{\#} r_j, \\ \mathcal{Q} = (\text{np}(\mathcal{U}_{\mathcal{P}}(t^{\#})))^{-1}, \\ \delta = \text{mgu}(t^{\#}, \text{Cap}_{\mathcal{Q}}(\ell'^{\#})), \{\ell\delta, \ell'\delta\} \subseteq \text{ANF}_{\mathcal{P}} \end{array} \right\} \end{array} \right.$$

Example 41. Consider the PTRS \mathcal{P} with the rules

$$\begin{array}{ll} f(x) \rightarrow \{1/2 : \mathfrak{g}(x), 1/2 : \mathfrak{h}(x)\} & \mathfrak{q}(\mathfrak{a}) \rightarrow \{1 : \mathfrak{a}\} \\ \mathfrak{g}(\mathfrak{a}) \rightarrow \{1 : \mathfrak{f}(\mathfrak{q}(\mathfrak{a}))\} & \mathfrak{q}(\mathfrak{b}) \rightarrow \{1 : \mathfrak{b}\} \\ \mathfrak{h}(\mathfrak{b}) \rightarrow \{1 : \mathfrak{f}(\mathfrak{q}(\mathfrak{b}))\} & \end{array}$$

After the usable rules and the usable terms processor, we obtain the ADPs

$$\begin{array}{ll} f(x) \rightarrow \{1/2 : \mathfrak{G}(x), 1/2 : \mathfrak{H}(x)\}^{\text{false}} & \mathfrak{q}(\mathfrak{a}) \rightarrow \{1 : \mathfrak{a}\}^{\text{true}} \\ \mathfrak{g}(\mathfrak{a}) \rightarrow \{1 : \mathfrak{F}(\mathfrak{q}(\mathfrak{a}))\}^{\text{false}} & \mathfrak{q}(\mathfrak{b}) \rightarrow \{1 : \mathfrak{b}\}^{\text{true}} \\ \mathfrak{h}(\mathfrak{b}) \rightarrow \{1 : \mathfrak{F}(\mathfrak{q}(\mathfrak{b}))\}^{\text{false}} & \end{array}$$

In this case, the instantiation processor is useless because both, the $\text{mgu}(\text{Cap}_{\mathcal{P}}(\mathfrak{F}(\mathfrak{q}(\mathfrak{a})))) = \mathfrak{F}(y)$ and $\mathfrak{F}(x)$, and the mgu of $\text{Cap}_{\mathcal{P}}(\mathfrak{F}(\mathfrak{q}(\mathfrak{b}))) = \mathfrak{F}(z)$ and $\mathfrak{F}(x)$, do not modify $\mathfrak{F}(x)$. However, the forward instantiation processor can replace the original f-ADP with $\mathfrak{f}(\mathfrak{a}) \rightarrow \{1/2 : \mathfrak{G}(\mathfrak{a}), 1/2 : \mathfrak{H}(\mathfrak{a})\}^{\text{false}}$ and $\mathfrak{f}(\mathfrak{b}) \rightarrow \{1/2 : \mathfrak{G}(\mathfrak{b}), 1/2 : \mathfrak{H}(\mathfrak{b})\}^{\text{false}}$. We can now remove the annotations of the normal forms $\mathfrak{G}(\mathfrak{b})$ and $\mathfrak{H}(\mathfrak{a})$ with the usable terms processor and apply the reduction pair processor with the polynomial interpretation that maps every function symbol to the constant 1, in order to remove all annotations of both new f-ADPs. Finally, we can remove all remaining annotations and prove iAST using the dependency graph processor. Note that without the forward instantiation processor, we would have to find a polynomial interpretation that is at least linear.

Ex. 41 shows that the processors that instantiate a given ADP are not only needed in some cases for a successful innermost almost-sure termination proof (as in **Ex. 38**), but sometimes they can also ease the search for polynomials in the reduction pair processor, the most time-consuming part of the ADP framework.

We now prove **Thm. 40**, i.e., soundness and completeness of **Thm. 40**. The construction in the proof is very similar to the one for the proof of **Thm. 37**.

Proof. As in the proof of **Thm. 37**, let $\overline{\mathcal{P}'} = \mathcal{P}' \cup N \cup \{\ell \rightarrow \{p_1 : \mathfrak{b}(r_1), \dots, p_k : \mathfrak{b}(r_k)\}^m\}$ and $\overline{\mathcal{P}} = \overline{\mathcal{P}'} \cup \{\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m\}$.

B.3 Narrowing

The narrowing processor [2, 18] can only be used in a weaker version in the probabilistic setting. Let $\mathcal{P} = \mathcal{P}' \uplus \{\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m\}$ be an ADP problem. For each $1 \leq j \leq k$ and each $t \leq_{\#} r_j$, we define its *narrowing substitutions* and its *narrowing results* (as in [37], where narrowing was adapted to dependency tuples for complexity analysis of ordinary term rewriting). If we have to perform rewrite steps on (an instance of) t in order to reach the next ADP usage at an annotated position, then the idea of the narrowing processor is to perform the first step of this reduction already on the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$. So whenever there is a $t \leq_{\#} r_j$ and a position τ in t such that $t|_{\tau}$ unifies with the left-hand side ℓ' of some rule $\ell' \rightarrow \{p_1 : r'_1, \dots, p_k : r'_k\}^{m'} \in \text{vr}(\mathcal{P})$ using an mgu δ such that $\ell\delta, \ell'\delta \in \text{ANF}_{\mathcal{P}}$, then δ is a *narrowing substitution* of t . While the corresponding *narrowing result* could also be defined for probabilistic rules, to simplify the presentation let us assume for the moment that the rule just has the form $\ell' \rightarrow \{1 : r'\}$. Then the corresponding *narrowing result* is $s = t[r']_{\tau}\delta$ if we rewrite at an annotated position of t , and $s = t[b(r')]_{\tau}\delta$ otherwise.

If $\delta_1, \dots, \delta_d$ are all narrowing substitutions of t with the corresponding narrowing results s_1, \dots, s_d , then one would like to define a narrowing processor that replaces $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ by $\ell\delta_e \rightarrow \{p_1 : r_1\delta_e, \dots, p_j : s_e, \dots, p_k : r_k\delta_e\}$, for all $1 \leq e \leq d$.

In addition, there could be another subterm $t' \leq_{\#} r_j$ (with $t' \neq t$) which was involved in a CT (i.e., $t' \# \sigma \xrightarrow{\text{np}(\mathcal{P})}^* \ell' \sigma'$ for some substitutions σ, σ'), but this CT is no longer possible when instantiating t' to $t'\delta_1, \dots, t'\delta_d$. We say that t' is *captured* by $\delta_1, \dots, \delta_d$ if for each narrowing substitution ρ of t' , there is a δ_e with $1 \leq e \leq d$ such that δ_e is more general than ρ (i.e., $\rho = \delta_e \rho'$ for some substitution ρ'). So the narrowing processor has to add another ADP $\ell \rightarrow \{p_1 : \#_{\text{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \#_{\text{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m$, where $\text{capt}_i(\delta_1, \dots, \delta_d)$ contains all positions of subterms $t' \leq_{\#} r_i$ which are not captured by the narrowing substitutions $\delta_1, \dots, \delta_d$ of t . (Therefore, in contrast to instantiation and forward instantiation, here we do not have to add another copy of the original rule without annotations.)

However, the main idea of the narrowing processor, i.e., performing the first rewrite step directly on the ADPs, is unsound for probabilistic ADP problems, as shown by the following example.

Example 42. Consider the PTRS \mathcal{R} with the rules

$$\begin{array}{ll} f(\mathbf{b}_1, \mathbf{d}_1) \rightarrow \{1 : f(\mathbf{a}, \mathbf{e})\} & \mathbf{e} \rightarrow \{1 : \mathbf{d}_1\} \\ f(\mathbf{b}_2, \mathbf{d}_2) \rightarrow \{1 : f(\mathbf{a}, \mathbf{e})\} & \mathbf{e} \rightarrow \{1 : \mathbf{d}_2\} \\ \mathbf{a} \rightarrow \{^{1/2} : \mathbf{b}_1, ^{1/2} : \mathbf{b}_2\} & \end{array}$$

This PTRS is not iAST. The usable rules and the usable terms processor transform the initial ADP problem into \mathcal{P} with

$$\begin{array}{ll} f(\mathbf{b}_1, \mathbf{d}_1) \rightarrow \{1 : F(\mathbf{a}, \mathbf{e})\}^{\text{false}} & \mathbf{e} \rightarrow \{1 : \mathbf{d}_1\}^{\text{true}} \\ f(\mathbf{b}_2, \mathbf{d}_2) \rightarrow \{1 : F(\mathbf{a}, \mathbf{e})\}^{\text{false}} & \mathbf{e} \rightarrow \{1 : \mathbf{d}_2\}^{\text{true}} \\ \mathbf{a} \rightarrow \{^{1/2} : \mathbf{b}_1, ^{1/2} : \mathbf{b}_2\}^{\text{true}} & \end{array}$$

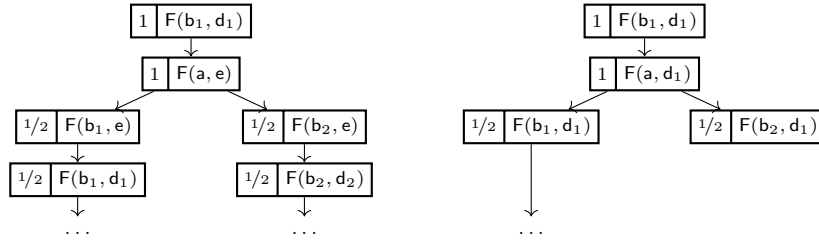
Indeed, this ADP problem allows for the CT below on the left without any leaves.

Note that all occurring terms are ground terms, hence all narrowing substitutions are just the identity function.

But if we apply the narrowing processor to the ADPs in order to rewrite e , then we obtain the four new ADPs

$$\begin{aligned} f(b_1, d_1) &\rightarrow \{1 : F(a, d_1)\}^{\text{false}} & f(b_2, d_2) &\rightarrow \{1 : F(a, d_1)\}^{\text{false}} \\ f(b_1, d_1) &\rightarrow \{1 : F(a, d_2)\}^{\text{false}} & f(b_2, d_2) &\rightarrow \{1 : F(a, d_2)\}^{\text{false}} \end{aligned}$$

This new ADP problem is iAST, as we will reach a normal form with a probability of $1/2$ after each application of an ADP. For example, if we use the first ADP, then we get the CT below on the right. There we reach the normal form $F(b_2, d_1)$ with probability $1/2$.



The difference is that when narrowing the ADP, we have to decide how to rewrite e before we split the term into different ones with a certain probability, i.e., before we rewrite a to $\{1/2 : b_1, 1/2 : b_2\}$.

Thus, in the probabilistic setting, we can only transform the ADP by applying a rewrite rule if we can ensure the same conditions as for the rewriting processor. So in the probabilistic setting, the narrowing processor can only instantiate the ADP by the narrowing substitutions, but it must not perform any rewrite step. Instead, the rewrite steps have to be done via the rewriting processor afterwards. Thus, instead of calling it *narrowing processor*, we call it *rule overlap instantiation processor* as it only instantiates the ADPs but does not perform any rewrite steps.

Theorem 43 (Rule Overlap Instantiation Processor). *Let \mathcal{P} be an ADP problem with $\mathcal{P} = \mathcal{P}' \uplus \{\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m\}$, let $1 \leq j \leq k$, and let $t \trianglelefteq_{\#} r_j$. Let $\delta_1, \dots, \delta_d$ be all narrowing substitutions of t , where $d \geq 0$. Then $\text{Proc}_{\text{roi}} = \{\mathcal{P}' \cup N\}$ is sound and complete, where*

$$\begin{aligned} N = & \left\{ \ell \delta_e \rightarrow \{p_1 : r_1 \delta_e, \dots, p_k : r_k \delta_e\} \mid 1 \leq e \leq d \right\} \\ & \cup \left\{ \ell \rightarrow \{p_1 : \#_{\text{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \#_{\text{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m \right\} \end{aligned}$$

Example 44. Consider \mathcal{R} with

$$\begin{aligned} f(d(x)) &\rightarrow \{3/4 : e(f(g(x)), f(h(x))), 1/4 : a\} & g(a) &\rightarrow \{1 : d(a)\} \\ & & h(b) &\rightarrow \{1 : d(b)\} \end{aligned}$$

The usable rules and the usable terms processor transform the initial ADP problem into \mathcal{P} with

$$\begin{aligned} f(d(x)) &\rightarrow \{3/4 : e(F(g(x)), F(h(x))), 1/4 : a\}^{\text{false}} & g(a) &\rightarrow \{1 : d(a)\}^{\text{true}} \\ & & h(b) &\rightarrow \{1 : d(b)\}^{\text{true}} \end{aligned}$$

The ADP problem \mathcal{P} (and thus also the original PTRS \mathcal{R}) is iAST because for every instantiation, at most one of the two “recursive F-calls” in the right-hand side of the f-ADP can be applied. The reason is that we can either use the g-rule if the variable x is instantiated with \mathbf{a} , or we can apply the h-rule if the variable is instantiated with \mathbf{b} , but not both. We apply Proc_{roi} using the term $F(\mathbf{g}(x))$, whose only narrowing substitution is $\delta = \{x/\mathbf{a}\}$. The other subterm $F(\mathbf{h}(x))$ with annotated root is not captured by this substitution, and hence, we have to generate an additional ADP where this second subterm is annotated. Thus, we replace the former f-ADP by the following two new ADPs.

$$\begin{aligned} f(\mathbf{d}(\mathbf{a})) &\rightarrow \{3/4 : \mathbf{e}(F(\mathbf{g}(\mathbf{a})), F(\mathbf{h}(\mathbf{a}))), 1/4 : \mathbf{a}\}^{\text{false}} \\ f(\mathbf{d}(x)) &\rightarrow \{3/4 : \mathbf{e}(f(\mathbf{g}(x)), F(\mathbf{h}(x))), 1/4 : \mathbf{a}\}^{\text{false}} \end{aligned}$$

Now one can remove the annotation of $F(\mathbf{h}(\mathbf{a}))$ from the first ADP by the usable terms processor and then apply the reduction pair processor with the polynomial interpretation that maps F to 1 and all other symbols to 0 to remove all annotations, which proves iAST. Again, proving iAST with such a simple polynomial interpretation would not be possible without the rule overlap instantiation processor.

We now prove [Thm. 43](#), i.e., soundness and completeness of the rule overlap instantiation processor.

Proof.

Soundness: We use the same idea as in the proof of soundness for [Thm. 40](#) but at a node v that uses the position $\pi \in \text{pos}_{\mathcal{D}\#}(t_x)$ we do not look at the next node that rewrites at an annotated position below or equal to π , but at all such nodes that can be either annotated or not.

Let \mathcal{P} be not iAST. Then by [Lemma 34](#) there exists a \mathcal{P} -CT $\mathfrak{T} = (V, E, L, P)$ that converges with probability < 1 that starts with $(1 : t)$ and $t = s\theta \in \text{ANF}_{\mathcal{P}}$ for a substitution θ and an ADP $s \rightarrow \dots \in \mathcal{P}$, and $\text{pos}_{\mathcal{D}\#}(t) = \{\varepsilon\}$. Let $\overline{\mathcal{P}'} = \mathcal{P}' \cup N$. We will now create a $\overline{\mathcal{P}'}$ -CT $\mathfrak{T}' = (V, E, L', P)$, 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 $|\mathfrak{T}'|_{\text{Leaf}} = |\mathfrak{T}|_{\text{Leaf}} < 1$, and hence $\overline{\mathcal{P}'}$ is not iAST either.

The core idea of this construction is that every rewrite step with $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ can also be done with a rule from N . If we use $\ell \rightarrow \{p_1 : \#_{\text{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \#_{\text{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m \in N$, we may create fewer annotations than we did when using the old ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$. However, we will never rewrite at the position of the annotations that do not get created in the CT \mathfrak{T} , hence we can ignore them. We construct the new labeling L' for the $\overline{\mathcal{P}'}$ -CT \mathfrak{T}' inductively such that for all nodes $x \in V \setminus \text{Leaf}$ with $xE = \{y_1, \dots, y_m\}$ we have $t'_x \xrightarrow{i}_{\overline{\mathcal{P}'}} \{ \frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_m}}{p_x} : t'_{y_m} \}$. Let $X \subseteq V$ be the set of nodes x where we have already defined the labeling $L'(x)$. During our construction, we ensure that the following property holds:

$$\text{For every } x \in X \text{ we have } t_x \doteq t'_x \text{ and } \text{pos}_{\mathcal{D}\#}(t_x) \setminus \text{Junk}(t_x) \subseteq \text{pos}_{\mathcal{D}\#}(t'_x). \quad (28)$$

Here, we define $\text{Junk}(t_x)$ as in the proof of [Thm. 16](#).

For the construction, we start with the same term at the root. Here, (28) obviously holds. 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 $xE = \{y_1, \dots, y_m\}$ be the set of its successors. We need to define the corresponding sets $t'_{y_1}, \dots, t'_{y_m}$ for the nodes y_1, \dots, y_m . Since x is not a leaf and \mathfrak{T} is a \mathcal{P} -CT, we have $t_x \xrightarrow{i}_{\mathcal{P}} \left\{ \frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_m}}{p_x} : t_{y_m} \right\}$. We have the following three cases:

- (A) If it is a step with $\xrightarrow{i}_{\mathcal{P}}$ using an ADP that is different from $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ in \mathfrak{T} , then we perform a rewrite step with the same ADP, the same redex, and the same substitution in \mathfrak{T}' . Analogous to Case (A) of the soundness proof for [Thm. 37](#), we can show (28) for the resulting terms.
- (B) If it is a step with $\xrightarrow{i}_{\mathcal{P}}$ using the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ at a position $\pi \notin \text{pos}_{\mathcal{D}\#}(t_x)$ in \mathfrak{T} , then we perform a rewrite step with $\ell \rightarrow \{p_1 : \#_{\text{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \#_{\text{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m$, same redex, same substitution, and same position in \mathfrak{T}' . Analogous to Case (B) of the soundness proof for [Thm. 37](#), we can show (28) for the resulting terms. Note that the rule that we use contains fewer annotations than the original rule, but since $\pi \notin \text{pos}_{\mathcal{D}\#}(t_x)$, we remove all annotations from the rule during the application of the rewrite step anyway.
- (C) If it is a step with $\xrightarrow{i}_{\mathcal{P}}$ using the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ at a position $\pi \in \text{pos}_{\mathcal{D}\#}(t_x)$ in \mathfrak{T} , then we look at specific successors to find a substitution δ such that $\ell\delta \rightarrow \{p_1 : r_1\delta, \dots, p_k : r_k\delta\}^m \in N$ or we detect that we can use the ADP $\ell \rightarrow \{p_1 : \#_{\text{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \#_{\text{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m$ and perform a rewrite step with this new ADP in \mathfrak{T}' .

So it remains to consider Case (C) in detail. Here, we have $t_x \xrightarrow{i}_{\mathcal{P}} \left\{ \frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_k}}{p_x} : t_{y_k} \right\}$ using the ADP $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$, the position $\pi \in \text{pos}_{\mathcal{D}\#}(t_x)$, and a substitution σ such that $\text{b}(t_x|_{\pi}) = \ell\sigma \in \text{ANF}_{\mathcal{P}}$.

We first consider the case where there is no successor v of x where an ADP is applied at an annotated position below or at π , or an ADP is applied on a position above π before reaching such a node v . Then, we can use $\ell \rightarrow \{p_1 : \#_{\text{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \#_{\text{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m$ instead, because the annotations will never be used, so they do not matter.

Otherwise, there exists a successor v of x where an ADP is applied at an annotated position below or at π , and no ADP is applied on a position above π before. Let v_1, \dots, v_n be all (not necessarily direct) successors that rewrite below position π , or rewrite at position π , and on the path from x to v there is no other node with this property, and no node that performs a rewrite step above π . Furthermore, let t_1, \dots, t_n be the used redexes and ρ_1, \dots, ρ_n be the used substitutions.

- **(C1)** If none of the redexes t_1, \dots, t_n are captured by t , then we use the ADP $\ell \rightarrow \{p_1 : \#_{\text{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \#_{\text{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m$ with the position $\pi \in \text{pos}_{\mathcal{D}\#}(t_x) \setminus \text{Junk}(t_x) \subseteq_{(IH)} \text{pos}_{\mathcal{D}\#}(t'_x)$ and the substitution σ . Once again, (28) is satisfied for our resulting terms.
- **(C2)** If $t = t_i$ for some $1 \leq i \leq n$, then we can find a narrowing substitution δ_e of t that is more general than σ , i.e., we have $\delta_e\gamma = \sigma$. Now, we use the

- ADP $\ell\delta_e \rightarrow \{p_1 : r_1\delta_e, \dots, p_k : r_k\delta_e\}^m$ with the position $\pi \in \text{pos}_{\mathcal{D}^\#}(t_x) \setminus \text{Junk}(t_x) \subseteq_{(IH)} \text{pos}_{\mathcal{D}^\#}(t'_x)$ and the substitution γ such that $\flat(t_x|_\pi) = \ell\delta_e\gamma = \ell\sigma \in \text{ANF}_{\mathcal{P}}$. Once again, (28) is satisfied for our resulting terms.
- **(C3)** If $t \neq t_i$ for all $1 \leq i \leq n$ but there is an $1 \leq i \leq n$ such that t_i is captured, then, since t_i is captured, there exists a narrowing substitution δ_e of t that is more general than ρ_i , i.e., there exists a substitution κ_1 with $\delta_e\kappa_1 = \rho_i$, and since we use ρ_i later on we additionally have that ρ_i is more general than σ , i.e., there exists a substitution κ_2 with $\rho_i\kappa_2 = \sigma$. Now, we use the ADP $\ell\delta_e \rightarrow \{p_1 : r_1\delta_e, \dots, p_k : r_k\delta_e\}^m$ with the position $\pi \in \text{pos}_{\mathcal{D}^\#}(t_x) \setminus \text{Junk}(t_x) \subseteq_{(IH)} \text{pos}_{\mathcal{D}^\#}(t'_x)$ and the substitution $\kappa_1\kappa_2$ such that $\flat(t_x|_\pi) = \ell\delta_e\kappa_1\kappa_2 = \ell\sigma \in \text{ANF}_{\mathcal{P}}$. Once again, (28) is satisfied for our resulting terms.

Completeness: The proof is analogous to the completeness proof of [Thm. 37](#). We can replace each ADP $\ell\delta_e \rightarrow \{p_1 : r_1\delta_e, \dots, p_k : r_k\delta_e\}^m$ with the more general one $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$, and each ADP $\ell \rightarrow \{p_1 : \#_{\text{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \#_{\text{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m$ can be replaced by $\ell \rightarrow \{p_1 : r_1, \dots, p_k : r_k\}^m$ as well, leading to more annotations than before. \square

C Examples

In this section, we present several examples to illustrate specific strengths and weaknesses of the new transformational processors.

C.1 Probabilistic Quicksort

In [Sect. 6](#) we have already seen a version of the probabilistic quicksort algorithm where iAST can only be proved if we use the new transformational processors. The following PTRS represents the full implementation of this probabilistic quicksort algorithm.

```

rotate(nil) → {1 : nil}
rotate(cons(x, xs)) → {1/2 : cons(x, xs), 1/2 : rotate(app(xs, cons(x, nil)))}
empty(nil) → {1 : true}
empty(cons(x, xs)) → {1 : false}
qsrt(xs) → {1 : if(empty(xs), low(hd(xs), tl(xs)), hd(xs), high(hd(xs), tl(xs)))}
if(true, xs, x, ys) → {1 : nil}
if(false, xs, x, ys) → {1 : app(qsrt(rotate(xs)), cons(x, qsrt(rotate(ys))))}
hd(cons(x, xs)) → {1 : x}
tl(cons(x, xs)) → {1 : xs}
low(x, nil) → {1 : nil}
low(x, cons(y, ys)) → {1 : ifLow(leq(x, y), x, cons(y, ys))}
ifLow(true, x, cons(y, ys)) → {1 : low(x, ys)}
ifLow(false, x, cons(y, ys)) → {1 : cons(y, low(x, ys))}
high(x, nil) → {1 : nil}
high(x, cons(y, ys)) → {1 : ifHigh(leq(x, y), x, cons(y, ys))}
ifHigh(true, x, cons(y, ys)) → {1 : cons(y, high(x, ys))}
ifHigh(false, x, cons(y, ys)) → {1 : high(x, ys)}
leq(0, x) → {1 : true}

```

$$\begin{aligned}
&\text{leq}(s(x), 0) \rightarrow \{1 : \text{false}\} \\
&\text{leq}(s(x), s(y)) \rightarrow \{1 : \text{leq}(x, y)\} \\
&\text{app}(\text{nil}, ys) \rightarrow \{1 : ys\} \\
&\text{app}(\text{cons}(x, xs), ys) \rightarrow \{1 : \text{cons}(x, \text{app}(xs, ys))\}
\end{aligned}$$

This quicksort algorithm searches for a random pivot element using the first two `rotate` rules. Here, we rotate the list and always move the head element to the end of the list. With a chance of $1/2$ we stop this iteration and use the current head element as the next pivot element. The rest of the rules represents the classical quicksort algorithm without any probabilities. Here, `app` computes list concatenation, `low`(x, xs) returns all elements of the list xs that are smaller than x , and `high` works analogously. Furthermore, `hd` returns the head element of a list and `tl` returns the rest of the list without the head. Finally, `empty` checks whether the list is empty or not.

Using our new ADP framework with the new transformational processors, AProVE can automatically prove that this PTRS $\mathcal{R}_{\text{qsort}}$ is iAST. In particular, we need the transformations (i.e., the rewriting processor) to evaluate the `hd`, `tl`, and `empty` functions, and we need the rule overlap instantiation processor to determine all possible terms that these functions can actually be applied on, e.g., we need to detect that if we have the term `empty`(x) for a variable x , then in order to apply any rewrite step, the variable x needs to be instantiated with either `nil` or `cons`(y, ys) for some new variables y and ys .

C.2 Moving Elements in Lists Probabilistically

Another interesting probabilistic algorithm that deals with lists is the following: We are given two lists L_1 and L_2 . If one of the two lists is empty, then the algorithm terminates. Otherwise, we either move the head of list L_1 to L_2 or vice versa, both with a chance of $1/2$, and then we repeat this procedure. This algorithm is represented by the following PTRS.

$$\begin{aligned}
&\text{or}(\text{false}, \text{false}) \rightarrow \{1 : \text{false}\} \\
&\text{or}(\text{true}, x) \rightarrow \{1 : \text{true}\} \\
&\text{or}(x, \text{true}) \rightarrow \{1 : \text{true}\} \\
&\text{moveElements}(xs, ys) \rightarrow \{1 : \text{if}(\text{or}(\text{empty}(xs), \text{empty}(ys)), xs, ys)\} \\
&\text{if}(\text{true}, xs, ys) \rightarrow \{1 : xs\} \\
&\text{if}(\text{false}, xs, ys) \rightarrow \{1/2 : \text{moveElements}(\text{tl}(xs), \text{cons}(\text{hd}(xs), ys)), \\
&\quad 1/2 : \text{moveElements}(\text{cons}(\text{hd}(ys), xs), \text{tl}(ys))\} \\
&\text{empty}(\text{nil}) \rightarrow \{1 : \text{true}\} \\
&\text{empty}(\text{cons}(x, xs)) \rightarrow \{1 : \text{false}\} \\
&\text{hd}(\text{cons}(x, xs)) \rightarrow \{1 : x\} \\
&\text{tl}(\text{cons}(x, xs)) \rightarrow \{1 : xs\}
\end{aligned}$$

This algorithm is iAST (and even AST) because we can view this as a classical random walk on the number of elements in the first list that is both bounded from below by 0 and from above by the sum of the length of both lists. In order to prove this automatically, we again have to use some kind of instantiation processor, e.g.,

the rule overlap instantiation processor, to find all possible terms that the functions `hd`, `tl`, and `empty` can actually be applied on. In addition, we also need the rewriting processor again to then evaluate these functions. Once the `moveElements-ADP` contains the symbols `cons` and `nil` in the left-hand side, we can detect the structure of the random walk using an application of the reduction pair processor that removes all annotations of this ADP. After that, there is no SCC in the dependency graph left, and we have proven iAST.

C.3 Conditions on Numbers

Another important task of termination analysis is to be able to handle conditions on numbers. These occur in nearly every program and often impact the termination behavior. The same is true for probabilistic programs. For a successful proof of iAST without transformations, we need that the rules of the PTRS have these conditions integrated in their left-hand sides of the rules, i.e., if one wants to check whether a number is zero or not, then one needs two different rules $f(0) \rightarrow \dots$ and $f(s(x)) \rightarrow \dots$ to perform this case analysis. However, in programs, conditions are mostly implemented by an if-construct, where one could use, e.g., an additional function `gt` to check whether a number is greater than another. The same is true for conditions on other data structures than numbers, as we have seen above. If one wants to check whether the list is empty or not, then without transformations, one needs two rules (for `nil` and `cons`), whereas with transformations, one can use conditions and auxiliary functions like `empty`.

The following PTRS depicts the classical random walk, but we check the condition $x > 0$ not directly in the left-hand side of the rule but with an additional function `gt(x, y)` which checks whether $x > y$. Additionally, in order to decrease the value of a number by one, we use the predecessor function $p(x) = x - 1$.

$$\begin{aligned} & \text{gt}(0, 0) \rightarrow \{1 : \text{false}\} \\ & \text{gt}(s(x), 0) \rightarrow \{1 : \text{true}\} \\ & \text{gt}(0, s(y)) \rightarrow \{1 : \text{false}\} \\ & \text{gt}(s(x), s(y)) \rightarrow \{1 : \text{gt}(x, y)\} \\ & p(0) \rightarrow \{1 : 0\} \\ & p(s(x)) \rightarrow \{1 : x\} \\ & \text{loop}(x) \rightarrow \{1 : \text{if}(\text{gt}(x, 0), x)\} \\ & \text{if}(\text{false}, x) \rightarrow \{1 : \text{stop}\} \\ & \text{if}(\text{true}, x) \rightarrow \{1/2 : \text{loop}(p(x)), 1/2 : \text{loop}(s(x))\} \end{aligned}$$

In this case, we need the rewriting processor to evaluate the functions `gt` and `p` and once again we need the rule overlap instantiation processor to check for all possible terms that these functions can actually be applied on.

C.4 Limits of the Instantiation Processors

Whenever we have an ADP where the left-hand side of the rule is also contained in the support of the right-hand side, then instantiations become useless, because we will always have at least the same ADP again after applying the processor.

For example, we need to apply one of the instantiation processors in order to prove termination for the TRS with the rules $f(x, y, z) \rightarrow g(x, y, z)$ and $g(a, b, z) \rightarrow f(z, z, z)$. If we make the rules probabilistic by adding the possibility to do nothing in a rewrite step with the probability $1/2$, then we result in the following PTRS.

$$\begin{aligned} f(x, y, z) &\rightarrow \{1/2 : g(x, y, z), 1/2 : f(x, y, z)\} \\ g(a, b, z) &\rightarrow \{1/2 : f(z, z, z), 1/2 : g(a, b, z)\} \end{aligned}$$

This PTRS is iAST (since the original TRS was innermost terminating) but we are unable to show this using the instantiation processor, because if one tries to instantiate any of the rules, this will result in at least the same rule after the processor. In contrast, in [Ex. 38](#) we had nearly the same PTRS but the first rule remained non-probabilistic. There, we were able to apply the instantiation processor and prove iAST using the ADP Framework. (However, while instantiation does not help in our example above, we can prove iAST using the rule overlap instantiation processor.)

C.5 Transformations do not Suffice for Inductive Reasoning

Transformational processors are useful to perform a case analysis, but they do not suffice for PTRSs where one needs inductive reasoning for the termination analysis. For example, we cannot show iAST of the following PTRS even though the if-structure seems similar to the one of the probabilistic quicksort example. This example was proposed to us by Johannes Niederhauser at the 0-th probabilistic termination competition in August 2023.

$$\begin{aligned} \text{even}(0) &\rightarrow \{1 : \text{true}\} & \text{even}(s(0)) &\rightarrow \{1 : \text{false}\} \\ \text{even}(s(s(x))) &\rightarrow \{1 : \text{even}(x)\} & \text{loop}(x) &\rightarrow \{1 : \text{if}(\text{even}(x), x)\} \\ \text{if}(\text{false}, x) &\rightarrow \{1 : \text{stop}\} & \text{if}(\text{true}, x) &\rightarrow \{1/2 : \text{loop}(x), 1/2 : \text{loop}(s(x))\} \end{aligned}$$

The idea here is that the recursion of `loop` stops if its argument contains an even number. If it is not even, then we either increase the value by 1 or use the same value again. Here, a simple case analysis does not suffice, but we actually have to show (inductively), that if a number x is odd, then $x + 1$ is even. This is not possible with the new transformational processors but needs other types of processors for inductive reasoning (e.g., as in [\[15\]](#) for the non-probabilistic DP framework).