Dependency Pairs for Expected Innermost Runtime Complexity and Strong Almost-Sure Termination of Probabilistic Term Rewriting

Jan-Christoph Kassing
kassing@cs.rwth-aachen.de
RWTH Aachen University
Aachen, Germany

Leon Spitzer
leon.spitzer@rwth-aachen.de
RWTH Aachen University
Aachen, Germany

Jürgen Giesl
giesl@informatik.rwth-aachen.de
RWTH Aachen University
Aachen, Germany

Abstract

The dependency pair (DP) framework is one of the most powerful techniques for automatic termination and complexity analysis of term rewrite systems. While DPs were extended to prove almost-sure termination of *probabilistic* term rewrite systems (PTRSs), automatic complexity analysis for PTRSs is largely unexplored. We introduce the first DP framework for analyzing expected complexity and for proving *positive* or *strong* almost-sure termination (SAST) of innermost rewriting with PTRSs, i.e., finite expected runtime. We implemented our framework in the tool AProVE and demonstrate its power compared to existing techniques for proving SAST.

CCS Concepts

• **Theory of computation → Equational logic and rewriting**; **Probabilistic computation**; **Program analysis**.

Keywords

Term Rewriting, Dependency Pairs, Probabilistic Programs, Termination Analysis, Complexity Analysis

ACM Reference Format:
Jan-Christoph Kassing, Leon Spitzer, and Jürgen Giesl. 2025. Dependency Pairs for Expected Innermost Runtime Complexity and Strong Almost-Sure Termination of Probabilistic Term Rewriting. In *Proceedings of the 27th International Symposium on Principles and Practice of Declarative Programming (PPDP '25).* ACM, New York, NY, USA, 20 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Probabilistic programming integrates probabilistic branching into traditional computer models, with applications in many areas [24]. Probabilities do not only handle uncertainty in data, but they can also be used to decrease the expected runtime of algorithms.

Term rewriting [9] is a fundamental concept to transform and evaluate expressions, which is used, e.g., for symbolic computation, automated theorem proving, and automatic program analysis. There exist many approaches to prove termination or infer bounds on the runtime complexity of TRSs, for example, via *ranking functions* like polynomial interpretations [44]. One of the most powerful approaches to analyze termination and runtime complexity of TRSs is the *dependency pair* (DP) framework, see, e.g., [3, 4, 21, 26, 53]. It uses a divide-and-conquer approach to transform termination or complexity problems into simpler subproblems repeatedly (via so-called *DP processors*). Indeed, DPs are used in essentially all current termination and complexity tools for TRSs, e.g., AProVE [22], MU-TERM [25], NaTT [57], TcT [5], $\mathsf{T_TT_2}$ [41], etc.

Probabilistic TRSs (PTRSs) have been introduced in [7, 13, 14]. A PTRS $\mathcal{R}$ is *almost-surely terminating* (AST) if every evaluation (or "reduction") terminates with probability 1. A strictly stronger notion is *positive* AST (PAST), where every reduction must consist of a finite expected number of rewrite steps. An even stronger notion is *strong* AST (SAST) which requires that the expected number of rewrite steps is bounded by a finite value for every start term. It is well known that SAST implies PAST and that PAST implies AST. In this paper, we develop an approach to prove SAST for PTRSs under an innermost evaluation strategy where we only consider reductions starting with *basic* terms (which represent the application of an algorithm to data objects). Moreover, our approach computes upper bounds on the *expected innermost runtime complexity* of PTRSs. *Runtime complexity* is one of the standard notions of complexity for non-probabilistic TRSs [27], and it was adapted to *expected runtime complexity* for PTRSs in [35].

Related Work: There are numerous techniques to prove (P)AST for *imperative programs on numbers*, e.g., [1, 2, 8, 15, 16, 18, 29–31, 46–52]. In particular, there also exist several *tools* to analyze (P)AST and expected costs for imperative probabilistic programs, e.g., Absynth [52], Amber [51], Eco-Imp [8], and KoAT [46, 50]. In addition, there are also several related approaches for recursive programs, e.g., to analyze probabilistic higher-order programs based on types or martingales [6, 11, 39, 42, 43, 54], or probabilistic imperative languages with recursion [40]. However, only few approaches analyze probabilistic programs on recursive *data structures*, e.g., [10, 45, 56]. While [10] uses pointers to represent data structures like tables and lists, [45, 56] consider a probabilistic programming language with matching similar to term rewriting and develop an automatic amortized resource analysis via fixed template potential functions. However, these works are mostly targeted towards specific data structures, whereas our aim is a fully automatic approach for general PTRSs that can model arbitrary data structures.

Currently, the only approach to analyze SAST of PTRSs automatically is the direct application of polynomial or matrix interpretations [17] to the whole PTRS [7], implemented in NaTT. However, already for non-probabilistic TRSs such a direct application of orderings is limited in power. For a powerful approach, one should combine orderings in a modular way, as in the DP framework.

Therefore, we already adapted the DP framework to the probabilistic setting in order to prove AST, both for innermost [32, 36] and full rewriting [37]. Moreover, in the non-probabilistic setting, DPs were extended to analyze complexity instead of just termination, see, e.g., [4, 53]. But up to now there did not exist any DP framework to prove SAST or PAST, or to infer bounds on the expected runtime of PTRSs. In this paper we show that the DP framework for AST from [36] which uses *annotated dependency pairs* can be lifted to a novel DP framework for expected complexity of PTRSs.

Moreover, in [34] we presented criteria for classes of PTRSs where (P)AST for innermost rewriting implies (P)AST for full rewriting, and in [35] these criteria were extended to SAST and expected runtime complexity. Thus, they can also be used in order to infer SAST and expected runtime complexity for full instead of innermost rewriting via our novel DP framework, see Sect. 5.

Main Results of the Paper:

- We develop the first DP framework for SAST and expected innermost runtime complexity of probabilistic TRSs.
- We introduce several processors for our novel DP framework.
- To evaluate the power of our novel framework, we implemented it in the tool AProVE.

Structure: We recapitulate (probabilistic) term rewriting in Sect. 2. In Sect. 3, we introduce our novel DP framework for SAST and expected runtime complexity. Afterwards, we present several processors that can be used in our framework in Sect. 4. In Sect. 5, we give experimental results, also in comparison to the technique of [7]. The proofs of all our results can be found in Sect. A.

2 Preliminaries

We recapitulate ordinary and probabilistic TRSs in Sect. 2.1 and 2.2.

2.1 Term Rewriting

We assume some familiarity with term rewriting [9], but recapitulate all needed notions. For any relation $\to \subseteq A \times A$ on a set A and $n \in \mathbb{N}$, we define $\to^n$ as $\to^0 = \{(a,a) \mid a \in A\}$ and $\to^{n+1} = \to^n \circ \to$, where "$\circ$" denotes composition of relations. Moreover, $\to^* = \bigcup_{n\in\mathbb{N}} \to^n$, i.e., $\to^*$ is the reflexive and transitive closure of $\to$.

The set $\mathcal{T} = \mathcal{T}(\Sigma, \mathcal{V})$ of all *terms* over a finite set of *function symbols* $\Sigma = \biguplus_{k\in\mathbb{N}} \Sigma_k$ and a (possibly infinite) set of *variables* $\mathcal{V}$ is the smallest set with $\mathcal{V} \subseteq \mathcal{T}$, and if $f \in \Sigma_k$ and $t_1, \ldots, t_k \in \mathcal{T}$ then $f(t_1, \ldots, t_k) \in \mathcal{T}$. The *arity* of a function symbol $f \in \Sigma_k$ is k. For example, consider the signature $\Sigma_{\mathsf{q}} = \{\mathsf{q}, \mathsf{start}, \mathsf{s}, \mathsf{0}\}$, where $\mathsf{0}$ has arity 0, s has arity 1, start has arity 2, and q has arity 3. Then, for $x \in \mathcal{V}$, $\mathsf{start}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))$ and $\mathsf{q}(\mathsf{0}, x, x)$ are terms in $\mathcal{T}(\Sigma_{\mathsf{q}}, \mathcal{V})$. A term without variables is called a *ground* term. The *size* $|t|$ of a term t is the number of occurrences of function symbols and variables in t, i.e., $|t| = 1$ if $t \in \mathcal{V}$, and $|t| = 1 + \sum_{j=1}^{k} |t_j|$ if $t = f(t_1, \ldots, t_k)$. Thus, $|\mathsf{start}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))| = 5$ and $|\mathsf{q}(\mathsf{0}, x, x)| = 4$. A *substitution* is a function $\sigma : \mathcal{V} \to \mathcal{T}$ with $\sigma(x) = x$ for all but finitely many $x \in \mathcal{V}$, and we often write $x\sigma$ instead of $\sigma(x)$. Substitutions homomorphically extend to terms: If $t = f(t_1, \ldots, t_k) \in \mathcal{T}$ then $t\sigma = f(t_1\sigma, \ldots, t_k\sigma)$. Thus, for a substitution σ with $\sigma(x) = \mathsf{s}(x)$ we obtain $\mathsf{q}(\mathsf{0}, x, x)\sigma = \mathsf{q}(\mathsf{0}, \mathsf{s}(x), \mathsf{s}(x))$. For any term $t \in \mathcal{T}$, the set of *positions* $\mathrm{Pos}(t)$ is the smallest subset of $\mathbb{N}^*$ satisfying $\varepsilon \in \mathrm{Pos}(t)$, and if $t = f(t_1, \ldots, t_k)$ then for all $1 \leq j \leq k$ and all $\pi \in \mathrm{Pos}(t_j)$ we have $j.\pi \in \mathrm{Pos}(t)$. A position π_1 is *above* π_2 if π_1 is a prefix of π_2. If $\pi \in \mathrm{Pos}(t)$ then $t|_\pi$ denotes the subterm starting at position π and $t[r]_\pi$ denotes the term that results from replacing the subterm $t|_\pi$ at position π with the term $r \in \mathcal{T}$. We write $s \trianglelefteq t$ if s is a subterm of t and $s \triangleleft t$ if s is a *proper* subterm of t (i.e., if $s \trianglelefteq t$ and $s \neq t$). For example, we have $\mathrm{Pos}(\mathsf{q}(\mathsf{0}, x, x)) = \{\varepsilon, 1, 2, 3\}$, $\mathsf{q}(\mathsf{0}, x, x)|_2 = x$, $\mathsf{q}(\mathsf{0}, x, x)[\mathsf{s}(x)]_2 = \mathsf{q}(\mathsf{0}, \mathsf{s}(x), x)$, and $\mathsf{s}(x) \triangleleft \mathsf{q}(\mathsf{0}, \mathsf{s}(x), x)$.

A *rewrite rule* $\ell \to r$ is a pair of terms $(\ell, r) \in \mathcal{T} \times \mathcal{T}$ such that $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ and $\ell \notin \mathcal{V}$, where $\mathcal{V}(t)$ denotes the set of all variables occurring in $t \in \mathcal{T}$. A *term rewrite system* (TRS) is a finite set of rewrite rules. As an example, consider the following TRS $\mathcal{R}_{\mathsf{q}}$ that is used to compute the rounded **q**uotient of two natural numbers (represented by the successor function s and $\mathsf{0}$) [3].

$$\begin{array}{rr} \mathsf{start}(x,y) \to \mathsf{q}(x,y,y) & \mathsf{q}(x,\mathsf{0},\mathsf{s}(z)) \to \mathsf{s}(\mathsf{q}(x,\mathsf{s}(z),\mathsf{s}(z))) \\ \mathsf{q}(\mathsf{s}(x),\mathsf{s}(y),z) \to \mathsf{q}(x,y,z) & \mathsf{q}(\mathsf{0},\mathsf{s}(y),\mathsf{s}(z)) \to \mathsf{0} \end{array}$$

A TRS $\mathcal{R}$ induces a *rewrite relation* $\to_{\mathcal{R}} \subseteq \mathcal{T} \times \mathcal{T}$ on terms where $s \to_{\mathcal{R}} t$ holds if there is a position $\pi \in \mathrm{Pos}(s)$, a rule $\ell \to r \in \mathcal{R}$, and a substitution σ such that $s|_\pi = \ell\sigma$ and $t = s[r\sigma]_\pi$. Let $\mathsf{NF}_{\mathcal{R}}$ denote the set of all terms that are in normal form w.r.t. $\to_{\mathcal{R}}$.

Such a rewrite step $s \to_{\mathcal{R}} t$ is an *innermost* rewrite step (denoted $s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t$) if $\ell\sigma \in \mathsf{ANF}_{\mathcal{R}}$, where $\mathsf{ANF}_{\mathcal{R}}$ is the set of all terms in *argument normal form* w.r.t. $\to_{\mathcal{R}}$, i.e., $t \in \mathsf{ANF}_{\mathcal{R}}$ iff $t' \in \mathsf{NF}_{\mathcal{R}}$ for all proper subterms $t' \triangleleft t$. The TRS $\mathcal{R}_{\mathsf{q}}$ computes $\lfloor \frac{n}{m} \rfloor$ when starting with the term $\mathsf{start}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$, where $\mathsf{s}^n(\ldots)$ denotes $n \in \mathbb{N}$ successive s-function symbols. For example, we have $\lfloor \frac{1}{1} \rfloor = 1$ and

$$\begin{array}{l} \mathsf{start}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0})) \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{q}}} \mathsf{q}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0})) \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{q}}} \mathsf{q}(\mathsf{0},\mathsf{0},\mathsf{s}(\mathsf{0})) \\ \qquad \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{q}}} \mathsf{s}(\mathsf{q}(\mathsf{0},\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0}))) \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{q}}} \mathsf{s}(\mathsf{0}). \end{array}$$

Already for non-probabilistic TRSs, the techniques for termination and complexity analysis of *innermost* rewriting are significantly stronger than the ones for "full" rewriting where arbitrary rewrite sequences are allowed (the same holds for the probabilistic DP framework for AST in [32, 36, 37]). Moreover, innermost evaluation is the standard strategy for most programming languages. Hence, in the remainder, we restrict ourselves to innermost rewriting.

The *derivation height* [28] of a term t is the length of the longest $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$-sequence starting with t, i.e.,

$$\mathrm{dh}_{\mathcal{R}}(t) = \sup\{n \in \mathbb{N} \mid \exists t' \in \mathcal{T} \text{ such that } t \xrightarrow{\mathrm{i}}{}^{n}_{\mathcal{R}} t'\} \in \mathbb{N} \cup \{\omega\}.$$

For example, $\mathrm{dh}_{\mathcal{R}_{\mathsf{q}}}(\mathsf{start}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0}))) = 4$. We have $\mathrm{dh}_{\mathcal{R}}(t) = 0$ iff $t \in \mathsf{NF}_{\mathcal{R}}$ and $\mathrm{dh}_{\mathcal{R}}(t) = \omega$ iff t starts an infinite sequence of $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$-steps, as we restricted ourselves to finite TRSs.

We decompose the signature $\Sigma = \mathcal{D}_{\mathcal{R}} \uplus \mathcal{C}_{\mathcal{R}}$ into *defined symbols* $\mathcal{D}_{\mathcal{R}} = \{\mathrm{root}(\ell) \mid \ell \to r \in \mathcal{R}\}$ and *constructors* $\mathcal{C}_{\mathcal{R}}$. If $\mathcal{R}$ is clear from the context, we just write $\mathcal{C}$ and $\mathcal{D}$. A term $f(t_1, \ldots, t_k)$ is *basic* if $f \in \mathcal{D}_{\mathcal{R}}$ and $t_1, \ldots, t_k \in \mathcal{T}(\mathcal{C}_{\mathcal{R}}, \mathcal{V})$, i.e., $t_1, \ldots, t_k$ do not contain defined symbols. Thus, basic terms represent an algorithm f that is applied to data $t_1, \ldots, t_k$. So for $\mathcal{R}_{\mathsf{q}}$, $\mathsf{q}(\mathsf{0}, x, x)$ is basic, but $\mathsf{q}(\mathsf{q}(\mathsf{0}, x, x), x, x)$ is not. Let $\mathcal{TB}_{\mathcal{R}}$ denote the set of basic terms for the TRS $\mathcal{R}$.

The *runtime complexity* $\mathrm{rc}_{\mathcal{R}}$ is a function that maps any $n \in \mathbb{N}$ to the maximum derivation height for basic terms of size $\leq n$.

Definition 2.1 (Runtime Complexity, $\mathrm{rc}_{\mathcal{R}}$ [27]). For a TRS $\mathcal{R}$, its *runtime complexity function* $\mathrm{rc}_{\mathcal{R}} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ is:

$$\mathrm{rc}_{\mathcal{R}}(n) = \sup\{\mathrm{dh}_{\mathcal{R}}(t) \mid t \in \mathcal{TB}_{\mathcal{R}}, |t| \leq n\}$$

Given a TRS $\mathcal{R}$, the goal is to determine an upper bound on the *asymptotic complexity* of the function $\mathrm{rc}_{\mathcal{R}}$.

Definition 2.2 (Asymptotic Complexities). We consider a set of complexities $\mathfrak{C} = \{\mathrm{Pol}_0, \mathrm{Pol}_1, \mathrm{Pol}_2, \ldots, \mathrm{Exp}, \text{2-Exp}, \mathrm{Fin}, \omega\}$ with the order $\mathrm{Pol}_0 \sqsubset \mathrm{Pol}_1 \sqsubset \mathrm{Pol}_2 \sqsubset \ldots \sqsubset \mathrm{Exp} \sqsubset \text{2-Exp} \sqsubset \mathrm{Fin} \sqsubset \omega$, where $\sqsubseteq$ is the reflexive closure of $\sqsubset$. For any function $f : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$, we define its *complexity* $\iota(f) \in \mathfrak{C}$ as follows:

$$\iota(f) = \begin{cases} \mathrm{Pol}_a & \text{if } a \in \mathbb{N} \text{ is the smallest number with } f(n) \in O(n^a) \\ \mathrm{Exp} & \text{if no such } a \text{ exists, but there is a} \\ & \text{polynomial pol}(n) \text{ such that } f(n) \in O(2^{\mathrm{pol}(n)}) \\ \text{2-Exp} & \text{if no such polynomial exists, but there is a} \\ & \text{polynomial pol}(n) \text{ such that } f(n) \in O(2^{2^{\mathrm{pol}(n)}}) \\ \mathrm{Fin} & \text{if no such polynomial exists,} \\ & \text{but there is no } n \in \mathbb{N} \text{ with } f(n) = \omega \\ \omega & \text{if there is an } n \in \mathbb{N} \text{ with } f(n) = \omega \end{cases}$$

For any TRS $\mathcal{R}$, we define its *runtime complexity* $\iota_{\mathcal{R}}$ as $\iota(\mathrm{rc}_{\mathcal{R}})$.

The TRS $\mathcal{R}_{\mathrm{q}}$ has linear runtime complexity, i.e., $\iota_{\mathcal{R}_{\mathrm{q}}} = \iota(\mathrm{rc}_{\mathcal{R}_{\mathrm{q}}}) = \mathrm{Pol}_1$. For example, any rewrite sequence starting with $\mathsf{start}(\mathsf{s}^n(0), \mathsf{s}^m(0))$ has at most $2n + 2$ rewrite steps.

Finally, we recapitulate a basic approach to prove termination and to infer upper bounds on the runtime complexity via polynomial[1] interpretations. A *polynomial interpretation* is a Σ-Algebra $\mathcal{I} : \Sigma \to \mathbb{N}[\mathcal{V}]$ that maps every function symbol $f \in \Sigma_k$ to a polynomial $\mathcal{I}_f$ over k variables with natural coefficients. As usual, $\mathcal{I}$ is homomorphically extended to terms. $\mathcal{I}$ is *monotonic* if $x > y$ implies $\mathcal{I}_f(\ldots, x, \ldots) > \mathcal{I}_f(\ldots, y, \ldots)$ for all $f \in \Sigma$ and $x, y \in \mathbb{N}$. We call $\mathcal{I}$ a *complexity polynomial interpretation* (CPI) if for all constructors $f \in \mathcal{C}$ we have $\mathcal{I}_f(x_1, \ldots, x_k) = a_1 x_1 + \ldots + a_k x_k + b$, where $b \in \mathbb{N}$ and $a_i \in \{0, 1\}$.[2] While arbitrary monotonic polynomial interpretations can be used to prove termination of TRSs [44], monotonic CPIs are needed to infer a polynomial runtime bound from such a termination proof [12, 28].

More precisely, if there is a monotonic polynomial interpretation $\mathcal{I}$ such that $\mathcal{I}(\ell) > \mathcal{I}(r)$ holds for every rule $\ell \to r \in \mathcal{R}$, then $\mathcal{R}$ is terminating and $\iota_{\mathcal{R}} \sqsubseteq$ 2-Exp. If all constructors are interpreted by linear polynomials, then we have $\iota_{\mathcal{R}} \sqsubseteq \mathrm{Exp}$.

But if $\mathcal{I}$ is a monotonic CPI, we even have $\iota_{\mathcal{R}} \sqsubseteq \mathrm{Pol}_a$ if for all $f \in \mathcal{D}$, the polynomial $\mathcal{I}_f$ has at most degree a. The reason is that this implies $\mathcal{I}(t) \in O(|t|^a)$ for all basic ground terms $t \in \mathcal{TB}_{\mathcal{R}}$. (More precisely, the function that maps any $n \in \mathbb{N}$ to $\sup\{\mathcal{I}(t) \mid t \in \mathcal{TB}_{\mathcal{R}}, \mathcal{V}(t) = \varnothing, |t| \leq n\}$ is in $O(n^a)$.) Since every rewrite step decreases the interpretation of the term by at least 1, the length of each rewrite sequence starting with a basic ground term of size $\leq n$ is in $O(n^a)$.

This direct approach is only feasible for simple examples like the TRS $\mathcal{R}_{\mathsf{plus}} = \{\mathsf{plus}(0, y) \to y,\ \mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{plus}(x, y))\}$, computing the addition of two natural numbers. Let $\mathcal{I}$ be a monotonic CPI with $\mathcal{I}_0 = 0$, $\mathcal{I}_{\mathsf{s}}(x) = x + 1$, and $\mathcal{I}_{\mathsf{plus}}(x, y) = 2x + y + 1$. We have $\mathcal{I}(\mathsf{plus}(\mathsf{s}(x), y)) = 2x + y + 3 > 2x + y + 2 = \mathcal{I}(\mathsf{s}(\mathsf{plus}(x, y)))$, and $\mathcal{I}(\mathsf{plus}(0, y)) = y + 1 > y = \mathcal{I}(y)$. Thus, $\mathcal{R}_{\mathsf{plus}}$ is terminating and has at most linear runtime complexity, i.e., $\iota_{\mathcal{R}_{\mathsf{plus}}} \sqsubseteq \mathrm{Pol}_1$ (in fact, we have $\iota_{\mathcal{R}_{\mathsf{plus}}} = \mathrm{Pol}_1$). To automate this approach, one can use SMT solvers to search for a suitable CPI $\mathcal{I}$. However, such a direct application of polynomials fails to prove termination or to infer a polynomial upper bound on $\iota_{\mathcal{R}_{\mathrm{q}}}$. In contrast, $\iota_{\mathcal{R}_{\mathrm{q}}} \sqsubseteq \mathrm{Pol}_1$ can be proved by more elaborate techniques like dependency pairs, see, e.g., [53]. Indeed, we will show how to analyze a probabilistic version of $\mathcal{R}_{\mathrm{q}}$ with our novel DP framework.

[1] In this paper, we focus on polynomial interpretations with natural coefficients for simplicity, but our results can be extended to other interpretations where one can define an addition and expected value operation $\mathbb{E}(\cdot)$, e.g., matrix interpretations [17].
[2] For monotonic CPIs one must have $a_i = 1$, but in Sect. 4.3 we will consider weakly monotonic CPIs, where $a_i \in \{0, 1\}$ is possible.

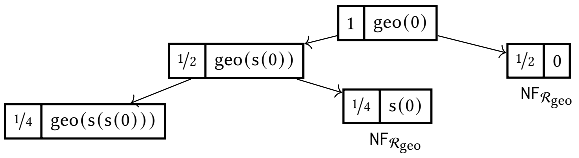

1
geo(0)
1/2
geo(s(0))
1/2
0
NF$_{\mathcal{R}_{\mathrm{geo}}}$
1/4
geo(s(s(0)))
1/4
s(0)
NF$_{\mathcal{R}_{\mathrm{geo}}}$

Figure 1: $\mathcal{R}_{\mathrm{geo}}$-Rewrite sequence tree starting with $\mathsf{geo}(0)$

2.2 Probabilistic Rewriting

A probabilistic TRS has finite multi-distributions on the right-hand sides of its rewrite rules. A finite *multi-distribution* μ on a set $A \neq \varnothing$ is a finite multiset of pairs $(p : a)$, where $0 < p \leq 1$ is a probability and $a \in A$, such that $\sum_{(p:a) \in \mu} p = 1$. Let $\mathrm{FDist}(A)$ denote the set of all finite multi-distributions on A. For $\mu \in \mathrm{FDist}(A)$, its *support* is the multiset $\mathrm{Supp}(\mu) = \{a \mid (p : a) \in \mu \text{ for some } p\}$. A *probabilistic rewrite rule* $\ell \to \mu$ is a pair $(\ell, \mu) \in \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ such that $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ for every $r \in \mathrm{Supp}(\mu)$. A *probabilistic TRS* (PTRS) is a finite set of probabilistic rewrite rules. Similar to TRSs, a PTRS $\mathcal{R}$ induces a *(probabilistic) rewrite relation* $\to_{\mathcal{R}} \subseteq \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ where $s \to_{\mathcal{R}} \{p_1 : t_1, \ldots, p_k : t_k\}$ if there is a position $\pi \in \mathrm{Pos}(s)$, a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{R}$, and a substitution σ such that $s|_\pi = \ell\sigma$ and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$. We call $s \to_{\mathcal{R}} \mu$ an *innermost* rewrite step (denoted $s \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mu$) if $\ell\sigma \in \mathsf{ANF}_{\mathcal{R}}$. Consider the PTRS $\mathcal{R}_{\mathrm{geo}}$ with the only rule $\mathsf{geo}(x) \to \{1/2 : \mathsf{geo}(\mathsf{s}(x)),\ 1/2 : x\}$. When starting with the term $\mathsf{geo}(0)$, it computes the representation $\mathsf{s}^k(0)$ of the number $k \in \mathbb{N}$ with a probability of $(1/2)^{k+1}$, i.e., a geometric distribution.

To track innermost rewrite sequences with their probabilities, we consider *rewrite sequence trees (RSTs)* [36]. The nodes v of an $\mathcal{R}$-RST are labeled by pairs $(p_v : t_v)$ of a probability $p_v \in (0, 1]$ and a term t_v, where the root always has the probability 1. For each node v with successors $w_1, \ldots, w_k$, the edge relation represents an innermost rewrite step, i.e., $t_v \xrightarrow{\mathrm{i}}_{\mathcal{R}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \ldots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$. For an $\mathcal{R}$-RST $\mathfrak{T}$, $V^{\mathfrak{T}}$ denotes its set of nodes, $\mathrm{root}(\mathfrak{T})$ is the term at its root, and $\mathrm{Leaf}^{\mathfrak{T}}$ denotes its set of leaves. An RST for $\mathcal{R}_{\mathrm{geo}}$ is shown in Fig. 1.

A PTRS $\mathcal{R}$ is *almost-surely terminating* (AST) if $\sum_{v \in \mathrm{Leaf}^{\mathfrak{T}}} p_v = 1$ holds for all $\mathcal{R}$-RSTs $\mathfrak{T}$, i.e., if the probability of termination is always 1. This notion of AST for PTRSs is equivalent to the ones in [7, 14, 32] where AST is defined via a lifting of $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ to multisets or via stochastic processes. However, AST is not sufficient to guarantee that the expected runtime complexity of a PTRS is finite. To define this concept formally, we first introduce the *expected derivation length* of an $\mathcal{R}$-RST $\mathfrak{T}$ as

$$\mathrm{edl}(\mathfrak{T}) = \textstyle\sum_{v \in V^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}}} p_v.$$

So $\mathrm{edl}(\mathfrak{T})$ adds up the probabilities of all rewrite steps in $\mathfrak{T}$. Thus, for the RST $\mathfrak{T}$ in Fig. 1 we obtain $\mathrm{edl}(\mathfrak{T}) = 1 + 1/2 = 3/2$, i.e., in expectation we perform $3/2$ rewrite steps in $\mathfrak{T}$. Then a PTRS $\mathcal{R}$ is *positively almost-surely terminating* (PAST) if $\mathrm{edl}(\mathfrak{T})$ is finite for all $\mathcal{R}$-RSTs $\mathfrak{T}$. Again, this notion of PAST for PTRSs is equivalent to the ones in [7, 14]. Clearly, PAST implies AST, but not vice versa (e.g., a

PTRS with the rule $\mathsf{g} \to \{1/2 : \mathsf{c}(\mathsf{g},\mathsf{g}),\ 1/2 : 0\}$ which represents a symmetric random walk is AST, but not PAST).

To adapt the notions for complexity from TRSs to PTRSs, recall that in the non-probabilistic setting, the *derivation height* of a term t does not consider a fixed rewrite sequence, but all possible rewrite sequences starting with t and takes the supremum of their lengths. Similarly, while edl considers a fixed RST $\mathfrak{T}$, for the *expected derivation height* of a term t, we consider all possible RSTs with root t and take the supremum of their expected derivation lengths. So the *expected derivation height* of a term t is

$$\mathrm{edh}_{\mathcal{R}}(t) = \sup\{\mathrm{edl}(\mathfrak{T}) \mid \mathfrak{T} \text{ is an } \mathcal{R}\text{-RST with } \mathrm{root}(\mathfrak{T}) = t\}.$$

Now we can adapt the notion of runtime complexity to PTRSs.

Definition 2.3 (Expected Runtime Complexity, $\mathrm{erc}_{\mathcal{R}}$). For a PTRS $\mathcal{R}$, its *expected runtime complexity function* $\mathrm{erc}_{\mathcal{R}} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ is:

$$\mathrm{erc}_{\mathcal{R}}(n) = \sup\{\mathrm{edh}_{\mathcal{R}}(t) \mid t \in \mathcal{TB}_{\mathcal{R}}, |t| \leq n\}$$

Moreover, we define $\mathcal{R}$'s *runtime complexity* $\iota_{\mathcal{R}}$ as $\iota(\mathrm{erc}_{\mathcal{R}})$.

A PTRS $\mathcal{R}$ is *strongly* or *bounded almost-surely terminating* (SAST) if $\mathrm{edh}_{\mathcal{R}}(t)$ is finite for every term t. So in contrast to PAST, here one requires a finite bound on the expected derivation lengths of all RSTs with the same term at the root. Such notions of SAST were defined in, e.g., [7, 19, 47]. SAST implies PAST, but not vice versa (a PTRS with finitely many rules that is PAST but not SAST is given in [35]). However, as also shown in [35], SAST and PAST are "almost always" equivalent for finite PTRSs (e.g., whenever the signature contains a function symbol of arity ≥ 2).

As mentioned, in this paper we restrict ourselves to innermost reductions that start with *basic* terms. So we regard a PTRS $\mathcal{R}$ to be SAST if $\mathrm{edh}_{\mathcal{R}}(t)$ is finite for all *basic* terms t, or equivalently, $\mathrm{erc}_{\mathcal{R}}(n) \neq \omega$ for all $n \in \mathbb{N}$. Thus, we use the following definition of SAST.

Definition 2.4 (Strong Almost-Sure Termination, SAST). A PTRS $\mathcal{R}$ is called *strongly almost-surely terminating* (SAST) if $\iota(\mathcal{R}) \sqsubseteq \mathrm{Fin}$.

Example 2.5 (Leading Examples). Consider the following PTRS $\mathcal{R}_1$, which is based on the previously defined systems $\mathcal{R}_{\mathrm{q}}$ and $\mathcal{R}_{\mathrm{geo}}$.

$$\begin{aligned}
\mathsf{start}(x,y) &\to \{1 : \mathsf{q}(\mathsf{geo}(x),y,y)\}\\
\mathsf{geo}(x) &\to \{1/2 : \mathsf{geo}(\mathsf{s}(x)),\ 1/2 : x\}\\
\mathsf{q}(\mathsf{s}(x),\mathsf{s}(y),z) &\to \{1 : \mathsf{q}(x,y,z)\}\\
\mathsf{q}(x,0,\mathsf{s}(z)) &\to \{1 : \mathsf{s}(\mathsf{q}(x,\mathsf{s}(z),\mathsf{s}(z)))\}\\
\mathsf{q}(0,\mathsf{s}(y),\mathsf{s}(z)) &\to \{1 : 0\}
\end{aligned}$$

When starting with $\mathsf{start}(\mathsf{s}^n(0),\mathsf{s}^m(0))$, $\mathcal{R}_1$ computes $\lfloor \frac{n+\mathrm{geo}(0)}{m} \rfloor$, i.e., it first increases n according to a geometric distribution, and then computes the quotient like $\mathcal{R}_{\mathrm{q}}$. Thus, $\iota(\mathcal{R}_1) = \mathrm{Pol}_1$, since $\mathcal{R}_{\mathrm{q}}$ has linear runtime complexity and the geometric distribution only increases n by 2 in expectation. So in particular, $\mathcal{R}_1$ is SAST.

Moreover, consider the PTRS $\mathcal{R}_2$ with the rules:

$$\begin{aligned}
\mathsf{start} &\to \{1 : \mathsf{f}(\mathsf{geo}(0))\}\\
\mathsf{geo}(x) &\to \{1/2 : \mathsf{geo}(\mathsf{s}(x)),\ 1/2 : x\}\\
\mathsf{f}(\mathsf{s}(x)) &\to \{1 : \mathsf{f}(\mathsf{c}(x,x))\}\\
\mathsf{f}(\mathsf{c}(x,y)) &\to \{1 : \mathsf{c}(\mathsf{f}(x),\mathsf{f}(y))\}
\end{aligned}$$

The two f-rules have exponential runtime complexity, as a reduction starting in $\mathsf{f}(\mathsf{s}^n(0))$ creates a full binary tree of height n and visits every inner node once. When beginning with the term start, $\mathcal{R}_2$ first generates the term $\mathsf{f}(\mathsf{s}^k(0))$ with probability $(1/2)^{k+1}$ and then takes at least 2^k steps to terminate. The expected derivation length of the corresponding RST is at least $\sum_{k=0}^{\infty}(1/2)^{k+1} \cdot 2^k = \sum_{k=0}^{\infty} 1/2 = \omega$. Hence, $\iota(\mathcal{R}_2) = \omega$, i.e., $\mathcal{R}_2$ is not SAST.

3 Annotated Dependency Pairs

In Sect. 3.1 we define *annotated dependency pairs*. While such dependency pairs were used to prove AST in [36] and relative termination of TRSs in [33], we now develop a new criterion in order to use them for complexity analysis of PTRSs. Afterwards, in Sect. 3.2 we introduce the general idea of our novel framework in order to derive upper bounds on the expected runtime complexity of PTRSs.

3.1 ADP Problems

The core idea of the *dependency pair framework* for termination of TRSs [3, 21] is the following: *A function is terminating iff the arguments of each recursive function call are decreasing w.r.t. some well-founded ordering.* Hence, for every defined symbol $f \in \mathcal{D}$ one introduces a fresh *tuple* or *annotated* symbol $f^{\sharp}$ that is used to compare the arguments of two successive calls of f. Let $\Sigma^{\sharp} = \Sigma \cup \mathcal{D}^{\sharp}$ with $\mathcal{D}^{\sharp} = \{f^{\sharp} \mid f \in \mathcal{D}\}$, and for any $\Sigma' \subseteq \Sigma \cup \mathcal{V}$, let $\mathrm{Pos}_{\Sigma'}(t)$ be all positions of t with symbols or variables from Σ'. For any $t = f(t_1,\ldots,t_k) \in \mathcal{T}$ with $f \in \mathcal{D}$, let $t^{\sharp} = f^{\sharp}(t_1,\ldots,t_k)$. For termination analysis, one considers each function call in a right-hand side of a rewrite rule on its own, i.e., for each rule $\ell \to r$ with $\mathrm{Pos}_{\mathcal{D}} = \{\pi_1,\ldots,\pi_n\}$, one obtains n *dependency pairs* $\ell^{\sharp} \to r|_{\pi_i}^{\sharp}$ for all $1 \leq i \leq n$. However, for complexity analysis, one has to consider all function calls in a right-hand side simultaneously. Thus, when adapting DPs for complexity analysis in [53], a single *dependency tuple* (DT) $\ell^{\sharp} \to [r|_{\pi_1}^{\sharp},\ldots,r|_{\pi_n}^{\sharp}]$ is constructed instead of the n dependency pairs. By analyzing the dependency tuples (together with the original rewrite rules), [53] presented a modular *DT framework* that can be used to infer an upper bound on the runtime complexity. However, in contrast to the *chain criterion* of dependency pairs (which states that termination of a TRS is equivalent to the absence of infinite chains of DPs), the *chain criterion* of this dependency tuple framework yields an over-approximation. More precisely, the upper bounds on the runtime complexity obtained via dependency tuples are only tight for *confluent* TRSs.

Recently, we introduced *annotated dependency pairs* (ADPs) to analyze *almost-sure termination* of PTRSs [36]. We now show that by using ADPs instead of dependency tuples, the corresponding chain criterion for (expected) complexity becomes an equivalence again, i.e., it can be used to compute *tight* complexity bounds (irrespective of confluence). Instead of extracting the function calls of right-hand sides and coupling them together in a fresh dependency tuple, in ADPs we annotate these function calls in the original rewrite rule directly, i.e., we keep its original structure.

Definition 3.1 (Annotations). For $t \in \mathcal{T}^{\sharp} = \mathcal{T}(\Sigma^{\sharp},\mathcal{V})$ and a set of positions $\Phi \subseteq \mathrm{Pos}_{\mathcal{D}\cup\mathcal{D}^{\sharp}}(t)$, let $\sharp_{\Phi}(t)$ be the variant of t where the symbols at positions from Φ in t are annotated and all other annotations are removed. So $\mathrm{Pos}_{\mathcal{D}^{\sharp}}(\sharp_{\Phi}(t)) = \Phi$ and $\sharp_{\varnothing}(t)$ removes all annotations from t. We often write $\sharp_{\mathcal{D}}(t)$ instead of $\sharp_{\mathrm{Pos}_{\mathcal{D}}(t)}(t)$ to annotate all defined symbols in t, and $\flat(t)$ instead of $\sharp_{\varnothing}(t)$, where we extend $\flat$ to multi-distributions, rules, and sets of rules by removing the annotations of all occurring terms. Moreover, $\flat_{\pi}^{\uparrow}(t)$ results from removing all annotations from t that are strictly above the position π. We write $t \trianglelefteq_{\sharp}^{\pi} s$ if $\pi \in \mathrm{Pos}_{\mathcal{D}^\sharp}(s)$ and $t = \flat(s|_\pi)$, i.e., t results from a subterm of s with annotated root symbol by removing its annotation. If π is not of interest, we just write $t \trianglelefteq_\sharp s$.

We often write F instead of $\mathsf{f}^\sharp$ for $\mathsf{f} \in \mathcal{D}$ (e.g., Geo instead of $\mathsf{geo}^\sharp$).

Example 3.2 (Annotations). If $\mathsf{f} \in \mathcal{D}$, then we have $\sharp_{\{1\}}(\mathsf{f}(\mathsf{f}(x))) = \sharp_{\{1\}}(\mathsf{F}(\mathsf{F}(x))) = \mathsf{f}(\mathsf{F}(x))$, $\sharp_{\mathcal{D}}(\mathsf{f}(\mathsf{f}(x))) = \sharp_{\{\varepsilon,1\}}(\mathsf{f}(\mathsf{f}(x))) = \mathsf{F}(\mathsf{F}(x))$, $\flat(\mathsf{F}(\mathsf{F}(x))) = \mathsf{f}(\mathsf{f}(x))$, $\flat_1^{\uparrow}(\mathsf{F}(\mathsf{F}(x))) = \mathsf{f}(\mathsf{F}(x))$, and $\mathsf{f}(x) \trianglelefteq_\sharp \mathsf{f}(\mathsf{F}(x))$.

The annotations indicate which function calls need to be regarded for complexity analysis. To transform a PTRS into ADPs, initially we annotate all defined symbols in the right-hand sides of rules, since all function calls need to be considered at the start of our analysis. The left-hand side of an ADP is just the left-hand side of the original rule (i.e., in contrast to the DPs of [3, 21], we do not annotate symbols in left-hand sides). The DP and the DT framework work on pairs $\langle \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{R}$ contains the original rewrite rules and $\mathcal{P}$ is the set of dependency pairs or tuples. In contrast, ADPs already represent the original rewrite rules themselves. We simply add a Boolean flag $m \in \{\mathsf{true}, \mathsf{false}\}$ to indicate whether we still need to consider the corresponding original rewrite rule for our analysis. Initially, the flag is true for all ADPs.

Definition 3.3 (Annotated Dependency Pairs). An *annotated dependency pair* (ADP) has the form $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$, where $\ell \in \mathcal{T}$ with $\ell \notin \mathcal{V}$, $m \in \{\mathsf{true}, \mathsf{false}\}$, and for all $1 \leq j \leq k$ we have $r_j \in \mathcal{T}^\sharp$ with $\mathcal{V}(r_j) \subseteq \mathcal{V}(\ell)$.

The *canonical ADP* of a probabilistic rule $\ell \to \mu = \{p_1 : r_1, \ldots, p_k : r_k\}$ is $\mathcal{A}(\ell \to \mu) = \ell \to \{p_1 : \sharp_{\mathcal{D}}(r_1), \ldots, p_k : \sharp_{\mathcal{D}}(r_k)\}^{\mathsf{true}}$. The canonical ADPs of a PTRS $\mathcal{R}$ are $\mathcal{A}(\mathcal{R}) = \{\mathcal{A}(\ell \to \mu) \mid \ell \to \mu \in \mathcal{R}\}$.

For a set of ADPs we can define the defined symbols, constructors, and basic terms as for a TRS, because the left-hand sides of the ADPs are the left-hand sides of the original rewrite rules.

Example 3.4 (Canonical Annotated Dependency Pairs). The canonical ADPs $\mathcal{A}(\mathcal{R}_1)$ and $\mathcal{A}(\mathcal{R}_2)$ of $\mathcal{R}_1$ and $\mathcal{R}_2$ from Ex. 2.5 are:

$$\begin{array}{llr}
\mathcal{A}(\mathcal{R}_1): & \mathsf{start}(x,y) \to \{1 : \mathsf{Q}(\mathsf{Geo}(x), y, y)\}^{\mathsf{true}} & (1)\\
& \mathsf{geo}(x) \to \{1/2 : \mathsf{Geo}(\mathsf{s}(x)), 1/2 : x\}^{\mathsf{true}} & (2)\\
& \mathsf{q}(\mathsf{s}(x), \mathsf{s}(y), z) \to \{1 : \mathsf{Q}(x, y, z)\}^{\mathsf{true}} & (3)\\
& \mathsf{q}(x, \mathsf{0}, \mathsf{s}(z)) \to \{1 : \mathsf{s}(\mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)))\}^{\mathsf{true}} & (4)\\
& \mathsf{q}(\mathsf{0}, \mathsf{s}(y), \mathsf{s}(z)) \to \{1 : \mathsf{0}\}^{\mathsf{true}} & (5)\\
\mathcal{A}(\mathcal{R}_2): & \mathsf{start} \to \{1 : \mathsf{F}(\mathsf{Geo}(\mathsf{0}))\}^{\mathsf{true}} & \\
& \mathsf{geo}(x) \to \{1/2 : \mathsf{Geo}(\mathsf{s}(x)), 1/2 : x\}^{\mathsf{true}} & \\
& \mathsf{f}(\mathsf{s}(x)) \to \{1 : \mathsf{F}(\mathsf{c}(x, x))\}^{\mathsf{true}} & (6)\\
& \mathsf{f}(\mathsf{c}(x, y)) \to \{1 : \mathsf{c}(\mathsf{F}(x), \mathsf{F}(y))\}^{\mathsf{true}} &
\end{array}$$

Since the original rule and all corresponding dependency pairs are encoded in a single ADP, when rewriting with ADPs we have to distinguish whether we mean to rewrite with the original rule or with a dependency pair. This is important as our analysis should only focus on the complexity of rewriting at annotated positions, i.e., of those function calls that we still need to analyze.

Definition 3.5 ($\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{P}}$). Let $\mathcal{P}$ be a finite set of ADPs. A term $s \in \mathcal{T}^\sharp$ rewrites with $\mathcal{P}$ to $\mu = \{p_1 : t_1, \ldots, p_k : t_k\}$ (denoted $s \overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{P}} \mu$) if there is a position $\pi \in \mathrm{Pos}_{\mathcal{D} \cup \mathcal{D}^\sharp}(s)$, a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$, and a substitution σ such that $\flat(s|_\pi) = \ell\sigma \in \mathsf{ANF}_{\mathcal{P}}$ (i.e., all proper subterms are in normal form w.r.t. $\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{P}}$), and for all $1 \leq j \leq k$, t_j is defined as follows, depending on the flag m and on whether $\pi \in \mathrm{Pos}_{\mathcal{D}^\sharp}(s)$ holds:

	$\pi \in \mathrm{Pos}_{\mathcal{D}^\sharp}(s)$	$\pi \notin \mathrm{Pos}_{\mathcal{D}^\sharp}(s)$
$m = \mathsf{true}$	$t_j = \quad s[r_j\sigma]_\pi$ **(at)**	$t_j = \quad s[\flat(r_j)\sigma]_\pi$ **(nt)**
$m = \mathsf{false}$	$t_j = \flat_\pi^{\uparrow}(s[r_j\sigma]_\pi)$ **(af)**	$t_j = \flat_\pi^{\uparrow}(s[\flat(r_j)\sigma]_\pi)$ **(nf)**

Rewriting with $\mathcal{P}$ is like ordinary probabilistic term rewriting while considering and modifying annotations. We distinguish between **a**-steps (**a**nnotation) and **n**-steps (**n**o annotation). Similar to the DP and the DT framework for non-probabilistic TRSs, for complexity we only "count" **a**-steps (on positions with **a**nnotated symbols) that apply dependency pairs, and between two **a**-steps there can be several **n**-steps where rules are applied below the position of the next **a**-step (which evaluate the arguments of the function call to normal forms). The flag $m \in \{\mathsf{true}, \mathsf{false}\}$ indicates whether the ADP may be used for such **n**-steps on the arguments before an **a**-step on an annotated symbol above.

During an **(at)**-step (for annotation and true), all annotations are kept except those in subterms that correspond to variables in the applied rule. Those subterms are normal forms as we consider innermost rewriting. An **(at)**-step at a position π represents an **a**-step as it rewrites at the position of an annotation, but in addition, it can also represent an **n**-step if an annotated symbol is later rewritten at a position above π. An example for an **(at)**-step is:

$$\mathsf{F}(\mathsf{s}(\mathsf{F}(\mathsf{s}(\mathsf{0})))) \overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}(\mathcal{R}_2)} \{1 : \mathsf{F}(\mathsf{s}(\mathsf{F}(\mathsf{c}(\mathsf{0},\mathsf{0}))))\}$$

using ADP (6). Here, we have $\pi = 1.1$, $\flat(s|_{1.1}) = \mathsf{f}(\mathsf{s}(\mathsf{0})) = \ell\sigma$, where σ instantiates x with the normal form $\mathsf{0}$, and $r_1 = \mathsf{F}(\mathsf{c}(\mathsf{0},\mathsf{0}))$.

A step of the form **(nt)** (for **n**o annotation and true) performs a rewrite step at the position of a non-annotated defined symbol. This represents only an **n**-step, and thus all annotations on the right-hand side r_j are removed. An example for such a step is:

$$\mathsf{F}(\mathsf{s}(\mathsf{f}(\mathsf{s}(\mathsf{0})))) \overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}(\mathcal{R}_2)} \{1 : \mathsf{F}(\mathsf{s}(\mathsf{f}(\mathsf{c}(\mathsf{0},\mathsf{0}))))\}$$

using ADP (6). Here, we have the same π, $\ell\sigma$, and σ as above, but use the right-hand side $\flat(r_1) = \mathsf{f}(\mathsf{c}(\mathsf{0},\mathsf{0}))$ without annotations.

An **(af)**-step (for **a**nnotation and **f**alse) at a position π only represents an **a**-step, but not an **n**-step to rewrite the arguments of a function that is evaluated later on an annotated position above. Therefore, we remove all annotations above π, as no **a**-step is allowed to occur above π afterwards. If $\mathcal{A}(\mathcal{R}_2)'$ contains $\mathsf{f}(\mathsf{s}(x)) \to \{1 : \mathsf{F}(\mathsf{c}(x,x))\}^{\mathsf{false}}$, then a step of the form **(af)** would be:

$$\mathsf{F}(\mathsf{s}(\mathsf{F}(\mathsf{s}(\mathsf{0})))) \overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}(\mathcal{R}_2)'} \{1 : \mathsf{f}(\mathsf{s}(\mathsf{F}(\mathsf{c}(\mathsf{0},\mathsf{0}))))\}$$

Finally, a step of the form **(nf)** (for **n**o annotation and **f**alse) is irrelevant for proving an upper bound on the expected runtime complexity since there can never be another **a**-step at a position above. These steps are only included to ensure that the innermost evaluation strategy is not affected if one modifies the annotations or the flag of ADPs (such modifications will be done by our ADP processors in Sect. 4). An example would be

$$\mathsf{F}(\mathsf{s}(\mathsf{f}(\mathsf{s}(\mathsf{0})))) \overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{A}(\mathcal{R}_2)'} \{1 : \mathsf{f}(\mathsf{s}(\mathsf{f}(\mathsf{c}(\mathsf{0},\mathsf{0}))))\}$$

with $\mathsf{f}(\mathsf{s}(x)) \to \{1 : \mathsf{F}(\mathsf{c}(x,x))\}^{\mathsf{false}}$ at Position 1.1 again.

Next, we lift RSTs to so-called *chain trees* that consider rewriting with $\overset{\mathsf{i}}{\hookrightarrow}_{\mathcal{P}}$ instead of $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$.

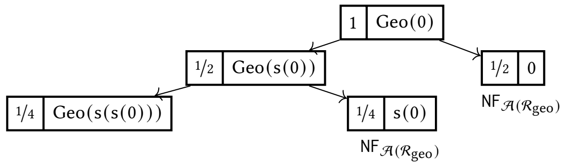

1 Geo(0)
1/2 Geo(s(0))
1/2 0
1/4 Geo(s(s(0)))
1/4 s(0)
$\mathsf{NF}_{\mathcal{A}(\mathcal{R}_{\mathrm{geo}})}$
$\mathsf{NF}_{\mathcal{A}(\mathcal{R}_{\mathrm{geo}})}$

Figure 2: $\mathcal{A}(\mathcal{R}_{\mathrm{geo}})$-Chain tree starting with $\mathrm{Geo}(0)$

Definition 3.6 (Chain Tree). Let $\mathfrak{T} = (V, E, L)$ be a (possibly infinite) labeled and directed tree with nodes $V \neq \varnothing$ and edges $E \subseteq V \times V$, where $vE = \{w \mid (v, w) \in E\}$ is finite for every $v \in V$. We say that $\mathfrak{T}$ is a $\mathcal{P}$*-chain tree* ($\mathcal{P}$-CT) if

- $L : V \longrightarrow (0, 1] \times \mathcal{T}^{\sharp}$ 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 $p_v = 1$.
- If $vE = \{w_1, \dots, w_k\}$, then $t_v \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{\frac{p_{w_1}}{p_v} : t_{w_1}, \dots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$.

For every inner node v, let $\mathcal{P}(v) \in \mathcal{P} \times \{(\mathbf{at}), (\mathbf{af}), (\mathbf{nt}), (\mathbf{nf})\}$ be the ADP and the kind of step used for rewriting t_v.

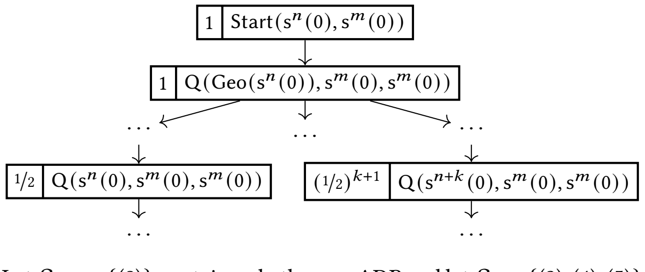
The $\mathcal{A}(\mathcal{R}_{\mathrm{geo}})$-CT in Fig. 2 corresponds to the $\mathcal{R}_{\mathrm{geo}}$-RST in Fig. 1.

In contrast to the *expected derivation length* of RSTs, for the expected derivation length of CTs, we only consider $(\mathbf{at})$- or $(\mathbf{af})$-steps, i.e., steps at the position of an annotated symbol.[3] Moreover, sometimes we do not want to count the application of *all* ADPs, but only the ADPs from some subset. Thus, similar to the adaption of DPs for complexity analysis in [53], in our ADP framework we do not consider a single set of ADPs $\mathcal{P}$, but we use a second set $\mathcal{S} \subseteq \mathcal{P}$ of those ADPs that (still) have to be taken into account. So one only has to add the probabilities of $\mathbf{a}$-steps with ADPs from $\mathcal{S}$ in the chain tree to determine its *expected derivation length.* Thus, our ADP framework uses *ADP problems* $\langle \mathcal{P}, \mathcal{S} \rangle$ where $\mathcal{S} \subseteq \mathcal{P}$, and our analysis ends once we have $\mathcal{S} = \varnothing$. Such ADP problems are called *solved.* In the remainder, we fix an arbitrary ADP problem $\langle \mathcal{P}, \mathcal{S} \rangle$.

Definition 3.7 (Exp. Derivation Length for Chain Trees, $\mathrm{edl}_{\langle \mathcal{P}, \mathcal{S} \rangle}$*).* Let $\mathfrak{T} = (V, E, L)$ be a $\mathcal{P}$-chain tree. The *expected derivation length* of $\mathfrak{T}$, where we only count steps with $\mathcal{S}$ at annotated symbols, is

$$\mathrm{edl}_{\langle \mathcal{P}, \mathcal{S} \rangle}(\mathfrak{T}) = \textstyle\sum_{v \in V \setminus \mathrm{Leaf}^{\mathfrak{T}},\ \mathcal{P}(v) \in \mathcal{S} \times \{(\mathbf{at}), (\mathbf{af})\}}\ p_v$$

Example 3.8 (Expected Derivation Length for Chain Trees). Reconsider the PTRS $\mathcal{R}_1$ and the following $\mathcal{A}(\mathcal{R}_1)$-chain tree $\mathfrak{T}$.


1 Start(s^n(0), s^m(0))
1 Q(Geo(s^n(0)), s^m(0), s^m(0))
...
1/2 Q(s^n(0), s^m(0), s^m(0))
(1/2)^{k+1} Q(s^{n+k}(0), s^m(0), s^m(0))

Let $\mathcal{S}_{\mathrm{geo}} = \{(2)\}$ contain only the geo-ADP and let $\mathcal{S}_{\mathrm{q}} = \{(3), (4), (5)\}$ contain the q-ADPs. Computing the expected derivation length of $\mathfrak{T}$ w.r.t. $\mathcal{S}_{\mathrm{geo}}$ or w.r.t. all ADPs results in

$$\mathrm{edl}_{\langle \mathcal{A}(\mathcal{R}_1), \mathcal{S}_{\mathrm{geo}} \rangle}(\mathfrak{T}) = 1 + \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{8} + \dots = 1 + \textstyle\sum_{k=1}^{\infty} \frac{1}{2^k} = 2$$

$$\begin{aligned} \mathrm{edl}_{\langle \mathcal{A}(\mathcal{R}_1), \mathcal{A}(\mathcal{R}_1) \rangle}(\mathfrak{T}) &= 1 + \mathrm{edl}_{\langle \mathcal{A}(\mathcal{R}_1), \mathcal{S}_{\mathrm{geo}} \rangle}(\mathfrak{T}) + \mathrm{edl}_{\langle \mathcal{A}(\mathcal{R}_1), \mathcal{S}_{\mathrm{q}} \rangle}(\mathfrak{T}) \\ &\leq 1 + 2 + \textstyle\sum_{k=1}^{\infty} (1/2)^{k+1} \cdot (2(n+k) + 2) \\ &= 3 + \textstyle\sum_{k=1}^{\infty} k/2^k + \sum_{k=1}^{\infty} (1/2)^k \cdot (n+1) \\ &= 3 + 2 + 1 \cdot (n+1) = n + 6 \end{aligned}$$

Next, we define expected derivation *height* via chain trees by considering all possible CTs with the root $t^{\sharp}$ for a basic term t, and taking the supremum of their expected derivation lengths.

Definition 3.9 (Exp. Derivation Height via Chain Trees, $\mathrm{edh}_{\langle \mathcal{P}, \mathcal{S} \rangle}$*).* For $t \in \mathcal{TB}_{\mathcal{P}}$, the *expected derivation height* $\mathrm{edh}_{\langle \mathcal{P}, \mathcal{S} \rangle}(t) \in \mathbb{N} \cup \{\omega\}$ is the supremum obtained when adding all probabilities for $\mathbf{a}$-steps with $\mathcal{S}$ in any chain tree $\mathfrak{T}$ with root $t^{\sharp}$:

$$\mathrm{edh}_{\langle \mathcal{P}, \mathcal{S} \rangle}(t) = \sup\{\mathrm{edl}_{\langle \mathcal{P}, \mathcal{S} \rangle}(\mathfrak{T}) \mid \mathfrak{T} \text{ is a } \mathcal{P}\text{-chain tree with } \mathrm{root}(\mathfrak{T}) = t^{\sharp}\}$$

Example 3.10 (Expected Derivation Height w.r.t. Chain Trees). Consider the term $t = \mathrm{start}(\mathrm{s}^n(0), \mathrm{s}^m(0))$ and its corresponding $\mathcal{A}(\mathcal{R}_1)$-CT from Ex. 3.8. As this is the only $\mathcal{A}(\mathcal{R}_1)$-CT with root $t^{\sharp}$, we obtain $\mathrm{edh}_{\langle \mathcal{A}(\mathcal{R}_1), \mathcal{A}(\mathcal{R}_1) \rangle}(t) \leq n + 6$ and $\mathrm{edh}_{\langle \mathcal{A}(\mathcal{R}_1), \mathcal{S}_{\mathrm{geo}} \rangle}(t) = 2$.

Now we can define expected runtime complexity for ADP problems.

Definition 3.11 (Expected Runtime Complexity for ADP Problems, $\mathrm{erc}_{\langle \mathcal{P}, \mathcal{S} \rangle}$*).* The *expected runtime complexity* function of an ADP problem $\langle \mathcal{P}, \mathcal{S} \rangle$ is defined as

$$\mathrm{erc}_{\langle \mathcal{P}, \mathcal{S} \rangle}(n) = \sup\{\mathrm{edh}_{\langle \mathcal{P}, \mathcal{S} \rangle}(t) \mid t \in \mathcal{TB}_{\mathcal{P}}, |t| \leq n\}$$

and we define the *runtime complexity* $\iota_{\langle \mathcal{P}, \mathcal{S} \rangle}$ of $\langle \mathcal{P}, \mathcal{S} \rangle$ as $\iota(\mathrm{erc}_{\langle \mathcal{P}, \mathcal{S} \rangle})$.

Example 3.12 (Expected Runtime Complexity for ADP Problems). For a basic term $\mathrm{start}(t_1, t_2)$, $\mathcal{A}(\mathcal{R}_1)$ first computes a geometric distribution starting in t_1. This needs 2 steps in expectation, and increases t_1 by only 2 in expectation. In the resulting term $\mathrm{q}(t_{\mathrm{geo}}, t_2, t_2)$, where t_{geo} is the normal form resulting from $\mathrm{geo}(t_1)$, we decrease t_{geo} until we reach 0. Therefore, the expected derivation height is linear in the size of the start term, i.e., $\iota_{\langle \mathcal{A}(\mathcal{R}_1), \mathcal{A}(\mathcal{R}_1) \rangle} = \mathrm{Pol}_1$. If we only consider $\mathcal{S}_{\mathrm{geo}}$ for the complexity, then $\iota_{\langle \mathcal{A}(\mathcal{R}_1), \mathcal{S}_{\mathrm{geo}} \rangle} = \mathrm{Pol}_0$.

With our new concepts, we obtain the following novel *chain criterion* for complexity analysis of PTRSs. It shows that to analyze the expected runtime complexity of a PTRS $\mathcal{R}$, it suffices to analyze the expected runtime complexity of its *canonical ADP problem* $\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle$, i.e., in the beginning all ADPs are considered for complexity. In the canonical ADP problem, all defined symbols in right-hand sides are annotated. Thus, one can only perform $(\mathbf{at})$-steps, because due to the innermost strategy, annotations are only removed from subterms in normal form. Hence, the rewrite steps with $\mathcal{R}$ and the ones with $\mathcal{A}(\mathcal{R})$ directly correspond to each other.

Theorem 3.13 (Chain Criterion). *Let $\mathcal{R}$ be a PTRS. Then for all basic terms $t \in \mathcal{TB}_{\mathcal{R}}$ we have*

$$\mathrm{edh}_{\mathcal{R}}(t) = \mathrm{edh}_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}(t)$$

and therefore $\iota_{\mathcal{R}} = \iota_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}$.

In contrast to the chain criterion of [53] for complexity analysis in the non-probabilistic setting, Thm. 3.13 yields a *tight* bound ($\iota_{\mathcal{R}} = \iota_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}$) for arbitrary PTRSs due to the usage of ADPs instead of dependency tuples (with dependency tuples one would only obtain an upper bound, i.e., $\iota_{\mathcal{R}} \leq \iota_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}$).

[3]Since $(\mathbf{nt})$- and $(\mathbf{nf})$-steps are disregarded for the expected derivation length of CTs, in contrast to the chain trees used for proving AST in [36], we do not have to require that every infinite path contains infinitely many $(\mathbf{at})$- or $(\mathbf{af})$-steps.

$$\begin{tikzcd}
& \text{Pol}_1\ \langle\mathcal{P},\mathcal{S}\rangle \arrow[dl] \arrow[dr] & \\
\text{Pol}_2\ \langle\mathcal{P}_1,\mathcal{S}_1\rangle \arrow[d] & & \langle\mathcal{P}_2,\varnothing\rangle\ \text{Pol}_0 \\
\text{Pol}_0\ \langle\mathcal{P}_3,\varnothing\rangle & &
\end{tikzcd}$$

Figure 3: Example of a proof tree

3.2 ADP Framework

Like the original DP framework of [21], our ADP framework is a *divide-and-conquer* approach which applies *processors* to simplify ADP problems until all subproblems are solved. As in [53], a processor also returns a complexity $c \in \mathfrak{C}$.

Definition 3.14 (Processor). An *(ADP) processor* Proc is a function $\text{Proc}(\langle\mathcal{P},\mathcal{S}\rangle) = (c, \{\langle\mathcal{P}_1,\mathcal{S}_1\rangle, \ldots, \langle\mathcal{P}_n,\mathcal{S}_n\rangle\})$ mapping an ADP problem $\langle\mathcal{P},\mathcal{S}\rangle$ to a complexity $c \in \mathfrak{C}$ and a set of ADP problems.

The intuition for Proc is that in addition to the complexities of $\langle\mathcal{P}_i,\mathcal{S}_i\rangle$, the complexity c is also used to obtain an upper bound on the complexity of $\langle\mathcal{P},\mathcal{S}\rangle$. During the analysis with our ADP framework, we construct a *proof tree* that contains all subproblems and complexities resulting from the application of processors.

Definition 3.15 (Proof Tree). A *proof tree* is a labeled, finite tree $(V, E, L_{\mathcal{A}}, L_C)$ with a labeling $L_{\mathcal{A}}$ that maps each node to an ADP problem and a second labeling L_C that maps each node to a complexity from $\mathfrak{C}$. Each edge represents an application of a processor, i.e., if $vE = \{w_1, \ldots, w_n\}$, then $\text{Proc}(L_{\mathcal{A}}(v)) = (L_C(v), \{L_{\mathcal{A}}(w_1), \ldots, L_{\mathcal{A}}(w_n)\})$ for some processor Proc, and we require that the complexity of all leaves v is $L_C(v) = \text{Pol}_0$ if the corresponding ADP problem $L_{\mathcal{A}}(v)$ is solved, and $L_C(v) = \omega$ otherwise. We call a proof tree *solved* if all ADP problems in its leaves are solved.

Fig. 3 shows an example of a solved proof tree where the complexities given by the labeling L_C are depicted in red. So here, a processor with $\text{Proc}(\langle\mathcal{P},\mathcal{S}\rangle) = (\text{Pol}_1, \{\langle\mathcal{P}_1,\mathcal{S}_1\rangle, \langle\mathcal{P}_2,\varnothing\rangle\})$ was used for the step from the root to its two children.

To ensure that the maximum of all complexities in a proof tree is an upper bound on the complexity of the ADP problem at the root, we require that proof trees are *well formed*, i.e., that for every node v with $L_{\mathcal{A}}(v) = \langle\mathcal{P},\mathcal{S}\rangle$, the expected complexity of $\langle\mathcal{P},\mathcal{S}\rangle$ is bounded by the L_C-labels of the subtree starting at v and the L_C-labels on the path from the root to v. However, since no processor has been applied on the leaves of the proof tree (yet), for leaves we use the actual expected complexity instead of the L_C-label. Moreover, for well-formed proof trees we require that the ADPs from $\mathcal{P}\setminus\mathcal{S}$ have already been taken into account in the path from the root to v. This will be exploited, e.g., in the *knowledge propagation processor* of Sect. 4.4.

Definition 3.16 ($\oplus$, Well-Formed Proof Tree). Let $\oplus$ be the maximum operator on complexities, i.e., for $c, d \in \mathfrak{C}$, let $c \oplus d = d$ if $c \sqsubseteq d$ and $c \oplus d = c$ otherwise (so, e.g., $\text{Pol}_2 \oplus \text{Pol}_1 = \text{Pol}_2$).

A proof tree $(V, E, L_{\mathcal{A}}, L_C)$ *well formed* if for every node v with $L_{\mathcal{A}}(v) = \langle\mathcal{P},\mathcal{S}\rangle$ and path $v_1, \ldots, v_k = v$ from the root v_1 to v, we have

$$\iota_{\langle\mathcal{P},\mathcal{S}\rangle} \sqsubseteq L_C(v_1) \oplus \cdots \oplus L_C(v_{k-1}) \oplus \max\{L'_C(w) \mid (v, w) \in E^*\}$$
$$\iota_{\langle\mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle} \sqsubseteq L_C(v_1) \oplus \cdots \oplus L_C(v_{k-1})$$

Here, E^* is the reflexive-transitive closure of the edge relation, i.e., $(v, w) \in E^*$ if v reaches w in the proof tree. Moreover, let $L'_C(v) = L_C(v)$ for inner nodes v and $L'_C(v) = \iota_{L_{\mathcal{A}}(v)}$ for leaves v.

The following theorem shows that for well-formed proof trees, the complexity of the ADP problem at the root is indeed bounded by the maximum of all complexities at the nodes.

Corollary 3.17 (Complexity Bound from Well-Formed Proof Tree). *Let $\mathfrak{P} = (V, E, L_{\mathcal{A}}, L_C)$ be a well-formed proof tree with $L_{\mathcal{A}}(v_1) = \langle\mathcal{P},\mathcal{S}\rangle$ for the root v_1 of $\mathfrak{P}$. Then,*

$$\iota_{\langle\mathcal{P},\mathcal{S}\rangle} \sqsubseteq \max\{L_C(v) \mid v \in V\}.$$

Now we can define when a processor is *sound.*

Definition 3.18 (Soundness of Proc.). A processor $\text{Proc}(\langle\mathcal{P},\mathcal{S}\rangle) = (c, \{\langle\mathcal{P}_1,\mathcal{S}_1\rangle, \ldots, \langle\mathcal{P}_n,\mathcal{S}_n\rangle\})$ is *sound* if for all well-formed proof trees $(V, E, L_{\mathcal{A}}, L_C)$ and all nodes $v \in V$, we have: If $L_{\mathcal{A}}(v) = \langle\mathcal{P},\mathcal{S}\rangle$ and $v_1, \ldots, v_k = v$ is the path from the root node v_1 to v, then

$$\iota_{\langle\mathcal{P},\mathcal{S}\rangle} \sqsubseteq L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1}) \oplus c \oplus \iota_{\langle\mathcal{P}_1,\mathcal{S}_1\rangle} \oplus \ldots \oplus \iota_{\langle\mathcal{P}_n,\mathcal{S}_n\rangle} \quad (7)$$
$$\iota_{\langle\mathcal{P}_i,\mathcal{P}_i\setminus\mathcal{S}_i\rangle} \sqsubseteq L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1}) \oplus c \qquad \text{for all } 1 \leq i \leq n \quad (8)$$

So (7) requires that the complexity of the considered ADPs $\mathcal{S}$ must be bounded by the maximum of all "previous" complexities $L_C(v_1), \ldots, L_C(v_{k-1})$, the newly derived complexity c, and the complexity of the remaining ADP problems $\langle\mathcal{P}_1,\mathcal{S}_1\rangle, \ldots, \langle\mathcal{P}_n,\mathcal{S}_n\rangle$. Moreover, (8) ensures that in the remaining ADP problems $\langle\mathcal{P}_i,\mathcal{S}_i\rangle$, the complexity of the "non-considered" ADPs $\mathcal{P}_i \setminus \mathcal{S}_i$ is bounded by the maximum of all previous complexities and the newly derived complexity c. This ensures that well-formedness of proof trees is preserved when extending them by applying sound processors.

Lemma 3.19 (Sound Processors Preserve Well-Formedness). *Let $\mathfrak{P} = (V, E, L_{\mathcal{A}}, L_C)$ be a proof tree with a leaf v where $L_{\mathcal{A}}(v)$ is not solved, and let* Proc *be a sound processor such that* $\text{Proc}(L_{\mathcal{A}}(v)) = (c, \{\langle\mathcal{P}_1,\mathcal{S}_1\rangle, \ldots, \langle\mathcal{P}_n,\mathcal{S}_n\rangle\})$. *Let $\mathfrak{P}'$ result from $\mathfrak{P}$ by adding fresh nodes $w_1, \ldots, w_n$ and edges $(v, w_1), \ldots, (v, w_n)$, where the labeling is extended such that $L_{\mathcal{A}}(w_i) = \langle\mathcal{P}_i,\mathcal{S}_i\rangle$ for all $1 \leq i \leq n$ and $L_C(v) = c$. Then $\mathfrak{P}'$ is also well formed.*

To determine an upper bound on the expected runtime complexity $\iota_{\mathcal{R}}$ of a PTRS $\mathcal{R}$, our ADP framework starts with the canonical ADP problem $\langle\mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R})\rangle$ and applies sound processors repeatedly until all problems are solved. Then by Thm. 3.13 and Cor. 3.17, the runtime complexity $\iota_{\mathcal{R}}$ is bounded by the maximum of all complexities occurring in the corresponding proof tree.

Corollary 3.20 (Soundness of the ADP Framework for Runtime Complexity). *Let $\mathcal{R}$ be a PTRS and $\mathfrak{P} = (V, E, L_{\mathcal{A}}, L_C)$ be a well-formed solved proof tree where $L_{\mathcal{A}}(v_1) = \langle\mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R})\rangle$ for the root v_1 of $\mathfrak{P}$. Then we have*

$$\iota_{\mathcal{R}} \sqsubseteq \max\{L_C(v) \mid v \in V\}.$$

Remark 3.21. While our framework is inspired by the DT framework of [53] for complexity analysis of non-probabilistic TRSs, our adaption to PTRSs differs from [53] in several aspects. Apart from using ADPs instead of dependency tuples (which results in a "tight" chain criterion instead of an over-approximation), we use proof trees (instead of just proof chains, which allows us to use processors that return several subproblems), and we introduced the novel concept of *well-formed* proof trees and require that sound processors preserve well-formedness. This will allow us to define a *knowledge propagation processor* in Sect. 4.4 which takes the knowledge provided by well-formed proof trees into account. In contrast to [53], we obtain such a processor without extending our ADP problems by an additional component $\mathcal{K}$ that contains those dependency tuples which were already taken into account in the proof up to now, since in our setting we would always have $\mathcal{K} = \mathcal{P} \setminus \mathcal{S}$.

4 ADP Processors

In this section, we adapt the main processors of the DP and the DT framework [21, 53] in order to analyze expected runtime complexity of probabilistic TRSs. Throughout the section, we illustrate our processors with the PTRS $\mathcal{R}_1$ from Ex. 2.5. The resulting solved proof tree for the initial ADP problem $\langle \mathcal{A}(\mathcal{R}_1), \mathcal{A}(\mathcal{R}_1) \rangle$ that we construct during the section is depicted in Fig. 6.

4.1 Usable Rules Processor

We start with a processor that considers the *usable rules*. Usable rules over-approximate the set of those rules that can be used to evaluate the arguments of annotated function symbols if their variables are instantiated by normal forms (as required in innermost evaluations). Essentially, the usable rules of a term t consist of all rules for the defined symbols f occurring in t and all rules that are usable for the terms in the right-hand sides of f-rules.

Definition 4.1 (Usable Rules). For every $f \in \Sigma^{\#}$ and set of ADPs $\mathcal{P}$, let $\mathrm{Rules}_{\mathcal{P}}(f) = \{\ell \to \mu^{\mathrm{true}} \in \mathcal{P} \mid \mathrm{root}(\ell) = f\}$. Moreover, for every $t \in \mathcal{T}^{\#}$, the *usable rules* $\mathcal{U}_{\mathcal{P}}(t)$ of t w.r.t. $\mathcal{P}$ are defined as:

$$\begin{aligned} \mathcal{U}_{\mathcal{P}}(t) &= \varnothing, \qquad\qquad \text{if } t \in \mathcal{V} \text{ or } \mathcal{P} = \varnothing \\ \mathcal{U}_{\mathcal{P}}(f(t_1, \dots, t_n)) &= \mathrm{Rules}_{\mathcal{P}}(f) \cup \textstyle\bigcup_{1 \le j \le n} \mathcal{U}_{\mathcal{P}'}(t_j) \\ &\quad \cup \textstyle\bigcup_{\ell \to \mu^{\mathrm{true}} \in \mathrm{Rules}_{\mathcal{P}}(f),\, r \in \mathrm{Supp}(\mu)} \mathcal{U}_{\mathcal{P}'}(\flat(r)) \end{aligned}$$

where $\mathcal{P}' = \mathcal{P} \setminus \mathrm{Rules}_{\mathcal{P}}(f)$. The usable rules of $\mathcal{P}$ are

$$\mathcal{U}(\mathcal{P}) = \textstyle\bigcup_{\ell \to \mu^m \in \mathcal{P},\, r \in \mathrm{Supp}(\mu),\, t \trianglelefteq_{\#} r} \mathcal{U}_{\mathcal{P}}(t^{\#}).$$

Similar to the usable rules processor for AST in [36], our usable rules processor sets the flag of all non-usable rules in $\mathcal{P}$ to false to indicate that they cannot be used to evaluate arguments of annotated functions that are rewritten afterwards. The rules in $\mathcal{P}$'s subset $\mathcal{S}$ are changed analogously (since the purpose of $\mathcal{S}$ is only to indicate which ADPs must still be counted for complexity).

Theorem 4.2 (Usable Rules Pr.). *For an ADP problem $\langle \mathcal{P}, \mathcal{S} \rangle$, let*

$$\begin{aligned} \mathcal{P}' &= \mathcal{U}(\mathcal{P}) \cup \{\ell \to \mu^{\mathrm{false}} \mid \ell \to \mu^m \in \mathcal{P} \setminus \mathcal{U}(\mathcal{P})\}, \\ \mathcal{S}' &= (\mathcal{S} \cap \mathcal{U}(\mathcal{P})) \cup \{\ell \to \mu^{\mathrm{false}} \mid \ell \to \mu^m \in \mathcal{S} \setminus \mathcal{U}(\mathcal{P})\}. \end{aligned}$$

Then, $\mathrm{Proc}_{\mathsf{UR}}(\langle \mathcal{P}, \mathcal{S} \rangle) = (\mathrm{Pol}_0, \{\langle \mathcal{P}', \mathcal{S}' \rangle\})$ *is sound.*

Example 4.3 (Usable Rules Processor). Consider the ADP problem $\langle \mathcal{A}(\mathcal{R}_1), \mathcal{A}(\mathcal{R}_1) \rangle$ from Ex. 3.4. There is only one term r in the right-hand sides with an annotated subterm $t \trianglelefteq_{\#} r$ where $t^{\#}$ has a defined symbol below the annotated root, viz. $t^{\#} = \mathsf{Q}(\mathsf{geo}(x), y, y)$. Thus, only the geo-ADP (2) is usable and we can set the flag of all other ADPs to false. Hence, we get $\mathrm{Proc}_{\mathsf{UR}}(\langle \mathcal{A}(\mathcal{R}_1), \mathcal{A}(\mathcal{R}_1) \rangle) = (\mathrm{Pol}_0, \{\langle \mathcal{P}_1, \mathcal{P}_1 \rangle\})$ where $\mathcal{P}_1 = \{(9) - (13)\}$ with

$$\mathsf{start}(x, y) \to \{1 : \mathsf{Q}(\mathsf{Geo}(x), y, y)\}^{\mathrm{false}} \tag{9}$$
$$\mathsf{geo}(x) \to \{1/2 : \mathsf{Geo}(\mathsf{s}(x)),\ 1/2 : x\}^{\mathrm{true}} \tag{10}$$
$$\mathsf{q}(\mathsf{s}(x), \mathsf{s}(y), z) \to \{1 : \mathsf{Q}(x, y, z)\}^{\mathrm{false}} \tag{11}$$
$$\mathsf{q}(x, 0, \mathsf{s}(z)) \to \{1 : \mathsf{s}(\mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)))\}^{\mathrm{false}} \tag{12}$$
$$\mathsf{q}(0, \mathsf{s}(y), \mathsf{s}(z)) \to \{1 : 0\}^{\mathrm{false}} \tag{13}$$

Fewer rules with the flag true have advantages, e.g., for the *dependency graph* and the *reduction pair processor*, see Sect. 4.2 and 4.3.

Example 4.4 (Basic Start Terms). The restriction to basic start terms is not only required to infer polynomial upper bounds from CPIs (see Sect. 2.1), but it is also essential for the soundness of the usable rules processor. To see this, let $\mathcal{R}_3$ contain all rules of $\mathcal{R}_2$ from Ex. 2.5 except $\mathsf{start} \to \{1 : \mathsf{f}(\mathsf{geo}(0))\}$. If we do not require basic start terms, then we can start an evaluation with the term $\mathsf{f}(\mathsf{geo}(0))$, i.e., then $\mathcal{R}_3$ is not SAST. The canonical ADPs $\mathcal{A}(\mathcal{R}_3)$ are the same as for $\mathcal{R}_2$ in Ex. 3.4 just without the start-ADP. Thus, $\mathcal{A}(\mathcal{R}_3)$ has no usable rules and $\mathrm{Proc}_{\mathsf{UR}}$ sets the flag of all ADPs to false. When starting with a term like $\mathsf{F}(\mathsf{Geo}(0))$, then one application of the geo-ADP now removes the annotations of the F-symbols above it, as the geo-ADP now has the flag false. So for the resulting ADP problem, only Geo is annotated in chain trees and thus, they all have finite expected derivation length. Hence, we would now falsely infer that $\mathcal{R}_3$ is SAST w.r.t. arbitrary start terms.

4.2 Dependency Graph Processor

The *dependency graph* is a control flow graph that indicates which function calls can occur after each other. This does not depend on the probabilities, and we can consider each function call on its own. Hence, we can use the ordinary dependency graph of the corresponding (non-probabilistic) dependency pairs. To also detect the predecessors of ADPs $\alpha : \ \ell \to \mu$ without annotations, we add a dependency pair $\ell^{\#} \to \bot$ for a fresh symbol $\bot$ in that case.

Definition 4.5 (Non-Probabilistic Variant, Dependency Pairs). For a set of ADPs $\mathcal{P}$, let $\mathrm{np}(\mathcal{P}) = \{\ell \to \flat(r_j) \mid \ell \to \{p_1 : r_1, \dots, p_k : r_k\}^{\mathrm{true}} \in \mathcal{P}, 1 \le j \le k\}$ denote its *non-probabilistic rule variant.* So $\mathrm{np}(\mathcal{P})$ is an ordinary TRS which only considers the ADPs with the flag true. For any ADP $\alpha = \ell \to \{p_1 : r_1, \dots, p_k : r_k\}^m$, let $\mathrm{dp}(\alpha) = \{\ell^{\#} \to t^{\#} \mid 1 \le j \le k, t \trianglelefteq_{\#} r_j\}$. Moreover, let $\mathrm{dp}^{\bot}(\alpha) = \{\ell^{\#} \to \bot\}$ if $\mathrm{dp}(\alpha) = \varnothing$, and $\mathrm{dp}^{\bot}(\alpha) = \mathrm{dp}(\alpha)$, otherwise. For $\mathcal{P}$, let $\mathrm{dp}(\mathcal{P}) = \bigcup_{\alpha \in \mathcal{P}} \mathrm{dp}^{\bot}(\alpha)$ denote its *non-probabilistic DP variant*, which is a set of dependency pairs as in [3, 21].

So each dependency pair from $\mathrm{dp}(\alpha)$ corresponds to a single annotation on the right-hand side of the ADP α. In the dependency graph, the edges indicate whether one DP can follow another when the instantiated arguments are evaluated with $\mathrm{np}(\mathcal{P})$.

Definition 4.6 (Dependency Graph). The $\mathcal{P}$*-dependency graph* has the nodes $\mathrm{dp}(\mathcal{P})$ and there is an edge from $\ell_1^{\#} \to t_1^{\#}$ to $\ell_2^{\#} \to \dots$ if there are substitutions σ_1, σ_2 such that $t_1^{\#}\sigma_1 \xrightarrow{\mathrm{i}}{}^{*}_{\mathrm{np}(\mathcal{P})} \ell_2^{\#}\sigma_2$ and both $\ell_1^{\#}\sigma_1$ and $\ell_2^{\#}\sigma_2$ are in argument normal form, i.e., $\ell_1^{\#}\sigma_1, \ell_2^{\#}\sigma_2 \in \mathsf{ANF}_{\mathcal{P}}$.

While the dependency graph is not computable in general, several techniques have been developed to compute over-approximations of the graph automatically, e.g., [3, 21, 26].

Example 4.7 (Dependency Graph). We continue with $\langle \mathcal{P}_1, \mathcal{P}_1 \rangle$ from Ex. 4.3, where $\mathcal{P}_1 = \{(9) - (13)\}$. We have

$$\mathrm{dp}(\mathcal{P}_1) = \{\ \mathsf{Start}(x, y) \to \mathsf{Q}(\mathsf{geo}(x), y, y), \tag{14}$$
$$\mathsf{Start}(x, y) \to \mathsf{Geo}(x), \tag{15}$$
$$\mathsf{Geo}(x) \to \mathsf{Geo}(\mathsf{s}(x)), \tag{16}$$
$$\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathsf{Q}(x, y, z), \tag{17}$$
$$\mathsf{Q}(x, 0, \mathsf{s}(z)) \to \mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)), \tag{18}$$
$$\mathsf{Q}(0, \mathsf{s}(y), \mathsf{s}(z)) \to \bot\ \}. \tag{19}$$

The $\mathcal{P}_1$-dependency graph is depicted in Fig. 4.

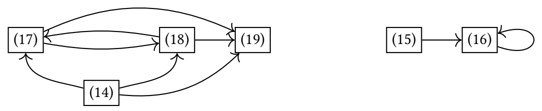


Figure 4: $\mathcal{P}_1$-dependency graph

The idea of the *dependency graph processor* for termination analysis is to analyze each strongly connected component (SCC)[4] of the dependency graph separately. However (already in the non-probabilistic setting, e.g., [53]), this is not possible when analyzing complexity. There are examples where all SCCs have linear complexity but the full system has quadratic complexity, or where all individual SCCs are SAST but the full system is not (see Ex. 4.11).

The problem is that when considering an SCC individually, then we lose the information how often and with which instantiations of the variables this SCC is "called". For that reason, we now present a novel dependency graph processor which regards each SCC *together with its "prefix"*, i.e., together with all nodes of the dependency graph that can reach the SCC. As usual, we say that a node *reaches* an SCC if there is a path from the node to the SCC in the dependency graph (where the path has length ≥ 0, i.e., each node also reaches itself). However, prefixes which are independent from each other can be regarded separately, i.e., we only regard SCC-prefixes $\mathcal{J}$ where for all nodes $\alpha, \beta \in \mathcal{J}$, α reaches β or β reaches α.

Definition 4.8 (SCC-Prefix). Let $\mathcal{P}$ be a set of ADPs. Then $\mathcal{J}$ is an *SCC-prefix* of the $\mathcal{P}$-dependency graph if there exists an SCC $\mathcal{G} \subseteq \mathcal{J}$ where $\mathcal{J} \subseteq \mathrm{dp}(\mathcal{P})$ is a maximal set such that all DPs of $\mathcal{J}$ reach $\mathcal{G}$ and for all $\alpha, \beta \in \mathcal{J}$, α reaches β or β reaches α.

For example, the $\mathcal{P}_1$-dependency graph of Fig. 4 has two SCCs $\{(16)\}$ and $\{(17), (18)\}$, and two SCC-prefixes $\mathcal{J}_1 = \{(15), (16)\}$ and $\mathcal{J}_2 = \{(14), (17), (18)\}$. In this example, $\mathcal{J}_1$ and $\mathcal{J}_2$ represent two completely independent parts of the dependency graph.

The dependency graph processor now handles each SCC-prefix $\mathcal{J}$ separately. To consider only the effects of the DPs $\mathcal{J}$ in the ADPs of $\mathcal{P}$, we replace every ADP α by the variant $\alpha|_{\mathcal{J}}$ where only those symbols are annotated that correspond to the DPs from $\mathcal{J}$. For example, for $\mathcal{J}_1 = \{(15), (16)\}$ and the ADP

$$\alpha = \qquad \mathsf{start}(x, y) \to \{1 : \mathsf{Q}(\mathsf{Geo}(x), y, y)\}^{\mathsf{false}}, \tag{9}$$

in $\alpha|_{\mathcal{J}_1}$ we only annotate Geo but not Q in the right-hand side, due to the DP (15) which results from its subterm $\mathsf{Geo}(x)$, i.e., $\alpha|_{\mathcal{J}_1} = \mathsf{start}(x, y) \to \{1 : \mathsf{q}(\mathsf{Geo}(x), y, y)\}^{\mathsf{false}}$.

Theorem 4.9 (Dependency Graph Proc.). *Let $\langle \mathcal{P}, \mathcal{S} \rangle$ be an ADP problem and let $\mathcal{J}$ be an SCC-prefix of the $\mathcal{P}$-dependency graph. For any ADP $\alpha = \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$ let $\alpha|_{\mathcal{J}} = \ell \to \{p_1 : \sharp_{\Phi_1}(r_1), \ldots, p_k : \sharp_{\Phi_k}(r_k)\}^m$ where for $1 \leq j \leq k$, we have $\pi \in \Phi_j$ iff there exists an $\ell^\sharp \to t^\sharp \in \mathcal{J}$ such that $t \trianglelefteq_\sharp^\pi r_j$. Similarly, let $\mathcal{P}|_{\mathcal{J}} = \{\alpha|_{\mathcal{J}} \mid \alpha \in \mathcal{P}\}$ and $\mathcal{S}|_{\mathcal{J}} = \{\alpha|_{\mathcal{J}} \mid \alpha \in \mathcal{S}\}$.*[5]

Then $\mathrm{Proc}_{\mathtt{DG}}(\langle \mathcal{P}, \mathcal{S} \rangle) = (\mathrm{Pol}_0, \{\langle \mathcal{P}|_{\mathcal{J}}, \mathcal{S}|_{\mathcal{J}} \rangle \mid \mathcal{J}$ *is an SCC-prefix of the $\mathcal{P}$-dependency graph* $\})$ *is sound.*

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

[5]Note that if a DP from $\mathcal{J}$ could originate from several subterms $r_j|_\pi$ in several ADPs α, then we would annotate all their positions in $\mathcal{P}|_{\mathcal{J}}$. A slightly more powerful variant of the processor could be obtained by storing for every DP of $\mathcal{J}$ from which subterm of which ADP it originates, and only annotating this position. We did not do this in Thm. 4.9 to ease the presentation.

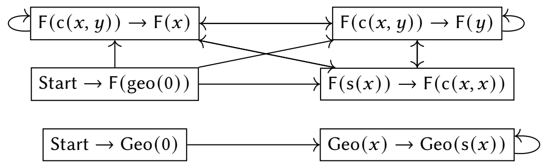


Figure 5: $\mathcal{A}(\mathcal{R}_2)$-dependency graph

Example 4.10 (Dependency Graph Proc.). Due to the two SCC-prefixes $\mathcal{J}_1 = \{(15), (16)\}$ and $\mathcal{J}_2 = \{(14), (17), (18)\}$, the dependency graph processor transforms the ADP problem $\langle \mathcal{P}_1, \mathcal{P}_1 \rangle$ from Ex. 4.3 into $\langle \mathcal{P}_1|_{\mathcal{J}_1}, \mathcal{P}_1|_{\mathcal{J}_1} \rangle$ (corresponding to the Geo-SCC-Prefix) and $\langle \mathcal{P}_1|_{\mathcal{J}_2}, \mathcal{P}_1|_{\mathcal{J}_2} \rangle$ (corresponding to the Q-SCC-Prefix) with

$$\mathcal{P}_1|_{\mathcal{J}_1}: \qquad \mathsf{start}(x, y) \to \{1 : \mathsf{q}(\mathsf{Geo}(x), y, y)\}^{\mathsf{false}} \tag{20}$$
$$\mathsf{geo}(x) \to \{1/2 : \mathsf{Geo}(\mathsf{s}(x)),\ 1/2 : x\}^{\mathsf{true}} \tag{21}$$
$$\mathsf{q}(\mathsf{s}(x), \mathsf{s}(y), z) \to \{1 : \mathsf{q}(x, y, z)\}^{\mathsf{false}} \tag{22}$$
$$\mathsf{q}(x, 0, \mathsf{s}(z)) \to \{1 : \mathsf{s}(\mathsf{q}(x, \mathsf{s}(z), \mathsf{s}(z)))\}^{\mathsf{false}} \tag{23}$$
$$\mathsf{q}(0, \mathsf{s}(y), \mathsf{s}(z)) \to \{1 : 0\}^{\mathsf{false}} \tag{24}$$

$$\mathcal{P}_1|_{\mathcal{J}_2}: \qquad \mathsf{start}(x, y) \to \{1 : \mathsf{Q}(\mathsf{geo}(x), y, y)\}^{\mathsf{false}} \tag{25}$$
$$\mathsf{geo}(x) \to \{1/2 : \mathsf{geo}(\mathsf{s}(x)),\ 1/2 : x\}^{\mathsf{true}} \tag{26}$$
$$\mathsf{q}(\mathsf{s}(x), \mathsf{s}(y), z) \to \{1 : \mathsf{Q}(x, y, z)\}^{\mathsf{false}} \tag{27}$$
$$\mathsf{q}(x, 0, \mathsf{s}(z)) \to \{1 : \mathsf{s}(\mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)))\}^{\mathsf{false}} \tag{28}$$
$$\mathsf{q}(0, \mathsf{s}(y), \mathsf{s}(z)) \to \{1 : 0\}^{\mathsf{false}} \tag{29}$$

Our novel dependency graph processor subsumes several previous processors from the literature, like the "leaf removal processor" of [53]. Leaves of the dependency graph like (19) are not part of any SCC. Hence, they are never contained in SCC-prefixes and thus, the annotations that only correspond to such leaves are always removed. For a similar reason, $\mathrm{Proc}_{\mathtt{DG}}$ from Thm. 4.9 subsumes the "rhs simplification processor" of [53] and the related "usable terms processor" of [36], both of which share the same underlying idea.

Ex. 4.11 shows that only considering SCCs without prefixes would be unsound for analyzing complexity and proving SAST.

Example 4.11 (Combining Non-Connected SCCs may Increase Complexity). Recall $\mathcal{R}_2$ from Ex. 2.5 which is not SAST and $\mathcal{A}(\mathcal{R}_2)$ from Ex. 3.4. The $\mathcal{A}(\mathcal{R}_2)$-dependency graph is depicted in Fig. 5. When only considering the SCCs $\mathcal{G}_{\mathrm{Geo}}$ (containing the Geo-DP) and $\mathcal{G}_{\mathsf{F}}$ (containing the three F-DPs), then we could falsely "prove" SAST since $\iota_{\langle \mathcal{A}(\mathcal{R}_2)|_{\mathcal{G}_{\mathrm{Geo}}}, \mathcal{A}(\mathcal{R}_2)|_{\mathcal{G}_{\mathrm{Geo}}} \rangle} = \mathrm{Pol}_0$ and $\iota_{\langle \mathcal{A}(\mathcal{R}_2)|_{\mathcal{G}_{\mathsf{F}}}, \mathcal{A}(\mathcal{R}_2)|_{\mathcal{G}_{\mathsf{F}}} \rangle} = \mathrm{Exp}$. The problem is that for the F-SCC $\mathcal{G}_{\mathsf{F}}$, one also has to consider the DP $\mathsf{Start} \to \mathsf{F}(\mathsf{geo}(0))$ which determines with which instantiations of the variables the F-SCC is called (i.e., it ensures that the geo-rules become usable). Indeed, we have $\iota(\langle \mathcal{A}(\mathcal{R}_2), \mathcal{A}(\mathcal{R}_2) \rangle) = \omega$.

4.3 Reduction Pair Processor

Now we lift the direct application of polynomial interpretations explained in Sect. 2.1 to an ADP processor for complexity analysis of PTRSs, which allows us to apply polynomial interpretations $\mathcal{I}$ in a modular way. As in the classical DP approach [3, 21], here it suffices if $\mathcal{I}$ is *weakly* monotonic, i.e., if $x \geq y$ implies $\mathcal{I}_f(\ldots, x, \ldots) \geq \mathcal{I}_f(\ldots, y, \ldots)$ for all $f \in \Sigma$ and $x, y \in \mathbb{N}$. Moreover, as in [7, 36], to ensure "weak monotonicity" w.r.t. expected values we restrict ourselves to interpretations with multilinear polynomials, where all monomials have the form $c \cdot x_1^{e_1} \cdot \ldots \cdot x_k^{e_k}$ with $c \in \mathbb{N}$ and $e_1, \ldots, e_k \in \{0, 1\}$.

The *reduction pair processor* imposes three requirements on $\mathcal{I}$:
(1) All rules with the flag true must be weakly decreasing in expectation when removing all annotations. Due to weak monotonicity, this ensures that evaluating the arguments of a function call (i.e., applying the rule in a context) also decreases weakly in expectation.
(2) All ADPs must be weakly decreasing when comparing the annotated left-hand side $\mathcal{I}(\ell^\sharp)$ with the expected value of the annotated subterms of the right-hand side $\{p_1 : r_1, \ldots, p_k : r_k\}$. To measure the value of a term r_j, here we consider all its subterms $t \trianglelefteq_\sharp r_j$ at annotated positions and add the polynomial interpretations of all such $t^\sharp$, i.e., we consider $\mathcal{I}^\sharp_\Sigma(r_j) = \sum_{t \trianglelefteq_\sharp r_j} \mathcal{I}(t^\sharp)$. Regarding this sum instead of the interpretation $\mathcal{I}(r_j)$ of the whole term r_j is the reason why only need *weak* monotonicity.
(3) Finally, the processor removes all strictly decreasing ADPs from the component $\mathcal{S}$ of the ADP problem. However, the ADPs are still kept in $\mathcal{P}$, because they may still be used in reductions. Moreover, if $\mathcal{I}$ is a CPI, then the processor infers a polynomial bound corresponding to the degrees of the polynomials used for annotated symbols. Otherwise, it only infers an exponential bound (which is still useful when analyzing SAST).

Theorem 4.12 (Reduction Pair Proc.). *Let $\mathcal{I} : \Sigma^\sharp \to \mathbb{N}(\mathcal{V})$ be a weakly monotonic, multilinear polynomial interpretation. Let $\langle \mathcal{P}, \mathcal{S} \rangle$ be an ADP problem where $\mathcal{P} = \mathcal{P}_\geq \uplus \mathcal{P}_>$ and $\mathcal{P}_> \cap \mathcal{S} \neq \varnothing$ such that:*
(1) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{\mathsf{true}} \in \mathcal{P}$: $\mathcal{I}(\ell) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathcal{I}(\flat(r_j))$
(2) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}_\geq$: $\mathcal{I}(\ell^\sharp) \geq \sum_{1 \leq j \leq k} p_j \cdot \mathcal{I}^\sharp_\Sigma(r_j)$
(3) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}_>$: $\mathcal{I}(\ell^\sharp) > \sum_{1 \leq j \leq k} p_j \cdot \mathcal{I}^\sharp_\Sigma(r_j)$, where $\mathcal{I}^\sharp_\Sigma(r_j) = \sum_{t \trianglelefteq_\sharp r_j} \mathcal{I}(t^\sharp)$.

Then $\mathrm{Proc}_{\mathsf{RP}}(\langle \mathcal{P}, \mathcal{S} \rangle) = (c, \langle \mathcal{P}, \mathcal{S} \setminus \mathcal{P}_> \rangle)$ is sound, where the complexity $c \in \mathfrak{C}$ is determined as follows: If $\mathcal{I}$ is a CPI and for all annotated symbols $f^\sharp \in \mathcal{D}^\sharp$, the polynomial $I_{f^\sharp}$ has at most degree a, then $c = \mathrm{Pol}_a$. If $\mathcal{I}$ is not a CPI, then $c = \mathrm{Exp}$ if all constructors are interpreted by linear polynomials, and otherwise $c = 2\text{-}\mathrm{Exp}$.

In contrast to the reduction pair processor for proving AST from [36], our processor cannot remove any annotation from $\mathcal{P}$. The reason is that the ADPs from $\mathcal{P}_>$ may still be needed to reach all annotated terms that are relevant for the complexity of $\mathcal{P}_\geq$. (This problem already occurs when analyzing complexity in the non-probabilistic setting, see, e.g., [53, Ex. 25].) But we can remove strictly decreasing ADPs from $\mathcal{S}$ and, therefore, do not have to count them anymore for the complexity. The complexity of the removed ADPs is accounted for by $c \in \{\mathrm{Pol}_a, \mathrm{Exp}, 2\text{-}\mathrm{Exp}\}$.

Example 4.13 (Reduction Pair Processor). Consider $\langle \mathcal{P}_1|_{\mathcal{J}_1}, \mathcal{P}_1|_{\mathcal{J}_1} \rangle$ from Ex. 4.10, and a polynomial interpretation $\mathcal{I}$ with $\mathcal{I}_{\mathsf{Start}}(x, y) = 2$, $\mathcal{I}_{\mathsf{Geo}}(x) = \mathcal{I}_{\mathsf{Q}} = 1$, and $\mathcal{I}_{\mathsf{s}}(x) = \mathcal{I}_{\mathsf{geo}}(x) = x + 1$. Then $\mathsf{geo}(x) \to \{1/2 : \mathsf{Geo}(\mathsf{s}(x)), 1/2 : x\}^{\mathsf{true}}$ (21) is weakly decreasing in expectation when disregarding annotations, since $\mathcal{I}(\mathsf{geo}(x)) = x + 1 = 1/2 \cdot \mathcal{I}(\mathsf{geo}(\mathsf{s}(x))) + 1/2 \cdot \mathcal{I}(x)$. Moreover, when regarding annotations, then all ADPs are strictly decreasing: For (21) we have $\mathcal{I}(\mathsf{Geo}(x)) = 1 > 1/2 = 1/2 \cdot \mathcal{I}(\mathsf{Geo}(\mathsf{s}(x)))$, for $\mathsf{start}(x, y) \to \{1 : \mathsf{q}(\mathsf{Geo}(x), y, y)\}^{\mathsf{false}}$ (20) we have $\mathcal{I}(\mathsf{Start}(x, y)) = 2 > 1 = \mathcal{I}(\mathsf{Geo}(x))$, and for (22)-(24) we have $\mathcal{I}(\mathsf{Q}(\ldots)) = 1 > 0$ (as their right-hand sides do not contain annotations). Since $\mathcal{I}$ is a CPI which interprets all annotated symbols as constants, we obtain $\mathrm{Proc}_{\mathsf{RP}}(\langle \mathcal{P}_1|_{\mathcal{J}_1}, \mathcal{P}_1|_{\mathcal{J}_1} \rangle) = (\mathrm{Pol}_0, \langle \mathcal{P}_1|_{\mathcal{J}_1}, \varnothing \rangle)$, i.e., a solved ADP problem.

For the other ADP problem $\langle \mathcal{P}_1|_{\mathcal{J}_2}, \mathcal{P}_1|_{\mathcal{J}_2} \rangle$ from Ex. 4.10, we use a polynomial interpretation with $\mathcal{I}_{\mathsf{Start}}(x, y) = x + 3$, $\mathcal{I}_{\mathsf{Geo}}(x) = 1$, and $\mathcal{I}_{\mathsf{Q}}(x, y, z) = \mathcal{I}_{\mathsf{s}}(x) = \mathcal{I}_{\mathsf{geo}}(x) = x + 1$. Then (26) is again weakly decreasing when disregarding annotations. When regarding the annotations, then the ADP (28) is weakly decreasing (since $\mathcal{I}(\mathsf{Q}(x, 0, \mathsf{s}(z))) = x + 1 = \mathcal{I}(\mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)))$), and all other ADPs are strictly decreasing. Since $\mathcal{I}$ is a CPI where Start and Q are interpreted as linear polynomials, we get $\mathrm{Proc}_{\mathsf{RP}}(\langle \mathcal{P}_1|_{\mathcal{J}_2}, \mathcal{P}_1|_{\mathcal{J}_2} \rangle) = (\mathrm{Pol}_1, \langle \mathcal{P}_1|_{\mathcal{J}_2}, \{(28)\} \rangle)$. However, there is no polynomial interpretation which orients (28) strictly and the other ADPs weakly. Thus, we need another processor to solve the remaining problem.

4.4 Knowledge Propagation Processor

The dependency graph can not only be used to decompose an ADP problem $\langle \mathcal{P}, \mathcal{S} \rangle$ according to the SCC-prefixes via the dependency graph processor, but it can also be used to remove an ADP α from $\mathcal{S}$ if all "*predecessors*" of α have already been taken into account. More precisely, let $\mathrm{Pre}(\alpha) \subseteq \mathcal{P}$ contain all ADPs that can "generate" a redex for a step with α at an annotated position, i.e., $\mathrm{Pre}(\alpha)$ consists of all ADPs $\beta \in \mathcal{P}$ such that there is an edge from some DP in $\mathrm{dp}^\perp(\beta)$ to some DP in $\mathrm{dp}^\perp(\alpha)$ in the $\mathcal{P}$-dependency graph. Note that $\mathrm{dp}^\perp(\alpha) \neq \varnothing$ for all ADPs α. If d is the maximal number of annotated symbols in any term on a right-hand side of an ADP from $\mathcal{P}$, then in any $\mathcal{P}$-chain tree $\mathfrak{T}$, the probabilities of α-steps can be over-approximated as follows. Except for the very first step, every **(at)**- or **(af)**-step with α is preceded by a step with some ADP β from $\mathrm{Pre}(\alpha)$. Every term in β's right-hand side can trigger at most d α-steps. If the β-step had probability p, then adding all probabilities for these α-steps yields at most $d \cdot p$. Since the very first step of the tree might also be an α-step, one obtains

$$\begin{aligned} \mathrm{edl}_{\langle \mathcal{P}, \{\alpha\} \rangle}(\mathfrak{T}) &\leq 1 + \textstyle\sum_{v \in V \setminus \mathrm{Leaf}^{\mathfrak{T}},\, \mathcal{P}(v) \in \mathrm{Pre}(\alpha) \times \{\mathbf{(at)}, \mathbf{(af)}\}} d \cdot p_v \\ &= 1 + d \cdot \mathrm{edl}_{\langle \mathcal{P}, \mathrm{Pre}(\alpha) \rangle}(\mathfrak{T}). \end{aligned}$$

This in turn implies $\mathrm{erc}_{\langle \mathcal{P}, \{\alpha\} \rangle}(n) \leq 1 + d \cdot \mathrm{erc}_{\langle \mathcal{P}, \mathrm{Pre}(\alpha) \rangle}(n)$ for all $n \in \mathbb{N}$, and thus, $\iota_{\langle \mathcal{P}, \{\alpha\} \rangle} \sqsubseteq \iota_{\langle \mathcal{P}, \mathrm{Pre}(\alpha) \rangle}$.

Hence, if $\alpha \in \mathcal{S}$ and $\mathrm{Pre}(\alpha) \cap \mathcal{S} = \varnothing$ (i.e., $\mathrm{Pre}(\alpha) \subseteq \mathcal{P} \setminus \mathcal{S}$), then in any well-formed proof tree with a node v where $L_{\mathcal{A}}(v) = \langle \mathcal{P}, \mathcal{S} \rangle$, the ADPs from $\mathrm{Pre}(\alpha)$ have already been taken into account in the path $v_1, \ldots, v_k = v$ from the root node v_1 to v, i.e.,

$$\iota_{\langle \mathcal{P}, \{\alpha\} \rangle} \sqsubseteq \iota_{\langle \mathcal{P}, \mathrm{Pre}(\alpha) \rangle} \sqsubseteq \iota_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle} \sqsubseteq L_C(v_1) \oplus \cdots \oplus L_C(v_{k-1}).$$

As the proof tree already contains knowledge about $\mathrm{Pre}(\alpha)$'s complexity, the *knowledge propagation processor* removes α from $\mathcal{S}$.

Theorem 4.14 (Knowledge Propagation Proc.). *Let $\langle \mathcal{P}, \mathcal{S} \rangle$ be an ADP problem, let $\alpha \in \mathcal{S}$ and $\mathrm{Pre}(\alpha) \cap \mathcal{S} = \varnothing$, where $\mathrm{Pre}(\alpha)$ consists of all ADPs $\beta \in \mathcal{P}$ such that there is an edge from some DP in $\mathrm{dp}^\perp(\beta)$ to some DP in $\mathrm{dp}^\perp(\alpha)$ in the $\mathcal{P}$-dependency graph. Then the following processor is sound:*

$$\mathrm{Proc}_{\mathsf{KP}}(\langle \mathcal{P}, \mathcal{S} \rangle) = (\mathrm{Pol}_0, \langle \mathcal{P}, \mathcal{S} \setminus \{\alpha\} \rangle)$$

Example 4.15 (Knowledge Propagation Processor). We still have to solve the ADP problem $\langle \mathcal{P}_1|_{\mathcal{J}_2}, \{(28)\} \rangle$ from Ex. 4.13. We have $\mathrm{dp}^\perp((28)) = \{(18)\}$ and the only DPs with edges to (18) in the dependency graph of Fig. 4 are (14) and (17), where $(14) \in \mathrm{dp}^\perp((25))$ and $(17) \in \mathrm{dp}^\perp((27))$. Thus, $\mathrm{Pre}((28)) = \{(25), (27)\}$, i.e., in particular $(28) \notin \mathrm{Pre}((28))$. Hence, we can apply the knowledge propagation

$$\begin{tikzcd}
 & \mathrm{Pol}_0 \ \langle \mathcal{A}(\mathcal{R}_1), \mathcal{A}(\mathcal{R}_1)\rangle \arrow[d, "\mathrm{Proc}_{\mathsf{UR}}"] & \\
 & \mathrm{Pol}_0 \ \langle \{(9)-(13)\}, \{(9)-(13)\}\rangle \arrow[dl] \arrow[dr, "\mathrm{Proc}_{\mathsf{DG}}"'] & \\
\mathrm{Pol}_0 \ \langle \{(20)-(24)\}, \{(20)-(24)\}\rangle \arrow[d, "\mathrm{Proc}_{\mathsf{RP}}"] & & \mathrm{Pol}_0 \ \langle \{(25)-(29)\}, \{(25)-(29)\}\rangle \arrow[d, "\mathrm{Proc}_{\mathsf{RP}}"] \\
\mathrm{Pol}_0 \ \langle \{(20)-(24)\}, \varnothing\rangle & & \mathrm{Pol}_1 \ \langle \{(25)-(29)\}, \{(28)\}\rangle \arrow[d, "\mathrm{Proc}_{\mathsf{KP}}"] \\
 & & \mathrm{Pol}_0 \ \langle \{(25)-(29)\}, \varnothing\rangle
\end{tikzcd}$$

Figure 6: Solved proof tree for $\mathcal{R}_1$

processor and obtain $\mathrm{Proc}_{\mathsf{KP}}(\langle \mathcal{P}_1|_{\mathcal{J}_2}, \{(28)\}\rangle) = (\mathrm{Pol}_0, \langle \mathcal{P}_1|_{\mathcal{J}_2}, \varnothing\rangle)$, i.e., the resulting ADP problem is solved.

The solved proof tree is shown in Fig. 6. Thus, we inferred that $\mathcal{R}_1$ is SAST and its complexity is at most linear, i.e., $\iota_{\mathcal{R}_1} \sqsubseteq \mathrm{Pol}_1$.

4.5 Probability Removal Processor

Our framework may yield ADP (sub)problems with non-probabilistic structure, i.e., where every ADP has the form $\ell \to \{1 : r\}^m$. Then, the *probability removal processor* can switch to ordinary (non-probabilistic) DT problems for complexity analysis from [53].

These DT problems have four components $(\mathcal{P}, \mathcal{S}, \mathcal{K}, \mathcal{R})$: A set of dependency tuples $\mathcal{P}$, the subset $\mathcal{S} \subseteq \mathcal{P}$ that is counted for complexity, a subset $\mathcal{K}$ whose complexity has already been taken into account (see Remark 3.21), and a set of rewrite rules $\mathcal{R}$.

Theorem 4.16 (Probability Removal Processor). *Let $\langle \mathcal{P}, \mathcal{S}\rangle$ be an ADP problem where every ADP in $\mathcal{P}$ has the form $\ell \to \{1 : r\}^m$. Let $\mathrm{dt}(\ell \to \{1 : r\}^m) = \ell^\sharp \to [t_1^\sharp, \ldots, t_n^\sharp]$ if $\{t \mid t \trianglelefteq_\sharp r\} = \{t_1, \ldots, t_n\}$, and let $\mathrm{dt}(\mathcal{P}) = \{\mathrm{dt}(\alpha) \mid \alpha \in \mathcal{P}\}$. Then the expected runtime complexity of $\langle \mathcal{P}, \mathcal{S}\rangle$ is equal to the runtime complexity of the non-probabilistic DT problem $\beta = (\mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{dt}(\mathcal{P} \setminus \mathcal{S}), \mathrm{np}(\mathcal{P}))$. So the processor $\mathrm{Proc}_{\mathsf{PR}}(\langle \mathcal{P}, \mathcal{S}\rangle) = (c, \varnothing)$ is sound if the DT framework returns c as an upper bound on the runtime complexity of β.*

When proving AST as in [36], one should move to the non-probabilistic DP framework for termination whenever possible, because then one can analyze function calls in right-hand sides of rules separately. In contrast, in the non-probabilistic DT framework for complexity analysis, one also has to consider all function calls from a right-hand side simultaneously. However, the switch to the non-probabilistic setting is still advantageous, since, e.g., the reduction pair processor of the non-probabilistic DT framework allows more orderings than just multilinear polynomial interpretations.

4.6 Transformational Processors

The DP and DT frameworks of [3, 21, 53] also provide several processors that use narrowing, rewriting, or instantiations to *transform* DP/DT problems to simplify the task of proving termination or complexity. To show how to adapt such transformational processors for our novel ADP framework, we consider the *narrowing processor.*

Let $\mathcal{P} = \mathcal{P}' \uplus \{\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m\}$ be a set of ADPs and let $t \trianglelefteq_\sharp r_j$ for some $1 \leq j \leq k$. If we have to perform rewrite steps on (an instance of) t in order to enable the next application of an ADP at an annotated position, then the idea of the narrowing processor is to perform the first step of this reduction already on the ADP $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$ via narrowing. So whenever there is a $t \trianglelefteq_\sharp r_j$ and a non-variable position τ in t such that $t|_\tau$ unifies with the left-hand side ℓ' of some (variable-renamed) ADP $\ell' \to \{p'_1 : r'_1, \ldots, p'_{k'} : r'_{k'}\}^{m'} \in \mathcal{P}$ using an mgu δ such that $\ell\delta, \ell'\delta \in \mathsf{ANF}_{\mathcal{P}}$, then δ is a *narrowing substitution* of t. This is analogous to the narrowing substitutions defined in [53] for DTs.

As shown in [38], in the probabilistic setting, the narrowing processor can only be used in a weaker version. Hence, here it was renamed to the *rule overlap instantiation processor.* While we can apply the narrowing substitution to the ADP, we cannot perform any rewrite steps. To be precise, if $\delta_1, \ldots, \delta_d$ are all narrowing substitutions of t, then we can replace $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$ by $\ell\delta_e \to \{p_1 : r_1\delta_e, \ldots, p_k : r_k\delta_e\}$, for all $1 \leq e \leq d$, but we cannot perform any rewrite steps on $t\delta_e$ directly.

Moreover, there could be another subterm $t' \trianglelefteq_\sharp r_j$ (with $t' \neq t$) which was involved in a chain tree (i.e., $t'^\sharp\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathrm{np}(\mathcal{P})} \tilde{\ell}\tilde{\sigma}$ for some substitutions $\sigma, \tilde{\sigma}$ and a left-hand side $\tilde{\ell}$ of an ADP, but this reduction is no longer possible when replacing t' by the instantiations $t'\delta_1, \ldots, t'\delta_d$. We say that t' is *captured* by $\delta_1, \ldots, \delta_d$ 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 $\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, p_k : \sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}^m$, where $\mathrm{capt}_j(\delta_1, \ldots, \delta_d)$ contains all positions of subterms $t' \trianglelefteq_\sharp r_j$ which are not captured by the narrowing substitutions $\delta_1, \ldots, \delta_d$ of t.

Theorem 4.17 (Rule Overlap Instantiation Processor). *Let $\langle \mathcal{P}, \mathcal{S}\rangle$ be an ADP problem with $\mathcal{P} = \mathcal{P}' \uplus \{\alpha\}$ for $\alpha = \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m$, let $1 \leq j \leq k$, and let $t \trianglelefteq_\sharp r_j$. Let $\delta_1, \ldots, \delta_d$ be all narrowing substitutions of t. Then $\mathrm{Proc}_{\mathsf{ROI}}(\langle \mathcal{P}, \mathcal{S}\rangle) = (\mathrm{Pol}_0, \{\langle \mathcal{P}' \cup N, \widetilde{\mathcal{S}}\rangle\})$ is sound, where*

$$N = \{\ell\delta_e \to \{p_1 : r_1\delta_e, \ldots, p_k : r_k\delta_e\}^m \mid 1 \leq e \leq d\} \cup \{\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, p_k : \sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}^m\}$$

$$\widetilde{\mathcal{S}} = \begin{cases} (\mathcal{S} \setminus \{\alpha\}) \cup N, & \text{if } \alpha \in \mathcal{S} \\ \mathcal{S}, & \text{otherwise} \end{cases}$$

Example 4.18 (Rule Overlap Inst. Proc.). Consider $\mathcal{R}_{\mathsf{ROI}}$ from [38].

$$\mathsf{f}(\mathsf{d}(x)) \to \{{}^3\!/\!_4 : \mathsf{e}(\,\mathsf{f}(\mathsf{g}(x)), \mathsf{f}(\mathsf{h}(x))\,), {}^1\!/\!_4 : \mathsf{a}\} \quad \mathsf{g}(\mathsf{a}) \to \{1 : \mathsf{d}(\mathsf{a})\}$$
$$\mathsf{h}(\mathsf{b}) \to \{1 : \mathsf{d}(\mathsf{b})\}$$

$\mathcal{R}_{\mathsf{ROI}}$ has constant runtime complexity, i.e., $\iota_{\mathcal{R}_{\mathsf{ROI}}} = \mathrm{Pol}_0$, because for every instantiation, at most one of the two recursive f-calls in the right-hand side of the f-rule can be evaluated. The reason is that we can either use the g-rule if x is instantiated with a, or we can apply the h-rule if x is instantiated with b, but not both.

With the rule overlap instantiation processor, our ADP framework can determine this constant complexity automatically. Using the dependency graph processor on the canonical ADP problem $\langle \mathcal{A}(\mathcal{R}_{\mathsf{ROI}}), \mathcal{A}(\mathcal{R}_{\mathsf{ROI}})\rangle$ yields $\langle \mathcal{P}_{\mathsf{ROI}}, \mathcal{P}_{\mathsf{ROI}}\rangle$, where $\mathcal{P}_{\mathsf{ROI}}$ consists of

$$\mathsf{f}(\mathsf{d}(x)) \to \{{}^3\!/\!_4 : \mathsf{e}(\,\mathsf{F}(\mathsf{g}(x)), \mathsf{F}(\mathsf{h}(x))\,), {}^1\!/\!_4 : \mathsf{a}\}^{\mathsf{true}}$$
$$\mathsf{g}(\mathsf{a}) \to \{1 : \mathsf{d}(\mathsf{a})\}^{\mathsf{true}}$$
$$\mathsf{h}(\mathsf{b}) \to \{1 : \mathsf{d}(\mathsf{b})\}^{\mathsf{true}}$$

We apply $\mathrm{Proc}_{\mathsf{ROI}}$ using the term $t = \mathsf{f}(\mathsf{g}(x))$ whose only narrowing substitution is $\delta = \{x/\mathsf{a}\}$. For the other subterm $\mathsf{F}(\mathsf{h}(x))$ with annotated root, $\mathsf{f}(\mathsf{h}(x))$ is not captured by δ. Hence, we generate an additional ADP where this second subterm is annotated. Thus, we replace the former f-ADP by the following two new ADPs.

$$\mathsf{f}(\mathsf{d}(\mathsf{a})) \to \{\sfrac{3}{4} : \mathsf{e}(\,\mathsf{F}(\mathsf{g}(\mathsf{a})),\ \mathsf{F}(\mathsf{h}(\mathsf{a}))\,),\ \sfrac{1}{4} : \mathsf{a}\}^{\mathsf{true}}$$
$$\mathsf{f}(\mathsf{d}(x)) \to \{\sfrac{3}{4} : \mathsf{e}(\,\mathsf{f}(\mathsf{g}(x)),\ \mathsf{F}(\mathsf{h}(x))\,),\ \sfrac{1}{4} : \mathsf{a}\}^{\mathsf{true}}$$

Now one can remove the annotation of $\mathsf{F}(\mathsf{h}(\mathsf{a}))$ from the first ADP by the dependency graph processor and then apply the reduction pair processor with the interpretation that maps F to 1 and all other symbols to 0 to remove all annotations. This proves $\iota_{\langle \mathcal{P}_{\mathrm{ROI}}, \mathcal{P}_{\mathrm{ROI}} \rangle} = \iota_{\mathcal{R}_{\mathrm{ROI}}} = \mathrm{Pol}_0$. Using a constant polynomial interpretation would not be possible without the rule overlap instantiation processor.

5 Evaluation

We implemented our new DP framework for upper bounds on the expected innermost runtime complexity in our termination prover AProVE [22]. To this end, AProVE first creates the canonical ADPs and then applies processors according to the following strategy:

First, we try to apply the dependency graph processor $\mathrm{Proc}_{\mathsf{DG}}$, the usable rules processor $\mathrm{Proc}_{\mathsf{UR}}$, the knowledge propagation processor $\mathrm{Proc}_{\mathsf{KP}}$, and the probability removal processor $\mathrm{Proc}_{\mathsf{PR}}$ in this order. The advantage of these processors is that they do not rely on searching (i.e., they are very fast) and they simplify the ADP problem whenever they are applicable. If none of these processors can be applied anymore, then we search for CPIs for the reduction pair processor $\mathrm{Proc}_{\mathsf{RP}}$ in order to derive polynomial complexity bounds for certain ADPs (otherwise, we try to apply $\mathrm{Proc}_{\mathsf{RP}}$ with a non-CPI polynomial interpretation to derive an exponential bound). As soon as one of the processors is applied successfully, we re-start the strategy again, since other processors might be applicable again on the simplified subproblems. Moreover, before the first application of the reduction pair processor, we use the rule overlap instantiation processor $\mathrm{Proc}_{\mathsf{ROI}}$. Since it does not always help in inferring an upper bound and often increases the number of ADPs, we use $\mathrm{Proc}_{\mathsf{ROI}}$ only once on a fixed number of terms.

For every PTRS, the user can indicate whether one wants to analyze termination or complexity, consider arbitrary or only basic start terms, and whether one wants to analyze innermost or full rewriting (with an arbitrary rewrite strategy). Since our novel DP framework only works for innermost rewriting and basic start terms, if the user asks for complexity analysis or SAST of full rewriting, we check whether the PTRS belongs to a known class where, e.g., upper bounds on the expected innermost runtime complexity are upper bounds w.r.t. an arbitrary rewrite strategy as well. Such properties were studied in [34, 35]. If one wants to consider arbitrary instead of basic start terms, we perform the transformation of [20] (adapted to PTRSs, see [34, 35]) in order to move from derivational to runtime complexity, i.e., the PTRS $\mathcal{R}$ is transformed into a new PTRS $\mathcal{R}'$ such that the complexity of $\mathcal{R}'$ on basic start terms is a bound on the complexity of $\mathcal{R}$ on all start terms.

For our evaluation, we used the benchmark set of all 128 PTRSs from the *Termination Problem Data Base* [55], i.e., the benchmarks considered for the annual *Termination and Complexity Competition* [23], containing 128 typical probabilistic programs, including examples with complicated probabilistic structure and probabilistic algorithms on lists and trees. Note that this set also contains many examples that are AST, but not SAST. Therefore, we extended the collection by 10 additional examples that are interesting for expected complexity analysis (including all examples from our paper).

To evaluate how our novel framework proves SAST, we compare AProVE with its previous version (called "POLO" in the table below) whose only way to prove SAST was to search for a monotonic, multilinear polynomial interpretation such that all rules of the PTRS are strictly decreasing [7], and with the tool NaTT [57] that implements polynomial and matrix interpretations to prove SAST. Thus, POLO and NaTT neither consider a specific rewrite strategy nor start terms. As shown by the experiments on all 138 PTRSs in the first table below, our novel ADP framework increases the power of proving SAST significantly.

Strategy	Start Terms	POLO	NaTT	AProVE
Full	Arbitrary	30	33	35
Full	Basic	30	33	44
Innermost	Arbitrary	30	33	54
Innermost	Basic	30	33	62

Strategy	Start Terms	Pol_0	Pol_1	Pol_2	Exp	2-Exp	ω
Full	Arbitrary	2	21	0	11	1	103
Full	Basic	15	25	1	3	0	94
Innermost	Arbitrary	2	47	0	5	0	84
Innermost	Basic	25	35	0	2	0	76

The second table shows the upper bounds inferred by AProVE. So AProVE obtains numerous constant and/or linear bounds, even for full rewriting and/or arbitrary start terms. Note that in contrast to the non-probabilistic setting, a PTRS with expected constant runtime is not necessarily trivial as it can have evaluations of unbounded (and even infinite) length. However, the transformation of [20] to move from arbitrary to basic start terms may add rules with linear runtime to the PTRS. This explains the low number of constant upper bounds for arbitrary start terms. Due to the restriction to multilinear polynomial interpretations, Pol_a for $a > 1$ can currently only be inferred from multilinear, but non-linear interpretations like $\mathcal{I}_{f^\sharp}(x, y) = x \cdot y$. In the future, we intend to extend our implementation to also use, e.g., matrix orderings [17] in order to improve the inference of polynomial bounds of higher degrees.

For more details on our experiments, the collection of examples, and for instructions on how to run our implementation in AProVE via its *web interface* or locally, we refer to:

https://aprove-developers.github.io/PTRSExpectedRuntime/

6 Conclusion

In this paper, we presented the first DP framework to infer upper bounds on the expected innermost runtime complexity of PTRSs automatically. Our implementation in AProVE is the first tool for automatic complexity analysis of PTRSs and it improves substantially over previous tools to analyze SAST of PTRSs.

There are several directions for future work, e.g., by extending the reduction pair processor of Thm. 4.12 to other orderings, by adapting further transformational processors to our new ADP framework, and by developing variants of our framework that are directly applicable for full instead of innermost rewriting and/or for arbitrary instead of basic start terms.

Acknowledgments

This work is supported by the DFG Research Training Group 2236 UnRAVeL.

References

[1] Alessandro Abate, Mirco Giacobbe, and Diptarko Roy. 2021. Learning Probabilistic Termination Proofs. In *Proc. CAV '21 (LNCS 12760)*. 3–26. doi:10.1007/978-3-030-81688-9_1

[2] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2018. Lexicographic Ranking Supermartingales: An Efficient Approach to Termination of Probabilistic Programs. *Proc. ACM Program. Lang.* 2, POPL (2018). doi:10.1145/3158122

[3] Thomas Arts and Jürgen Giesl. 2000. Termination of Term Rewriting Using Dependency Pairs. *Theor. Comput. Sc.* 236, 1-2 (2000), 133–178. doi:10.1016/S0304-3975(99)00207-8

[4] Martin Avanzini and Georg Moser. 2016. A Combination Framework for Complexity. *Inf. Comput.* 248 (2016), 22–55. doi:10.1016/j.ic.2015.12.007

[5] Martin Avanzini, Georg Moser, and Michael Schaper. 2016. TcT: Tyrolean Complexity Tool. In *Proc. TACAS '16 (LNCS 9636)*. 407–423. doi:10.1007/978-3-662-49674-9_24

[6] Martin Avanzini, Ugo Dal Lago, and Alexis Ghyselen. 2019. Type-Based Complexity Analysis of Probabilistic Functional Programs. In *Proc. LICS '19*. doi:10.1109/LICS.2019.8785725

[7] Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. 2020. On Probabilistic Term Rewriting. *Sci. Comput. Program.* 185 (2020). doi:10.1016/j.scico.2019.102338

[8] Martin Avanzini, Georg Moser, and Michael Schaper. 2020. A Modular Cost Analysis for Probabilistic Programs. *Proc. ACM Program. Lang.* 4, OOPSLA (2020). doi:10.1145/3428240

[9] Franz Baader and Tobias Nipkow. 1998. *Term Rewriting and All That.* Cambridge University Press. doi:10.1017/CBO9781139172752

[10] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Lena Verscht. 2023. A Calculus for Amortized Expected Runtimes. *Proc. ACM Program. Lang.* 7, POPL (2023), 1957–1986. doi:10.1145/3571260

[11] Raven Beutner and Luke Ong. 2021. On Probabilistic Termination of Functional Programs with Continuous Distributions. In *Proc. PLDI '21*. 1312–1326. doi:10.1145/3453483.3454111

[12] Guillaume Bonfante, Adam Cichon, Jean-Yves Marion, and Hélène Touzet. 2001. Algorithms with Polynomial Interpretation Termination Proof. *Journal of Functional Programming* 11, 1 (2001), 33–53. doi:10.1017/S0956796800003877

[13] Olivier Bournez and Claude Kirchner. 2002. Probabilistic Rewrite Strategies. Applications to ELAN. In *Rewriting Techniques and Applications*. 252–266.

[14] Olivier Bournez and Florent Garnier. 2005. Proving Positive Almost-Sure Termination. In *Proc. RTA '05 (LNCS 3467)*. 323–337. doi:10.1007/978-3-540-32033-3_24

[15] Krishnendu Chatterjee, Hongfei Fu, and Petr Novotný. 2020. Termination Analysis of Probabilistic Programs with Martingales. In *Foundations of Probabilistic Programming*, Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva (Eds.). Cambridge University Press, 221–258. doi:10.1017/9781108770750.008

[16] Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, Jiri Zárevúcky, and Dorde Zikelic. 2023. On Lexicographic Proof Rules for Probabilistic Termination. *Formal Aspects Comput.* 35, 2 (2023). doi:10.1145/3585391

[17] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. 2008. Matrix Interpretations for Proving Termination of Term Rewriting. *J. Autom. Reason.* 40, 2-3 (2008), 195–220. doi:10.1007/S10817-007-9087-9

[18] Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Proc. POPL '15*. 489–501. doi:10.1145/2676726.2677001

[19] Hongfei Fu and Krishnendu Chatterjee. 2019. Termination of Nondeterministic Probabilistic Programs. In *Proc. VMCAI '19 (LNCS 11388)*. 468–490. doi:10.1007/978-3-030-11245-5_22

[20] Carsten Fuhs. 2019. Transforming Derivational Complexity of Term Rewriting to Runtime Complexity. In *Proc. FroCoS '19 (LNCS 11715)*. 348–364. doi:10.1007/978-3-030-29007-8_20

[21] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. 2006. Mechanizing and Improving Dependency Pairs. *J. Autom. Reason.* 37, 3 (2006), 155–203. doi:10.1007/s10817-006-9057-7

[22] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. 2017. Analyzing Program Termination and Complexity Automatically with AProVE. *J. Autom. Reason.* 58, 1 (2017), 3–31. doi:10.1007/s10817-016-9388-y

[23] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. 2019. The Termination and Complexity Competition. In *Proc. TACAS '19 (LNCS 11429)*. 156–166. doi:10.1007/978-3-030-17502-3_10 Website of *TermComp*: https://termination-portal.org/wiki/Termination_Competition.

[24] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic Programming. In *Proc. FOSE '14*. 167–181. doi:10.1145/2593882.2593900

[25] Raúl Gutiérrez and Salvador Lucas. 2020. MU-TERM: Verify Termination Properties Automatically (System Description). In *Proc. IJCAR '20 (LNCS 12167)*. 436–447. doi:10.1007/978-3-030-51054-1_28

[26] Nao Hirokawa and Aart Middeldorp. 2005. Automating the Dependency Pair Method. *Inf. Comput.* 199, 1-2 (2005), 172–199. doi:10.1016/j.ic.2004.10.004

[27] Nao Hirokawa and Georg Moser. 2008. Automated Complexity Analysis Based on the Dependency Pair Method. In *Proc. IJCAR '08 (LNCS 5195)*. 364–379. doi:10.1007/978-3-540-71070-7_32

[28] Dieter Hofbauer and Clemens Lautemann. 1989. Termination Proofs and the Length of Derivations (Preliminary Version). In *Proc. RTA '89 (LNCS 355)*. 167–177. doi:10.1007/3-540-51081-8_107

[29] Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2019. Modular Verification for Almost-Sure Termination of Probabilistic Programs. *Proc. ACM Program. Lang.* 3, OOPSLA (2019). doi:10.1145/3360555

[30] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* 65, 5 (2018). doi:10.1145/3208102

[31] Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2020. Expected Runtime Analyis by Program Verification. In *Foundations of Probabilistic Programming*, Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva (Eds.). Cambridge University Press, 185–220. doi:10.1017/9781108770750.007

[32] Jan-Christoph Kassing and Jürgen Giesl. 2023. Proving Almost-Sure Innermost Termination of Probabilistic Term Rewriting Using Dependency Pairs. In *Proc. CADE '23 (LNCS 14132)*. 344–364. doi:10.1007/978-3-031-38499-8_20

[33] Jan-Christoph Kassing, Grigory Vartanyan, and Jürgen Giesl. 2024. A Dependency Pair Framework for Relative Termination of Term Rewriting. In *Proc. IJCAR '24 (LNCS 14740)*. 360–380. doi:10.1007/978-3-031-63501-4_19

[34] Jan-Christoph Kassing, Florian Frohn, and Jürgen Giesl. 2024. From Innermost to Full Almost-Sure Termination of Probabilistic Term Rewriting. In *Proc. FoSSaCS '24 (LNCS 14575)*. 206–228. doi:10.1007/978-3-031-57231-9_10

[35] Jan-Christoph Kassing and Jürgen Giesl. 2024. From Innermost to Full Probabilistic Term Rewriting: Almost-Sure Termination, Complexity, and Modularity. *CoRR* abs/2409.17714 (2024). doi:10.48550/arXiv.2409.17714 Long version of [34].

[36] Jan-Christoph Kassing, Stefan Dollase, and Jürgen Giesl. 2024. A Complete Dependency Pair Framework for Almost-Sure Innermost Termination of Probabilistic Term Rewriting. In *Proc. FLOPS '24 (LNCS 14659)*. 62–80. doi:10.1007/978-981-97-2300-3_4

[37] Jan-Christoph Kassing and Jürgen Giesl. 2024. Annotated Dependency Pairs for Full Almost-Sure Termination of Probabilistic Term Rewriting. In *Principles of Verification (LNCS 15260)*. 339–366. doi:10.1007/978-3-031-75783-9_14

[38] Jan-Christoph Kassing and Jürgen Giesl. 2024. The Annotated Dependency Pair Framework for Almost-Sure Termination of Probabilistic Term Rewriting. *CoRR* abs/2412.20220 (2024). doi:10.48550/arXiv.2412.20220 Long version of [36] and [37].

[39] Andrew Kenyon-Roberts and C.-H. Luke Ong. 2021. Supermartingales, Ranking Functions and Probabilistic Lambda Calculus. In *Proc. LICS '21*. doi:10.1109/LICS52264.2021.9470550

[40] Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. 2020. On the Termination Problem for Probabilistic Higher-Order Recursive Programs. *Log. Methods Comput. Sci.* 16, 4 (2020). doi:10.23638/LMCS-16(4:2)2020

[41] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. 2009. Tyrolean Termination Tool 2. In *Proc. RTA '09 (LNCS 5595)*. 295–304. doi:10.1007/978-3-642-02348-4_21

[42] Ugo Dal Lago and Charles Grellois. 2019. Probabilistic Termination by Monadic Affine Sized Typing. *ACM Trans. Program. Lang. Syst.* 41, 2 (2019). doi:10.1145/3293605

[43] Ugo Dal Lago, Claudia Faggian, and Simona Ronchi Della Rocca. 2021. Intersection Types and (Positive) Almost-Sure Termination. *Proc. ACM Program. Lang.* 5, POPL (2021). doi:10.1145/3434313

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

[45] Lorenz Leutgeb, Georg Moser, and Florian Zuleger. 2022. Automated Expected Amortised Cost Analysis of Probabilistic Data Structures. In *Proc. CAV '22 (LNCS 13372)*. 70–91. doi:10.1007/978-3-031-13188-2_4

[46] Nils Lommen, Éléanore Meyer, and Jürgen Giesl. 2024. Control-Flow Refinement for Complexity Analysis of Probabilistic Programs in KoAT (Short Paper). In *Proc. IJCAR '24 (LNCS 14739)*. 233–243. doi:10.1007/978-3-031-63498-7_14

[47] Rupak Majumdar and V. R. Sathiyanarayana. 2024. Positive Almost-Sure Termination: Complexity and Proof Rules. *Proc. ACM Program. Lang.* 8, POPL (2024), 1089–1117. doi:10.1145/3632879

[48] Rupak Majumdar and V. R. Sathiyanarayana. 2025. Sound and Complete Proof Rules for Probabilistic Termination. *Proc. ACM Program. Lang.* 9, POPL (2025), 1871–1902. doi:10.1145/3704899

[49] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2018. A New Proof Rule for Almost-Sure Termination. *Proc. ACM Program. Lang.* 2, POPL (2018). doi:10.1145/3158121

[50] Fabian Meyer, Marcel Hark, and Jürgen Giesl. 2021. Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes. In *Proc. TACAS '21 (LNCS 12651)*. 250–269. doi:10.1007/978-3-030-72016-2_14

[51] Marcel Moosbrugger, Ezio Bartocci, Joost-Pieter Katoen, and Laura Kovács. 2021. Automated Termination Analysis of Polynomial Probabilistic Programs. In *Proc.*

ESOP '21 (LNCS 12648). 491–518. doi:10.1007/978-3-030-72019-3_18

[52] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded Expectations: Resource Analysis for Probabilistic Programs. In *Proc. PLDI '18*. 496–512. doi:10.1145/3192366.3192394

[53] Lars Noschinski, Fabian Emmes, and Jürgen Giesl. 2013. Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs. *J. Autom. Reason.* 51 (2013), 27–56. doi:10.1007/s10817-013-9277-6

[54] Vineet Rajani, Gilles Barthe, and Deepak Garg. 2024. A Modal Type Theory of Expected Cost in Higher-Order Probabilistic Programs. *Proc. ACM Program. Lang.* 8, OOPSLA2 (2024), 389–414. doi:10.1145/3689725

[55] TPDB. 2025. Termination Problem Data Base. https://github.com/TermCOMP/TPDB-ARI

[56] Di Wang, David M. Kahn, and Jan Hoffmann. 2020. Raising Expectations: Automating Expected Cost Analysis with Types. *Proc. ACM Program. Lang.* 4, ICFP (2020), 31 pages. doi:10.1145/3408992

[57] Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. 2014. Nagoya Termination Tool. In *Proc. RTA-TLCA '14 (LNCS 8560)*. 466–475. doi:10.1007/978-3-319-08918-8_32

A Additional Theory and Proofs

In this appendix, we first present some additional theory on the size of a term under a polynomial interpretation in Sect. A.1. Afterwards, we prove all our theorems and lemmas in Sect. A.2.

A.1 Size of Polynomial Interpretations

We first analyze the size of $\mathcal{I}_0(t)$ for every term t and every polynomial interpretation $\mathcal{I}$. Here, $\mathcal{I}_0$ denotes the interpretation that behaves like $\mathcal{I}$, and in addition, maps every variable to 0.

A.1.1 Size of Constructor Terms. Let $\mathcal{I}$ be a CPI. Recall that a CPI $\mathcal{I}$ guarantees that $\mathcal{I}(f(x_1, \ldots, x_n)) = a_1x_1 + \ldots + a_nx_n + b$, with $a_i \in \{0,1\}, b \in \mathbb{N}$ for all constructors $f \in \Sigma_C$. Let $s \in \mathcal{T}(C, \mathcal{V})$ be a constructor term, i.e., a term containing only constructors and variables. Let b_{max} be the maximum of all b for all constructors $f \in \Sigma_C$. For $|s| = 1$ we get $\mathcal{I}_0(s) \leq b_{max}$, and for $|s| = n+1$ we get $\mathcal{I}_0(s) \leq \mathcal{I}_0(s') + b_{max}$ for some constructor term s' with $|s'| = n$. Hence, for every term $s \in \mathcal{T}(C, \mathcal{V})$ we obtain $\mathcal{I}_0(s) \leq b_{max} \cdot |s|$.

Next, let $\mathcal{I}$ not be a CPI but linear on constructors, i.e., $\mathcal{I}(f(x_1, \ldots, x_n)) = a_1x_1 + \ldots + a_nx_n + b$, with $a_1, \ldots, a_n, b \in \mathbb{N}$ for all constructors $f \in \Sigma_C$. Let b_{max} be the maximum of all b and a_{max} be the maximum of all a. Again, let $s \in \mathcal{T}(C, \mathcal{V})$. For $|s| = 1$ we get $\mathcal{I}_0(s) \leq b_{max}$, and for $|s| = n+1$ we get $\mathcal{I}_0(s) \leq a_{max} \cdot \mathcal{I}_0(s') + b_{max}$ for some constructor term s' with $|s'| = n$. This is a first-order non-homogeneous linear recurrence relation, which can be upper-bounded by $2^{\mathrm{pol}(|s|)}$, for some polynomial $\mathrm{pol}(n)$ in n, i.e., we have $\mathcal{I}_0(s) \in \mathcal{O}(2^{\mathrm{pol}(|s|)})$.

Finally, if $\mathcal{I}$ is not a CPI and not even linear on constructors, e.g., $\mathcal{I}(f(x_1, x_2)) = x_1 \cdot x_2$ or $\mathcal{I}(f(x_1, x_2)) = x_1^2 + x_2^2$, then we can only guarantee that $\mathcal{I}_0(s) \leq 2^{2^{\mathrm{pol}(|s|)}}$, for some polynomial $\mathrm{pol}(n)$ in n. To be precise, let a_{max} be the maximum coefficient of all non-constant monomials in all interpretations I_f of constructors f, and let b_{max} be the maximum constant monomial in all these interpretations. Moreover, let c_{max} be the maximum degree of all $\mathcal{I}(f(x_1, \ldots, x_n))$, and k be the maximal arity for all constructors $f \in \Sigma_C$. For $|s| = 1$ we again have $\mathcal{I}_0(s) \leq b_{max}$, and for $|s| = n+1$ we get $\mathcal{I}_0(s) \leq 2^k \cdot a_{max} \cdot \mathcal{I}_0(s')^{c_{max}} + b_{max}$ for some constructor term s' with $|s'| = n$. After conversion to homogeneous form, and taking the log on both sides, we result in a first-order linear recurrence relation again. Hence, we have $\log(\mathcal{I}_0(s)) \leq 2^{\mathrm{pol}(|s|)}$, for some polynomial $\mathrm{pol}(n)$, i.e., we have $\mathcal{I}_0(s) \in \mathcal{O}(2^{2^{\mathrm{pol}'(|s|)}})$, for some polynomial $\mathrm{pol}'(n)$.

A.1.2 Size of Basic Terms. First, let $\mathcal{I}$ be a CPI where $\mathcal{I}(f(x_1, \ldots, x_n)) = a_1x_1 + \ldots + a_nx_n + b$, with $a_i \in \{0,1\}, b \in \mathbb{N}$ for all constructors $f \in \Sigma_C$. Again, let b_{max} be the maximum of all b for all constructors $f \in \Sigma_C$. Moreover, let $t = f(t_1, \ldots, t_n) \in \mathcal{TB}_{\mathcal{R}}$ of size $|t| = n$. Here, we have

$$\begin{aligned}\mathcal{I}_0(t) &= \mathcal{I}_f(\mathcal{I}_0(t_1), \ldots, \mathcal{I}_0(t_n))\\ &\leq \mathcal{I}_f(b_{max} \cdot |t_1|, \ldots, b_{max} \cdot |t_n|)\\ &\leq \mathcal{I}_f(b_{max} \cdot |t|, \ldots, b_{max} \cdot |t|)\\ &\leq b_{max}^m \cdot \mathcal{I}_f(|t|, \ldots, |t|), \quad \text{where } m \text{ is the degree of } \mathcal{I}_f\end{aligned}$$

Hence, we have $\mathcal{I}_0(t) \in \mathcal{O}(|t|^m)$.

If $\mathcal{I}$ is only linear on constructors, we get $\mathcal{I}_0(t) \in \mathcal{O}(2^{\mathrm{pol}(|t|)})$, and if $\mathcal{I}$ is not linear, we obtain $\mathcal{I}_0(t) \in \mathcal{O}(2^{2^{\mathrm{pol}(|t|)}})$ in a similar fashion.

A.1.3 Size of Arbitrary Terms. If we have $\mathcal{I}(f(x_1, \ldots, x_n)) = a_1x_1 + \ldots + a_nx_n + b$ with $a_i \in \{0,1\}, b \in \mathbb{N}$ for all $f \in \Sigma$, then $\mathcal{I}_0(t) \in \mathcal{O}(|t|)$. If $\mathcal{I}$ is linear for all $f \in \Sigma$, then $\mathcal{I}_0(t) \in \mathcal{O}(2^{\mathrm{pol}(|t|)})$, and if $\mathcal{I}$ is not linear, we obtain $\mathcal{I}_0(t) \in \mathcal{O}(2^{2^{\mathrm{pol}(|t|)}})$.

A.1.4 Bounds on Expected Complexity via Polynomials. Assume that there exists a monotonic polynomial interpretation $\mathcal{I}$ such that $\mathcal{I}(\ell) > \mathcal{I}(r)$ for every rule $\ell \to r \in \mathcal{R}$. Since the interpretation is decreasing for every rule and bounded from below by 0, the interpretation $\mathcal{I}(t)$ of a term t gives an upper bound on its derivation height. Hence, e.g., if $\mathcal{I}$ is a CPI, then the runtime complexity is at most Pol_m, where m is the highest degree of $\mathcal{I}_f$ for any defined symbol f.

A.2 Proofs

To prove the soundness of the chain criterion, we proceed as follows: Given an $\mathcal{R}$-RST $\mathfrak{T}$ that starts with the basic term t, we create an $\mathcal{A}(\mathcal{R})$-CT $\mathfrak{T}'$ that starts with $t^\sharp$ and mirrors every rewrite step by using the corresponding ADP for the used rewrite rule. We will show that each rewrite step takes place at an annotated position, i.e., we always use case (**at**) with the ADPs. Since the resulting CT has the same structure as the original RST (same nodes with the same probabilities) and we only have (**at**)-steps (that we count for the expected derivation length of a CT), we get $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}(\mathfrak{T}')$ for every such RST $\mathfrak{T}$, and hence $\mathrm{edh}_{\mathcal{R}}(t) \leq \mathrm{edh}_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}(t)$ for every basic term t. For the other direction, i.e., $\mathrm{edh}_{\mathcal{R}}(t) \geq \mathrm{edh}_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}(t)$, we transform every $\mathcal{A}(\mathcal{R})$-CT $\mathfrak{T}$ into an $\mathcal{R}$-RST $\mathfrak{T}'$ by simply removing all occurring annotations. Then, we directly get $\mathrm{edl}(\mathfrak{T}') = \mathrm{edl}_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}(\mathfrak{T})$ again.

When applying an ADP, we may remove annotations of normal forms, hence, we define the set of all positions of subterms that are not in normal form. During the construction of the proof of the chain criterion, we show that at least those positions are annotated. In the following, as usual we say that two positions π_1 and π_2 are *orthogonal* (or *parallel*) if π_1 is not above π_2 and π_2 is not above π_1.

Definition A.1 ($\mathrm{Pos}_{\mathcal{D} \wedge \neg \mathsf{NF}_{\mathcal{R}}}$). Let $\mathcal{R}$ be a PTRS. For a term $t \in \mathcal{T}$ we define $\mathrm{Pos}_{\mathcal{D} \wedge \neg \mathsf{NF}_{\mathcal{R}}}(t) = \{\pi \mid \pi \in \mathrm{Pos}_{\mathcal{D}}(t), t|_\pi \notin \mathsf{NF}_{\mathcal{R}}\}$.

THEOREM 3.13 (CHAIN CRITERION). *Let $\mathcal{R}$ be a PTRS. Then for all basic terms $t \in \mathcal{TB}_{\mathcal{R}}$ we have*

$$\mathrm{edh}_{\mathcal{R}}(t) = \mathrm{edh}_{\langle \mathcal{A}(\mathcal{R}), \mathcal{A}(\mathcal{R}) \rangle}(t)$$

and therefore $\iota_{\mathcal{R}} = \iota_{\langle \mathcal{A}(\mathcal{R}),\mathcal{A}(\mathcal{R})\rangle}$.

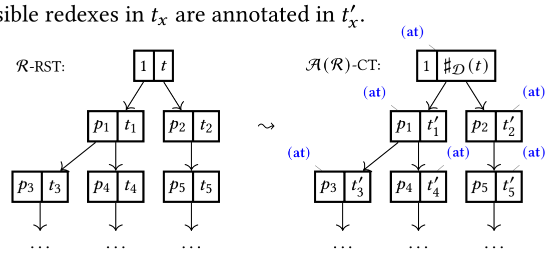
Proof. In the following, we will often implicitly use that for an annotated term $t \in \mathcal{T}^{\sharp}$, we have $\flat(t) \in \mathsf{ANF}_{\mathcal{R}}$ iff $t \in \mathsf{ANF}_{\mathcal{A}(\mathcal{R})}$ since a rewrite rule and its corresponding canonical annotated dependency pair have the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

Soundness ($\mathrm{edh}_{\mathcal{R}}(t) \leq \mathrm{edh}_{\langle \mathcal{A}(\mathcal{R}),\mathcal{A}(\mathcal{R})\rangle}(t)$): Let $\mathfrak{T} = (V,E,L)$ be an $\mathcal{R}$-RST whose root is labeled with $(1 : t)$ for some term $t \in \mathcal{TB}_{\mathcal{R}}$. We create a $\mathcal{A}(\mathcal{R})$-CT $\mathfrak{T}' = (V,E,L')$ that starts with $t^{\sharp}$ and mirrors every rewrite step by using the ADP corresponding to the used rewrite rule. Moreover, we ensure that every rewrite step is an **(at)**-step, and hence, we have $\mathrm{edl}(\mathfrak{T}) = \mathrm{edl}_{\langle \mathcal{A}(\mathcal{R}),\mathcal{A}(\mathcal{R})\rangle}(\mathfrak{T}')$.

We construct the new labeling L' for the $\mathcal{A}(\mathcal{R})$-CT inductively such that for all inner nodes $x \in V \setminus \mathrm{Leaf}$ with children nodes $xE = \{y_1,\ldots,y_k\}$ we have $t'_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{A}(\mathcal{R})} \{\frac{p_{y_1}}{p_x} : t'_{y_1},\ldots,\frac{p_{y_k}}{p_x} : t'_{y_k}\}$ and use Case **(at)**. 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 for all $x \in X$:

$$\flat(t_x) = \flat(t'_x) \wedge \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t_x) \subseteq \mathrm{Pos}_{\mathcal{D}^{\sharp}}(t'_x). \qquad (30)$$

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.

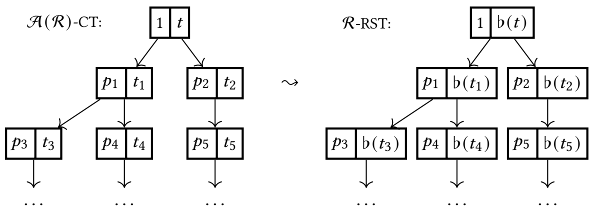


We label the root of $\mathfrak{T}'$ with $\sharp_{\mathcal{D}}(t)$. Here, we obviously have $\flat(t) = \flat(\sharp_{\mathcal{D}}(t))$ and $\mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t) \subseteq \mathrm{Pos}_{\mathcal{D}}(t) = \mathrm{Pos}_{\mathcal{D}^{\sharp}}(\sharp_{\mathcal{D}}(t))$. 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 $xE = \{y_1,\ldots,y_k\}$ be the set of its successors. We need to define the corresponding terms $t'_{y_1},\ldots,t'_{y_k}$ for the nodes $y_1,\ldots,y_k$. Since x is not a leaf, we have $t_x \xrightarrow{\mathrm{i}}_{\mathcal{R}} \{\frac{p_{y_1}}{p_x} : t_{y_1},\ldots,\frac{p_{y_k}}{p_x} : t_{y_k}\}$. This means that there is a rule $\ell \to \{p_1 : r_1,\ldots,p_k : r_k\} \in \mathcal{R}$, a position π, and a substitution σ such that $t_x|_{\pi} = \ell\sigma \in \mathsf{ANF}_{\mathcal{R}}$. 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 $\ell \to \{p_1 : \sharp_{\mathcal{D}}(r_1),\ldots,p_k : \sharp_{\mathcal{D}}(r_k)\}^{\mathsf{true}}$. Furthermore, $\pi \in \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t_x) \subseteq \mathrm{Pos}_{\mathcal{D}^{\sharp}}(t'_x)$ and $\flat(t_x) = \flat(t'_x)$ by the induction hypothesis. Hence, we can rewrite t'_x with $\ell \to \{p_1 : \sharp_{\mathcal{D}}(r_1),\ldots,p_k : \sharp_{\mathcal{D}}(r_k)\}^{\mathsf{true}}$, using the position π and the substitution σ, and Case **(at)** applies. We get $t'_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{A}(\mathcal{R})} \{p_1 : t'_{y_1},\ldots,p_k : t'_{y_k}\}$ with $t'_{y_j} = t'_x[\sharp_{\mathcal{D}}(r_j)\sigma]_{\pi}$. This means that we have $\flat(t_{y_j}) = \flat(t'_{y_j})$. It remains to prove $\mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t_{y_j}) \subseteq \mathrm{Pos}_{\mathcal{D}^{\sharp}}(t'_{y_j})$ for all $1 \leq j \leq k$. For all $\tau \in \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t_{y_j}) = \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t_x[r_j\sigma]_{\pi})$ that are orthogonal or above π, we have $\tau \in \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t_x,\mathcal{R}) \subseteq \mathrm{Pos}_{\mathcal{D}^{\sharp}}(t'_x)$ by the induction hypothesis, and all annotations orthogonal or above π remain in t'_{y_j} as they were in t'_x. For all positions $\tau \in \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t_{y_j}) = \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(t_x[r_j\sigma]_{\pi})$ 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 $\tau \in \mathrm{Pos}_{\mathcal{D}^{\sharp}}(t'_{y_j})$, as all defined symbols of r_j are annotated in $t'_{y_j} = t'_x[\sharp_{\mathcal{D}}(r_j)\sigma]_{\pi}$.

Completeness ($\mathrm{edh}_{\mathcal{R}}(t) \geq \mathrm{edh}_{\langle \mathcal{A}(\mathcal{R}),\mathcal{A}(\mathcal{R})\rangle}(t)$): Let $\mathfrak{T} = (V,E,L)$ be an $\mathcal{A}(\mathcal{R})$-RST whose root is labeled with $(1 : t^{\sharp})$ for some term $t \in \mathcal{TB}_{\mathcal{R}}$. We create a $\mathcal{R}$-RST $\mathfrak{T}' = (V,E,L')$ that starts with t and mirrors every rewrite step by using the corresponding original probabilistic rule for the used ADP. Hence, we get $\mathrm{edl}_{\langle \mathcal{A}(\mathcal{R}),\mathcal{A}(\mathcal{R})\rangle}(\mathfrak{T}) = \mathrm{edl}(\mathfrak{T}')$.

We label all nodes $x \in V$ in $\mathfrak{T}'$ with $\flat(t_x)$, where t_x is the term for the node x in $\mathfrak{T}$, i.e., we remove all annotations. We only have to show that $\mathfrak{T}'$ is indeed a valid RST, i.e., that the edge relation represents valid rewrite steps with $\to_{\mathcal{R}}$, but this follows directly from the fact that if we remove all annotations in Def. 3.5, then we get the ordinary probabilistic term rewrite relation again.



■

Lemma 3.19 (Sound Processors Preserve Well-Formedness). *Let $\mathfrak{P} = (V,E,L_{\mathcal{A}},L_C)$ be a proof tree with a leaf v where $L_{\mathcal{A}}(v)$ is not solved, and let* Proc *be a sound processor such that* $\mathrm{Proc}(L_{\mathcal{A}}(v)) = (c,\{\langle \mathcal{P}_1,\mathcal{S}_1\rangle,\ldots,\langle \mathcal{P}_n,\mathcal{S}_n\rangle\})$. *Let $\mathfrak{P}'$ result from $\mathfrak{P}$ by adding fresh nodes $w_1,\ldots,w_n$ and edges $(v,w_1),\ldots,(v,w_n)$, where the labeling is extended such that $L_{\mathcal{A}}(w_i) = \langle \mathcal{P}_i,\mathcal{S}_i\rangle$ for all $1 \leq i \leq n$ and $L_C(v) = c$. Then $\mathfrak{P}'$ is also well formed.*

Proof. For the first condition required for well-formed proof trees, note that in $\mathfrak{P}$, we had $L'_C(v) = \iota_{\langle \mathcal{P},\mathcal{S}\rangle}$, because v was a leaf. Since v is not a leaf anymore in $\mathfrak{P}'$, here we have $L'_C(v) = L_C(v) = c$. However, v now has the children $w_1,\ldots,w_n$. Let $v_1,\ldots,v_k = v$ be the path from the root to v. Hence, we obtain

$$\begin{aligned}
&L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1}) \oplus \iota_{\langle \mathcal{P},\mathcal{S}\rangle}\\
\sqsubseteq\; &L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1}) \oplus c \oplus \iota_{\langle \mathcal{P}_1,\mathcal{S}_1\rangle} \oplus \ldots \oplus \iota_{\langle \mathcal{P}_n,\mathcal{S}_n\rangle} \qquad (\dagger)\\
=\; &L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1}) \oplus L_C(v_k) \oplus L'_C(w_1) \oplus \ldots L'_C(w_n)
\end{aligned}$$

Here, (†) holds due to the first condition (7) of Def. 3.18 Thus, $\mathfrak{P}'$ also satisfies the first condition for well-formed proof trees.

Now we consider the second condition required for well-formed proof trees. Since $\mathfrak{P}$ is well formed and Proc is sound, the second condition (8) of Def. 3.18 implies that $\iota_{\langle \mathcal{P}_i,\mathcal{P}_i\setminus\mathcal{S}_i\rangle} \sqsubseteq L_C(v_1) \oplus \cdots \oplus L_C(v_{k-1}) \oplus L_C(v_k)$ with $L_C(v_k) = c$ holds for all $1 \leq i \leq n$. Hence, $\mathfrak{P}'$ is well-formed as well. ■

Theorem 4.2 (Usable Rules Pr.). *For an ADP problem $\langle \mathcal{P},\mathcal{S}\rangle$, let*

$$\begin{aligned}
\mathcal{P}' &= \quad \mathcal{U}(\mathcal{P}) \quad \cup \{\ell \to \mu^{\mathsf{false}} \mid \ell \to \mu^{m} \in \mathcal{P} \setminus \mathcal{U}(\mathcal{P})\},\\
\mathcal{S}' &= (\mathcal{S} \cap \mathcal{U}(\mathcal{P})) \cup \{\ell \to \mu^{\mathsf{false}} \mid \ell \to \mu^{m} \in \mathcal{S} \setminus \mathcal{U}(\mathcal{P})\}.
\end{aligned}$$

Then, $\mathrm{Proc}_{\mathsf{UR}}(\langle \mathcal{P},\mathcal{S}\rangle) = (\mathrm{Pol}_0,\{\langle \mathcal{P}',\mathcal{S}'\rangle\})$ *is sound.*

Proof. Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{P}, \mathcal{S} \rangle$ and let $v_1, \ldots, v_k = v$ be the path from the root node v_1 to v. Moreover, let $L_{\mathcal{A}}(w) = \langle \mathcal{P}', \mathcal{S}' \rangle$ for the only successor w of v in the proof tree.

We first show that (7) holds, i.e.,

$$\iota_{\langle \mathcal{P}, \mathcal{S} \rangle} \sqsubseteq L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1}) \oplus \mathrm{Pol}_0 \oplus \iota_{\langle \mathcal{P}', \mathcal{S}' \rangle}$$

Every $\mathcal{P}$-CT can also be seen as a $\mathcal{P}'$-CT, since in innermost reductions, variables are always instantiated with normal forms. Thus, the only rules applicable to the right-hand side of ADPs are the usable rules. Additionally, we start with basic terms, and hence, in every $\mathcal{P}$-CT only usable rules can be applied below the root of an annotated subterm. Thus, we have $\iota_{\langle \mathcal{P}, \mathcal{S} \rangle} = \iota_{\langle \mathcal{P}', \mathcal{S}' \rangle}$, which directly implies (7).

The condition (8) follows by the same reasoning: Every $\mathcal{P}'$-CT can also be seen as a $\mathcal{P}$-CT, so that $\iota_{\langle \mathcal{P}', \mathcal{P}' \setminus \mathcal{S}' \rangle} = \iota_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle}$. By well-formedness of $\mathfrak{P}$, we have $\iota_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle} \sqsubseteq L_C(v_1) \oplus \cdots \oplus L_C(v_{k-1})$. Thus, $\iota_{\langle \mathcal{P}', \mathcal{P}' \setminus \mathcal{S}' \rangle} = \iota_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle}$ implies (8). ■

Theorem 4.9 (Dependency Graph Proc.). *Let $\langle \mathcal{P}, \mathcal{S} \rangle$ be an ADP problem and let $\mathcal{J}$ be an SCC-prefix of the $\mathcal{P}$-dependency graph. For any ADP $\alpha = \ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^m \in \mathcal{P}$ let $\alpha|_{\mathcal{J}} = \ell \to \{p_1 : \sharp_{\Phi_1}(r_1), \ldots, p_k : \sharp_{\Phi_k}(r_k)\}^m$ where for $1 \leq j \leq k$, we have $\pi \in \Phi_j$ iff there exists an $\ell^\sharp \to t^\sharp \in \mathcal{J}$ such that $t \trianglelefteq^{\pi}_{\sharp} r_j$. Similarly, let $\mathcal{P}|_{\mathcal{J}} = \{\alpha|_{\mathcal{J}} \mid \alpha \in \mathcal{P}\}$ and $\mathcal{S}|_{\mathcal{J}} = \{\alpha|_{\mathcal{J}} \mid \alpha \in \mathcal{S}\}$.*

Then $\mathrm{Proc}_{\mathtt{DG}}(\langle \mathcal{P}, \mathcal{S} \rangle) = (\mathrm{Pol}_0, \{\langle \mathcal{P}|_{\mathcal{J}}, \mathcal{S}|_{\mathcal{J}} \rangle \mid \mathcal{J}$ is an SCC-prefix of the $\mathcal{P}$-dependency graph $\})$ is sound.

Proof. Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{P}, \mathcal{S} \rangle$, let $v_1, \ldots, v_k = v$ be the path from the root node v_1 to v, and let $c_{(v_1, \ldots, v_k)} = L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1})$.

Let $\{\mathcal{J}_1, \ldots, \mathcal{J}_n\}$ be the set of all SCC-prefixes of the $\mathcal{P}$-dependency graph. To show that (7) holds, it suffices to show

$$\iota_{\langle \mathcal{P}, \mathcal{S} \rangle} \sqsubseteq \iota_{\langle \mathcal{P}|_{\mathcal{J}_1}, \mathcal{S}|_{\mathcal{J}_1} \rangle} \oplus \ldots \oplus \iota_{\langle \mathcal{P}|_{\mathcal{J}_n}, \mathcal{S}|_{\mathcal{J}_n} \rangle} \oplus c_{(v_1, \ldots, v_k)}$$

Let $\mathfrak{T} = (V, E, L)$ be a $\mathcal{P}$-CT that starts with the term $t^\sharp$ at the root. We will create n trees $\mathfrak{T}_1, \ldots, \mathfrak{T}_n$ from $\mathfrak{T}$ such that $\mathfrak{T}_i$ is a $\mathcal{P}|_{\mathcal{J}_i}$-CT that starts with $t^\sharp$ and

$$\begin{aligned} &\mathrm{edl}_{\langle \mathcal{P}, \mathcal{S} \rangle}(\mathfrak{T}) & (31)\\ &\leq \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_1}, \mathcal{S}|_{\mathcal{J}_1} \rangle}(\mathfrak{T}_1) + \ldots + \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_n}, \mathcal{S}|_{\mathcal{J}_n} \rangle}(\mathfrak{T}_n) + B \cdot Q + D, \end{aligned}$$

where $B, D \in \mathbb{N}$ are constants and

$$Q = \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_1}, \mathcal{S}|_{\mathcal{J}_1} \rangle}(\mathfrak{T}_1) + \ldots + \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_n}, \mathcal{S}|_{\mathcal{J}_n} \rangle}(\mathfrak{T}_n) + \mathrm{edl}_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle}(\mathfrak{T}).$$

While "$\mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i}, \mathcal{S}|_{\mathcal{J}_i} \rangle}(\mathfrak{T}_i)$" considers all $(\mathbf{at})$- and $(\mathbf{af})$-steps performed with ADPs corresponding to DPs from $\mathcal{J}_i$, "$B \cdot Q + D$" considers all $(\mathbf{at})$- and $(\mathbf{af})$-steps that do not occur in any $\mathfrak{T}_i$ anymore. As (31) holds for every $\mathcal{P}$-CT $\mathfrak{T}$, we get $\iota_{\langle \mathcal{P}, \mathcal{S} \rangle} \sqsubseteq \iota_{\langle \mathcal{P}|_{\mathcal{J}_1}, \mathcal{S}|_{\mathcal{J}_1} \rangle} \oplus \ldots \oplus \iota_{\langle \mathcal{P}|_{\mathcal{J}_n}, \mathcal{S}|_{\mathcal{J}_n} \rangle} \oplus \iota_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle}$ and by well-formedness of $\mathfrak{P}$ we obtain $\iota_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle} \sqsubseteq c_{(v_1, \ldots, v_k)}$, which implies (7).

We first present the construction to get from $\mathfrak{T}$ to $\mathfrak{T}_1, \ldots, \mathfrak{T}_n$, and afterwards, we show that the number of $(\mathbf{at})$- and $(\mathbf{af})$-steps performed in $\mathfrak{T}$ that are not performed in any $\mathfrak{T}_1, \ldots, \mathfrak{T}_n$ is $\leq B \cdot Q + D$ for some constants $B, D \in \mathbb{N}$, which implies (31)

1. Every $\mathcal{P}$-CT gives rise to n CTs $\mathfrak{T}_1, \ldots, \mathfrak{T}_n$

Let $1 \leq i \leq n$. We will construct the $\mathcal{P}|_{\mathcal{J}_i}$-CT $\mathfrak{T}_i = (V, E, L_i)$ using the same underlying tree structure, an adjusted labeling such that $p_x^{\mathfrak{T}} = p_x^{\mathfrak{T}'}$ for all $x \in V$, and the term at the root of $\mathfrak{T}_i$ will still be labeled with $(1 : t^\sharp)$.

We now recursively define the new labeling L' for the $\mathcal{P}|_{\mathcal{J}_i}$-CT $\mathfrak{T}_i$. 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 all $x \in X$

$$\flat(t_x) = \flat(t'_x) \wedge \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x) \setminus \mathrm{Junk}(t_x, \mathcal{J}_i) \subseteq \mathrm{Pos}_{\mathcal{D}^\sharp}(t'_x). \qquad (32)$$

Here, for any annotated term t_x, let $\mathrm{Junk}(t_x, \mathcal{J}_i)$ denote the set of all positions of annotations in t_x that will never be used for a rewrite step in $\mathfrak{T}$ with some ADP that corresponds to the DPs from $\mathcal{J}_i$. We define $\mathrm{Junk}(t_x, \mathcal{J}_i)$ recursively: For the term t at the root, we define $\mathrm{Junk}(t, \mathcal{J}_i) = \varnothing$. For a node y_j for some $1 \leq j \leq k$ with predecessor x such that $t_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \ldots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$ at position π, we define $\mathrm{Junk}(t_{y_j}, \mathcal{J}_i) = \{\rho \mid \rho \in \mathrm{Junk}(t_x, \mathcal{J}_i), \pi \not< \rho\}$ if $\pi \notin \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x)$, and otherwise we define $\mathrm{Junk}(t_{y_j}, \mathcal{J}_i)$ to be the union of $\{\rho \mid \rho \in \mathrm{Junk}(t_x, \mathcal{J}_i), \pi \not< \rho\}$ (all positions that were already in $\mathrm{Junk}(t_x, \mathcal{J}_i)$ and are not below π), and $\{\pi.\rho \mid \rho \in \mathrm{Pos}_{\mathcal{D}^\sharp}(\alpha), \rho \notin \mathrm{Pos}_{\mathcal{D}^\sharp}(\alpha|_{\mathcal{J}_i})\}$ (all annotated positions where the annotation was removed by the dependency graph processor within the ADP). Here, as usual, $\pi < \rho$ means that π is strictly above ρ (i.e., π is a proper prefix of ρ).

We start with the same term t at the root. Here, our property (32) is clearly satisfied. As long as there is still an inner node $x \in X$ such that its successors are not contained in X, we do the following:

Let $xE = \{y_1, \ldots, y_k\}$ be the set of its successors. We need to define the terms for the nodes $y_1, \ldots, y_k$ in $\mathfrak{T}_i$. Since x is not a leaf and $\mathfrak{T}$ is a $\mathcal{P}$-CT, we have $t_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \ldots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$.

If we performed a step with $\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}}$ using the ADP α, the position π and the substitution σ in $\mathfrak{T}$, then we can use the ADP $\alpha|_{\mathcal{J}_i}$ with the same position π and the same substitution σ.

Now, we directly obtain (32) for all t_{y_j} with $1 \leq j \leq k$, since the original rule contains the same terms with more annotations, but all missing annotations are in $\mathrm{Junk}(t_x, \mathcal{J}_i)$.

2. Removed $(\mathbf{at})$- and $(\mathbf{af})$-steps are bounded by $B \cdot Q + D$.

During the above construction, it can happen that $(\mathbf{at})$- or $(\mathbf{af})$-steps from $\mathfrak{T}$ are replaced by only $(\mathbf{nt})$- and $(\mathbf{nf})$-steps in all $\mathfrak{T}_1, \ldots, \mathfrak{T}_n$. The number of those rewrite steps is bounded by $B \cdot Q + D$. To be precise, we infer a bound the number of rewrite steps taking place at a term t_x in node $x \in X$ at a position $\pi \in \bigcup_{1 \leq i \leq n} \mathrm{Junk}(t_x, \mathcal{J}_i)$ with an ADP from $\mathcal{S}$. Note that those rewrite steps belong to DPs that do not reach any SCC within the dependency graph, e.g., leaves. Moreover, we can bound the number of such rewrite steps, similar to the bound used for the knowledge propagation processor, by its (not necessarily direct) predecessors from an SCC-prefix or the ADP initially used at the root.

Let $\alpha \in \mathcal{S}$ be an ADP such that all DPs in $\mathrm{dp}^\perp(\alpha)$ cannot reach any SCC within the dependency graph. We will obtain a bound on $\mathrm{edl}_{\langle \mathcal{P}, \{\alpha\} \rangle}(\mathfrak{T})$ using the complexity of the "predecessor SCC-prefixes", i.e., those SCC-prefixes that can lead to an α-step at an annotated position later in the chain tree.

Let d be the maximal number of annotated symbols in any term on a right-hand side of an ADP from $\mathcal{P}$. Every term in the multi-distribution of some right-hand side of some ADP β with a DP in $\mathrm{dp}^\perp(\beta)$ that is a direct predecessor of some DP in $\mathrm{dp}^\perp(\alpha)$ can trigger at most d α-steps. Recall that even if α contains no annotations, we still have $\mathrm{dp}^\perp(\alpha) \neq \varnothing$. Moreover, every ADP with a DP in $\mathrm{dp}^\perp(\beta)$

that is a 2-step predecessor of some DP in $\mathrm{dp}^{\perp}(\alpha)$ (i.e., we can reach a DP in $\mathrm{dp}^{\perp}(\alpha)$ from a DP in $\mathrm{dp}^{\perp}(\beta)$ in at most 2 steps) can trigger at most d^2 α-steps. In general, every e-step predecessor can trigger at most d^e α-steps. A path in the dependency graph starting at a DP from some SCC-prefix that reaches a DP from $\mathrm{dp}^{\perp}(\alpha)$ without node repetition has a length of at most $|\mathrm{dp}(\mathcal{P})|$. Moreover, we can also create α-steps without going through an SCC-prefix, by following a path from the initially used ADP to α. Overall, we get

$$\mathrm{edl}_{\langle \mathcal{P},\{\alpha\}\rangle}(\mathfrak{T}) \leq d^{|\mathrm{dp}(\mathcal{P})|} \cdot (1 + \sum_{i=1}^{n} \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{P}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i)). \tag{33}$$

Since SCC-prefixes may contain nodes from $\mathrm{dp}(\mathcal{S})$ and $\mathrm{dp}(\mathcal{P} \setminus \mathcal{S})$ we have

$$\begin{aligned} &\mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{P}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i) \\ =\ &\mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{S}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i) + \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{P}|_{\mathcal{J}_i}\setminus\mathcal{S}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i) \\ \leq\ &\mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{S}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i) + \mathrm{edl}_{\langle \mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle}(\mathfrak{T}). \end{aligned} \tag{34}$$

Hence, we even have

$$\begin{aligned} &\mathrm{edl}_{\langle \mathcal{P},\{\alpha\}\rangle}(\mathfrak{T}) \\ &\leq d^{|\mathrm{dp}(\mathcal{P})|} \cdot (1 + \textstyle\sum_{j=1}^{n} \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{P}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i)) \quad \text{by (33)} \\ &\leq d^{|\mathrm{dp}(\mathcal{P})|} \cdot (1 + \textstyle\sum_{j=1}^{n} (\mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{S}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i) + \mathrm{edl}_{\langle \mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle}(\mathfrak{T}))) \quad \text{by (34)} \\ &= d^{|\mathrm{dp}(\mathcal{P})|} \cdot (1 + n \cdot \mathrm{edl}_{\langle \mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle}(\mathfrak{T}) + \textstyle\sum_{j=1}^{n} \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{S}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i)) \\ &\leq d^{|\mathrm{dp}(\mathcal{P})|} + d^{|\mathrm{dp}(\mathcal{P})|} \cdot n \cdot (\mathrm{edl}_{\langle \mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle}(\mathfrak{T}) + \textstyle\sum_{j=1}^{n} \mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{S}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}_i)). \end{aligned}$$

So the number of $(\mathbf{at})$- and $(\mathbf{af})$-steps with such an ADP α is $\leq d^{|\mathrm{dp}(\mathcal{P})|} \cdot n \cdot Q + d^{|\mathrm{dp}(\mathcal{P})|}$. Since there are at most $|\mathcal{P}|$ such ADPs α, the final constants B and D for (31) are $B = |\mathcal{P}| \cdot d^{|\mathrm{dp}(\mathcal{P})|} \cdot n$ and $D = |\mathcal{P}| \cdot d^{|\mathrm{dp}(\mathcal{P})|}$.
Now we show that (8) holds. For all $1 \leq i \leq n$ we have to show

$$\iota_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{P}|_{\mathcal{J}_i}\setminus\mathcal{S}|_{\mathcal{J}_i}\rangle} \sqsubseteq c_{(v_1,\ldots,v_k)}.$$

Every $\mathcal{P}|_{\mathcal{J}_i}$-CT $\mathfrak{T}$ gives rise to a $\mathcal{P}$-CT $\mathfrak{T}'$ by using the same ADPs just with (possibly) more annotations. Hence, for every such $\mathfrak{T}$ we get $\mathrm{edl}_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{P}|_{\mathcal{J}_i}\setminus\mathcal{S}|_{\mathcal{J}_i}\rangle}(\mathfrak{T}) \leq \mathrm{edl}_{\langle \mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle}(\mathfrak{T}')$, and thus, we have $\iota_{\langle \mathcal{P}|_{\mathcal{J}_i},\mathcal{P}|_{\mathcal{J}_i}\setminus\mathcal{S}|_{\mathcal{J}_i}\rangle} \sqsubseteq \iota_{\langle \mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle}$. So (8) holds by well-formedness of $\mathfrak{P}$, which implies $\iota_{\langle \mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle} \sqsubseteq c_{(v_1,\ldots,v_k)}$. ■

Theorem 4.12 (Reduction Pair Proc.). *Let $\mathcal{I} : \Sigma^{\#} \to \mathbb{N}(\mathcal{V})$ be a weakly monotonic, multilinear polynomial interpretation. Let $\langle \mathcal{P}, \mathcal{S}\rangle$ be an ADP problem where $\mathcal{P} = \mathcal{P}_{\geq} \uplus \mathcal{P}_{>}$ and $\mathcal{P}_{>} \cap \mathcal{S} \neq \varnothing$ such that:*
(1) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{\mathsf{true}} \in \mathcal{P}$: $\mathcal{I}(\ell) \geq \sum_{1\leq j\leq k} p_j \cdot \mathcal{I}(\flat(r_j))$
(2) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{m} \in \mathcal{P}_{\geq}$: $\mathcal{I}(\ell^{\#}) \geq \sum_{1\leq j\leq k} p_j \cdot \mathcal{I}^{\#}_{\Sigma}(r_j)$
(3) For every $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\}^{m} \in \mathcal{P}_{>}$: $\mathcal{I}(\ell^{\#}) > \sum_{1\leq j\leq k} p_j \cdot \mathcal{I}^{\#}_{\Sigma}(r_j)$, where $\mathcal{I}^{\#}_{\Sigma}(r_j) = \sum_{t \trianglelefteq_{\#} r_j} \mathcal{I}(t^{\#})$.

Then $\mathrm{Proc}_{\mathsf{RP}}(\langle \mathcal{P}, \mathcal{S}\rangle) = (c, \langle \mathcal{P}, \mathcal{S} \setminus \mathcal{P}_{>}\rangle)$ is sound, where the complexity $c \in \mathfrak{C}$ is determined as follows: If $\mathcal{I}$ is a CPI and for all annotated symbols $f^{\#} \in \mathcal{D}^{\#}$, the polynomial $I_{f^{\#}}$ has at most degree a, then $c = \mathrm{Pol}_a$. If $\mathcal{I}$ is not a CPI, then $c = \mathrm{Exp}$ if all constructors are interpreted by linear polynomials, and otherwise $c = 2\text{-}\mathrm{Exp}$.

Proof. Let $\mathcal{I}^{\#}_{0}$ behave exactly as $\mathcal{I}^{\#}_{\Sigma}$ but in addition, it maps every variable to 0. Moreover, let poloC : $\mathbb{N} \to \mathbb{N} \cup \{\omega\}$ be the function that maps n to the maximal interpretation of any basic term of size $\leq n$ which is annotated at the root. Thus, $\mathrm{poloC}(n) = \sup\{\mathcal{I}^{\#}_{0}(t^{\#}) \mid t \in \mathcal{TB} \text{ and } |t| \leq n\}$.

As in [36], the conditions (1), (2), (3) from Thm. 4.12 can be lifted to rewrite steps with $\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}}$ instead of just rules and, therefore, to edges of a CT. For each ADP $\alpha = \ell \to \{p_1 : r_j, \ldots, p_k : r_k\}^m \in \mathcal{P}_{>}$ we can find an $\varepsilon_\alpha > 0$ such that $\mathcal{I}(\ell^{\#}) \geq \varepsilon_\alpha + \sum_{1\leq j\leq k} p_j \cdot \mathcal{I}^{\#}_{\Sigma}(r_j)$. Let $\varepsilon = \min\{\varepsilon_\alpha \mid \alpha \in \mathcal{P}_{>}\}$ be the smallest such number. After the lifting, we get:

(1) For every $s \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \mu$ with $(\mathbf{nt})$ or $(\mathbf{nf})$, we have
$$\mathcal{I}^{\#}_{\Sigma}(s) \geq \textstyle\sum_{1\leq j\leq k} p_j \cdot \mathcal{I}^{\#}_{\Sigma}(t)$$
(2) For every $s \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \mu$ with $(\mathbf{at})$ or $(\mathbf{af})$ and an ADP from $\mathcal{P}_{\geq}$, we have
$$\mathcal{I}^{\#}_{\Sigma}(s) \geq \textstyle\sum_{1\leq j\leq k} p_j \cdot \mathcal{I}^{\#}_{\Sigma}(t)$$
(3) For every $s \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \mu$ with $(\mathbf{at})$ or $(\mathbf{af})$ and an ADP from $\mathcal{P}_{>}$, we have
$$\mathcal{I}^{\#}_{\Sigma}(s) \geq \varepsilon + \textstyle\sum_{1\leq j\leq k} p_j \cdot \mathcal{I}^{\#}_{\Sigma}(t)$$

We only have to prove that the complexity of $\mathcal{P}$ when counting only $\mathcal{P}_{>}$-rules is bounded by ι_{poloC}, i.e, $\iota_{\langle \mathcal{P},\mathcal{P}_{>}\rangle} \sqsubseteq \iota_{\mathrm{poloC}}$. Then, $\iota_{\mathrm{poloC}} \sqsubseteq c$ follows by the same reasoning that was used in Sect. A.1 to find upper bounds on the polynomial interpretation of terms of size n. Instead of basic terms, here we only need to consider terms that have an annotation at the root, and constructor terms as proper subterms.

Afterwards, we can conclude that (7) holds, i.e.,

$$\iota_{\langle \mathcal{P},\mathcal{S}\rangle} \sqsubseteq c \oplus \iota_{\langle \mathcal{P},\mathcal{S}\setminus\mathcal{P}_{>}\rangle},$$

because

$$\begin{aligned} &\iota_{\langle \mathcal{P},\mathcal{S}\rangle} \\ &= \iota_{\langle \mathcal{P},\mathcal{S}\cap\mathcal{P}_{>}\rangle} \oplus \iota_{\langle \mathcal{P},\mathcal{S}\setminus\mathcal{P}_{>}\rangle} \\ &\sqsubseteq \iota_{\langle \mathcal{P},\mathcal{P}_{>}\rangle} \oplus \iota_{\langle \mathcal{P},\mathcal{S}\setminus\mathcal{P}_{>}\rangle} \\ &\sqsubseteq c \oplus \iota_{\langle \mathcal{P},\mathcal{S}\setminus\mathcal{P}_{>}\rangle} \end{aligned}$$

Similarly, one can also conclude that (8) holds. Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{P}, \mathcal{S}\rangle$, let $v_1, \ldots, v_k = v$ be the path from the root node v_1 to v, and let $c_{(v_1,\ldots,v_k)} = L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1})$. Then we have

$$\iota_{\langle \mathcal{P},\mathcal{P}\setminus(\mathcal{S}\setminus\mathcal{P}_{>})\rangle} \sqsubseteq c_{(v_1,\ldots,v_k)} \oplus c$$

because $\iota_{\langle \mathcal{P},\mathcal{P}\setminus(\mathcal{S}\setminus\mathcal{P}_{>})\rangle} \sqsubseteq \iota_{\langle \mathcal{P},(\mathcal{P}\setminus\mathcal{S})\cup\mathcal{P}_{>}\rangle}$ and

$$\iota_{\langle \mathcal{P},(\mathcal{P}\setminus\mathcal{S})\cup\mathcal{P}_{>}\rangle} = \iota_{\langle \mathcal{P},\mathcal{P}\setminus\mathcal{S}\rangle} \oplus \iota_{\langle \mathcal{P},\mathcal{P}_{>}\rangle} \sqsubseteq c_{(v_1,\ldots,v_k)} \oplus c$$

So it remains to show that $\iota_{\langle \mathcal{P},\mathcal{P}_{>}\rangle} \sqsubseteq \iota_{\mathrm{poloC}}$ holds. Consider a basic term t and a $\mathcal{P}$-CT $\mathfrak{T}$ with $t^{\#}$ at the root. We want to show that up to a factor, $\mathcal{I}^{\#}_{0}(t^{\#})$ is a bound on the expected derivation length of $\mathfrak{T}$.

We have $\mathcal{I}^{\#}_{0}(t^{\#})$ as the initial value at the root. Whenever we perform a rewrite step with $(\mathbf{nt})$ or $(\mathbf{nf})$ or with an ADP from $\mathcal{P}_{\geq}$, we weakly decrease the value in expectation. And whenever we perform a rewrite step with $(\mathbf{at})$ or $(\mathbf{af})$ and an ADP from $\mathcal{P}_{>}$ at a node v, we strictly decrease the value by at least $\varepsilon \cdot p_v$ in expectation.

This behavior is illustrated in Fig. 7. Since the value is bounded from below by 0, we get:

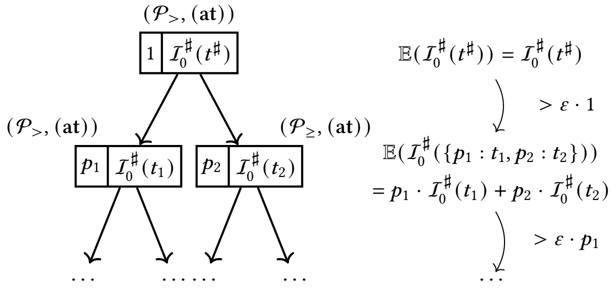
$$\mathcal{I}_0^\sharp(t^\sharp) - \sum_{v \in V \setminus \mathrm{Leaf}^{\mathfrak{T}}} \sum_{\mathcal{P}(v) \in \mathcal{P}_> \times \{(\mathbf{at}),(\mathbf{af})\}} \varepsilon \cdot p_v \geq 0$$

$$\iff \sum_{v \in V \setminus \mathrm{Leaf}^{\mathfrak{T}}} \sum_{\mathcal{P}(v) \in \mathcal{P}_> \times \{(\mathbf{at}),(\mathbf{af})\}} \varepsilon \cdot p_v \leq \mathcal{I}_0^\sharp(t^\sharp) \qquad (35)$$

Therefore, we get:

$$\begin{aligned} &\varepsilon \cdot \mathrm{edl}_{\langle \mathcal{P},\mathcal{P}_> \rangle}(\mathfrak{T}) \\ &= \sum_{v \in V \setminus \mathrm{Leaf}^{\mathfrak{T}}} \sum_{\mathcal{P}(v) \in \mathcal{P}_> \times \{(\mathbf{at}),(\mathbf{af})\}} \varepsilon \cdot p_v && \text{(definition of edl)} \\ &\leq \mathcal{I}_0^\sharp(t^\sharp) && \text{(with (35))} \end{aligned}$$

and therefore, $\mathrm{edl}_{\langle \mathcal{P},\mathcal{P}_> \rangle}(\mathfrak{T}) \leq {}^1/\varepsilon \cdot \mathcal{I}_0^\sharp(t^\sharp)$ and thus, $\iota_{\langle \mathcal{P},\mathcal{P}_> \rangle} \sqsubseteq \iota_{\mathrm{poloC}}$. ∎



$(\mathcal{P}_>, (\mathbf{at}))$
1
$\mathcal{I}_0^\sharp(t^\sharp)$
$(\mathcal{P}_>, (\mathbf{at}))$
$(\mathcal{P}_\geq, (\mathbf{at}))$
p_1
$\mathcal{I}_0^\sharp(t_1)$
p_2
$\mathcal{I}_0^\sharp(t_2)$
$\mathbb{E}(\mathcal{I}_0^\sharp(t^\sharp)) = \mathcal{I}_0^\sharp(t^\sharp)$
$> \varepsilon \cdot 1$
$\mathbb{E}(\mathcal{I}_0^\sharp(\{p_1 : t_1, p_2 : t_2\}))$
$= p_1 \cdot \mathcal{I}_0^\sharp(t_1) + p_2 \cdot \mathcal{I}_0^\sharp(t_2)$
$> \varepsilon \cdot p_1$

Figure 7: Expected Decrease of $\mathcal{I}_0^\sharp$ in a Chain Tree.

Theorem 4.14 (Knowledge Propagation Proc.). *Let $\langle \mathcal{P}, \mathcal{S} \rangle$ be an ADP problem, let $\alpha \in \mathcal{S}$ and $\mathrm{Pre}(\alpha) \cap \mathcal{S} = \varnothing$, where $\mathrm{Pre}(\alpha)$ consists of all ADPs $\beta \in \mathcal{P}$ such that there is an edge from some DP in $\mathrm{dp}^\perp(\beta)$ to some DP in $\mathrm{dp}^\perp(\alpha)$ in the $\mathcal{P}$-dependency graph. Then the following processor is sound:*

$$\mathrm{Proc}_{\mathsf{KP}}(\langle \mathcal{P}, \mathcal{S} \rangle) = (\mathrm{Pol}_0, \langle \mathcal{P}, \mathcal{S} \setminus \{\alpha\} \rangle)$$

Proof. Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{P}, \mathcal{S} \rangle$, let $v_1, \ldots, v_k = v$ be the path from the root node v_1 to v, and let $c_{(v_1,\ldots,v_k)} = L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1})$.

We first show that (7) holds, i.e.,

$$\iota_{\langle \mathcal{P},\mathcal{S} \rangle} \sqsubseteq c_{(v_1,\ldots,v_k)} \oplus \mathrm{Pol}_0 \oplus \iota_{\langle \mathcal{P},\mathcal{S} \setminus \{\alpha\} \rangle}$$

Let $\mathfrak{T} = (V, E, L)$ be a $\mathcal{P}$-CT. Let d be the maximal number of annotated symbols in any term on a right-hand side of an ADP from $\mathcal{P}$. Recall that

$$\begin{aligned} \mathrm{edl}_{\langle \mathcal{P},\{\alpha\} \rangle}(\mathfrak{T}) &\leq 1 + \textstyle\sum_{v \in V \setminus \mathrm{Leaf}^{\mathfrak{T}},\, \mathcal{P}(v) \in \mathrm{Pre}(\alpha) \times \{(\mathbf{at}),(\mathbf{af})\}} d \cdot p_v \\ &= 1 + d \cdot \mathrm{edl}_{\langle \mathcal{P},\mathrm{Pre}(\alpha) \rangle}(\mathfrak{T}), \end{aligned}$$

which implies $\mathrm{erc}_{\langle \mathcal{P},\{\alpha\} \rangle}(n) \leq 1 + d \cdot \mathrm{erc}_{\langle \mathcal{P},\mathrm{Pre}(\alpha) \rangle}(n)$ for all $n \in \mathbb{N}$ and thus, $\iota_{\langle \mathcal{P},\{\alpha\} \rangle} \sqsubseteq \iota_{\langle \mathcal{P},\mathrm{Pre}(\alpha) \rangle}$. Hence, (7) holds, because of well-formedness of $\mathfrak{P}$, i.e.,

$$\begin{aligned} \iota_{\langle \mathcal{P},\mathcal{S} \rangle} &= \iota_{\langle \mathcal{P},\mathcal{S} \setminus \{\alpha\} \rangle} \oplus \iota_{\langle \mathcal{P},\{\alpha\} \rangle} \sqsubseteq \iota_{\langle \mathcal{P},\mathcal{S} \setminus \{\alpha\} \rangle} \oplus \iota_{\langle \mathcal{P},\mathrm{Pre}(\alpha) \rangle} \\ &\sqsubseteq \iota_{\langle \mathcal{P},\mathcal{S} \setminus \{\alpha\} \rangle} \oplus \iota_{\langle \mathcal{P},\mathcal{P} \setminus \mathcal{S} \rangle} \sqsubseteq \iota_{\langle \mathcal{P},\mathcal{S} \setminus \{\alpha\} \rangle} \oplus c_{(v_1,\ldots,v_k)} \end{aligned}$$

Now we show (8), i.e.,

$$\iota_{\langle \mathcal{P},\mathcal{P} \setminus (\mathcal{S} \setminus \{\alpha\}) \rangle} \sqsubseteq c_{(v_1,\ldots,v_k)} \oplus \mathrm{Pol}_0$$

This holds again because $\mathfrak{P}$ is well formed:

$$\begin{aligned} \iota_{\langle \mathcal{P},\mathcal{P} \setminus (\mathcal{S} \setminus \{\alpha\}) \rangle} &= \iota_{\langle \mathcal{P},(\mathcal{P} \setminus \mathcal{S}) \cup \{\alpha\} \rangle} = \iota_{\langle \mathcal{P},\mathcal{P} \setminus \mathcal{S} \rangle} \oplus \iota_{\langle \mathcal{P},\{\alpha\} \rangle} \\ &\sqsubseteq c_{(v_1,\ldots,v_k)} \oplus c_{(v_1,\ldots,v_k)} = c_{(v_1,\ldots,v_k)} \end{aligned}$$

∎

Next, we prove the soundness of probability removal processor. Here, we follow the notation of [53] for the non-probabilistic dependency tuple framework.

In [53], the runtime complexity for a DT problem $(\mathcal{P}, \mathcal{S}, \mathcal{K}, \mathcal{R})$ is defined via *$\langle \mathcal{P}, \mathcal{R} \rangle$-DT chain trees.* Nodes of these trees are labeled by pairs $(\ell^\sharp \to [t_1^\sharp, \ldots, t_n^\sharp], \sigma)$ of a DT and a substitution such that $\ell^\sharp\sigma \in \mathsf{NF}_{\mathcal{R}}$, and if a node $(\ell^\sharp \to [t_1^\sharp, \ldots, t_n^\sharp], \sigma)$ has children $(\ell_1^\sharp \to \ldots, \delta_1), \ldots, (\ell_k^\sharp \to \ldots, \delta_k)$, then there are pairwise different $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $t_{i_j}^\sharp \sigma \xrightarrow{\mathrm{i}}{}_{\mathcal{R}}^{*} \ell_j^\sharp \delta_j$ for all $1 \leq j \leq k$.[6] The *$\mathcal{S}$-derivation length* $\mathrm{dl}_{\langle \mathcal{P},\mathcal{S},\mathcal{R} \rangle}(\mathfrak{T})$ of a $\langle \mathcal{P}, \mathcal{R} \rangle$-DT chain tree $\mathfrak{T}$ is the number of its nodes labeled by DTs from $\mathcal{S}$. The *derivation height* of a term $t^\sharp$ w.r.t. $\langle \mathcal{P}, \mathcal{S}, \mathcal{R} \rangle$ is the supremum over all $\mathcal{S}$-derivation lengths of all $\langle \mathcal{P}, \mathcal{R} \rangle$-DT chain trees starting with $t^\sharp$, i.e.,

$$\begin{aligned} &\mathrm{dh}_{\langle \mathcal{P},\mathcal{S},\mathcal{R} \rangle}(t^\sharp) = \\ &\quad \sup\{\mathrm{dl}_{\langle \mathcal{P},\mathcal{S},\mathcal{R} \rangle}(\mathfrak{T}) \mid \mathfrak{T} \text{ is a } \langle \mathcal{P}, \mathcal{R} \rangle\text{-DT chain tree starting with } t^\sharp\}. \end{aligned}$$

Finally, the runtime complexity function of $\langle \mathcal{P}, \mathcal{S}, \mathcal{R} \rangle$ is defined in a similar way as before

$$\mathrm{rc}_{\langle \mathcal{P},\mathcal{S},\mathcal{R} \rangle}(n) = \sup\{\mathrm{dh}_{\langle \mathcal{P},\mathcal{S},\mathcal{R} \rangle}(t^\sharp) \mid t \in \mathcal{TB}_{\mathcal{R}}, |t| \leq n\},$$

and its *runtime complexity* $\iota_{\langle \mathcal{P},\mathcal{S},\mathcal{R} \rangle}$ is $\iota(\mathrm{rc}_{\langle \mathcal{P},\mathcal{S},\mathcal{R} \rangle})$.

As mentioned before, in the non-probabilistic DT framework one has two components $\mathcal{S}$ and $\mathcal{K}$ for those DTs that we still need to count and for those DTs for which we already have a complexity bound on how often this DT can occur in a $\langle \mathcal{P}, \mathcal{R} \rangle$-DT chain tree, respectively. The overall complexity of a DT problem $(\mathcal{P}, \mathcal{S}, \mathcal{K}, \mathcal{R})$ is $\iota_{(\mathcal{P},\mathcal{S},\mathcal{K},\mathcal{R})} = \iota_{\langle \mathcal{P},\mathcal{S},\mathcal{R} \rangle} \ominus \iota_{\langle \mathcal{P},\mathcal{K},\mathcal{R} \rangle}$.[7] Here, $c \ominus d = c$ if $d \sqsubset c$ and $c \ominus d = \mathrm{Pol}_0$ otherwise (so, e.g., $\mathrm{Pol}_2 \ominus \mathrm{Pol}_1 = \mathrm{Pol}_2$ and $\mathrm{Pol}_1 \ominus \mathrm{Pol}_2 = \mathrm{Pol}_0$)

Theorem 4.16 (Probability Removal Processor). *Let $\langle \mathcal{P}, \mathcal{S} \rangle$ be an ADP problem where every ADP in $\mathcal{P}$ has the form $\ell \to \{1 : r\}^m$. Let $\mathrm{dt}(\ell \to \{1 : r\}^m) = \ell^\sharp \to [t_1^\sharp, \ldots, t_n^\sharp]$ if $\{t \mid t \trianglelefteq_\sharp r\} = \{t_1, \ldots, t_n\}$, and let $\mathrm{dt}(\mathcal{P}) = \{\mathrm{dt}(\alpha) \mid \alpha \in \mathcal{P}\}$. Then the expected runtime complexity of $\langle \mathcal{P}, \mathcal{S} \rangle$ is equal to the runtime complexity of the non-probabilistic DT problem $\beta = (\mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{dt}(\mathcal{P} \setminus \mathcal{S}), \mathrm{np}(\mathcal{P}))$. So the processor $\mathrm{Proc}_{\mathsf{PR}}(\langle \mathcal{P}, \mathcal{S} \rangle) = (c, \varnothing)$ is sound if the DT framework returns c as an upper bound on the runtime complexity of β.*

Proof. First, every $\mathcal{P}$-CT $\mathfrak{T}$ gives rise to a $\langle \mathrm{dt}(\mathcal{P}), \mathrm{np}(\mathcal{P}) \rangle$-DT chain tree $\mathfrak{T}'$ such that $\mathrm{edh}_{\langle \mathcal{P},X \rangle}(\mathfrak{T}) = \mathrm{dh}_{\langle \mathrm{dt}(\mathcal{P}),\mathrm{dt}(X),\mathrm{np}(\mathcal{P}) \rangle}(\mathfrak{T}')$ for every $X \subseteq \mathcal{P}$, and vice versa. To see this, note that every $\mathcal{P}$-CT is a single (not necessarily finite) path, due to the trivial probabilities in its rules. Moreover, the DTs α in $\mathfrak{T}'$ correspond to those ADPs β that are used at annotated subterms, i.e., with Case $(\mathbf{at})$ or $(\mathbf{af})$, and we

[6]So the branching in $\langle \mathcal{P}, \mathcal{R} \rangle$-DT chain trees is not due to probabilities but due to the right-hand sides of DTs containing several terms that can be evaluated.
[7]We do not need such a $\ominus$-operation due to our notion of well-formed proof trees that did not exist in [53].

have $\alpha = \mathrm{dt}(\beta)$. The substitution is the corresponding substitution used for the rewrite step at this node.

To be precise, if we have $t \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{1 : s\}$ at the root of $\mathfrak{T}$, where we use an ADP $\alpha = \ell \to \{1 : r\}^m \in \mathcal{P}$ at position π with the substitution σ such that $\flat(t|_\pi) = \ell\sigma \in \mathsf{ANF}_{\mathcal{P}}$, then we start with $(\mathrm{dt}(\alpha), \sigma)$ at the root of $\mathfrak{T}'$. Let $\mathrm{dt}(\alpha) = \ell^\sharp \to [r_1^\sharp, \dots, r_n^\sharp]$. If we eventually rewrite at the subterm corresponding to some $r_j^\sharp$ in $\mathfrak{T}$ with an ADP $\alpha' = \ell' \to \{1 : r'\}^{m'} \in \mathcal{P}$ and the substitution σ' and this subterm still contains its annotation at the root, then we must have $r_j^\sharp\sigma \xrightarrow{\mathrm{i}}_{\mathrm{np}(\mathcal{P})} \ell'^\sharp\sigma'$. Hence, we can create a child node of $(\mathrm{dt}(\alpha), \sigma)$ labeled by $(\mathrm{dt}(\alpha'), \sigma')$. We can construct the whole tree $\mathfrak{T}'$ inductively. Note that whenever we perform a rewrite step at an annotated position in $\mathfrak{T}$, we create a corresponding node in $\mathfrak{T}'$, hence we have $\mathrm{edh}_{\langle \mathcal{P}, X\rangle}(\mathfrak{T}) = \mathrm{dh}_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(X), \mathrm{np}(\mathcal{P})\rangle}(\mathfrak{T}')$.

For the converse, i.e., to get from a $\langle \mathrm{dt}(\mathcal{P}), \mathrm{np}(\mathcal{P})\rangle$-DT chain tree $\mathfrak{T}'$ to the corresponding $\mathcal{P}$-CT $\mathfrak{T}$, we simply perform all rewrite steps with $\mathcal{R}$ in $\mathfrak{T}$ that are omitted in $\mathfrak{T}'$. To be precise, if we have $(\mathrm{dt}(\alpha), \sigma)$ at the root of $\mathfrak{T}'$ for some ADP $\alpha = \ell \to \{1 : r\}^m \in \mathcal{P}$, then we can perform the rewrite step $\ell^\sharp\sigma \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{1 : r\sigma\}$ at the root of $\mathfrak{T}$. Let $\mathrm{dt}(\alpha) = \ell^\sharp \to [r_1^\sharp, \dots, r_n^\sharp]$. If there is a child labeled by $(\mathrm{dt}(\alpha'), \sigma')$ of the root in $\mathfrak{T}'$, then we have $r_j^\sharp\sigma \xrightarrow{\mathrm{i}}_{\mathrm{np}(\mathcal{P})} \ell'^\sharp\sigma'$ for some $1 \leq j \leq n$, where ℓ' is the left-hand side of α'. We can simply perform those rewrite steps in $\mathfrak{T}$. Since $\mathrm{np}(\mathcal{P})$ only contains rules resulting from ADPs with the flag $m = \mathrm{true}$, this does not remove any annotations from subterms that are not in normal form. Hence, we can do this iteratively for every such child. In this way, we can create the whole tree $\mathfrak{T}$ inductively. Again, for every node of the $\langle \mathrm{dt}(\mathcal{P}), \mathrm{np}(\mathcal{P})\rangle$-DT chain tree, we perform a rewrite step with Case **(at)** or **(af)** in the $\mathcal{P}$-CT. Therefore, $\mathrm{edh}_{\langle \mathcal{P}, X\rangle}(\mathfrak{T}) = \mathrm{dh}_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(X), \mathrm{np}(\mathcal{P})\rangle}(\mathfrak{T}')$.

As this relation holds for arbitrary $\mathcal{P}$-CTs and $\langle \mathrm{dt}(\mathcal{P}), \mathrm{np}(\mathcal{P})\rangle$-DT chain trees, we obtain $\iota_{\langle \mathcal{P}, X\rangle} = \iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(X), \mathrm{np}(\mathcal{P})\rangle}$ for every $X \subseteq \mathcal{P}$.

We can now conclude soundness: Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{P}, \mathcal{S}\rangle$, let $v_1, \dots, v_k = v$ be the path from the root node v_1 to v, and let $c_{(v_1, \dots, v_k)} = L_C(v_1) \oplus \dots \oplus L_C(v_{k-1})$.

Since the resulting ADP problems are solved, we only have to show (7). Let $\mathcal{K} = \mathcal{P} \setminus \mathcal{S}$. If we have $\iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{np}(\mathcal{P})\rangle} \sqsubseteq \iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{K}), \mathrm{np}(\mathcal{P})\rangle}$, then $\iota_{\langle \mathcal{P}, \mathcal{S}\rangle} \sqsubseteq \iota_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S}\rangle} \sqsubseteq c_{(v_1, \dots, v_k)}$ by well-formedness of $\mathfrak{P}$, which proves that $\mathrm{Proc}_{\mathsf{PR}}(\langle \mathcal{P}, \mathcal{S}\rangle) = (c, \varnothing)$ is sound for every $c \in \mathcal{C}$.

Next, consider the case $\iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{K}), \mathrm{np}(\mathcal{P})\rangle} \sqsubset \iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{np}(\mathcal{P})\rangle}$. This implies that $\iota_{(\mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{dt}(\mathcal{K}), \mathrm{np}(\mathcal{P}))} = \iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{np}(\mathcal{P})\rangle} \ominus \iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{K}), \mathrm{np}(\mathcal{P})\rangle} = \iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{np}(\mathcal{P})\rangle}$, and therefore, we have $\iota_{\langle \mathcal{P}, \mathcal{S}\rangle} = \iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{np}(\mathcal{P})\rangle} = \iota_{\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{dt}(\mathcal{K}), \mathrm{np}(\mathcal{P})\rangle} \sqsubseteq c$, since c is returned by the DT framework as a bound on the runtime complexity of $\langle \mathrm{dt}(\mathcal{P}), \mathrm{dt}(\mathcal{S}), \mathrm{dt}(\mathcal{K}), \mathrm{np}(\mathcal{P})\rangle$. Again, this proves soundness of a processor with $\mathrm{Proc}_{\mathsf{PR}}(\langle \mathcal{P}, \mathcal{S}\rangle) = (c, \varnothing)$. ■

Theorem 4.17 (Rule Overlap Instantiation Processor). *Let $\langle \mathcal{P}, \mathcal{S}\rangle$ be an ADP problem with $\mathcal{P} = \mathcal{P}' \uplus \{\alpha\}$ for $\alpha = \ell \to \{p_1 : r_1, \dots, p_k : r_k\}^m$, let $1 \leq j \leq k$, and let $t \trianglelefteq_\sharp r_j$. Let $\delta_1, \dots, \delta_d$ be all narrowing substitutions of t. Then* $\mathrm{Proc}_{\mathtt{ROI}}(\langle \mathcal{P}, \mathcal{S}\rangle) = (\mathrm{Pol}_0, \{\langle \mathcal{P}' \cup N, \widetilde{\mathcal{S}}\rangle\})$ *is sound, where*

$$N = \{\ell\delta_e \to \{p_1 : r_1\delta_e, \dots, p_k : r_k\delta_e\}^m \mid 1 \leq e \leq d\} \cup \{\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \sharp_{\mathrm{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m\}$$

$$\widetilde{\mathcal{S}} = \begin{cases} (\mathcal{S} \setminus \{\alpha\}) \cup N, & \textit{if } \alpha \in \mathcal{S} \\ \mathcal{S}, & \textit{otherwise} \end{cases}$$

Proof. We first show that (7) holds, i.e., we show

$$\iota_{\langle \mathcal{P}, \mathcal{S}\rangle} \sqsubseteq \iota_{\langle \mathcal{P}' \cup N, \widetilde{\mathcal{S}}\rangle}$$

Every $\mathcal{P}$-CT $\mathfrak{T}$ gives rise to a $(\mathcal{P}' \cup N)$-CT $\mathfrak{T}'$, since every rewrite step with α can also be done with a rule from N. Moreover, if $\alpha \in \mathcal{S}$, then $N \subseteq \widetilde{\mathcal{S}}$, so every rewrite step that counts for the expected derivation length of $\mathfrak{T}$ w.r.t. $\langle \mathcal{P}, \mathcal{S}\rangle$ still counts for the expected derivation length of $\mathfrak{T}'$ w.r.t. $\langle \mathcal{P}' \cup N, \widetilde{\mathcal{S}}\rangle$. To be precise, we get $\mathrm{edl}_{\langle \mathcal{P}, \mathcal{S}\rangle}(\mathfrak{T}) \leq \mathrm{edl}_{\langle \mathcal{P}' \cup N, \widetilde{\mathcal{S}}\rangle}(\mathfrak{T})$.[8] As this holds for every $\mathcal{P}$-CT, we obtain $\iota_{\langle \mathcal{P}, \mathcal{S}\rangle} \sqsubseteq \iota_{\langle \mathcal{P}' \cup N, \widetilde{\mathcal{S}}\rangle}$, and (7) holds.

Now we explain the precise construction. Let $\mathfrak{T} = (V, E, L)$ be a $\mathcal{P}$-CT and let $\overline{\mathcal{P}'} = \mathcal{P}' \cup N$. We will create an $\overline{\mathcal{P}'}$-CT $\mathfrak{T}' = (V, E, L')$. As mentioned, the core idea of this construction is that every rewrite step with α can also be done with a rule from N. If we use $\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \sharp_{\mathrm{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m \in N$, we may create fewer annotations than we did when using the old ADP α. However, we will never rewrite at the position of the annotations that got removed 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 $x \in V \setminus \mathrm{Leaf}$ with $xE = \{y_1, \dots, y_m\}$ we have $t'_x \overset{\mathrm{i}}{\hookrightarrow}_{\overline{\mathcal{P}'}} \{\frac{p_{y_1}}{p_x} : t'_{y_1}, \dots, \frac{p_{y_m}}{p_x} : t'_{y_m}\}$. 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 for all $x \in X$:

$$\flat(t_x) = \flat(t'_x) \wedge \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x) \cap \mathrm{Pos}_{\mathcal{D} \wedge \neg \mathsf{NF}_{\mathcal{R}}}(\flat(t_x)) \subseteq \mathrm{Pos}_{\mathcal{D}^\sharp}(t'_x). \quad (36)$$

Thus, all annotations of root symbols of subterms that are not in normal form in t_x are still annotated in t'_x.

For the construction, we start with the same term at the root. Here, (36) 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 terms $t'_{y_1}, \dots, t'_{y_m}$ for the nodes $y_1, \dots, y_m$. Since x is not a leaf and $\mathfrak{T}$ is a $\mathcal{P}$-CT, we have $t_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \dots, \frac{p_{y_m}}{p_x} : t_{y_m}\}$. We have the following three cases:

(A) If it is a step with $\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}}$ using an ADP that is different from α in $\mathfrak{T}$, then we perform a rewrite step with the same ADP, the same redex, and the same substitution in $\mathfrak{T}'$. Then, it is easy to see that (36) for the resulting terms hold.

(B) If it is a step with $\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}}$ using α at a position $\pi \notin \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x)$ in $\mathfrak{T}$, then we perform a rewrite step with the new ADP $\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1, \dots, \delta_d)}(r_1), \dots, p_k : \sharp_{\mathrm{capt}_k(\delta_1, \dots, \delta_d)}(r_k)\}^m$, the same redex, same substitution, and same position in $\mathfrak{T}'$. Since the new rule has the same underlying terms as α, it is easy to see that (36) holds for the resulting terms. Note that the rule that we use contains fewer annotations

[8]Note that we only have "≤" and not "=" since we may add rules to $\mathcal{S}$ that would not count before, but finding a tree of equal or greater expected derivation length suffices.

than the original rule, but since $\pi \notin \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x)$, we remove all annotations from the rule during the application of the rewrite step anyway.

(C) If it is a step with $\overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}}$ using α at a position $\pi \in \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x)$ in $\mathfrak{T}$, then we look at the specific successors to find a substitution δ such that $\ell\delta \to \{p_1 : r_1\delta, \ldots, p_k : r_k\delta\}^m \in N$ or we detect that we can use the ADP $\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, p_k : \sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}^m$ and perform a rewrite step with this new ADP, at the same position in $\mathfrak{T}'$.

It remains to consider Case (C) in detail. Here, we have $t_x \overset{\mathrm{i}}{\hookrightarrow}_{\mathcal{P}} \{\frac{p_{y_1}}{p_x} : t_{y_1}, \ldots, \frac{p_{y_k}}{p_x} : t_{y_k}\}$ using the ADP α, the position $\pi \in \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x)$, and a substitution σ such that $\flat(t_x|_\pi) = \ell\sigma \in \mathsf{ANF}_{\mathcal{P}}$.

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 strictly above π before reaching such a node v. Then, we can use $\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, p_k : \sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}^m$ 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 strictly above π before. Let $v_1, \ldots, v_n$ 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 strictly above π. Furthermore, let $t_1, \ldots, t_n$ be the used redexes and $\rho_1, \ldots, \rho_n$ be the used substitutions.

- **(C1) If** none of the redexes $t_1, \ldots, t_n$ is captured by t, then we use $\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, p_k : \sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}^m$ with the position $\pi \in \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x) \cap \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(\flat(t_x)) \subseteq_{(IH)} \mathrm{Pos}_{\mathcal{D}^\sharp}(t'_x)$ and the substitution σ. Again, (36) 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 $\ell\delta_e \to \{p_1 : r_1\delta_e, \ldots, p_k : r_k\delta_e\}^m$ with the position $\pi \in \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x) \cap \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(\flat(t_x)) \subseteq_{(IH)} \mathrm{Pos}_{\mathcal{D}^\sharp}(t'_x)$ and the substitution γ such that $\flat(t_x|_\pi) = \ell\delta_e\gamma = \ell\sigma \in \mathsf{ANF}_{\mathcal{P}}$. Again, (36) 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 $\ell\delta_e \to \{p_1 : r_1\delta_e, \ldots, p_k : r_k\delta_e\}^m$ with the position $\pi \in \mathrm{Pos}_{\mathcal{D}^\sharp}(t_x) \cap \mathrm{Pos}_{\mathcal{D}\wedge\neg\mathsf{NF}_{\mathcal{R}}}(\flat(t_x)) \subseteq_{(IH)} \mathrm{Pos}_{\mathcal{D}^\sharp}(t'_x)$ and the substitution $\kappa_1\kappa_2$ such that $\flat(t_x|_\pi) = \ell\delta_e\kappa_1\kappa_2 = \ell\sigma \in \mathsf{ANF}_{\mathcal{P}}$. Again, (36) is satisfied for our resulting terms.

Now we show condition (8). Let $\mathfrak{P}$ be a well-formed proof tree with $L_{\mathcal{A}}(v) = \langle \mathcal{P}, \mathcal{S} \rangle$, let $v_1, \ldots, v_k = v$ be the path from the root node v_1 to v, and let $c_{(v_1,\ldots,v_k)} = L_C(v_1) \oplus \ldots \oplus L_C(v_{k-1})$. We have

$$\iota_{\langle \mathcal{P}' \cup N, (\mathcal{P}' \cup N) \setminus \widetilde{\mathcal{S}} \rangle} \sqsubseteq c_{(v_1,\ldots,v_k)},$$

because every $(\mathcal{P}' \cup N)$-CT $\mathfrak{T}$ gives rise to a $\mathcal{P}$-CT $\mathfrak{T}'$ with at least the same $(\mathbf{at})$- and $(\mathbf{af})$-steps. We can replace each usage of an ADP $\ell\delta_e \to \{p_1 : r_1\delta_e, \ldots, p_k : r_k\delta_e\}^m$ with the more general ADP α, and each ADP $\ell \to \{p_1 : \sharp_{\mathrm{capt}_1(\delta_1,\ldots,\delta_d)}(r_1), \ldots, p_k : \sharp_{\mathrm{capt}_k(\delta_1,\ldots,\delta_d)}(r_k)\}^m$ can be replaced by α as well, leading to more annotations than before.

If an $(\mathbf{at})$- or $(\mathbf{af})$-step is performed with an ADP $\beta \in (\mathcal{P}' \cup N) \setminus \widetilde{\mathcal{S}}$ in $\mathfrak{T}$, we have the following cases:

- If $\beta \in \mathcal{P}' \setminus \widetilde{\mathcal{S}}$, and $\alpha \in \mathcal{S}$, then

$$\beta \in \mathcal{P}' \setminus \widetilde{\mathcal{S}} = \mathcal{P}' \setminus ((\mathcal{S} \setminus \{\alpha\}) \cup N) = \mathcal{P}' \setminus (\mathcal{S} \cup N) \subseteq \mathcal{P} \setminus \mathcal{S}$$

- If $\beta \in \mathcal{P}' \setminus \widetilde{\mathcal{S}}$, and $\alpha \notin \mathcal{S}$, then

$$\beta \in \mathcal{P}' \setminus \widetilde{\mathcal{S}} = \mathcal{P}' \setminus \mathcal{S} \subseteq \mathcal{P} \setminus \mathcal{S}$$

- If $\beta \in N \setminus \widetilde{\mathcal{S}}$, then $\alpha \notin \mathcal{S}$, and we use α in $\mathfrak{T}'$.

In every case, if the step counts towards the expected derivation length w.r.t. $(\mathcal{P}' \cup N) \setminus \widetilde{\mathcal{S}}$ in $\mathfrak{T}$, it will count for the expected derivation length w.r.t. $\mathcal{P} \setminus \mathcal{S}$ in $\mathfrak{T}'$ as well. Therefore, we result in $\mathrm{edl}_{\langle \mathcal{P}' \cup N, (\mathcal{P}' \cup N) \setminus \widetilde{\mathcal{S}} \rangle}(\mathfrak{T}) \leq \mathrm{edl}_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle}(\mathfrak{T})$. As this holds for every $(\mathcal{P}' \cup N)$-CT, we get $\iota_{\langle \mathcal{P}' \cup N, (\mathcal{P}' \cup N) \setminus \widetilde{\mathcal{S}} \rangle} \sqsubseteq \iota_{\langle \mathcal{P}, \mathcal{P} \setminus \mathcal{S} \rangle} \sqsubseteq c_{(v_1,\ldots,v_k)}$, by well-formedness of $\mathfrak{P}$, and (8) holds. ■