

Newtonian Program Analysis of Probabilistic Programs

DI WANG, Peking University, China

THOMAS REPS, University of Wisconsin, USA

Due to their *quantitative* nature, probabilistic programs pose non-trivial challenges for designing compositional and efficient program analyses. Many analyses for probabilistic programs rely on *iterative* approximation. This article presents an interprocedural dataflow-analysis framework, called *NPA-PMA*, for designing and implementing (partially) *non-iterative* program analyses of probabilistic programs with unstructured-flow control-flow, nondeterminism, and general recursion. NPA-PMA is based on Newtonian Program Analysis (NPA), a generalization of Newton's method to solve equation systems over semirings. The key challenge for developing NPA-PMA is to handle multiple kinds of *confluences* in both the algebraic structures that specify analyses and the equation systems that encode control flow: semirings support a single confluence operation, whereas NPA-PMA involves three confluence operations (conditional, probabilistic, and nondeterministic).

Our work introduces ω -continuous pre-Markov algebras (ω PMAs) to factor out common parts of different analyses; adopts *regular infinite-tree expressions* to encode program-execution paths in control-flow hypergraphs; and presents a *linearization* method that makes Newton's method applicable to the setting of regular-infinite-tree equations over ω PMAs. NPA-PMA allows analyses to supply a non-iterative strategy to solve linearized equations. Our experimental evaluation demonstrates that (i) NPA-PMA holds considerable promise for outperforming Kleene iteration, and (ii) provides great generality for designing program analyses.

1 INTRODUCTION

Probabilistic programs, which augment deterministic programs with *data randomness* (e.g., sampling) and *control-flow randomness* (e.g., probabilistic branching), provide a rich framework for implementing and analyzing randomized algorithms and probabilistic systems [McIver and Morgan 2005]. To establish properties of probabilistic programs, people have developed many *program analyses*, as well as frameworks for encoding existing analyses and developing new analyses. Two prominent instances are Probabilistic Abstract Interpretation (PAI) [Cousot and Monerau 2012] and the Pre-Markov Algebra Framework (PMAF) [Wang et al. 2018]. Both frameworks derive from *abstract interpretation* [Cousot and Cousot 1977]: one proves the soundness of an analysis by establishing a *soundness relation* between concrete and abstract semantics, and implements the analysis via *iterative approximation* of fixed-points.

However, the *quantitative* nature of probabilistic programs usually causes iteration-based program analysis to be inefficient. For example, consider the probabilistic loop $P \triangleq \text{"while prob}(\frac{3}{4}) \text{ do } x := x + 1 \text{ od}"$, which increments program variable x by one with probability $\frac{3}{4}$ in each iteration, and otherwise exits the loop. Suppose that a program analysis aims to reason about the expected difference between the final and initial value of x . An iteration-based analysis would try to obtain the expected difference, denoted by r , by approximating the fixed-point of $r = f(r) \triangleq \frac{3}{4} \cdot (1+r) + \frac{1}{4} \cdot 0$, where f intuitively captures the fact that $P \equiv \text{"if prob}(\frac{3}{4}) \text{ then } x := x + 1; P \text{ else skip fi}"$. For example, *Kleene iteration* yields an approximation sequence $\{f^i(0)\}_{i \geq 0}$, which converges to the least fixed-point of f but will not converge in any finite number of steps. On the other hand, r is the solution of the linear equation $r = \frac{3}{4} \cdot (1+r) + \frac{1}{4} \cdot 0$, which directly yields $r = 3$. The following equation demonstrates the difference between the iterative and non-iterative computation:

$$\text{iterative } \left(\frac{3}{4}\right)^1 + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^3 + \left(\frac{3}{4}\right)^4 + \cdots = 3 \quad 3 = \frac{3}{4} \times \frac{1}{1-\frac{3}{4}} \quad \text{non-iterative} \quad (\star)$$

Authors' addresses: Di Wang, Peking University, China; Thomas Reps, University of Wisconsin, USA.

2018. 2475-1421/2018/1-ART1 \$15.00

<https://doi.org/>

This article presents a framework, which we call *NPA-PMA* (for Newtonian Program Analysis with Pre-Markov Algebras), for designing and implementing compositional and efficient *interprocedural* program analyses of probabilistic programs. In particular, NPA-PMA enables analyses with computations of the form shown on the right side of (\star) , and achieves three major desiderata:

- *Compositionality*. NPA-PMA analyzes a program by analyzing its constituents and then combining the analysis results.
- *Efficiency*. NPA-PMA allows an instance program analysis to supply a more efficient strategy (compared to Kleene iteration) for abstracting repetitive behavior. NPA-PMA then provides a unified and sound approach for efficient *intraprocedural* analysis (of loops) and *interprocedural* analysis (of recursion) via the analysis-supplied strategy.
- *Generality*. NPA-PMA covers a multitude of analyses of probabilistic programs. For our experimental evaluation (§5), we instantiated NPA-PMA for finding (i) probability distributions of program states, (ii) upper bounds on moments of random variables, and (iii) linear and (iv) non-linear expectation invariants. We reformulated two existing abstract domains for (i) and (ii), and designed two new abstract domains for (iii) and (iv).

Our approach is motivated by *Newtonian Program Analysis* (NPA) [Esparza et al. 2008b, 2010]. Many interprocedural dataflow analyses can be reduced to finding the least fixed-point of an equation system $\vec{X} = \vec{f}(\vec{X})$ over a semiring [Bouajjani et al. 2003; Reps et al. 2007, 2003]. Esparza et al. considered interprocedural analysis of *loop-free* programs and developed an iterative approximation method with the following scheme, where $\text{LINEARCORRECTION}(\vec{f}, \vec{v}^{(i)})$ is a correction term, which is the solution to a “linearized” equation system of \vec{f} at $\vec{v}^{(i)}$ (we will discuss linearization in §2.1):

$$\vec{v}^{(0)} = \vec{f}(\vec{1}), \quad \vec{v}^{(i+1)} = \vec{f}(\vec{v}^{(i)}) \sqcup \text{LINEARCORRECTION}(\vec{f}, \vec{v}^{(i)}). \quad (1)$$

NPA is *partially iterative* and *partially non-iterative*: Eqn. (1) defines the iteration that produces the successive values of the Newton sequence, whereas the solution to the linearized equation system of \vec{f} at each value $\vec{v}^{(i)}$ can be computed non-iteratively.

NPA-PMA generalizes variants of Newton’s method used for solving recursive Markov chains [Etessami and Yannakakis 2005] and probabilistic pushdown automata [Esparza et al. 2004]. In particular, those models did not support *nondeterminism*, i.e., the transition probability can be nondeterministic. Nondeterminism arises naturally when the analyzed program has an infinite state space, because an analysis usually relies on an *abstraction* of the state space, and consequently can only treat some conditional choices as nondeterministic ones, due to loss of information.

We implemented a prototype of NPA-PMA, with which our preliminary evaluation confirmed that the quantitative nature of probabilistic programs unleashes the potential of Newton’s method. As mentioned earlier, we instantiated NPA-PMA with two existing and two new abstract domains. For the two analyses based on existing domains, we evaluated them on synthetic multi-procedure benchmark programs. In addition, for one of the analyses we implemented two different analysis-supplied strategies, showing the generality of NPA-PMA. In general, NPA-PMA outperforms Kleene iteration. For example, for one domain, NPA-PMA greatly reduces the number of iterations (3,070.42 \rightarrow 9.15), contributing to an overall runtime speedup of 4.08x. For the two analyses based on new domains, we evaluated them on benchmark programs collected from the literature. NPA-PMA can derive non-trivial expectation invariants compared to state-of-the-art techniques, which shows that NPA-PMA is flexible and holds considerable promise for supporting program analyses of probabilistic programs.

Fig. 1 provides a roadmap for the presentation of our NPA-PMA framework for interprocedural analysis of probabilistic programs with nondeterminism and unstructured control-flow (lower-right quadrant). The starting point is the NPA framework for interprocedural analysis of loop-free programs (upper-left quadrant). The horizontal arrow stands for “adding probability;” the vertical

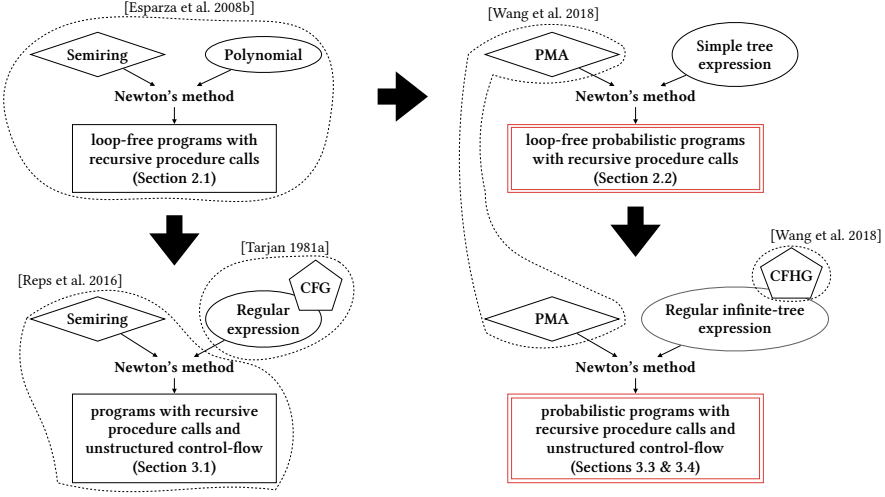


Fig. 1. A roadmap for the Overview sections (§2 and §3). We use dashed curves for concepts from prior work and red double borders for contributions of this work. For each variant of Newton’s method for solving $\vec{X} = \vec{f}(\vec{X})$, we use \diamond for the algebraic structure of the elements, \circ for the form of the right-hand sides $\vec{f}(\vec{X})$, and \square for the supported kind of programs. We put a \circ (for a program representation) on top of a \circ (for a form of right-hand sides) to denote that there is a sound transformation from \circ to \circ .

arrow stands for “adding unstructured control-flow.” In our development, we first move horizontally and then move vertically. Our goal is to apply the recipe of *algebraic program analysis* (lower-left quadrant) [Farzan and Kincaid 2013; Kincaid et al. 2021; Tarjan 1981a], in which program actions are represented by expressions over a suitable algebra, and then fit it into the NPA framework.

As observed by Wang et al. [2018, 2019b], because of multiple kinds of confluence operations, it is more suitable to use *control-flow hyper-graphs* (CFHGs), rather than ordinary control-flow graphs (CFGs), to represent probabilistic programs. In a CFHG, conditional-, probabilistic-, and nondeterministic-branching are represented by different kinds of *hyper-edges*, each of which consists of one source node and two destination nodes. In an ordinary CFG, a node can also have multiple successors, but the operation applied at confluence nodes is usually the same operation (e.g., join). Thus, the challenges we face are two-fold: (i) to find a suitable notion of “hyper-path expressions” that encode the control-flow hyper-path through a CFHG, and (ii) to devise an appropriate method—required by Newton’s method—of linearizing such hyper-path expressions.

Contributions. The article’s contributions include the following:

- We develop NPA-PMA, an algebraic framework for interprocedural program analyses of probabilistic programs. NPA-PMA provides a novel approach to analyzing loops and recursion in probabilistic programs with nondeterminism and unstructured control-flow (§2 and §3). Tab. 1 gives a comparison between NPA-PMA and prior analysis frameworks for probabilistic programs in terms of what program features and analysis features are supported.
- We adopt *regular infinite-tree expressions* to encode the hyper-path through a control-flow hyper-graph and devise a formulation of their differential (required by Newton’s method) in a new family of algebras, ω -continuous pre-Markov algebras (ω PMAs) (§4). We also develop a theory of regular hyper-paths that justifies the use of regular infinite-tree expressions in our framework.
- We created a prototype implementation of NPA-PMA and instantiated it with four different abstract domains for analyzing probabilistic programs, two of which are new (§5). Our preliminary

Table 1. Comparison in terms of supported program and analysis features. “ucf.” stands for “unstructured control-flow;” “nondet.” stands for “nondeterminism;” “recur.” stands for “recursion;” and “iter.” and “non-iter.” stand for “iterative” and “non-iterative,” respectively.

	Program features			Analysis features		
	ucf.	nondet.	recur.	loops	linear recur.	non-linear recur.
PAI [Cousot and Monerau 2012]	✓	✓		iter.	N/A	N/A
PMAF [Wang et al. 2018]	✓	✓	✓	iter.	iter.	iter.
NPA [Esparza et al. 2008b]			✓	non-iter.	non-iter. ¹	iter.
NPA-PMA (this work)	✓	✓	✓	non-iter.	non-iter. ²	iter.

evaluation shows that NPA-PMA outperforms Kleene iteration on the two existing abstract domains. We developed two new abstract domains for expectation-invariant analysis to show the generality of NPA-PMA for designing program analyses.

§6 discusses related work. §7 concludes.

2 OVERVIEW, PART I: INTERPROCEDURAL ANALYSIS OF LOOP-FREE PROGRAMS

This section explains the main ideas of the *interprocedural* part of NPA-PMA, i.e., from the upper-left quadrant to the upper-right quadrant in Fig. 1. §2.1 reviews Newtonian Program Analysis (NPA) for interprocedural dataflow analysis of loop-free programs. §2.2 discusses the limitations of applying NPA to analyze probabilistic programs with *nondeterminism* and motivates our extension to NPA.

2.1 Prior Work: Newtonian Program Analysis

A *semiring* is an algebraic structure $\mathcal{S} = \langle D, \oplus, \otimes, \underline{0}, \underline{1} \rangle$, where $\langle D, \otimes, \underline{1} \rangle$ is a monoid (i.e., \otimes is an associative binary operation with $\underline{1}$ as its identity element); $\langle D, \oplus, \underline{0} \rangle$ is a monoid in which \oplus (*combine*) is commutative; and \otimes (*extend*) distributes over \oplus . An ω -*continuous semiring* is a semiring in which (i) the relation $\sqsubseteq \triangleq \{(a, b) \in D \times D \mid \exists d: a \oplus d = b\}$ is a partial order, (ii) every ω -chain $a_0 \sqsubseteq a_1 \sqsubseteq \dots \sqsubseteq a_n \sqsubseteq \dots$ has a supremum with respect to \sqsubseteq , denoted by $\bigsqcup_{i \in \mathbb{N}} a_i$, and (iii) both \oplus and \otimes are ω -continuous in both arguments. A mapping $f: D \rightarrow D$ is ω -*continuous*, if f is monotone and for any ω -chain $(a_i)_{i \in \mathbb{N}}$ it holds that $f(\bigsqcup_{i \in \mathbb{N}} a_i) = \bigsqcup_{i \in \mathbb{N}} f(a_i)$.

Example 2.1 (The real semiring). A canonical example of ω -continuous semirings is the *real semiring* whose elements are nonnegative real numbers together with infinity: $\langle \mathbb{R}_{\geq 0} \cup \{\infty\}, +, \cdot, 0, 1 \rangle$.

NPA generalizes Newton’s method to solve interprocedural equation systems over a semiring. We demonstrate NPA with the finite-state probabilistic program shown in Fig. 2(a). Its termination probability can be obtained as the least solution to an equation system that encodes the program with respect to the real semiring. We encode procedure X as an equation $X = f(X)$ where $f(X) \triangleq \frac{1}{2} \oplus (\frac{1}{2} \otimes X \otimes X)$, where underlining denotes semiring constants. The equation can be viewed as a representation of procedure X ’s interprocedural control-flow graph (CFG); see Fig. 2(b).

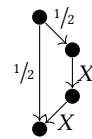
One can apply *Kleene iteration* to find the least fixed-point of $X = f(X)$ via the sequence $\kappa^{(0)} = \underline{0}$ and $\kappa^{(i+1)} = f(\kappa^{(i)})$ for $i \in \mathbb{N}$. With NPA, we

```

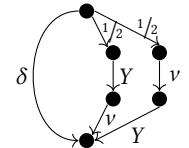
proc X() begin
  if prob( $\frac{1}{2}$ ) then
    skip
  else
    X(); X()
  fi
end

```

(a) An example program



(b) $X = f(X)$



(c) $Y = \delta \oplus Df|_v(Y)$

Fig. 2. The running example of NPA.

¹NPA creates subproblems that are equivalent to analyzing linearly recursive programs, but does not provide a non-iterative method for solving linear recursion. Reps et al. [2016] devised a non-iterative method for solving linear recursion (NPA-TP).

²We allow an instance analysis to supply a non-iterative linear-recursion-solving strategy. See §2.2 and §4.3.

solve the following sequence of subproblems for v :³

$$v^{(0)} = f(\underline{0}), \quad v^{(i+1)} = v^{(i)} \oplus \Delta^{(i)}, \quad (2)$$

where $\Delta^{(i)}$ is the least solution of

$$Y = \delta^{(i)} \oplus \mathcal{D}f|_{v^{(i)}}(Y), \quad (3)$$

$\delta^{(i)}$ is any element that satisfies $f(v^{(i)}) = v^{(i)} \oplus \delta^{(i)}$, and $\mathcal{D}f|_{v^{(i)}}(Y)$ is the *differential* of f at $v^{(i)}$.⁴

Definition 2.2 ([Esparza et al. 2008b, 2010]). Let $f(X)$ be a function expressed as an expression in an ω -continuous semiring. The *differential* of $f(X)$ at v , denoted by $\mathcal{D}f|_v(Y)$, is defined as follows:

$$\mathcal{D}f|_v(Y) \triangleq \begin{cases} \underline{0} & \text{if } f = a \in D \\ Y & \text{if } f = X \\ \mathcal{D}g|_v(Y) \oplus \mathcal{D}h|_v(Y) & \text{if } f = g \oplus h \\ (\mathcal{D}g|_v(Y) \otimes h(v)) \oplus (g(v) \otimes \mathcal{D}h|_v(Y)) & \text{if } f = g \otimes h \end{cases}$$

As discussed in §1, Eqns. (2) and (3) resemble Kleene iteration, except that in each iteration, $f(v^{(i)})$ is adjusted by a *linear* correction term $\mathcal{D}f|_{v^{(i)}}(Y)$. A consequence of Defn. 2.2 is that the right-hand side of Eqn. (3) is of the form $m_1 \oplus \dots \oplus m_\ell$, where $\ell \geq 1$, and m_1, \dots, m_ℓ are each *linear* expressions of the form $a \otimes Y \otimes b$ for $a, b \in D$.

Remark 2.3. Note how the case for “ $g \otimes h$ ” in Defn. 2.2 resembles the (univariate) Leibniz product rule from differential calculus: $d(g * h) = dg * h + g * dh$.

Remark 2.4. The key property of the differentials is that for any elements v and Δ , it holds that

$$f(v) \oplus \mathcal{D}f|_v(\Delta) \sqsubseteq f(v \oplus \Delta), \quad (4)$$

i.e., the linearization of $f(v \oplus \Delta)$ at v is always an **under-approximation**, thus the iteration defined in Eqn. (2) will not “overshoot” the least fixed-point. We include a formal discussion in §4.4.

Recall the interprocedural equation $X = f(X)$ of the example probabilistic program shown in Fig. 2(a) where $f(X) \triangleq \frac{1}{2} \oplus (\frac{1}{2} \otimes X \otimes X)$. The differential of f at v is $\mathcal{D}(\frac{1}{2} \oplus (\frac{1}{2} \otimes X \otimes X))|_v(Y) = \underline{0} \oplus ((\underline{0} \otimes v \otimes v) \oplus (\frac{1}{2} \otimes Y \otimes v) \oplus (\frac{1}{2} \otimes v \otimes Y)) = (\frac{1}{2} \otimes Y \otimes v) \oplus (\frac{1}{2} \otimes v \otimes Y)$. Let δ be an element that satisfies $f(v) = v \oplus \delta$. From Eqn. (3), we then obtain the following linearized equation, which is also illustrated graphically in Fig. 2(c):

$$Y = \delta \oplus (\frac{1}{2} \otimes Y \otimes v) \oplus (\frac{1}{2} \otimes v \otimes Y). \quad (5)$$

Recall that we interpret the algebraic equation with respect to the real semiring to compute the termination probability. Thus, on the $(i+1)^{\text{st}}$ Newton round, we can compute δ as $\delta \triangleq f(v^{(i)}) - v^{(i)}$, where $v^{(i)}$ is the value for v obtained in the i^{th} round. Consequently, on the $(i+1)^{\text{st}}$ Newton round, Eqn. (5) becomes the linear numeric equation $Y = ((\frac{1}{2} + \frac{1}{2} \cdot v^{(i)} \cdot v^{(i)}) - v^{(i)}) + \frac{1}{2} \cdot Y \cdot v^{(i)} + \frac{1}{2} \cdot v^{(i)} \cdot Y$, which means that $\Delta^{(i)} \triangleq (1 - v^{(i)})/2$, and thus the iterative step in Eqn. (2) is $v^{(i+1)} \leftarrow v^{(i)} \oplus \Delta^{(i)} = (1 + v^{(i)})/2$. Because $v^{(0)} \triangleq f(\underline{0}) = \frac{1}{2}$, we have $v^{(i)} = 1 - 2^{-(i+1)}$ for any $i \in \mathbb{N}$, i.e., the Newton sequence converges to 1 (so the example program shown in Fig. 2(a) terminates with probability one), and gains one bit of precision per Newton iteration. In contrast, Kleene iteration would yield a sequence such that $\kappa^{(i)} \leq 1 - \frac{1}{i+1}$ for $i \in \mathbb{N}$, and thus one needs to perform about 2^i Kleene rounds to obtain i bits of precision—i.e., Kleene iteration converges exponentially slower.

³Esparza et al. described a general technique for solving the multivariate case, i.e., finding the least fixed-point of $\vec{X} = \vec{f}(\vec{X})$. In §2.1, for brevity, we only describe their technique for solving the univariate case.

⁴When Eqn. (2) is compared with the high-level intuition given as Eqn. (1), the application of f to $v^{(i)}$ on the right-hand side of Eqn. (2) is a bit hidden. $\Delta^{(i)}$ is the least solution of Eqn. (3), and therefore $\Delta^{(i)} \sqsupseteq \delta^{(i)} \oplus \mathcal{D}f|_{v^{(i)}}(\underline{0})$. Because $f(v^{(i)}) = v^{(i)} \oplus \delta^{(i)}$, the right-hand side of Eqn. (2) can be seen as $v^{(i)} \oplus \Delta^{(i)} \sqsupseteq v^{(i)} \oplus \delta^{(i)} \oplus \mathcal{D}f|_{v^{(i)}}(\underline{0}) = f(v^{(i)}) \oplus \mathcal{D}f|_{v^{(i)}}(\underline{0})$.

2.2 This Work: Algebras and Expressions with Multiple Confluence Operations

A major limitation of NPA is that it cannot handle probabilistic programs with *nondeterminism*. As discussed in §1, nondeterminism arises naturally from abstractions of the state space of a program [McIver and Morgan 2005]. Consider a termination-probability analysis that computes the *lower bound* on the termination probability. The algebraic foundation of NPA—semirings—is not sufficient for expressing this analysis, because a semiring contains only *one combine* operation \oplus , whereas the analysis requires *two* different combine operations:

- For probabilistic-choice (e.g., “**if** $\text{prob}(\frac{1}{2})$ **then** ... **else** ... **fi**”), we need a combine operation that performs addition. We have shown the use of addition in §2.1 for probabilistic-choice.
- For nondeterministic-choice, we need a combine operation that computes the minimum of two values.

We refer to such operations generically as *confluence operations*.

In this article, we devise ω -continuous pre-Markov algebras (ω PMAs), which can be seen as a refinement of ω -continuous semirings and previously proposed pre-Markov algebras (PMAs) [Wang et al. 2018]. For the lower-bound-on-termination-probability analysis, we extend the real semiring to be an ω PMA $\mathcal{R} = \langle \mathbb{R}_{\geq 0} \cup \{\infty\}, \oplus, \otimes, \diamond, \ominus, \uplus, \underline{0}, \underline{1} \rangle$, where $\langle \mathbb{R}_{\geq 0} \cup \{\infty\}, \oplus, \otimes, \underline{0}, \underline{1} \rangle$ is the real semiring, and \diamond (parameterized by a logical condition φ), \ominus (parameterized by $p \in [0, 1]$), and \uplus are three binary confluence operations that correspond to conditional-choice, probabilistic-choice, and nondeterministic-choice, respectively. The confluence operations are defined as $r_1 \ominus_p r_2 \triangleq p \cdot r_1 + (1-p) \cdot r_2$ for any $p \in [0, 1]$ and $r_1 \uplus r_2 \triangleq \min(r_1, r_2)$. Because the example does not contain conditional branching, we do not need the \diamond operation. In general, for any r_1, r_2 that satisfy $r_1 \geq r_2$, we let $r_1 \ominus r_2$ denote any element d that satisfies $r_2 \oplus d = r_1$. Here, for nonnegative real numbers, we can define $r_1 \ominus r_2 \triangleq r_1 - r_2$.

We use the example program shown in Fig. 3(a) to demonstrate our design of the form for right-hand sides of equation systems. We use the \star symbol to denote nondeterministic-choice. In this work, we encode a loop-free probabilistic program as a *simple tree expression*, in which every node corresponds to a program command and every root-to-leaf path corresponds to a program path. Confluences are then encoded as internal nodes with multiple (typically two) child nodes in the tree. Fig. 3(b) presents the abstract syntax tree (AST) of the following simple tree expression that corresponds to the example program shown in Fig. 3(a), where $\underline{1}$ denotes **skip**, $\text{call}[X]$ is a procedure-call command, $\text{prob}[p]$ is a probabilistic-choice command that takes the left branch with probability p , and ndet is a nondeterministic-choice command:

$$X = f(X) \triangleq \text{prob}[\frac{1}{3}] \left(\underline{1}, \text{call}[X] \left(\text{ndet}(\text{call}[X](\underline{1}), \underline{1}) \right) \right). \quad (6)$$

We use a syntax-directed method to interpret simple tree expressions over \mathcal{R} , where we parameterize the interpretation by a value v for interpreting procedure calls to X :

$$\begin{aligned} \mathcal{R}[\underline{r}](v) &\triangleq r, & \mathcal{R}[\text{prob}[p](E_1, E_2)](v) &\triangleq \mathcal{R}[E_1](v) \oplus_p \mathcal{R}[E_2](v), \\ \mathcal{R}[\text{call}[X](E)](v) &\triangleq v \otimes \mathcal{R}[E](v), & \mathcal{R}[\text{ndet}(E_1, E_2)](v) &\triangleq \mathcal{R}[E_1](v) \uplus \mathcal{R}[E_2](v). \end{aligned}$$

The solution of an equation $X = f(X)$ is a value v that satisfies $v = \mathcal{R}[f(X)](v)$.

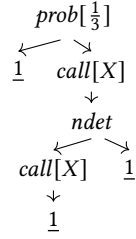
We now describe how our extension of NPA works by showing how NPA-PMA solves Eqn. (6) over \mathcal{R} . Recall that NPA requires a notion of *differential* (see Defn. 2.2 for the semiring case). We want to define the differential of $f(X)$ (expressed as a simple tree expression) at a value v , still

```

proc X() begin
  if prob( $\frac{1}{3}$ ) then skip
  else
    X();
  if  $\star$  then X() fi
fi
end

```

(a) An example program



(b) $f(X)$ as a tree

Fig. 3. A nondeterministic recursive program.

denoted by $\mathcal{D}f|_v(Y)$. Because \mathcal{R} is an extension of the real semiring, it is natural for us to define $\mathcal{D}f|_v(Y)$ as a conservative extension of Defn. 2.2:

- If $f = \underline{r}$: Because $\mathcal{R}[\underline{r}](v) = \underline{r}$ and Defn. 2.2 defines the differential of any constant to be zero, we derive $\mathcal{D}f|_v(Y) \triangleq \underline{0}$.
- If $f = \text{call}[X](g)$: Because $\mathcal{R}[\text{call}[X](g)](v) = v \otimes \mathcal{R}[g](v)$ and the differential of \otimes resembles the Leibniz product rule, we can treat $\text{call}[X](g)$ as $X \otimes g$ and its differential at v is $(Y \otimes g) \oplus (v \otimes \mathcal{D}g|_v(Y))$. The monomial $Y \otimes g$ can then be transformed back to the tree form $\text{call}[Y](g)$. For the monomial $v \otimes \mathcal{D}g|_v(Y)$, where v is a specific known value, we introduce a sequencing command $\text{seq}[\underline{r}]$ whose interpretation is $\mathcal{R}[\text{seq}[\underline{r}](E)](v) \triangleq \underline{r} \otimes \mathcal{R}[E](v)$. Thus, we derive $\mathcal{D}f|_v(Y) \triangleq \text{call}[Y](g(v)) \oplus \text{seq}[v](\mathcal{D}g|_v(Y))$.
- If $f = \text{ndet}(g, h)$: Recall that we want $\mathcal{D}f|_v(Y)$ to satisfy the under-approximation property Eqn. (4):

$$\text{ndet}(g(v), h(v)) \oplus \mathcal{D}f|_v(Y) \sqsubseteq \text{ndet}(g(v \oplus Y), h(v \oplus Y)).$$

Applying Eqn. (4) to the sub-tree-expressions g and h , we have

$$g(v) \oplus \mathcal{D}g|_v(Y) \sqsubseteq g(v \oplus Y), \quad h(v) \oplus \mathcal{D}h|_v(Y) \sqsubseteq h(v \oplus Y).$$

If ndet is a *monotone* operation, we have

$$\text{ndet}(g(v) \oplus \mathcal{D}g|_v(Y), h(v) \oplus \mathcal{D}h|_v(Y)) \sqsubseteq \text{ndet}(g(v \oplus Y), h(v \oplus Y)).$$

Thus, we can safely define $\mathcal{D}f|_v(Y) \triangleq \text{ndet}(g(v) \oplus \mathcal{D}g|_v(Y), h(v) \oplus \mathcal{D}h|_v(Y)) \ominus \text{ndet}(g(v), h(v))$.

- If $f = \text{prob}[p](g, h)$: We could repeat the derivation for ndet by assuming $\text{prob}[p]$ is a monotone operation; however, we take a different approach because of an observation that $\text{prob}[p]$ usually satisfies more algebraic properties than ndet , i.e., for any elements a, b, c, d ,

$$\text{prob}[p](a \oplus b, c \oplus d) = \text{prob}[p](a, c) \oplus \text{prob}[p](b, d).$$

For the ω PMA \mathcal{R} , the equation holds trivially because $p \cdot (a + b) + (1 - p) \cdot (c + d) = p \cdot a + (1 - p) \cdot c + p \cdot b + (1 - p) \cdot d$. We then have

$$\begin{aligned} \text{prob}[p](g(v \oplus Y), h(v \oplus Y)) &\supseteq \text{prob}[p](g(v) \oplus \mathcal{D}g|_v(Y), h(v) \oplus \mathcal{D}h|_v(Y)) \\ &= \text{prob}[p](g(v), h(v)) \oplus \text{prob}[p](\mathcal{D}g|_v(Y), \mathcal{D}h|_v(Y)). \end{aligned}$$

Thus, we can safely define $\mathcal{D}f|_v(Y) \triangleq \text{prob}[p](\mathcal{D}g|_v(Y), \mathcal{D}h|_v(Y))$.

To summarize, we have derived the following rules:

$\mathcal{D}f _v(Y) \triangleq$	$\underline{0}$	if $f = \underline{r}$
	$\text{call}[Y](g(v)) \oplus \text{seq}[v](\mathcal{D}g _v(Y))$	if $f = \text{call}[X](g)$
	$\text{ndet}(g(v) \oplus \mathcal{D}g _v(Y), h(v) \oplus \mathcal{D}h _v(Y)) \ominus \text{ndet}(g(v), h(v))$	if $f = \text{ndet}(g, h)$
	$\text{prob}[p](\mathcal{D}g _v(Y), \mathcal{D}h _v(Y))$	if $f = \text{prob}[p](g, h)$

We then apply Newton's method to find the least fixed-point of $X = f(X)$ by solving a sequence of subproblems for v : $v^{(0)} = f(\underline{0})$ and $v^{(i+1)} = v^{(i)} \oplus \Delta^{(i)}$ for $i \geq 0$, where $\Delta^{(i)}$ is the least solution of $Y = \delta^{(i)} \oplus \mathcal{D}f|_{v^{(i)}}(Y)$, and $\delta^{(i)} \triangleq f(v^{(i)}) \ominus v^{(i)}$. Fig. 4 graphically shows the subproblem in terms of Y for the example program shown in Fig. 3(a) when X has some value v . A key feature of the AST for the differential is that every root-to-leaf path contains *at most* one $\text{call}[\cdot]$ node; thus, the subproblem in terms of Y is again a *linearized* equation. We now unfold the definition of all algebraic operations and obtain the following linear numeric equation:

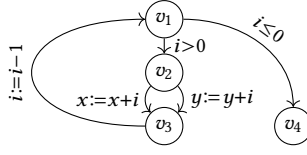
$$Y = \left(\frac{1}{3} + \frac{2}{3} \cdot v \cdot \min(v, 1) - v\right) + \frac{2}{3} \cdot (Y \cdot \min(v, 1) + v \cdot (\min(v + Y, 1) - \min(v, 1))).$$

Our NPA-PMA framework then uses an analysis-supplied strategy to solve linear equations like the one shown above. Here, because the equation is simple enough, we can solve it analytically. Observing probabilities are real numbers in the interval $[0, 1]$, we reduce the equation shown above to the following equivalent linear equation: $Y = \left(\frac{1}{3} + \frac{2}{3} \cdot v^2 - v\right) + \frac{4}{3} \cdot v \cdot Y$. Therefore,

```

while  $i > 0$  do
  if  $\star$  then  $x := x + i$ 
  else  $y := y + i$  fi;
   $i := i - 1$ 
od

```



$$\begin{aligned}
Z_{v_1} &= [i > 0] \cdot Z_{v_2} + [i \leq 0] \cdot Z_{v_4} \\
Z_{v_2} &= [x := x + i] \cdot Z_{v_3} + [y := y + i] \cdot Z_{v_3} \\
Z_{v_3} &= [i := i - 1] \cdot Z_{v_1} \\
Z_{v_4} &= \varepsilon
\end{aligned}$$

(a) An example program

(b) Control-flow graph (CFG)

(c) CFG as an equation system

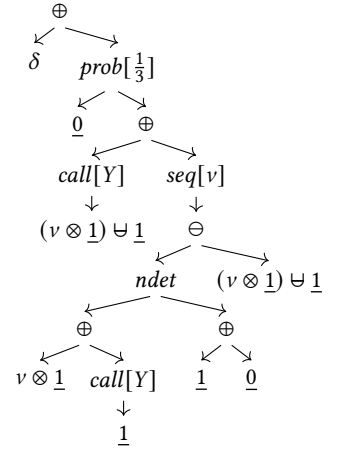
Fig. 5. The running example of algebraic program analysis.

on the $(i + 1)^{st}$ Newton round, we solve the equation above where v is set to $v^{(i)}$ to obtain the correction term $\Delta(i) \triangleq \frac{1+2(v^{(i)})^2-3v^{(i)}}{3-4v^{(i)}}$, and perform the assignment $v^{(i+1)} \leftarrow v^{(i)} + \Delta(i) = \frac{1-2(v^{(i)})^2}{3-4v^{(i)}}$. The first four elements of the Newton sequence are $v^{(0)} = \frac{1}{3}$, $v^{(1)} = \frac{7}{15}$, $v^{(2)} = \frac{127}{255}$, $v^{(3)} = \frac{32767}{65535} \approx 0.499992$, whereas the first four elements of the Kleene sequence are $\kappa^{(0)} = 0$, $\kappa^{(1)} = \frac{1}{3}$, $\kappa^{(2)} = \frac{11}{27}$, $\kappa^{(3)} = \frac{971}{2187} \approx 0.443987$, and the Newton sequence converges faster than the Kleene sequence.

3 OVERVIEW,

PART II: ANALYSIS OF UNSTRUCTURED CONTROL-FLOW

This section describes how we extend the approach from §2 to analyze probabilistic programs with arbitrary control-flow, i.e., from the upper-right quadrant to the lower-right quadrant in Fig. 1. We use *regular infinite-tree expressions* to encode the control-flow of probabilistic programs and then define differentials of such tree expressions. §3.1 reviews *algebraic program analysis* and an extension of NPA to support loops. §3.2 reviews *control-flow hyper-graphs*, which provide a suitable representation for probabilistic programs with multiple kinds of confluences. §3.3 and §3.4 describe NPA-PMA's intra- and inter-procedural parts, respectively.

Fig. 4. $\delta \oplus \mathcal{D}f|_v(Y)$ as a tree.

3.1 Prior Work: Algebraic Program Analysis

Algebraic program analysis [Farzan and Kincaid 2013; Kincaid et al. 2021; Tarjan 1981a] provides an alternative to classic iteration-based program analysis. We illustrate the recipe of algebraic program analysis with the program shown in Fig. 5(a), where the \star symbol denotes nondeterministic-choice. Fig. 5(b) presents the program's control-flow graph (CFG), and we can view the equation system in Fig. 5(c) as a grammar for program paths: for node v_i , we use a nonterminal Z_{v_i} , which generates a set of finite paths that start from v_i and ends at the exit node of the CFG (v_4 in this example).

An iterative program analysis would reinterpret the equation system in an abstract domain and employ an iterative approximation method (e.g., Kleene iteration) to find the (post)-fixed-point of the abstract equation system. In contrast, an algebraic approach reduces program analysis to solving *path problems* in graphs [Tarjan 1981a]: one uses *regular expressions* to represent the set of paths of a graph, and then reinterprets these expressions in an abstract domain to obtain the analysis result. The key difference is that the fixed-point computation becomes *internal* in the algebraic approach, because a regular expression can use the $*$ operation to represent repetitive behavior, whereas the iterative approach usually assumes the fixed-point computation is *external* to the abstract domain. For example, we can apply a path-expression algorithm (e.g., [Tarjan 1981b]) to compute, for each variable X_{v_i} in the equation system in Fig. 5(c), a regular expression that

recognizes the set of paths starting at v_i , where $body \triangleq [i > 0] \cdot ([x := x+1] + [y := y+i]) \cdot [i := i-1]$: $Z_{v_1} = body^* \cdot [i \leq 0]$, $Z_{v_2} = ([x := x+i] + [y := y+i]) \cdot [i := i-1] \cdot body^* \cdot [i \leq 0]$, $Z_{v_3} = [i := i-1] \cdot body^* \cdot [i \leq 0]$, $Z_{v_4} = \varepsilon$. The regular expressions are *closed-form* solutions to the concrete equation system; therefore, a program analysis can be performed by reinterpreting those expressions in an abstract domain equipped with a method for reinterpreting the $*$ operation.

Recently, [Reps et al. \[2016\]](#) proposed a technique to apply the NPA framework to analyze programs with both loops and procedure calls. Recall that in §2.1, we discussed that NPA solves equations whose right-hand sides are semiring expressions, i.e., expressions formed by operations \oplus, \otimes and semiring elements. [Reps et al.](#) extended NPA to solve equations whose right-hand sides are expressions in a regular algebra by defining the differential of Kleene-star as $\mathcal{D}f|_v(Y) \triangleq (g(v))^\otimes \otimes \mathcal{D}g|_v(Y) \otimes (g(v))^\otimes$, if $f = g^\otimes$. This rule—like the other rules in Defn. 2.2—produces a linear expression (but was developed for *idempotent* semirings; the semirings we work with, e.g., the real semiring, might not be idempotent). Thus, by integrating with a path-expression algorithm, one can apply NPA to analyze programs with procedure calls and arbitrary control-flow.

3.2 Prior Work: Probabilistic Programs and Control-Flow Hyper-Graphs

In this article, we use an imperative probabilistic programming language developed by [Wang et al. \[2018\]](#). The semantics of the language is parameterized by a set \mathcal{A} of *data actions* (such as assignment) and a set \mathcal{L} of *logical conditions* (such as comparison). The language uses control-flow hyper-graphs (CFHG) to encode unstructured control-flow and multiple kinds of confluences. A *hyper-graph* H is a quadruple $\langle V, E, v^{\text{entry}}, v^{\text{exit}} \rangle$, where V is a set of nodes, E is a set of hyper-edges, and $v^{\text{entry}}, v^{\text{exit}} \in V$ are distinguished entry and exit nodes, respectively. Each *hyper-edge* in E is a pair $\langle x, (y_1, \dots, y_k) \rangle$, where $k \geq 1$ and $x, y_1, \dots, y_k \in V$ (and the order of y_i 's is significant). We assume that v^{entry} (resp., v^{exit}) does not have incoming (resp., outgoing) hyper-edges.

A *probabilistic program* is then defined as a sequence of procedures $\{X_i \mapsto H_i\}_{i=1}^n$, where the i^{th} procedure has name X_i and CFHG $H_i = \langle V_i, E_i, v_i^{\text{entry}}, v_i^{\text{exit}} \rangle$, in which each node in $V_i \setminus \{v_i^{\text{exit}}\}$ has exactly one outgoing hyper-edge. We assume that V_1, \dots, V_n are pairwise disjoint. Commands are attached to hyper-edges via a mapping $\text{Cmd} : \bigcup_{i=1}^n E_i \rightarrow \text{Cmd}$, where each command in Cmd takes one of the following forms: $\text{seq}[\text{act}]$ for a data action $\text{act} \in \mathcal{A}$, $\text{cond}[\varphi]$ for a logical condition $\varphi \in \mathcal{L}$, $\text{prob}[p]$ for a number $p \in [0, 1]$, ndet , or $\text{call}[X_i]$ for a procedure X_i .

Example 3.1 (Probabilistic Boolean programs). Fig. 6(a) demonstrates a probabilistic Boolean program, i.e., a probabilistic program with Boolean-valued variables. The grammar below defines data actions and logical conditions for Boolean programs, where x is a program variable and $p \in [0, 1]$. The data action $[x \sim \text{BER}(p)]$ draws a random value from a Bernoulli distribution with mean p , i.e., the action assigns **true** to x with probability p , and otherwise assigns **false**.

$$\text{act} \in \mathcal{A} ::= x := \varphi \mid x \sim \text{BER}(p) \mid \text{skip} \quad \varphi \in \mathcal{L} ::= x \mid \text{true} \mid \text{false} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$

Fig. 6(b) shows the CFHG of the program in Fig. 6(a), where v_1 is the entry node and v_7 is the exit node. There are three hyper-edges, each with two destination nodes in the CFHG: edge $\langle v_2, (v_3, v_1) \rangle$ is associated with a conditional-branching command $\text{cond}[b_1 \vee b_2]$, edge $\langle v_3, (v_7, v_4) \rangle$ is associated with a probabilistic-branching command $\text{prob}[0.1]$, and edge $\langle v_4, (v_5, v_6) \rangle$ is associated with a conditional-branching command $\text{cond}[b_1]$. Note that **return** (and **goto**, etc.) are not data actions, and they are encoded as control-flow hyper-edges with a singleton-valued destination-node set.

3.3 This Work: Algebraic Analysis of CFHG via Regular Infinite-Tree Expressions

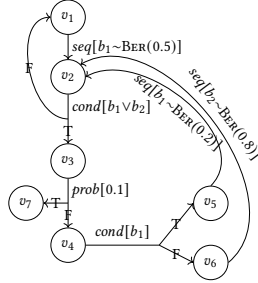
In this work, we develop new machinery to be able to apply algebraic program analysis to the CFHG of a probabilistic program. There are three steps: (i) compute a *regular infinite-tree expression* that

```

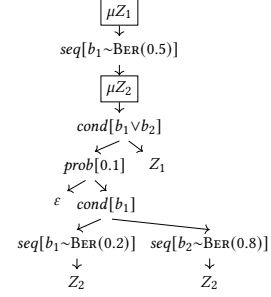
while true do
   $b_1 \sim \text{BER}(0.5)$ ;
  while  $b_1 \vee b_2$  do
    if prob(0.1) then return fi;
    if  $b_1$  then  $b_1 \sim \text{BER}(0.2)$ 
    else  $b_2 \sim \text{BER}(0.8)$ 
  fi
od
od

```

(a) An example program



(b) Control-flow hyper-graph



(c) Regular infinite-tree expression AST

$$\mu Z_1. \text{seq}[\text{act}_1] \left(\mu Z_2. \text{cond}[b_1 \vee b_2] \left(\text{prob}[0.1] \left(\epsilon, \text{cond}[b_1] \left(\text{seq}[\text{act}_2](Z_2), \text{seq}[\text{act}_3](Z_2) \right) \right), Z_1 \right) \right)$$

(d) Regular infinite-tree expression (where $\text{act}_1 \triangleq [b_1 \sim \text{BER}(0.5)]$, $\text{act}_2 \triangleq [b_1 \sim \text{BER}(0.2)]$, $\text{act}_3 \triangleq [b_2 \sim \text{BER}(0.8)]$)

Fig. 6. An example program with nested loops and unstructured control-flow.

recognizes the *hyper-path* through a CFHG, (ii) interpret that regular infinite-tree expression over an ω PMA and extract a system of equations whose right-hand sides are simple tree expressions, and (iii) apply an analysis-supplied strategy to solve that equation system.

Step (i): Compute a regular infinite-tree expression. Just as a path through a CFG is a concatenation of edges, a *hyper-path* through a CFHG is a concatenation of hyper-edges—each of which has one source node but possibly multiple destination nodes—and thus represents a *tree*, on which every root-to-leaf path corresponds to a program-execution path. For example, similar to the simple tree expressions introduced in §2.2, we can denote the hyper-path through “**if** prob(0.5) **then** $x := \text{true}$ **else** $x := \text{false}$ **fi**” by $\text{prob}[0.5](\text{seq}[x := \text{true}](\epsilon), \text{seq}[x := \text{false}](\epsilon))$, in the sense that every root-to-leaf path corresponds to a program path. However, the hyper-path through a loop “**while** x **do** $x \sim \text{BER}(0.5)$ **od**” needs to be an *infinite tree*, because of the infinite execution path

$$\text{cond}[x] \xrightarrow{T} \text{seq}[x \sim \text{BER}(0.5)] \rightarrow \text{cond}[x] \xrightarrow{T} \text{seq}[x \sim \text{BER}(0.5)] \rightarrow \text{cond}[x] \xrightarrow{T} \dots,$$

i.e., the value of variable x is always **true** during the execution.

We adopt a standard technique that represents such possibly-infinite trees as finite μ -terms [Courcelle 1983, 1990], namely *regular infinite-tree expressions*. Each regular infinite-tree expression is a finite representation of a possibly-infinite hyper-path. We use the expression $\mu Z. E$, called a μ -binder, where the *bound variable* Z can occur multiple times as a leaf node in tree expression E , to encode self-substituting E for Z an infinite number of times. For example, the infinite hyper-path through “**while** x **do** $x \sim \text{BER}(0.5)$ **od**” can be encoded as $\mu Z. \text{cond}[x](\text{seq}[x \sim \text{BER}(0.5)](Z), \epsilon)$.

Recall the probabilistic Boolean program shown in Fig. 6(a), which we discussed in Ex. 3.1. Applying Bekić’s theorem, which we will review in §4.1, we can compute a regular infinite-tree expression that represents the hyper-path in the CFHG that begins at v_1 and ends at v_7 . We present the regular infinite-tree expression in Fig. 6(d) and its AST in Fig. 6(c), where boxed nodes are μ -binders, which introduce bound variables that can be used as leaves in the sub-AST.

Step (ii): Interpretation and equation extraction. To demonstrate the interpretation of regular infinite-tree expressions with respect to ω PMAs, in this section, we consider the *Bayesian-inference* analysis of probabilistic Boolean programs in the style of prior work by Claret et al. [2013]; that is, we are interested in computing the exact probability distribution on variable valuations at the end of the program, conditioned on termination of the program. The inferred probability distribution is

called the *posterior distribution*.⁵ For a Boolean program with a set Var of variables, its probabilistic semantics can be modeled as a transformer from states (2^{Var}) to state-distributions ($2^{\text{Var}} \rightarrow [0, 1]$). Therefore, we can represent distribution transformers as matrices of type $2^{\text{Var}} \times 2^{\text{Var}} \rightarrow [0, 1]$. Wang et al. [2018] formulated a PMA on such matrices with the following definitions of operations and constants, and here we extend it to be an ω PMA $\mathcal{B} = \langle 2^{\text{Var}} \times 2^{\text{Var}} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}, \oplus, \otimes, \varphi \diamond, \oplus, \ominus, \underline{0}, \underline{1} \rangle$: $\underline{0} \triangleq \mathbf{0}$ (all-zero matrix), $\mathbf{A} \otimes \mathbf{B} \triangleq \mathbf{A} \cdot \mathbf{B}$ (matrix multiplication), $\mathbf{A} \oplus \mathbf{B} \triangleq p \cdot \mathbf{A} + (1-p) \cdot \mathbf{B}$, $\underline{1} \triangleq \mathbf{I}$ (identity matrix), $\mathbf{A} \ominus \mathbf{B} \triangleq \min(\mathbf{A}, \mathbf{B})$ (pointwise minimum), $\mathbf{A} \varphi \diamond \mathbf{B} \triangleq \Gamma_\varphi \cdot \mathbf{A} + \Gamma_{\neg\varphi} \cdot \mathbf{B}$, where Γ_φ is a diagonal matrix such that $\Gamma_\varphi(s, s) = 1$ if φ holds in s , and otherwise 0, for any state $s \in 2^{\text{Var}}$. We further define $\mathbf{A} \oplus \mathbf{B} \triangleq \mathbf{A} + \mathbf{B}$ to be pointwise addition. Wang et al. defined a mapping $\llbracket \cdot \rrbracket^{\mathcal{B}}$ from data actions to the algebra \mathcal{B} ; for example, let $\text{Var} \triangleq \{b_1, b_2\}$, we can encode sampling statements as matrices as follows, where $p, q \in [0, 1]$:

$$\llbracket b_1 \sim \text{BER}(p) \rrbracket^{\mathcal{B}} \triangleq \begin{array}{c} \text{TT} \quad \text{TF} \quad \text{FT} \quad \text{FF} \\ \text{TF} \begin{pmatrix} p & 0 & 1-p & 0 \\ 0 & p & 0 & 1-p \\ p & 0 & 1-p & 0 \\ 0 & p & 0 & 1-p \end{pmatrix}, \quad \llbracket b_2 \sim \text{BER}(q) \rrbracket^{\mathcal{B}} \triangleq \begin{array}{c} \text{TT} \quad \text{TF} \quad \text{FT} \quad \text{FF} \\ \text{TF} \begin{pmatrix} q & 1-q & 0 & 0 \\ q & 1-q & 0 & 0 \\ 0 & 0 & q & 1-q \\ 0 & 0 & q & 1-q \end{pmatrix} \end{array}$$

We denote by “ $\sigma_1 \sigma_2$ ” the program state $\{b_1 \mapsto \sigma_1, b_2 \mapsto \sigma_2\}$, where $\sigma_1, \sigma_2 \in \{\text{T}, \text{F}\}$, and rows and columns of the matrices stand for pre- and post-states, respectively. We can now use a syntax-directed method to interpret regular infinite-tree expressions without μ -binders. The method is similar to the interpretation of simple tree expressions (discussed in §2.2), except that we parameterize the interpretation by a *valuation* γ that maps bound variables to matrices:

$$\begin{aligned} \mathcal{B}_\gamma \llbracket \text{seq}[\text{act}](E) \rrbracket &\triangleq \llbracket \text{act} \rrbracket^{\mathcal{B}} \otimes \mathcal{B}_\gamma \llbracket E \rrbracket, & \mathcal{B}_\gamma \llbracket \text{prob}[p](E_1, E_2) \rrbracket &\triangleq \mathcal{B}_\gamma \llbracket E_1 \rrbracket \oplus \mathcal{B}_\gamma \llbracket E_2 \rrbracket, \\ \mathcal{B}_\gamma \llbracket \varepsilon \rrbracket &\triangleq \underline{1}, & \mathcal{B}_\gamma \llbracket Z \rrbracket &\triangleq \gamma(Z), & \mathcal{B}_\gamma \llbracket \text{cond}[\varphi](E_1, E_2) \rrbracket &\triangleq \mathcal{B}_\gamma \llbracket E_1 \rrbracket \varphi \diamond \mathcal{B}_\gamma \llbracket E_2 \rrbracket. \end{aligned}$$

We now reduce the interpretation of μ -binders to extracting and solving a system of equations whose right-hand sides are μ -free simple tree expressions. We observe that for an expression $\mu Z. E_o$, where E_o is μ -free, we can eliminate the μ -binder by substituting Z with a fresh variable Z' in E_o , written $E_o[Z'/Z]$, and then introducing an equation $Z' = E_o[Z'/Z]$, whose right-hand side is a μ -free simple tree expression. The interpretation of the original $\mu Z. E_o$ can then be given by the solution for Z' . To interpret a regular infinite-tree expression E that contains μ -binders as subexpressions, e.g., $E \triangleq \dots \mu Z. E_o \dots$, we proceed to interpret $E' \triangleq \dots Z' \dots$ where Z' is the solution to the equation $Z' = E_o[Z'/Z]$. In the general case, where a regular infinite-tree expression can contain nested μ -binders as subexpressions, the extracted equations may have right-hand sides that contain μ -binders. We iteratively apply a similar extraction method to those equations: if $V = \dots \mu Z. E \dots$ is an equation whose right-hand side contains μ -binders, then it becomes $V = \dots Z' \dots$ plus the equation $Z' = E[Z'/Z]$ with a fresh variable Z' . For example, to interpret the expression shown in Fig. 6(c), we extract the following equations, and the interpretation of the original expression is the value of Z'_1 from the solution to the equation system, i.e., a valuation $\gamma : \{Z'_1, Z'_2\} \rightarrow \mathcal{B}$ such that $\gamma(Z'_1) = \mathcal{B}_\gamma \llbracket f_1(Z'_1, Z'_2) \rrbracket$ and $\gamma(Z'_2) = \mathcal{B}_\gamma \llbracket f_2(Z'_1, Z'_2) \rrbracket$:

$$\begin{aligned} Z'_1 &\triangleq f_1(Z'_1, Z'_2) \triangleq \text{seq}[\text{act}_1](Z'_2), \\ Z'_2 &\triangleq f_2(Z'_1, Z'_2) \triangleq \text{cond}[b_1 \vee b_2](\text{prob}[0.1](\varepsilon, \text{cond}[b_1](\text{seq}[\text{act}_2](Z'_2), \text{seq}[\text{act}_3](Z'_2)), Z'_1)). \end{aligned} \quad (7)$$

Step (iii): Apply an analysis-supplied solving strategy. Recall that in §2.2, we say a simple tree expression is *linear* if every root-to-leaf path on the tree contains at most one $\text{call}[\cdot]$ node. We generalize the notion of linearity to count the number of *multiplicative* factors; in the tree expressions f_1 and f_2 , the variables Z'_1 and Z'_2 are the only multiplicative factors. Because bound variables can only appear as leaf nodes, it is clear that every root-to-leaf path on a tree can contain at most one bound variable, and hence at most one multiplicative factor. We can also illustrate the

⁵Bayesian inference usually uses *conditioning* to specify posterior distributions. For the analysis in §3.3 (and also §5.1), we can encode a conditioning failure as a non-terminating loop, i.e., $\text{condition}(\varphi) \triangleq \text{while } \neg\varphi \text{ do skip od}$.

linearity by interpreting the simple tree expressions f_1 and f_2 with respect to \mathcal{B} and observing that the interpretations are *linear* matrix expressions in Z'_1 and Z'_2 :

$$Z'_1 = \llbracket \text{act}_1 \rrbracket^{\mathcal{B}} \cdot Z'_2,$$

$$Z'_2 = \Gamma_{b_1 \vee b_2} \cdot (0.1 \cdot \mathbf{I} + 0.9 \cdot (\Gamma_{b_1} \cdot \llbracket \text{act}_2 \rrbracket^{\mathcal{B}} \cdot Z'_2 + \Gamma_{\neg b_1} \cdot \llbracket \text{act}_3 \rrbracket^{\mathcal{B}} \cdot Z'_2)) + \Gamma_{\neg(b_1 \vee b_2)} \cdot Z'_1.$$

Our NPA-PMA framework requires that each individual analysis supply a strategy for solving linear equation systems. For the example shown above, we can directly use linear algebra to solve the equation system. As we will show in §5.1, there are different strategies for Bayesian-inference analysis based on linear programming (LP) and algebraic decision diagrams (ADDs), respectively. Note that neither strategies analyze loops via iteration; in contrast, prior work [Claret et al. 2013; Wang et al. 2018] used Kleene-iteration-like schemes to analyze loops.

Remark 3.2. The reader may wonder whether regular infinite-tree expressions are really necessary, because one could directly extract a linear equation system from a CFHG (similar to the non-probabilistic case depicted in Fig. 5) and then solve the equations. This observation is correct for **intraprocedural** analysis. However, as we will develop in §3.4, in NPA-PMA we use regular infinite-tree expressions as right-hand sides of an **interprocedural** equation system. Such an intermediate representation enables us to generalize the notion of differentials, and thus makes it possible to have an algebraic framework for probabilistic programs with procedure calls and arbitrary control-flow.

3.4 This Work: Differentials of Regular Infinite-Tree Expressions

We now describe how our extension of NPA for solving equations with simple-tree-expression right-hand sides (discussed in §2.2) is generalized to solve equations with regular-infinite-tree-expression right-hand sides, i.e., we need to devise a method to compute the differential of regular infinite-tree expressions. Consider an example $X = f(X) \triangleq \mu Z. \text{cond}[\varphi](\text{call}[X](Z), \underline{1})$ for some condition φ , whose right-hand side encodes the program “**while** φ **do** $X()$ **od**.” Note that the expression takes a slightly different form from the expressions in §3.3: here we use *algebraic* regular infinite-tree expressions, in the sense that we transform ε to $\underline{1}$ and $\text{seq}[\text{act}]$ to $\text{seq}[c]$ where the interpretation of act is c . Fig. 7(a) shows the AST of $f(X)$ and Fig. 7(b) demonstrates its corresponding infinite hyper-path. Recall that in §2.2, we said that a simple tree expression is *linear* if every root-to-leaf path contains *at most one* $\text{call}[\cdot]$ node. We generalize the notion of linearity and say a regular infinite-tree expression is linear if every rooted path on its corresponding hyper-path contains at most one $\text{call}[\cdot]$ node. Note that Fig. 7(b) shows that $f(X)$ in our example is definitely not linear.

We develop the differential of μ -binders as if we are differentiating their corresponding hyper-paths, i.e., infinite trees, via the rules shown in §2.2. One would obtain another infinite tree similar to the one depicted in Fig. 7(d), where one can easily “fold” this infinite tree and get the regular infinite-tree expression in Fig. 7(c).

To formulate the process above, we parameterize the differentiation operator by a valuation γ that interprets bound variables, denoted by $\mathcal{D}f_\gamma|_\nu$, in the sense that the regular infinite-tree expression f being differentiated contains bound variables in $\text{dom}(\gamma)$. We write f_γ to denote the

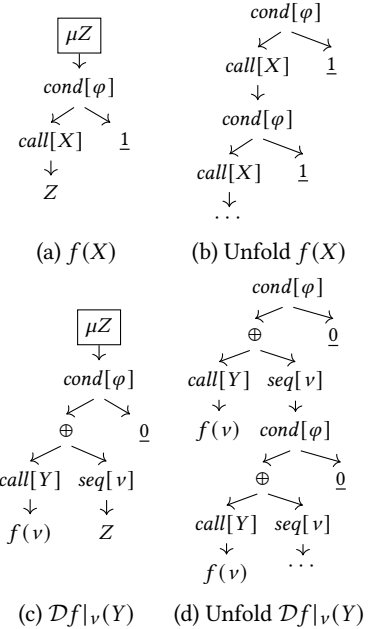


Fig. 7. An example of differentiating μ -binders.

interpretation of f under γ . Thus, the observation about “folding” leads us to define

$$\mathcal{D}f_{\gamma}|_v(Y) \triangleq \begin{cases} Z & \text{if } f = Z \\ \mu Z. \mathcal{D}g_{\gamma[Z \mapsto f_{\gamma}(v)]}|_v(Y) & \text{if } f = \mu Z. g \end{cases}$$

Recall the example $f(X) \triangleq \mu Z. \text{cond}[\varphi](\text{call}[X](Z), \underline{1})$. Let us define $g \triangleq \text{cond}[\varphi](\text{call}[X](Z), \underline{1})$ so that $f = \mu Z. g$. By the differential rule for closure expressions, we have $\mathcal{D}f_{\gamma}|_v(Y) = \mu Z. \mathcal{D}g_{\gamma[Z \mapsto f_{\gamma}(v)]}|_v(Y)$. Because g is essentially a simple tree expression, we obtain its differential as $\mathcal{D}g_{\gamma[Z \mapsto f_{\gamma}(v)]}|_v(Y) = \text{cond}[\varphi](\text{call}[Y](f_{\gamma}(v)) \oplus \text{seq}[v](Z), \underline{0})$. Fig. 7(a) shows the AST for f ; Fig. 7(c) shows the AST for the differential of f at a value v .

Recall that we apply Newton’s method to find the least fixed-point of $X = f(X)$, by solving a sequence of subproblems for v : $v^{(0)} = f_{\gamma}(\underline{0})$ and $v^{(i+1)} = v^{(i)} \oplus \Delta^{(i)}$ for $i \geq 0$, where $\Delta^{(i)}$ is the least solution of $Y = \delta^{(i)} \oplus \mathcal{D}f_{\gamma}|_{v^{(i)}}(Y)$, and $\delta^{(i)} \triangleq f_{\gamma}(v^{(i)}) \ominus v^{(i)}$. In the example above, the differential $\mathcal{D}f_{\gamma}|_v(Y)$ at some value v is $\mu Z. \text{cond}[\varphi](\text{call}[Y](f_{\gamma}(v)) \oplus \text{seq}[v](Z), \underline{0})$, i.e., a regular infinite-tree expression with a μ -binder. Thus, the equation we need to solve in a Newton round is

$$Y = \delta \oplus \mu Z. \text{cond}[\varphi](\text{call}[Y](f_{\gamma}(v)) \oplus \text{seq}[v](Z), \underline{0}).$$

We now apply the extraction method sketched in §3.3 to eliminate μ -binders and obtain an equation system whose right-hand sides are μ -free:

$$Y = \delta \oplus Z', \quad Z' = \text{cond}[\varphi](\text{call}[Y](f_{\gamma}(v)) \oplus \text{seq}[v](Z'), \underline{0}),$$

which are indeed *linear* equations, in the sense that in each of the right-hand sides—viewed as a simple tree expression—every root-to-leaf path contains at most one multiplicative factor (a $\text{call}[Y]$ node or a leaf node with the variable Z'). Therefore, the analysis-supplied strategy for solving linear equation systems can be applied to solve the linearized equation system in a Newton round.

4 TECHNICAL DETAILS

Fig. 8 illustrates the overall pipeline of interprocedural program analysis in NPA-PMA. The inputs of our framework are (i) a probabilistic program $\{X_i \mapsto H_i\}_{i=1}^n$, where each H_i is the CFHG of the procedure X_i , and (ii) an abstract domain for the program analysis. The output of our framework is a vector \vec{v} of elements in the abstract domain, where each v_i is the analysis result for the procedure X_i . NPA-PMA first translates the probabilistic program into an equation system (boxes 1 to 4); applies a differentiation process and a normalization transformation to extract a system of linear equations (boxes 5 to 7); and, repeatedly, updates some constants in the equations and solves the linear equations (box 8). In §2, we gave an overview of the pipeline under the assumption that each procedure X_i does not contain unstructured control-flow; consequently, it is straightforward to translate such a program into an equation system in which each right-hand side is a simple tree expression, and to then linearize the equations (see §2.2). In §3, we considered unstructured control-flow, which requires a more complex treatment for equation extraction (see §3.3) and differentiation (see §3.4).

We adopt existing techniques on regular infinite-tree expressions and give an algorithm that translates a probabilistic program $\{X_i \mapsto H_i\}_{i=1}^n$ into an equation system $\{X_i = E_{X_i}\}_{i=1}^n$ whose right-hand sides are regular infinite-tree expressions (§4.1). We then proceed to the novel parts of NPA-PMA. We devise a new family of algebraic structures, called ω PMAs, for specifying abstract

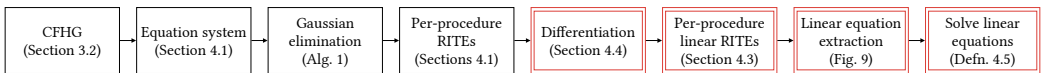


Fig. 8. The pipeline of NPA-PMA. “RITE” denotes “regular infinite-tree expression.” Red double borders indicate contributions of this work. The Newtonian phase repeatedly updates some constants in the linear equations and solves the linear equations.

domains of program analyses (§4.2). We then describe a method for solving a linear equation system over an ω PMA (§4.3). Next, we develop NPA-PMA's mechanism for solving interprocedural equation systems with regular-infinite-tree-expression right-hand sides (§4.4). We end this section with a discussion for the soundness of NPA-PMA (§4.5). Proofs are included in Appendices A to C.

4.1 Background: Regular Infinite-Tree Expressions

Let $\text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$ denote the set of *regular infinite-tree expressions* over a ranked alphabet \mathcal{F} and a set \mathcal{K} of free variables. A *ranked alphabet* is a pair $\langle \mathcal{F}, \text{Arity} \rangle$ where \mathcal{F} is a nonempty set and Arity is a mapping from \mathcal{F} to \mathbb{N} . The *arity* of a symbol $f \in \mathcal{F}$ is $\text{Arity}(f)$. The set of symbols of arity n is denoted by \mathcal{F}_n . We use parentheses and commas to specify symbols with their arity, e.g., $f(\cdot)$ specifies a binary symbol f . The set \mathcal{K} includes *free variables* that can be used as leaves (i.e., symbols with arity zero) and would not be bound by any μ -binder. We define $\text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$ to be inductively generated from the rules listed below, where \uplus stands for disjoint union. Each regular infinite-tree expression corresponds to exactly one hyper-path.

$$\begin{array}{c} \frac{a \in \mathcal{F}_0 \uplus \mathcal{K}}{a \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})} \text{ (LEAF)} \quad \frac{f \in \mathcal{F}_n \quad n > 0 \quad \forall i = 1, \dots, n: E_i \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})}{f(E_1, \dots, E_n) \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})} \text{ (NODE)} \\[10pt] \frac{E_1 \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K} \uplus \{Z\}) \quad E_2 \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})}{E_1 \#_Z E_2 \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})} \text{ (CONCAT)} \quad \frac{E \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K} \uplus \{Z\})}{\mu Z. E \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})} \text{ (MU)} \end{array}$$

We include the formal development of hyper-paths (as *possibly-infinite trees*) in Appendix A, following a well-studied theory of infinite trees [Courcelle 1983, 1990].

A free variable $Z \in \mathcal{K}$ intuitively represents a *substitution* placeholder, for which we introduce the following notation.

Definition 4.1. $E_1 \#_Z E_2$ represents the tree obtained by substituting the tree encoded by E_2 at each occurrence of a leaf with variable Z in the tree encoded by E_1 . A μ -binder $\mu Z. E$ represents the tree obtained by $(E \#_Z E \#_Z \dots \#_Z E \#_Z \dots)$, i.e., self-substituting E for the bound variable Z an infinite number of times.

To represent the control-flow hyper-path of a probabilistic program, we define the following ranked alphabet, where \mathcal{A} is the set of data actions and \mathcal{L} is the set of logical conditions, both introduced in §3.2:

$$\begin{aligned} \mathcal{F} \triangleq & \{\varepsilon\} \cup \{\text{seq}[\text{act}]() \mid \text{act} \in \mathcal{A}\} \cup \{\text{cond}[\varphi](\cdot) \mid \varphi \in \mathcal{L}\} \cup \{\text{prob}[p](\cdot) \mid p \in [0, 1]\} \cup \{\text{ndet}(\cdot)\} \\ & \cup \{\text{call}[X_i]() \mid X_i \text{ is a procedure}\}. \end{aligned}$$

We now define the correspondence between a regular infinite-tree expression and the CFHG of a probabilistic programs. Recall that a CFHG H is a quadruple $\langle V, E, v^{\text{entry}}, v^{\text{exit}} \rangle$, where each hyper-edge $e \in E$ is associated with a command $\text{Cmd}(e) \in \mathcal{F}$. Let Z_v for each $v \in V$ be a free variable. We then treat each hyper-edge $e = \langle v, (u_1, \dots, u_k) \rangle \in E$ as an *equation* $Z_v = \text{Cmd}(e) (Z_{u_1}, \dots, Z_{u_k})$, where the right-hand side is a regular infinite-tree expression in $\text{RegExp}^\infty(\mathcal{F}, \{Z_v \mid v \in V\})$ by the rule (NODE). Thus, we can extract the following equation system from a CFHG:

$$Z_v = \text{Cmd}(e) (Z_{u_1}, \dots, Z_{u_k}) \quad \text{for } e = \langle v, (u_1, \dots, u_k) \rangle \in E, \quad Z_{v^{\text{exit}}} = \varepsilon,$$

where each free variable Z_v appears exactly once as a left-hand side, because every non-exit node has exactly one outgoing hyper-edge (see §3.2).

We then apply Bekić's theorem [Bekić 1984] to compute a regular infinite-tree expression $E_v \in \text{RegExp}^\infty(\mathcal{F}, \emptyset)$ for each left-hand side Z_v ($v \in V$) as a closed-form solution to the equation system. Informally, Bekić's theorem allows splitting a mutual recursion into *univariate* recursions; in our setting, this amounts to treating the whole equation system extracted from a CFHG as a mutual recursion over $\{Z_v \mid v \in V\}$ and obtaining for each left-hand side Z_v , an expression where

Algorithm 1 Beki's theorem based on Gaussian elimination**Input:** An equation system $\{Z_i = R_i\}_{i=1}^n$ where each $R_i \in \text{RegExp}^\infty(\mathcal{F}, \{Z_i \mid i = 1, \dots, n\})$ **Output:** A closed-form solution $\{Z_i = E_i\}_{i=1}^n$ where each $E_i \in \text{RegExp}^\infty(\mathcal{F}, \emptyset)$

```

for  $i \leftarrow 1$  to  $n$  do                                ▶ Front-solving
  if  $Z_i$  appears in  $R_i$  then  $R_i \leftarrow \mu Z_i. R_i$       ▶ Loop-solving
  for each  $j > i$  such that  $Z_i$  appears in  $R_j$  do  $R_j \leftarrow R_j \dot{+}_{Z_i} R_i$ 
  for  $i \leftarrow n$  to  $2$  do                                ▶ Back-solving
    for each  $j < i$  such that  $Z_i$  appears in  $R_j$  do  $R_j \leftarrow R_j \dot{+}_{Z_i} R_i$ 
return  $\{Z_i = R_i\}_{i=1}^n$ 

```

all free variables are bound by univariate recursions, i.e., μ -binders. We present an implementation of Beki's theorem based on Gaussian elimination in Algorithm 1.

4.2 ω PMA's and Algebraic Regular Infinite-Tree Expressions

Throughout our development in the rest of §4, we fix a probabilistic program P as an equation system $P \triangleq \{X_i = E_{X_i}\}_{i=1}^n$, where each E_{X_i} is in $\text{RegExp}^\infty(\mathcal{F}, \emptyset)$ and represents different program analyses depending on how the data actions in the tree are interpreted. Toward this end, we devise ω -continuous pre-Markov algebras, which can be thought as a refinement of both pre-Markov algebras [Wang et al. 2018] and ω -continuous semirings, to specify program analyses. Earlier, we described two ω PMA's: one over real numbers (see §2.2), the other over matrices (see §3.3). Recall from §2.2 that an ω PMA has three binary confluence operators, $\varphi \diamond$ (parameterized by a logical condition φ), $p \oplus$ (parameterized by $p \in [0, 1]$), and \uplus , which correspond to conditional-choice, probabilistic-choice, and nondeterministic-choice, respectively.

Definition 4.2. An ω -continuous pre-Markov algebra (ω PMA) over a set \mathcal{L} of logical conditions is an 8-tuple $\mathcal{M} = \langle M, \oplus_M, \otimes_M, \varphi \diamond_M, p \oplus_M, \uplus_M, \underline{0}_M, \underline{1}_M \rangle$, where $\langle M, \oplus_M, \otimes_M, \underline{0}_M, \underline{1}_M \rangle$ forms an ω -continuous semiring (so it admits a partial order $\sqsubseteq_M \triangleq \{(a, b) \in M \times M \mid \exists d: a \oplus_M d = b\}$ and we can define $a \oplus_M b$ for $a \sqsubseteq_M b$ to be any element d that satisfies $b \oplus_M d = a$); the operations $\varphi \diamond_M, p \oplus_M, \uplus_M$ are ω -continuous with respect to \sqsubseteq_M ; and for all $a, b, c, d \in M$ and $\varphi \in \mathcal{L}, p \in [0, 1]$, it holds that $(a \oplus_M b) \varphi \diamond_M (c \oplus_M d) = (a \varphi \diamond_M c) \oplus_M (b \varphi \diamond_M d)$ and $(a \oplus_M b) p \oplus_M (c \oplus_M d) = (a p \oplus_M c) \oplus_M (b p \oplus_M d)$.

An ω PMA interpretation is a pair $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$, where $\llbracket \cdot \rrbracket^{\mathcal{M}}$ maps data actions to \mathcal{M} .

Given a regular infinite-tree expression $E \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$, an ω PMA interpretation $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$, a valuation $\gamma: \mathcal{K} \rightarrow \mathcal{M}$, and a procedure-summary vector $\vec{v} \in \mathcal{M}^n$, we define the interpretation of E under γ and \vec{v} , denoted by $\mathcal{M}_\gamma \llbracket E \rrbracket(\vec{v})$, as follows.

$$\begin{aligned}
\mathcal{M}_\gamma \llbracket \varepsilon \rrbracket(\vec{v}) &\triangleq \underline{1}_M, & \mathcal{M}_\gamma \llbracket Z \rrbracket(\vec{v}) &\triangleq \gamma(Z), & \mathcal{M}_\gamma \llbracket \text{cond}[\varphi](E_1, E_2) \rrbracket(\vec{v}) &\triangleq \mathcal{M}_\gamma \llbracket E_1 \rrbracket(\vec{v}) \varphi \diamond_M \mathcal{M}_\gamma \llbracket E_2 \rrbracket(\vec{v}), \\
\mathcal{M}_\gamma \llbracket \text{seq}[\text{act}](E) \rrbracket(\vec{v}) &\triangleq \llbracket \text{act} \rrbracket^{\mathcal{M}} \otimes_M \mathcal{M}_\gamma \llbracket E \rrbracket(\vec{v}), & \mathcal{M}_\gamma \llbracket \text{prob}[p](E_1, E_2) \rrbracket(\vec{v}) &\triangleq \mathcal{M}_\gamma \llbracket E_1 \rrbracket(\vec{v}) p \oplus_M \mathcal{M}_\gamma \llbracket E_2 \rrbracket(\vec{v}), \\
\mathcal{M}_\gamma \llbracket \text{call}[X_i](E) \rrbracket(\vec{v}) &\triangleq v_i \otimes_M \mathcal{M}_\gamma \llbracket E \rrbracket(\vec{v}), & \mathcal{M}_\gamma \llbracket \text{ndet}(E_1, E_2) \rrbracket(\vec{v}) &\triangleq \mathcal{M}_\gamma \llbracket E_1 \rrbracket(\vec{v}) \uplus_M \mathcal{M}_\gamma \llbracket E_2 \rrbracket(\vec{v}), \\
\mathcal{M}_\gamma \llbracket \mu Z. E \rrbracket(\vec{v}) &\triangleq \text{lfp}_{\underline{0}_M}^{\sqsubseteq_M} \lambda \theta. \mathcal{M}_\gamma \llbracket Z \mapsto \theta \rrbracket \llbracket E \rrbracket(\vec{v}), & \mathcal{M}_\gamma \llbracket E_1 \dot{+}_Z E_2 \rrbracket(\vec{v}) &\triangleq \mathcal{M}_\gamma \llbracket Z \mapsto \mathcal{M}_\gamma \llbracket E_2 \rrbracket(\vec{v}) \rrbracket \llbracket E_1 \rrbracket(\vec{v}).
\end{aligned}$$

For $E_1 \dot{+}_Z E_2$, the variable Z does *not* appear in E_2 , but it might appear in E_1 . Thus, we can interpret E_2 under the valuation γ , and to interpret E_1 , we extend γ by mapping Z to the interpretation of E_2 . For $\mu Z. E$, the bound variable Z might appear in E . In this case, we should find a value θ such that $\theta = \mathcal{M}_\gamma \llbracket Z \mapsto \theta \rrbracket \llbracket E \rrbracket$. Because \mathcal{M} is an ω -continuous semiring (thus it admits an ω -cpo), we define the interpretation for μ -binders using *least fixed-points*. For a procedure call $\text{call}[X_i](E)$, we call E the *continuation* of this call to X_i , and use the procedure summary v_i to interpret the call to X_i .

For the probabilistic program $P \triangleq \{X_i = E_{X_i}\}_{i=1}^n$ represented by an interprocedural equation system with regular-infinite-tree right-hand sides, we define its interpretation via a least fixed-point:

$$\mathcal{M}[P] \triangleq \text{lfp}_{\vec{\theta}}^{\mathcal{E}_M} \lambda \vec{\theta}. \langle \mathcal{M}_{\{X_i\}}[E_{X_i}](\vec{\theta}), \dots, \mathcal{M}_{\{X_n\}}[E_{X_n}](\vec{\theta}) \rangle.$$

For easier manipulation of regular infinite-tree expressions (as well as the differentials we will develop in §4.4), we define *algebraic regular infinite-tree expressions* over the following ranked alphabet, where we essentially change data actions to algebra elements:

$$\mathcal{F}^\alpha \triangleq \{c \mid c \in \mathcal{M}\} \cup \{\text{seq}[c]() \mid c \in \mathcal{M}\} \cup \{\text{cond}[\varphi](\cdot, \cdot) \mid \varphi \in \mathcal{L}\} \cup \{\text{prob}[p](\cdot, \cdot) \mid p \in [0, 1]\} \\ \cup \{\text{ndet}(\cdot, \cdot)\} \cup \{\text{call}[X_i]() \mid i = 1, \dots, n\},$$

Similar to the interpretation of regular infinite-tree expressions, we can define the interpretation $\mathcal{M}_\gamma[E](\vec{v})$ of an algebraic expression E under γ and \vec{v} by induction on the structure of E .

LEMMA 4.3. *For any expression $E \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$, there exists an expression $E' \in \text{RegExp}^\infty(\mathcal{F}^\alpha, \mathcal{K})$, such that for any $\gamma : \mathcal{K} \rightarrow \mathcal{M}$ and $\vec{v} \in \mathcal{M}^n$, it holds that $\mathcal{M}_\gamma[E](\vec{v}) = \mathcal{M}_\gamma[E'](\vec{v})$.*

In the rest of this section, we assume that the right-hand sides of equations are all algebraic.

4.3 Solving Linear Equations

In this section, we develop a mechanism to solve *linear* equation systems. This material describes boxes 6, 7, and 8 in Fig. 8. Recall that in §2.2, we informally said that a simple tree expression is *linear* if every root-to-leaf path contains *at most one* $\text{call}[\cdot]$ node. We now formalize the idea by introducing *linear algebraic regular regular-tree expressions* over the following ranked alphabet:

$$\mathcal{F}_{\text{lin}}^\alpha \triangleq \{c \mid c \in \mathcal{M}\} \cup \{\text{seq}[c]() \mid c \in \mathcal{M}\} \cup \{\text{cond}[\varphi](\cdot, \cdot) \mid \varphi \in \mathcal{L}\} \cup \{\text{prob}[p](\cdot, \cdot) \mid p \in [0, 1]\} \\ \cup \{\text{ndet}(\cdot, \cdot), \oplus(\cdot, \cdot), \ominus(\cdot, \cdot)\} \cup \{\text{call}_{\text{lin}}[Y_i; c] \mid i = 1, \dots, n, c \in \mathcal{M}\}.$$

In $\mathcal{F}_{\text{lin}}^\alpha$, we add two binary symbol \oplus, \ominus that correspond to \oplus_M, \ominus_M , respectively, and restrict general procedure calls to linear ones of the form $\text{call}_{\text{lin}}[Y_i; c]$ by restricting the continuation of a procedure call to Y_i to be a value $c \in \mathcal{M}$.⁶ We interpret linear algebraic regular infinite-tree expressions in $\text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \mathcal{K})$ in the same way that general algebraic regular infinite-tree expressions are interpreted, except that we remove the rule for $\text{call}[X_i](E)$, and add $\mathcal{M}_\gamma[\text{call}_{\text{lin}}[Y_i; c]](\vec{v}) \triangleq v_i \otimes_M c$,

$$\mathcal{M}_\gamma[\oplus(E_1, E_2)](\vec{v}) \triangleq \mathcal{M}_\gamma[E_1](\vec{v}) \oplus_M \mathcal{M}_\gamma[E_2](\vec{v}), \quad \mathcal{M}_\gamma[\ominus(E_1, E_2)](\vec{v}) \triangleq \mathcal{M}_\gamma[E_1](\vec{v}) \ominus_M \mathcal{M}_\gamma[E_2](\vec{v}).$$

We can now state the problem of solving a system of linear equations (box 6 in Fig. 8):

Solve $\{Y_i = E_{Y_i}\}_{i=1}^n$, where each Y_i is in $\text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \emptyset)$; i.e., find the least $\vec{\theta} \in \mathcal{M}^n$ that satisfies $\mathcal{M}_{\{Y_i\}}[E_{Y_i}](\vec{\theta}) = \theta_i$ for each i .

NPA-PMA relies on an analysis-supplied strategy to solve linear equations. To simplify matters, we work with equations whose right-hand sides are normalized to be simple tree expressions (§2.2). (The normalization transformation is similar to the standard program transformation that replaces loops with tail-recursion—see below—but is performed on the equation system.) Let $\text{RegExp}^{\text{cf}}$ denote μ -free regular infinite-tree expressions. The problem statement now becomes:

Solve $\{Y_i = E_{Y_i}\}_{i=1}^n$, where $Y_i \in \text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \emptyset)$, given a strategy that can solve the equation system $\{Y_i = F_{Y_i}\}_{i=1}^n \uplus \{Z = F_Z\}_{Z \in \mathcal{Z}}$, whose right-hand sides are in $\text{RegExp}^{\text{cf}}(\mathcal{F}_{\text{lin}}^\alpha, \mathcal{Z})$.

This formulation reflects that there are *two* sources of multiplicative factors: (i) $\text{call}_{\text{lin}}[Y_i; c]$ can be seen as $Y_i \otimes c$, and (ii) $\text{seq}[c](Z)$ can be seen as $c \otimes Z$ for a variable $Z \in \mathcal{Z}$.

The normalization transformation corresponds to box 7 in Fig. 8; it removes μ -binders by introducing a new equation with a fresh left-hand side variable for each μ -binder. Fig. 9 presents some

⁶We strategically change the procedure names from X_i 's to Y_i 's to signify that later in §4.4, linear expressions are derived by differentials of non-linear expressions—and thus a linearized equation system should be understood as being defined over a different set of variables from the original interprocedural equation system.

$$\begin{array}{c}
\frac{c \in \mathcal{M}}{\Gamma \vdash \text{call}_{\text{lin}}[Y_i; c] \Downarrow \text{call}_{\text{lin}}[Y_i; c] \mid \emptyset} \text{ (CALL-LIN)} \\
\frac{\Gamma[Z \mapsto Z] \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_2 \mid \Theta_2}{\Gamma \vdash E_1 \text{ ++ } E_2 \Downarrow F_1 \text{ ++ } F_2 \mid \Theta_1 \uplus \Theta_2} \text{ (CONCAT)} \\
\frac{Z' \text{ fresh} \quad \Gamma[Z \mapsto Z'] \vdash E \Downarrow F \mid \Theta}{\Gamma \vdash \mu Z. E \Downarrow Z' \mid \Theta \uplus \{Z' = F\}} \text{ (MU)}
\end{array}$$

Fig. 9. Selected rules for extracting a μ -free linear regular infinite-tree expression and an associated equation system from a linear regular infinite-tree expression.

rules for a routine $\Gamma \vdash E \Downarrow F \mid \Theta$ that extracts from a linear regular infinite-tree expression E a μ -free expression F and an equation system Θ , where Γ is a mapping on variables. For a μ -binder $\mu Z. E$, the rule (Mu) introduces a fresh variable Z' as the result and adds a new equation $Z' = F$, where F is the extracted μ -free expression from E with all occurrences of Z mapped to Z' .

Example 4.4. Recall the probabilistic Boolean program shown in Fig. 6(a) and the regular infinite-tree expression through its CFHG shown in Fig. 6(d). Let us denote an algebraic version of the regular infinite-tree expression by E , where $E \in \text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \emptyset)$, with respect to the ω PMA \mathcal{B} for Bayesian-inference analysis presented in §3.3, i.e., $\mu Z_1. \text{seq}[C_1](\mu Z_2. \text{cond}[b_1 \vee b_2](\text{prob}[0.1](\underline{1}, \text{cond}[b_1](\text{seq}[C_2](Z_2), \text{seq}[C_3](Z_2))), Z_1))$, where C_1, C_2, C_3 are the interpretations of $\text{act}_1, \text{act}_2, \text{act}_3$ (shown in Fig. 6), respectively. We now show how the normalization transformation works by establishing $\{\} \vdash E \Downarrow F \mid \Theta$ for some F and Θ . After applying the rule (Mu) twice, we derive the following normalization for a μ -free subexpression:

$$\{Z_1 \mapsto Z'_1, Z_2 \mapsto Z'_2\} \vdash \text{cond}[b_1 \vee b_2](\text{prob}[0.1](\underline{1}, \text{cond}[b_1](\text{seq}[C_2](Z_2), \text{seq}[C_3](Z_2))), Z_1) \Downarrow \\
\text{cond}[b_1 \vee b_2](\text{prob}[0.1](\underline{1}, \text{cond}[b_1](\text{seq}[C_2](Z'_2), \text{seq}[C_3](Z'_2))), Z'_1) \mid \{\}.$$

We then add a new equation to Θ for each application of the rule (Mu):

$$\begin{aligned}
\{\} \vdash E \Downarrow Z'_1 \mid \{Z'_1 = \text{seq}[C_1](Z'_2)\}, \\
Z'_2 = \text{cond}[b_1 \vee b_2](\text{prob}[0.1](\underline{1}_{\mathcal{B}}, \text{cond}[b_1](\text{seq}[C_2](Z'_2), \text{seq}[C_3](Z'_2))), Z'_1). \quad (8)
\end{aligned}$$

We now formalize *linear-recursion-solving strategies* (box 8 in Fig. 8).

Definition 4.5. An algorithm *solve* (parameterized by \mathcal{K}) is said to be a *linear-recursion-solving strategy* for an ω PMA \mathcal{M} , if for any equation system $\{Y_i = E_{Y_i}\}_{i=1}^n \uplus \{Z = E_Z\}_{Z \in \mathcal{Z}}$ where \mathcal{Z} is a finite set of variables that is disjoint from $(\mathcal{F}_{\text{lin}}^\alpha)_0 \cup \mathcal{K}$ and right-hand sides of the equations are in $\text{RegExp}^{\text{cf}}(\mathcal{F}_{\text{lin}}^\alpha, \mathcal{K} \uplus \mathcal{Z})$, and any valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$, the routine $\text{solve}_{\mathcal{K}}(\{Y_i = E_{Y_i}\}_{i=1}^n, \{Z = E_Z\}_{Z \in \mathcal{Z}}, \gamma)$ returns the *least* mapping $\iota : \mathcal{Z} \rightarrow \mathcal{M}$ and vector \vec{v} , such that $\iota(Z) = \mathcal{M}_{\gamma \uplus \iota}[\![E_Z]\!](\vec{v})$ for each $Z \in \mathcal{Z}$ and $\iota_i = \mathcal{M}_{\gamma \uplus \iota}[\![E_{Y_i}]\!](\vec{v})$ for each $i = 1, \dots, n$, with respect to the order $\sqsubseteq_{\mathcal{M}}$.

Our method for solving an equation system $\{Y_i = E_{Y_i}\}_{i=1}^n$ whose right-hand sides are in $\text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \mathcal{K})$ —given a valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$ —proceeds in three steps:

- (1) For each equation $Y_i = E_{Y_i}$, derive $\{Z \mapsto Z\}_{Z \in \mathcal{K}} \vdash E_{Y_i} \Downarrow F_{Y_i} \mid \Theta_i$.
- (2) Run $\text{solve}_{\mathcal{K}}(\{Y_i = F_{Y_i}\}_{i=1}^n, \biguplus_{i=1}^n \Theta_i, \gamma)$ to obtain a solution mapping $\iota : \text{dom}(\biguplus_{i=1}^n \Theta_i) \rightarrow \mathcal{M}$ and a procedure-summary vector $\vec{v} \in \mathcal{M}^n$.
- (3) Compute $\mathcal{M}_{\gamma \uplus \iota}[\![F_{Y_i}]\!](\vec{v})$ as the solution for Y_i for each $i = 1, \dots, n$.

THEOREM 4.6. *Given a procedure *solve* that meets the requirements of Defn. 4.5, the method presented above computes $\text{lfp}_{\vec{v}_M}^{\sqsubseteq_M} \lambda \vec{\theta}. \langle \mathcal{M}_\gamma[\![E_{Y_1}]\!](\vec{\theta}), \dots, \mathcal{M}_\gamma[\![E_{Y_n}]\!](\vec{\theta}) \rangle$.*

Example 4.7. Consider the regular infinite-tree expression E from Ex. 4.4 and an equation system $\{Y = E\}$. In step (1), we apply the rules in Fig. 9 to E and then derive the normalization judgment shown in Eqn. (8). The derived equations take a similar form to Eqn. (7). In step (2), we run the analysis-supplied strategy, i.e., $\text{solve}_{\{\}}(Y = Z'_1, \Theta, \{\})$, to obtain a solution mapping $\iota : \{Z'_1, Z'_2\} \rightarrow \mathcal{B}$

and a procedure-summary $\nu \in \mathcal{B}$ for Y . We will describe two such strategies for Bayesian-inference analysis in §5.1. Finally, in step (3), we compute $\mathcal{B}_i[\llbracket Z'_1 \rrbracket](\nu) = \iota(Z'_1)$ as the solution for Y .

4.4 Solving Equations via Newton's Method

In §4.3, we discussed solving linear equations. We now consider the problem of solving equation systems in the general case by extending Newtonian Program Analysis [Esparza et al. 2008a, 2010], i.e., we develop a differentiation process for regular infinite-tree expressions (box 5 in Fig. 8).

Let $f \in \text{RegExp}^\infty(\mathcal{F}^\alpha, \mathcal{K})$. For brevity, in this section, we use $f_Y(\vec{\nu})$ to denote $\mathcal{M}_Y[\llbracket f \rrbracket](\vec{\nu})$ for a valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$ and a procedure-summary vector $\vec{\nu}$. We also write $f(\vec{X})$ from time to time to indicate that the procedure-call symbols in the alphabet are $\{\text{call}[X_i]() \mid i = 1, \dots, n\}$.

Definition 4.8. Let $f(\vec{X}) \in \text{RegExp}^\infty(\mathcal{F}^\alpha, \mathcal{K})$. The *differential* of $f(\vec{X})$ with respect to X_j at $\vec{\nu} \in \mathcal{M}^n$ under a valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$, denoted by $\mathcal{D}_{X_j} f_Y|_{\vec{\nu}}(\vec{Y})$, is defined as follows:

$\mathcal{D}_{X_j} f_Y _{\vec{\nu}}(\vec{Y}) \triangleq$	$\begin{cases} \text{seq}[\nu_k](\mathcal{D}_{X_j} g_Y _{\vec{\nu}}(\vec{Y})) & \text{if } f = \text{call}[X_k](g) \text{ and } k \neq j \\ \oplus(\text{call}_{\text{lin}}[Y_j; g_Y(\vec{\nu})], \text{seq}[\nu_j](\mathcal{D}_{X_j} g_Y _{\vec{\nu}}(\vec{Y}))) & \text{if } f = \text{call}[X_j](g) \\ \oplus(\text{ndet}(\oplus(g_Y(\vec{\nu}), \mathcal{D}_{X_j} g_Y _{\vec{\nu}}(\vec{Y})), \oplus(h_Y(\vec{\nu}), \mathcal{D}_{X_j} h_Y _{\vec{\nu}}(\vec{Y}))), f_Y(\vec{\nu})) & \text{if } f = \text{ndet}(g, h) \\ Z & \text{if } f = Z \in \mathcal{K} \\ \mathcal{D}_{X_j} g_Y[Z \mapsto h_Y(\vec{\nu})] _{\vec{\nu}}(\vec{Y}) \oplus \mathcal{D}_{X_j} h_Y _{\vec{\nu}}(\vec{Y}) & \text{if } f = g \oplus_Z h \\ \mu Z. \mathcal{D}_{X_j} g_Y[Z \mapsto f_Y(\vec{\nu})] _{\vec{\nu}}(\vec{Y}) & \text{if } f = \mu Z. g \end{cases}$	
--	---	--

The complete definition is included in Appendix B. Let \vec{f} be a vector of algebraic regular infinite-tree expressions. The *multivariate differential* of \vec{f} at $\vec{\nu}$ under γ , denoted by $\mathcal{D}_{\vec{f}}|_{\vec{\nu}}(\vec{Y})$, is defined as

$$\mathcal{D}_{\vec{f}}|_{\vec{\nu}}(\vec{Y}) \triangleq \left\langle \mathcal{D}_{X_1}(f_1)_Y|_{\vec{\nu}}(\vec{Y}) \oplus \dots \oplus \mathcal{D}_{X_n}(f_1)_Y|_{\vec{\nu}}(\vec{Y}), \dots, \mathcal{D}_{X_1}(f_n)_Y|_{\vec{\nu}}(\vec{Y}) \oplus \dots \oplus \mathcal{D}_{X_n}(f_n)_Y|_{\vec{\nu}}(\vec{Y}) \right\rangle,$$
 where we use \oplus as an infix operator. We use $\mathcal{D}(f_i)_Y|_{\vec{\nu}}(\vec{Y})$ to denote the i^{th} component of $\mathcal{D}_{\vec{f}}|_{\vec{\nu}}(\vec{Y})$.

Example 4.9. Consider the equation $X = f(X)$ for a single-procedure program with $f(X) \triangleq \mu Z. \text{cond}[\varphi](\text{call}[X](\text{seq}[c_1](Z))), c_2)$, which represents the program “**while** φ **do** $X()$; c_1 **od**; c_2 ,” for some $\varphi \in \mathcal{L}$ and $c_1, c_2 \in \mathcal{M}$. Suppose we want the differential of f at ν . By Defn. 4.8, we obtain

$$\mathcal{D}f|_{\nu}^{(1)}(Y) = \mu Z. \text{cond}[\varphi](\oplus(\text{call}_{\text{lin}}[Y; c_1 \otimes_M f_{\{\}}(\nu)], \text{seq}[\nu](\text{seq}[c_1](Z))), \underline{0}_M),$$

which is a *linear* regular infinite-tree expression in $\text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \emptyset)$.

LEMMA 4.10. *If \vec{f} is a vector of regular infinite-tree expressions in $\text{RegExp}^\infty(\mathcal{F}^\alpha, \mathcal{K})$, it holds that $\mathcal{D}_{\vec{f}}|_{\vec{\nu}}(\vec{Y})$ is a vector of regular infinite-tree expressions in $\text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \mathcal{K})$.*

We now describe Newton's method for ω PMAs by defining a Newton sequence of Newton approximants. In this sense, NPA-PMA is still *iterative* when solving general recursion.

Definition 4.11. Let \vec{f} be a vector of regular infinite-tree expressions in $\text{RegExp}^\infty(\mathcal{F}^\alpha, \emptyset)$. A *Newton sequence* $\{\vec{\nu}^{(i)}\}_{i \in \mathbb{N}}$ is given by $\vec{\nu}^{(0)} \triangleq \vec{f}_{\{\}}(\underline{0}_M)$ and $\vec{\nu}^{(i+1)} \triangleq \vec{\nu}^{(i)} \oplus_M \vec{\Delta}^{(i)}$ for $i \geq 0$, where $\vec{\Delta}^{(i)}$ is the least solution of the following linearized equation system

$$\vec{Y} = \oplus(\vec{f}_{\{\}}(\vec{\nu}^{(i)}) \ominus_M \vec{\nu}^{(i)}, \mathcal{D}_{\vec{f}_{\{\}}}|_{\vec{\nu}^{(i)}}(\vec{Y})),$$

using the method for solving linear recursion developed in §4.3 and Thm. 4.6.

We show that the Newton sequence converges to the least fixed-point of \vec{f} .

THEOREM 4.12. *Let \vec{f} be a vector of regular infinite-tree expressions in $\text{RegExp}^\infty(\mathcal{F}^\alpha, \emptyset)$. Then the Newton sequence is monotonically increasing, and it converges to the least fixed-point as least as fast*

as the Kleene sequence, i.e., for all $i \in \mathbb{N}$, we have

$$\vec{\kappa}^{(i)} \sqsubseteq_M \vec{v}^{(i)} \sqsubseteq_M \vec{f}_{\{\}}(\vec{v}^{(i)}) \sqsubseteq_M \vec{v}^{(i+1)} \sqsubseteq_M \text{lf}_{\vec{0}_M}^{\sqsubseteq_M} \vec{f}_{\{\}} = \bigsqcup_{j \in \mathbb{N}}^{\uparrow} \vec{\kappa}^{(j)},$$

where the Kleene sequence is defined as $\vec{\kappa}^{(j)} \triangleq \vec{f}_{\{\}}^j(\vec{0}_M)$ for $j \in \mathbb{N}$. Specifically, the Newton sequence and the Kleene sequence converge to the same least fixed-point.

Finally, we connect the Newton sequence back to our original equation-solving problem.

COROLLARY 4.13. Consider the equation system $\{X_i = E_{X_i}\}_{i=1}^n$ where $E_{X_i} \in \text{RegExp}^\infty(\mathcal{F}^\alpha, \emptyset)$ for each i . Let $\vec{f} \triangleq \langle E_{X_i} \rangle_{i=1, \dots, n}$. Then by Thm. 4.12, the Newton sequence converges to the least $\vec{\theta} \in \mathcal{M}^n$ that satisfies $\vec{f}_{\{\}}(\vec{\theta}) = \vec{\theta}$, i.e., $\mathcal{M}_{\{\}}[\![E_{X_i}]\!](\vec{\theta}) = \theta_i$ for each i .

4.5 Soundness

In this section, we sketch our approach for proving the soundness of NPA-PMA, but include the details in Appendix C. We use a recently proposed family of algebraic structures, namely *Markov algebras* (MAs) [Wang et al. 2018], to specify concrete semantics of probabilistic programs. We then introduce *soundness relations* between MAs and ω PMAs and show that these relations guarantee the soundness of program analyses in NPA-PMA.

Semantic foundations. We use the interpretation of regular infinite-tree expressions over MAs to define concrete semantics of probabilistic programs. A *Markov algebra* (MA) $\mathcal{M} = \langle M, \sqsubseteq_M, \otimes_M, \diamond_M, p \oplus_M, \uplus_M, \underline{0}_M, \underline{1}_M \rangle$ over a set \mathcal{L} of logical conditions is almost the same as an ω PMA, except that it has an explicit partial order \sqsubseteq_M (instead of the \oplus_M operation), and satisfies a different set of algebraic laws: $\langle M, \sqsubseteq_M \rangle$ forms a directed-complete partial order with $\underline{0}_M$ as its least element; $\langle M, \otimes_M, \underline{1}_M \rangle$ forms a monoid; \uplus_M is idempotent, commutative, and associative, and for all $a, b \in M$ and $\varphi \in \mathcal{L}, p \in [0, 1]$ it holds that $a \varphi \diamond_M b, a p \oplus_M b \leq_M a \uplus_M b$, where \leq_M is the semilattice ordering induced by \uplus_M (i.e., $a \leq_M b$ if $a \uplus_M b = b$); and $\otimes_M, \varphi \diamond_M, p \oplus_M, \uplus_M$ are Scott-continuous. An *MA interpretation* is then a pair $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$ where $\llbracket \cdot \rrbracket^{\mathcal{M}}$ maps data actions to \mathcal{M} . Given a regular infinite-tree expression $E \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$, an MA interpretation $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$, a valuation $\gamma: \mathcal{K} \rightarrow \mathcal{M}$, and a procedure-summary vector $\vec{v} \in \mathcal{M}^n$, the interpretation of E under γ and \vec{v} , denoted by $\mathcal{M}_\gamma[\![E]\!](\vec{v})$, can be defined in the same way as the ω PMA interpretations presented in §4.2.

Example 4.14. Consider probabilistic Boolean programs with a set Var of program variables. Let $S \triangleq 2^{\text{Var}}$ denote the state space of such programs. We can formulate a demonic denotational semantics [McIver and Morgan 2005] by defining an MA interpretation $\mathcal{C} = \langle \mathcal{C}, \llbracket \cdot \rrbracket^{\mathcal{C}} \rangle$ where \mathcal{C} is an MA on $S \rightarrow \wp(S \rightarrow [0, 1])$, i.e., mappings from states to sets of state distributions.

Soundness relations. We adapt abstract interpretation [Cousot and Cousot 1977] to justify NPA-PMA by establishing a relation between the concrete and abstract semantics. We express the relation via a *soundness relation*, which is a binary relation between an MA interpretation and an ω PMA interpretation that is preserved by the algebraic operations. Intuitively, a (concrete, abstract) pair in the relation should be read as “the concrete element is approximated by the abstract element.”

Example 4.15. Let $\mathcal{C} = \langle \mathcal{C}, \llbracket \cdot \rrbracket^{\mathcal{C}} \rangle$ be an MA interpretation where \mathcal{C} is an MA for probabilistic Boolean programs, which is introduced in Ex. 4.14. Let $\mathcal{B} = \langle \mathcal{B}, \llbracket \cdot \rrbracket^{\mathcal{B}} \rangle$ be an ω PMA interpretation where \mathcal{B} is the ω PMA over matrices described in §3.3. We define the following approximation relation to indicate that the program analysis reasons about lower bounds:

$$r \Vdash \mathbf{A} \iff \forall s \in S: \forall \mu \in r(s): \forall s' \in S: \mu(s') \geq \mathbf{A}(s, s').$$

The correctness of an interpretation is then justified by the following soundness theorem, which followed by induction on the structure of regular infinite-tree expressions.

THEOREM 4.16. *Let $\mathcal{C} = \langle \mathcal{C}, \llbracket \cdot \rrbracket^{\mathcal{C}} \rangle$ be an MA interpretation. Let $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$ be an ω PMA interpretation. Let $\Vdash \subseteq \mathcal{C} \times \mathcal{M}$ be a soundness relation. Then for any regular infinite-tree expression $E \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$, $\gamma : \mathcal{K} \rightarrow \mathcal{C}$, $\gamma^\# : \mathcal{K} \rightarrow \mathcal{M}$ such that $\gamma(Z) \Vdash \gamma^\#(Z)$ for all $Z \in \mathcal{K}$, $\vec{v} : \mathcal{C}^n$, and $\vec{v}^\# : \mathcal{M}^n$ such that $v_i \Vdash v_i^\#$ for $i = 1, \dots, n$, we have $\mathcal{C}_\gamma \llbracket E \rrbracket(\vec{v}) \Vdash \mathcal{M}_{\gamma^\#} \llbracket E \rrbracket(\vec{v}^\#)$.*

5 CASE STUDIES

We implemented a prototype of NPA-PMA in OCaml; the core framework consists of around 1,000 lines of code. We conducted four case studies in which we instantiated NPA-PMA to perform different analyses of probabilistic programs. The purpose of this section is to present preliminary experimental results, and show the generality of NPA-PMA by sketching instantiations with existing abstract domains for (i) finding probability distributions of variable valuations (§5.1 & Appendix D) and (ii) upper bounds on moments (Appendix E), as well as with new abstract domains for (iii) finding linear expectation invariants (§5.2) and (iv) non-linear expectation invariants (§5.3). For some of those case studies, the subproblems generated from Newton’s method are reduced to *linear programming* (LP); in the implementations of those instantiations, we used the off-the-shelf LP solver CoinOr CLP [Clp team 2022]. All the benchmark programs are included in the supplementary material.

5.1 Bayesian-Inference Analysis

Matrix-based domain. In §3.3, we demonstrated the intraprocedural part of NPA-PMA using the Bayesian-inference analysis of probabilistic Boolean programs. The analysis is in the style of prior work [Claret et al. 2013; Etessami and Yannakakis 2015; Holtzen et al. 2020; Wang et al. 2018]; it computes the probability distribution of variable valuations conditioned on program termination. When the analyzed program contains nondeterminism, the analysis computes a lower bound on the probability distribution. We presented an ω PMA \mathcal{B} for this analysis in §3.3; the algebra \mathcal{B} further admits the partial order $\mathbf{A} \sqsubseteq_{\mathcal{B}} \mathbf{B} \triangleq \mathbf{A} \leq \mathbf{B}$ (pointwise comparison) and the subtraction operation $\mathbf{A} \ominus_{\mathcal{B}} \mathbf{B} \triangleq \mathbf{A} - \mathbf{B}$ (pointwise subtraction). We can then formulate an ω PMA interpretation $\mathcal{B} = \langle \mathcal{B}, \llbracket \cdot \rrbracket^{\mathcal{B}} \rangle$, where we sketched the definition of the semantic function $\llbracket \cdot \rrbracket^{\mathcal{B}} : \mathcal{A} \rightarrow \mathcal{B}$ for interpreting data actions in §3.3. In Ex. 4.15, we formulated a soundness relation between a concrete interpretation \mathcal{C} for probabilistic Boolean programs and the abstract interpretation \mathcal{B} .

We now describe a linear-recursion-solving strategy solve for this analysis. Consider $\text{solve}_{\mathcal{K}}(\{Y_i = E_{Y_i}\}_{i=1}^n, \{Z = E_Z\}_{Z \in \mathcal{Z}}, \gamma)$ where E_{Y_i} ’s and E_Z ’s are linear regular infinite-tree expressions in $\text{RegExp}^{\text{cf}}(\mathcal{F}_{\text{lin}}^\alpha, \mathcal{K} \uplus \mathcal{Z})$, and $\gamma : \mathcal{K} \rightarrow \mathcal{B}$ is a valuation. In §3.3, we used linear algebra to solve linear matrix equations; here, because the equations involve nondeterministic-choice, we reduce the equation-solving problem to *linear programming* (LP). We transform a min-linear matrix equation system to LP by introducing a fresh variable for each min-subexpression, and thus obtain a *normalized* matrix equation system $\vec{Z} = \vec{f}(\vec{Z})$ in which each $f_i(\vec{Z})$ is either a linear matrix expression over \vec{Z} or the minimum of two variables $\min(Z_j, Z_k)$, for some j, k . The least solution of $\vec{Z} = \vec{f}(\vec{Z})$ can then be obtained via the following LP:

$$\begin{array}{ll} \textbf{maximize} & \sum_i Z_i, \textbf{ s.t. } Z_i = C_i + \sum_{j,k} \mathbf{A}_{i,j,k} \cdot Z_j \cdot \mathbf{B}_{i,j,k} \quad \text{for } i \text{ such that } f_i(\vec{Z}) = C_i + \sum_{j,k} \mathbf{A}_{i,j,k} \cdot Z_j \cdot \mathbf{B}_{i,j,k}, \\ & Z_i \leq Z_j, Z_i \leq Z_k \quad \text{for } i \text{ such that } f_i(\vec{Z}) = \min(Z_j, Z_k), \\ & Z_i \geq 0 \quad \text{for each } i, \end{array}$$

where each Z_i can be represented by a matrix of numeric variables in an LP solver.

Our implementation of the abstract domain for Bayesian-inference analysis consists of about 400 lines of code. We evaluated the performance of NPA-PMA against Kleene iteration on 100 randomly generated probabilistic programs, each of which has two Boolean-valued variables and consists of 100 procedures, each of which takes one of the following forms: (i)

$\text{prob}[p](\text{seq}[x:=a](\text{call}[X_i](\varepsilon)), \text{seq}[y:=b](\text{call}[X_j](\varepsilon)))$ for some probability p , variables x, y , Boolean constants a, b , and procedures X_i, X_j ; (ii) $\text{prob}[p](\text{cond}[x](\text{call}[X_i](\varepsilon), \text{call}[X_j](\varepsilon)), \varepsilon)$ for some probability p , variable x , and procedures X_i, X_j ; or (iii) $\text{call}[X_i](\text{call}[X_j](\varepsilon))$ for some procedures X_i, X_j . On the benchmark suite of 100 programs, NPA-PMA and Kleene iteration derive the same analysis results (up to some negligible floating-point error). Such an experimental result is unsurprising because Thm. 4.12 ensures that Newton iteration and Kleene iteration converge to the same fixed-point. On this benchmark suite, the average number of Newton rounds performed by NPA-PMA is 9.15, whereas the average number of Kleene rounds is 3,070.42. Fig. 10 presents a scatter plot that compares the *running time* (in seconds) of Kleene iteration (x -axis) against NPA-PMA (y -axis), where a point in the lower-right triangle indicates that NPA-PMA has better performance on that benchmark. Overall, NPA-PMA outperforms Kleene iteration: the geometric mean of x/y is 4.08. Although a Newton round is usually more expensive than a Kleene round, NPA-PMA greatly reduces the number of rounds ($3,070.42 \rightarrow 9.15$), resulting in an average speedup of 4.08x.

Algebraic-decision-diagram-based domain. The matrix-based encoding leads to a very high computational complexity because the instantiation needs 2^n -by- 2^n matrices to analyze programs with n Boolean program variables. Recall that we consider the style of Bayesian-inference analysis from prior work by Claret et al. [2013]. Their implementation uses *Algebraic Decision Diagrams* (ADDs) [Bahar et al. 1997] to represent distributions compactly. In Appendix D, we develop an instantiation that uses ADDs to represent distribution transformers and carry out Bayesian-inference analysis of probabilistic programs *without* nondeterminism.

We evaluated the performance of the ADD-based domain on the same benchmark suite mentioned in §5.1. NPA-PMA achieves an overall 1.54x speedup compared with Kleene iteration. We adopted the technique of Repts et al. [2016] for solving linear equation systems using tensor products. They used their extension of NPA, called NPA-TP, to analyze non-probabilistic programs. In their experiments, NPA-TP was slower than chaotic iteration on a BDD-based predicate-abstraction domain, whereas our experimental result is much better than that. We think it is the quantitative nature of probabilistic programs that unleashes the potential of Newton’s method.

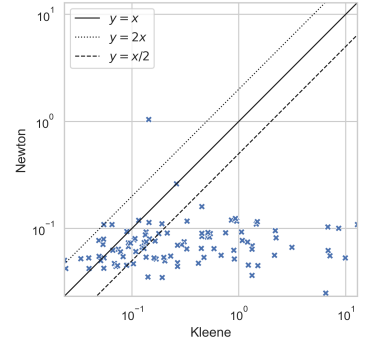


Fig. 10. Log-log scatter plots for running times (seconds) of Kleene iteration and NPA-PMA, where the solid line, dotted line, and dashed line indicate equal performance, 2x slowdown, and 2x speedup, respectively.

5.2 Expectation-Invariant Analysis

For a probabilistic program P with a set Var of numeric program variables, $\mathbb{E}[\mathcal{E}_2] \bowtie \mathcal{E}_1$ is called an *expectation invariant* [Chakarov and Sankaranarayanan 2014; Katoen et al. 2010; Wang et al. 2018], where (i) \mathcal{E}_1 (resp. \mathcal{E}_2) is an arithmetic expression over Var (resp., Var' , primed copies of the variables in Var), (ii) \bowtie is a comparison operator (e.g., \leq), and (iii) the expectation of \mathcal{E}_2 in the final valuation (i.e., the primed copies in Var' represent the values of variables at the termination of P) is related to the value of \mathcal{E}_1 in the initial valuation of P by \bowtie . We studied two kinds of expectation-invariant analysis: in this section, we consider linear expectation invariants of the form $\mathbb{E}[x'] \leq \mathcal{E}$, where x is a variable, for programs where all variables are nonnegative. In §5.3, we will describe an analysis for obtaining non-linear expectation invariants, but considers a more restricted setting in which randomness does *not* influence the flow of control in a program.

We consider programs with nonnegative program variables and we want to derive upper bounds on the expected value $\mathbb{E}[x']$ of every program variable x . In particular, we consider expectation

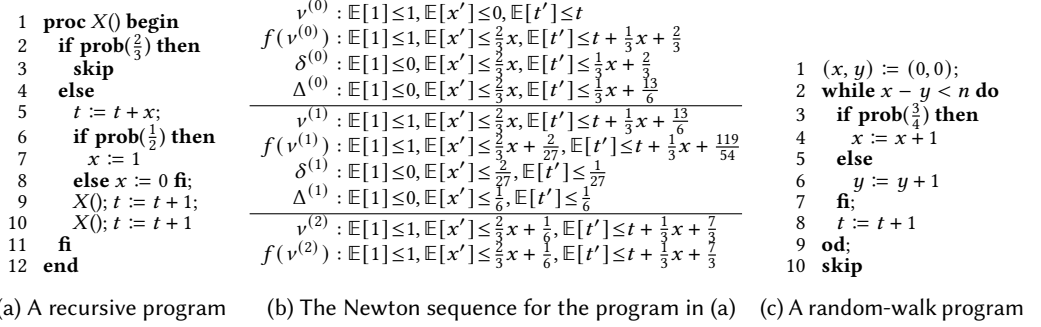


Fig. 11. Example programs for expected-value analysis.

invariants of the form $\mathbb{E}[x'] \leq \mathcal{E}$ where \mathcal{E} is a linear expression. Expected-value analysis is useful in many applications; for example, a recent work has used expected-value analysis as a subroutine for performing expected-cost analysis [Avanzini et al. 2020]. Suppose that a program uses k variables and they are represented by a vector \vec{x} . We can encode upper bounds on expected values of \vec{x} as a nonnegative matrix T of the form $\left(\frac{p}{\delta^T} \middle| \frac{\vec{b}}{M}\right)$, where p is a number in the unit interval $[0, 1]$, \vec{b} is a k -dimensional vector, and M is a k -by- k matrix, in the sense that $[\mathbb{E}[1] \mid \mathbb{E}[\vec{x}']] \leq [1 \mid \vec{x}] \cdot T$. Let V denote the space of such matrices.

We now define an ω PMA $\mathcal{V} = \langle V, \oplus_V, \otimes_V, \varphi \diamond_V, p \oplus_V, \cup_V, \underline{0}_V, \underline{1}_V \rangle$ as follows. Note that we ignore $\varphi \diamond_V$ and \cup_V temporarily because they are more involved; we will revisit them later.

$$\underline{0}_V \triangleq \left(\frac{0}{\delta^T} \middle| \frac{\vec{0}}{0}\right), \quad \underline{1}_V \triangleq \left(\frac{1}{\delta^T} \middle| \frac{\vec{0}}{1}\right), \quad T_1 \oplus_V T_2 \triangleq T_1 + T_2, \quad T_1 \otimes_V T_2 \triangleq T_1 \cdot T_2, \quad T_1 p \oplus_V T_2 \triangleq p \cdot T_1 + (1-p) \cdot T_2.$$

The ω PMA \mathcal{V} admits the partial order $T_1 \sqsubseteq_V T_2 \triangleq T_1 \leq T_2$ (i.e., pointwise comparison) and the subtraction operation $T_1 \ominus_V T_2 \triangleq T_1 - T_2$ (i.e., pointwise subtraction). Observing that the linear-recursion-solving strategy for \mathcal{V} needs to essentially solve a system of linear matrix equations (similar to the setting in §5.1), we again apply LP to solve linear recursion.

Consider the recursive procedure X shown in Fig. 11(a) with two program variables x and t . The variable t can be seen as a reward accumulator and thus its expected value corresponds to the expected accumulated reward. Fig. 11(b) presents the Newton sequence obtained when analyzing procedure X with respect to ω PMA \mathcal{V} . We observe that the Newton sequence converges in three iterations and the analysis derives expectation invariants $\mathbb{E}[x'] \leq \frac{2}{3}x + \frac{1}{6}$ and $\mathbb{E}[t'] \leq t + \frac{1}{3}x + \frac{7}{6}$. Note that the sequence shown in Fig. 11(b) does *not* start the iteration from $\underline{0}_V$; instead, we set $\nu^{(0)}$

to be $\frac{1}{t} \left(\frac{1}{\delta^T} \middle| \frac{x' \ t'}{0 \ 0 \ 0 \ 1}\right)$ because (i) we are interested in *partial correctness* (thus, we assume the program terminates with probability one by having $\mathbb{E}[1] \leq 1$ as an expectation invariant), and (ii) the program shown in Fig. 11(a) will only increment t during its execution (thus $\mathbb{E}[t'] \leq t$ is certainly an under-approximation of the final solution and we can safely set $\mathbb{E}[t'] \leq t$ in $\nu^{(0)}$).

A straightforward way to define the conditional-choice operation $\varphi \diamond_V$ is to use pointwise maximum of the operand matrices, i.e., $A \varphi \diamond_V B \triangleq \max(A, B)$. Alternatively, we can implement a more precise conditional-choice operation by adopting the idea of *rewrite functions* [Carboneaux et al. 2017; Ngo et al. 2018], which provide a mechanism to incorporate information about invariant conditions into an analysis. A rewrite function at a given location l in the program is based on a term t over program variables. When $t \geq 0$ is an invariant at l , t can be used as a *nonnegativity rewrite function*. When $t = 0$ is an invariant at l , t can be used as a *vanishing rewrite function*. The rewrite functions we use are linear expressions over program variables (and such an expression

Table 2. Selected evaluation results of expectation-invariant analysis. (Time is in seconds.)

#	Program	Time	Invariants (selected)
8	eg-tail	0.008772	$\mathbb{E}[x'] \leq x + 3, \mathbb{E}[y'] \leq y + 3, \mathbb{E}[z'] \leq 0.25z + 0.75$
12	recursive	0.001473	$\mathbb{E}[x'] \leq x + 9$
15	coupon-five-fsm	0.005259	$\mathbb{E}[t'] \leq t + 11.4167$
16	dice	0.001575	$\mathbb{E}[r'] \leq 3.5, \mathbb{E}[t'] \leq t + 1.33333$
17	exponential	0.040215	$\mathbb{E}[n'] \leq 1, \mathbb{E}[r'] \leq 0.2$
18	geometric	0.001840	$\mathbb{E}[i'] \leq n, \mathbb{E}[t'] \leq -2560i + 2560n + t$
19	non-linear-recursion	0.001642	$\mathbb{E}[t'] \leq t + 0.333333x + 2.33333, \mathbb{E}[x'] \leq 0.666667x + 0.166667$
22	unbiased	0.000624	$\mathbb{E}[r'] \leq 1.5, \mathbb{E}[t'] \leq t + 11.1111$

can contain negative coefficients). For example, consider the probabilistic program in Fig. 11(c), which implements a biased one-dimensional random walk, where the current position of the walk is represented by $x - y$. By a simple non-probabilistic program analysis, we can derive that (i) at the beginning of each loop iteration, the expression $n - x + y - 1$ is always nonnegative (and hence can be used as a nonnegativity rewrite function), and (ii) at the end of the loop, the expression $n - x + y$ is always zero (and hence can be used as a vanishing rewrite function). Consider computing $\mathbf{A}_{x-y < n} \diamond_V \mathbf{B}$, where \mathbf{A} and \mathbf{B} summarize the property of the computations starting from line 3 and line 10 of the program, respectively. Before performing the max operator, we can add $c_1 \cdot (n - x + y - 1)$ and $c_2 \cdot (n - x + y)$ to the upper bounds indicated by \mathbf{A} and \mathbf{B} , respectively, where $c_1 \geq 0$ and c_2 is an arbitrary number. For example, if \mathbf{A} implies $\mathbb{E}[t'] \leq t + 2n - 2x + 2y$ and \mathbf{B} implies $\mathbb{E}[t'] \leq t$, the direct maximum would yield $\mathbb{E}[t'] \leq t + 2n + 2y$, but we can add $2 \cdot (n - x + y)$ to the right-hand side of $\mathbb{E}[t'] \leq t$ and obtain the more precise upper bound $\mathbb{E}[t'] \leq t + 2n - 2x + 2y$. Using the technique of rewriting functions, we can again reduce the problem of computing $\varphi \diamond_V$ to LP. In practice, we assume that each $\text{cond}[\varphi](\cdot, \cdot)$ is associated with two sets of rewriting functions $\Gamma_\varphi, \Gamma_{\neg\varphi} \subseteq \mathbb{R}[\vec{x}]$ of linear expressions, in the sense that the expressions in Γ_φ (resp., $\Gamma_{\neg\varphi}$) are guaranteed to be always nonnegative if the conditional choice takes the “then” (resp., “else”) branch.⁷ Our expected-value analysis derives the following expectation invariants for the random-walk program: $\mathbb{E}[t'] \leq t + 2n, \mathbb{E}[x'] \leq \frac{3}{2}n, \mathbb{E}[y'] \leq \frac{1}{2}n$.

Our implementation of the abstract domain for linear expected-value analysis consists of about 500 lines of code. To evaluate its effectiveness, we collected 22 benchmark programs—13 of which come from prior work [Wang et al. 2018]. Tab. 2 presents selected evaluation results (the full table is included in Appendix F), including the derived expectation invariants and the time taken by the implementation. Note that the domain used here captures a more restrictive class of invariants than the one used in prior work [Wang et al. 2018]. Thus, a comparison of running times is not informative. Nevertheless, the analysis created via NPA-PMA still efficiently obtains non-trivial results. For this domain, Newton’s method is slower than Kleene iteration on most of the benchmarks; however, the benchmark programs are quite small and only 8 of the 22 programs involve recursion.

5.3 Non-Linear Expectation Invariants via Expectation-Recurrence Analysis

Prior work [Bartocci et al. 2019; Bouissou et al. 2016] has studied program analysis of probabilistic programs where the control flow of a program does *not* depend on randomness; i.e., the program does not branch on the values of random variables. Such probabilistic programs are not uncommon in applications such as robotics and control theory [Bouissou et al. 2016]. Fig. 12(a) shows a probabilistic program that models a simple lane-keeping controller: variable y tracks the distance between the controlled vehicle and the middle of the lane; during each iteration, the vehicle changes its position with respect to an angle stored in variable x , and then updates the angle by a random amount (see the random variable z). Expectation invariants that involve $\mathbb{E}[y']$ can be used to derive

⁷A vanishing rewrite function t can be represented by two nonnegativity rewrite functions t and $-t$.

tail bounds on the final position y' of the vehicle via *concentration-of-measure* inequalities [Dubhashi and Panconesi 2009], such as the Markov inequality.

When the control flow of analyzed programs is not probabilistic, we can categorize program variables into deterministic ones Var_{det} and random ones Var_{rnd} . Randomness can only flow to variables in Var_{rnd} , and branch conditions can only depend on variables in Var_{det} . We then apply the abstract domain of Compositional Recurrence Analysis (CRA) [Farzan and Kincaid 2015; Kincaid et al. 2018]—reviewed below—to analyze expectation recurrences.

Transition-formula analysis. We consider *transition formulas* over integer arithmetic. For the program in Fig. 5(a), a transition formula $\varphi(\text{Var}, \text{Var}')$ is a formula over the variables $\text{Var} \triangleq \{i, x, y\}$ and their primed copies $\text{Var}' \triangleq \{i', x', y'\}$, representing the values of the variables in the pre- and post-state of an execution.

For example, the transition formula of $[i := i - 1]$ is $i' = i - 1 \wedge x' = x \wedge y' = y$. The sequencing operation is defined as relational composition and choice is defined as disjunction: $\varphi_1 \otimes \varphi_2 \triangleq \exists \text{Var}'' : \varphi_1(\text{Var}, \text{Var}'') \wedge \varphi_2(\text{Var}'', \text{Var}')$, $\varphi_1 \oplus \varphi_2 \triangleq \varphi_1 \vee \varphi_2$.

The key innovation of CRA is a method for interpreting the iteration operation $^\circledast$ in the transition-formula domain. Given a formula φ_{body} that encodes the property of a loop body, CRA computes a formula $\varphi_{\text{body}}^\circledast$ that over-approximates any number of iterations of φ_{body} , by extracting recurrence relations from φ_{body} and computing their closed forms. For the program in Fig. 5(a), CRA first summarizes the loop body by the following transition formula:

$$\varphi_{\text{body}} : i > 0 \wedge ((x' = x + i \wedge y' = y) \vee (x' = x \wedge y' = y + i)) \wedge i' = i - 1.$$

CRA then extracts recurrences from φ_{body} and computes their closed forms; some instances are presented below, where $i^{(k)}$ denotes the value of i at the end of the k^{th} iteration of the loop.

Recurrence	Closed form	Recurrence	Closed form
$i' = i - 1$	$i^{(k)} = i^{(0)} - k$	$x' + y' = x + y + i$	$x^{(k)} + y^{(k)} = x^{(0)} + y^{(0)} + k(k - 1)/2 + ki^{(0)}$

Finally, CRA introduces an existentially quantified nonnegative variable k as the number of iterations and conjoins the closed forms to obtain a summary of the loop:

$$\varphi_{\text{body}}^\circledast : \exists k : k \geq 0 \wedge i' = i - k \wedge x' + y' = x + y + k(k - 1)/2 + ki.$$

Expectation-recurrence analysis. The abstract domain consists of transition formulas of the form $\varphi(\text{Var}_{\text{det}} \cup \text{Var}_{\text{rnd}}, \text{Var}'_{\text{det}} \cup \mathbb{E}[\text{Var}'_{\text{rnd}}])$, which describe non-probabilistic invariants for Var_{det} and expectation invariants for Var_{rnd} . For example, the transition formula of $[z \sim \text{UNIF}(-0.1, 0.1)]$ is $i' = i \wedge n' = n \wedge \mathbb{E}[x'] = x \wedge \mathbb{E}[y'] = y \wedge \mathbb{E}[z'] = 0$. Let $\langle I, \oplus_I, \otimes_I, \circledast_I, \ominus_I, \underline{0}_I, \underline{1}_I \rangle$ be the regular algebra for CRA where I is the set of transition formulas for both non-probabilistic and expectation invariants.

We define a PMA $\mathcal{I} = \langle I, \sqsubseteq_I, \otimes_I, \diamond_I, \rho \oplus_I, \cup_I, \underline{0}_I, \underline{1}_I \rangle$ with the following definitions of operations:

$$\varphi_1 \sqsubseteq_I \varphi_2 \triangleq (\varphi_1 \implies \varphi_2), \quad \varphi_1 \diamond_I \varphi_2 \triangleq (\psi \otimes_I \varphi_1) \oplus_I (\neg \psi \otimes_I \varphi_2), \quad \varphi_1 \cup_I \varphi_2 \triangleq \varphi_1 \oplus_I \varphi_2,$$

where we do not define $\rho \oplus_I$ because we assume that the control flow of programs is not random.

We now describe a loop-solving strategy solve for this analysis. Consider $\text{solve}_{\mathcal{K}}(\{X = E_X\}_{X \in \mathcal{X}}, \gamma)$ where E_X 's are regular infinite-tree expressions in $\text{RegExp}^{\text{cf}}(\mathcal{F}_{\text{intra}}, \mathcal{K} \uplus \mathcal{X})$. Because the ranked alphabet $\mathcal{F}_{\text{intra}}$ for intraprocedural analysis does not contain procedure-call symbols, the equation system $\{X = E_X\}_{X \in \mathcal{X}}$ can be interpreted as a system of *regular* equations over the algebra $\langle I, \oplus_I, \otimes_I, \underline{0}_I, \underline{1}_I \rangle$. Thus, we can use the approach for algebraic program analysis of non-probabilistic programs (which we reviewed in §3.1), i.e., apply Tarjan's path-expression algorithm [Tarjan 1981b]

```

assume( $n > 0$ );  $i := 0$ ;
while  $i < n$  do
   $y := y + 0.1 * x$ ;
   $z \sim \text{UNIF}(-0.1, 0.1)$ ;
   $x := 0.8 * x + z$ ;
   $i := i + 1$  od

```

(a) A lane-keeping model

```

assume( $n > 0$ );  $(i, x, y, z) := (0, 1, 1, 1)$ ;
while  $i < n$  do
   $x \sim \text{CAT}((x + 1, 0.5), (x + 2, 0.5))$ ;
   $y \sim \text{CAT}((y + z + x * x, \frac{1}{3}), (y - z - x, \frac{2}{3}))$ ;
   $z \sim \text{CAT}((z + y, \frac{1}{4}), (z - y, \frac{3}{4}))$ ;
   $i := i + 1$  od

```

(b) A program with non-linear arithmetic

Fig. 12. Example programs.

Table 3. Selected results from expectation-recurrence analysis. (Time is in seconds.)

#	Program	Time (our work / POLAR)	Invariants (selected) derived by our work
1	50coinflips	108.920 / 0.448	$\mathbb{E}[r0'] = 1 - (1/2)^n, \mathbb{E}[total'] = 50(1 - (1/2)^{n-1})$
3	dbn-component-health	0.109 / 0.181	$\mathbb{E}[obs'] = (2683/1000) \cdot (9/100)^n + 701/1000$
8	hermann3	T/O / 0.294	-
9	las-vegas-search	0.059 / 0.288	$\mathbb{E}[found'] = 1 - (20/21)^n, \mathbb{E}[attempts'] = 20n/21$
11	randomized-response	0.093 / 0.168	$\mathbb{E}[n1'] = 5n/4 - pn, \mathbb{E}[p1'] = n/4 + pn/2, \mathbb{E}[ntrue'] = 3n/4$
12	rock-paper-scissors	0.110 / 0.215	$\mathbb{E}[p1bal'] = n(p2(-q2 - 2q3 + 1) + p3(q2 - q3) + q3)$

to solve the regular equations. We do not describe a linear-recursion-solving strategy because it is complex and not central to this article. Interprocedural CRA is developed by Kincaid et al. [2017].

For the lane-keeping program from Fig. 12(a), our analysis obtains the following non-linear expectation invariants: $\mathbb{E}[z'] = 0, \mathbb{E}[x'] = (\frac{4}{5})^n \cdot x, \mathbb{E}[y'] = y + \frac{1}{2} \cdot x - \frac{1}{2} \cdot (\frac{4}{5})^n \cdot x$. Fig. 12(b) presents another example program where $CAT(\cdot)$ stands for categorical distributions. Our analysis derives $\mathbb{E}[x'] = \frac{3}{2} \cdot n + 1, \mathbb{E}[y'] = -9 \cdot n - \frac{37}{5} \cdot (\frac{2}{3})^n + \frac{67}{5} \cdot (\frac{3}{2})^n - 5, \mathbb{E}[z'] = \frac{9}{4} \cdot n^2 + \frac{19}{4} \cdot n - \frac{201}{10} \cdot (\frac{3}{2})^n - \frac{37}{5} \cdot (\frac{2}{3})^n + \frac{57}{2}$.

Our implementation of the abstract domain for expectation-recurrence analysis consists of about 200 lines of code (excluding the original CRA implementation). To evaluate the effectiveness of our expectation-recurrence analysis, we collected 14 benchmark programs—12 of which come from prior work on POLAR [Moosbrugger et al. 2022].⁸ Tab. 3 presents the results of the evaluation, including the derived expectation invariants and the time taken by the implementation. The analysis instantiated via NPA-PMA is capable of deriving complex non-linear expectation recurrences and achieves comparable performance with POLAR on many benchmark programs. Another take-away is that NPA-PMA is flexible enough to allow techniques originally developed for analyzing non-probabilistic programs to be extended to probabilistic programs.

6 RELATED WORK

Automata techniques. The theory of regular word languages has been lifted to languages that can contain infinite words, and several different kinds of automata have been defined [Pin and Perrin 2004]. Zhu and Kincaid [2021] observed that the capability of describing sets of infinite words makes ω -regular expressions suitable for reasoning about program *termination*, and proposed an algebraic termination analysis (for non-probabilistic programs) that uses ω -regular path expressions to represent sets of infinite execution paths in a control-flow graph. In our work, we have to consider multiple kinds of confluences and thus develop a theory of trees rather than words.

Bayesian inference of Boolean programs (§5.1). Claret et al. [2013] developed an (intraprocedural) dataflow analysis for computing posterior distributions. Holtzen et al. [2020] proposed a technique that reduces Bayesian inference to weighted model counting. There has been a line of work on inference in probabilistic logics and logic programs [Cleaveland et al. 2005; De Raedt et al. 2007; Fierens et al. 2015; Riguzzi and Swift 2011]. Probabilistic model checking [Dehnert et al. 2017; Kwiatkowska et al. 2011] can also be used to perform Bayesian inference of finite-state programs. It might be possible to obtain more efficient Newtonian analyzers for Boolean probabilistic programs by formulating the aforementioned techniques in the framework of NPA-PMA.

Expectation-invariant analysis (§5.2 & §5.3). There has been a line of work on generating expectation invariants (sometimes called stochastic invariants or probabilistic invariants), e.g. [Barthe et al. 2016; Bartocci et al. 2019; Chakarov and Sankaranarayanan 2013, 2014; Chatterjee et al. 2017; Katoen et al. 2010; Schreuder and Ong 2019; Wang et al. 2018]. One observation is that for a certain class of probabilistic programs, we can apply existing techniques of generating invariants for non-probabilistic programs to discover expectation invariants. Our future research would investigate using non-probabilistic recurrence-solving techniques (e.g., [Breck et al. 2020; Farzan and Kincaid 2015; Kincaid et al. 2017, 2019, 2018]) to derive higher-moment invariants of probabilistic programs.

⁸We excluded a few benchmark programs from POLAR, each of which involves non-trivial stochastic control flow.

7 CONCLUSION

In this article, we have developed NPA-PMA, an interprocedural dataflow-analysis framework for designing and implementing partially non-iterative program analyses of probabilistic programs. NPA-PMA features a new family of ω -continuous pre-Markov algebras for formulating abstract domains, an adoption of regular infinite-tree expressions for encoding unstructured control-flow, and a new differentiation method for linearizing interprocedural equation systems. We have instantiated NPA-PMA with four domains (two existing and two new) and demonstrated that NPA-PMA is both efficient and general. In the future, we plan to develop more advanced instantiations of NPA-PMA to carry out more powerful program analyses. We are also looking into adapting variants of Newton's method from numerical analysis to program analysis.

ACKNOWLEDGMENTS

REFERENCES

- Martin Avanzini, Georg Moser, and Michael Schaper. 2020. A Modular Cost Analysis for Probabilistic Programs. *Proc. ACM Program. Lang.* 4, 172 (November 2020). Issue OOPSLA. <https://doi.org/10.1145/3428240>
- R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. 1997. Algebraic Decision Diagrams and their Applications. *Formal Methods in System Design* 10 (April 1997). <https://doi.org/10.1023/A:1008699807402>
- Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. 2016. Synthesizing Probabilistic Invariants via Doob's Decomposition. In *Computer Aided Verif. (CAV'16)*. https://doi.org/10.1007/978-3-319-41528-4_3
- Ezio Bartocci, Laura Kovács, and Miroslav Stanković. 2019. Automatic Generation of Moment-Based Invariants for Prob-Solvable Loops. In *Automated Tech. for Verif. and Analysis (ATVA'19)*. https://doi.org/10.1007/978-3-030-31784-3_15
- Hans Bekić. 1984. Definable Operations in General Algebras, and the Theory of Automata and Flowcharts. In *Programming Languages and Their Definition*. Springer, 30–55. <https://doi.org/10.1007/BFb0048939>
- Ahmed Bouajjani, Javier Esparza, and Tayssir Touili. 2003. A Generic Approach to the Static Analysis of Concurrent Programs with Procedures. In *Princ. of Prog. Lang. (POPL'03)*. <https://doi.org/10.1145/604131.604137>
- Olivier Bouissou, Eric Goubault, Sylvie Putot, Aleksandar Chakarov, and Sriram Sankaranarayanan. 2016. Uncertainty Propagation Using Probabilistic Affine Forms and Concentration of Measure Inequalities. In *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'16)*. https://doi.org/10.1007/978-3-662-49674-9_13
- Jason Breck, John Cyphert, Zachary Kincaid, and Thomas Reps. 2020. Templates and Recurrences: Better Together. In *Prog. Lang. Design and Impl. (PLDI'20)*. 688–702. <https://doi.org/10.1145/3385412.3386035>
- Quentin Carboneaux, Jan Hoffmann, Thomas Reps, and Zhong Shao. 2017. Automated Resource Analysis with Coq Proof Objects. In *Computer Aided Verif. (CAV'17)*. https://doi.org/10.1007/978-3-319-63390-9_4
- Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verif. (CAV'13)*. 511–526. https://doi.org/10.1007/978-3-642-39799-8_34
- Aleksandar Chakarov and Sriram Sankaranarayanan. 2014. Expectation Invariants for Probabilistic Program Loops as Fixed Points. In *Static Analysis Symp. (SAS'14)*. 85–100. https://doi.org/10.1007/978-3-319-10936-7_6
- Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. 2017. Stochastic Invariants for Probabilistic Termination. In *Princ. of Prog. Lang. (POPL'17)*. 145–160. <https://doi.org/10.1145/3093333.3009873>
- Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon, and Johannes Borgström. 2013. Bayesian Inference using Data Flow Analysis. In *Found. of Softw. Eng. (FSE'13)*. 92–102. <https://doi.org/10.1145/2491411.2491423>
- Rance Cleaveland, S. Purushothaman Iyer, and Murali Narasimha. 2005. Probabilistic temporal logics via the modal mu-calculus. *Theor. Comp. Sci.* 342 (September 2005), 316–350. Issue 2–3. <https://doi.org/10.1016/j.tcs.2005.03.048>
- Clp team. 2022. COIN-OR Linear Programming Solver. Available on <https://projects.coin-or.org/Clp>.
- Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. 2007. Tree Automata Techniques and Applications. Available on <http://www.grappa.univ-lille3.fr/tata>.
- Bruno Courcelle. 1983. Fundamental Properties of Infinite Trees. *Theor. Comp. Sci.* 25 (March 1983), 95–169. Issue 2. [https://doi.org/10.1016/0304-3975\(83\)90059-2](https://doi.org/10.1016/0304-3975(83)90059-2)
- Bruno Courcelle. 1990. CHAPTER 9 - Recursive Applicative Program Schemes. In *Formal Models and Semantics*. Elsevier, 461–492. <https://doi.org/10.1016/B978-0-444-88074-1.50014-7>
- Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Princ. of Prog. Lang. (POPL'77)*. <https://doi.org/10.1145/512950.512973>
- Patrick Cousot and Michael Monerau. 2012. Probabilistic Abstract Interpretation. In *European Symp. on Programming (ESOP'12)*. 169–193. https://doi.org/10.1007/978-3-642-28869-2_9

- Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. ProbLog: A Probabilistic Prolog and its Application in Link Discovery. In *Int. Joint Conf. on Artif. Intelligence (IJCAI'07)*. <https://dl.acm.org/doi/10.5555/1625275.1625673>
- Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. 2017. A Storm is Coming: A Modern Probabilistic Model Checker. In *Computer Aided Verif. (CAV'17)*. https://doi.org/10.1007/978-3-319-63390-9_31
- Devdatt P. Dubhashi and Alessandro Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press. <https://doi.org/10.1017/CBO9780511581274>
- Javier Esparza, Thomas Gawlitza, Stefan Kiefer, and Helmut Seidl. 2008a. Approximative Methods for Monotone Systems of Min-Max-Polynomial Equations. In *Int. Colloq. on Automata, Langs., and Programming (ICALP'08)*. https://doi.org/10.1007/978-3-540-70575-8_57
- Javier Esparza, Stefan Kiefer, and Michael Luttenberger. 2008b. Newton's Method for ω -Continuous Semirings. In *Int. Colloq. on Automata, Langs., and Programming (ICALP'08)*. https://doi.org/10.1007/978-3-540-70583-3_2
- Javier Esparza, Stefan Kiefer, and Michael Luttenberger. 2010. Newtonian Program Analysis. *J. ACM* 57, 33 (October 2010). Issue 6. <https://doi.org/10.1145/1857914.1857917>
- Javier Esparza, Antonín Kučera, and Richard Mayr. 2004. Model Checking Probabilistic Pushdown Automata. In *Logic in Computer Science (LICS'04)*. <https://doi.org/10.1109/LICS.2004.1319596>
- Javier Esparza, Antonín Kučera, and Richard Mayr. 2005. Quantitative Analysis of Probabilistic Pushdown Automata: Expectations and Variances. In *Logic in Computer Science (LICS'05)*. <https://doi.org/10.1109/LICS.2005.39>
- Kousha Etessami, Dominik Wojtczak, and Mihalis Yannakakis. 2008. Recursive Stochastic Games with Positive Rewards. In *Int. Colloq. on Automata, Langs., and Programming (ICALP'08)*. https://doi.org/10.1007/978-3-540-70575-8_58
- Kousha Etessami and Mihalis Yannakakis. 2005. Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations. In *Symp. on Theor. Aspects of Comp. Sci. (STACS'05)*. https://doi.org/10.1007/978-3-540-31856-9_28
- Kousha Etessami and Mihalis Yannakakis. 2015. Recursive Markov Decision Processes and Recursive Stochastic Games. *J. ACM* 62, 11 (May 2015). Issue 2. <https://doi.org/10.1145/2699431>
- Azadeh Farzan and Zachary Kincaid. 2013. An Algebraic Framework For Compositional Program Analysis. <https://arxiv.org/abs/1310.3481>
- Azadeh Farzan and Zachary Kincaid. 2015. Compositional Recurrence Analysis. In *Formal Methods in Computer-Aided Design (FMCAD'15)*. <https://doi.org/10.1109/FMCAD.2015.7542253>
- Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. 2015. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming* 15 (May 2015). Issue 3. <https://doi.org/10.1017/S1471068414000076>
- Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. *Proc. ACM Program. Lang.* 4, 140 (November 2020). Issue OOPSLA. <https://doi.org/10.1145/3428208>
- Jean-Baptiste Jeannin, Dexter Kozen, and Alexandra Silva. 2017. CoCaml: Functional Programming with Regular Coinductive Types. *Fundamenta Informaticae* 150 (March 2017). Issue 3-4. <https://doi.org/10.3233/FI-2017-1473>
- Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. In *European Symp. on Programming (ESOP'16)*. 364–389. https://doi.org/10.1007/978-3-662-49498-1_15
- Joost-Pieter Katoen, Annabelle K. McIver, Larissa A. Meinicke, and Carroll C. Morgan. 2010. Linear-Invariant Generation for Probabilistic Programs: Automated Support for Proof-Based Methods. In *Static Analysis Symp. (SAS'10)*. 390–406. https://doi.org/10.1007/978-3-642-15769-1_24
- Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas Reps. 2017. Compositional Recurrence Analysis Revisited. In *Prog. Lang. Design and Impl. (PLDI'17)*. 248–262. <https://doi.org/10.1145/3062341.3062373>
- Zachary Kincaid, Jason Breck, John Cyphert, and Thomas Reps. 2019. Closed Forms for Numerical Loops. *Proc. ACM Program. Lang.* 3, 55 (January 2019). Issue POPL. <https://doi.org/10.1145/3290368>
- Zachary Kincaid, John Cyphert, Jason Breck, and Thomas Reps. 2018. Non-Linear Reasoning for Invariant Synthesis. *Proc. ACM Program. Lang.* 2, 54 (January 2018). Issue POPL. <https://doi.org/10.1145/3158142>
- Zachary Kincaid, Thomas Reps, and John Cyphert. 2021. Algebraic Program Analysis. In *Computer Aided Verif. (CAV'21)*. 46–83. https://doi.org/10.1007/978-3-030-81685-8_3
- Dexter Kozen and Alexandra Silva. 2017. Practical coinduction. *Math. Struct. Comp. Sci.* 27 (October 2017). Issue 7. <https://doi.org/10.1017/S09560129515000493>
- Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probability for Randomized Program Runtimes via Martingales for Higher Moments. In *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'19)*. https://doi.org/10.1007/978-3-030-17465-1_8
- Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verif. (CAV'11)*. https://doi.org/10.1007/978-3-642-22110-1_47
- Zhifei Li and Jason Eisner. 2009. First- and Second-Order Expectation Semirings with Applications to Minimum-Risk Training on Translation Forests. In *Empirical Methods in Natural Language Processing (EMNLP'09)*. <https://dl.acm.org/>

[doi/10.5555/1699510.1699517](https://doi.org/10.5555/1699510.1699517)

- Annabelle K. McIver and Carroll C. Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer Science+Business Media, Inc. <https://doi.org/10.1007/b138392>
- Marcel Moosbrugger, Miroslav Stankovič, Ezio Bartocci, and Laura Kovács. 2022. This Is the Moment for Probabilistic Loops. *Proc. ACM Program. Lang.* 6, 178 (October 2022). Issue OOPSLA2. <https://doi.org/10.1145/3563341>
- Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded Expectations: Resource Analysis for Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'18)*. 496–512. <https://doi.org/10.1145/3192366.3192394>
- Jean-Eric Pin and Dominique Perrin. 2004. *Infinite Words: Automata, Semigroups, Logic and Games*. Elsevier.
- Thomas Reps, Akash Lal, and Nick Kidd. 2007. Program Analysis Using Weighted Pushdown System. In *Found. of Soft. Tech. and Theor. Comput. Sci. (FSTTCS'07)*. https://doi.org/10.1007/978-3-540-77050-3_4
- Thomas Reps, Stefan Schwoon, and Somesh Jha. 2003. Weighted Pushdown Systems and their Application to Interprocedural Dataflow Analysis. In *Static Analysis Symp. (SAS'03)*. <https://dl.acm.org/doi/10.5555/1760267.1760283>
- Thomas Reps, Emma Turetsky, and Prathmesh Prabhu. 2016. Newtonian Program Analysis via Tensor Product. In *Princ. of Prog. Lang. (POPL'16)*. 663–677. <https://doi.org/10.1145/2837614.2837659>
- Fabrizio Riguzzi and Terrance Swift. 2011. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming* 11 (July 2011). Issue 4-5. <https://doi.org/10.1017/S147106841100010X>
- Anne Schreuder and Luke Ong. 2019. Polynomial Probabilistic Invariants and the Optional Stopping Theorem. <https://arxiv.org/abs/1910.12634>
- Robert Endre Tarjan. 1981a. A Unified Approach to Path Problems. *J. ACM* 28 (July 1981). Issue 3. <https://doi.org/10.1145/322261.322272>
- Robert Endre Tarjan. 1981b. Fast Algorithms for Solving Path Problems. *J. ACM* 28 (July 1981). Issue 3. <https://doi.org/10.1145/322261.322273>
- Di Wang, Jan Hoffmann, and Thomas Reps. 2018. PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'18)*. <https://doi.org/10.1145/3192366.3192408>
- Di Wang, Jan Hoffmann, and Thomas Reps. 2019b. A Denotational Semantics for Low-Level Probabilistic Programs with Nondeterminism. *Electr. Notes Theor. Comp. Sci.* 347 (November 2019). <https://doi.org/10.1016/j.entcs.2019.09.016>
Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics.
- Di Wang, Jan Hoffmann, and Thomas Reps. 2021. Central Moment Analysis for Cost Accumulators in Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'21)*. <https://doi.org/10.1145/3453483.3454062>
- Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019a. Cost Analysis of Nondeterministic Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'19)*. <https://doi.org/10.1145/3314221.3314581>
- Shaowei Zhu and Zachary Kincaid. 2021. Termination Analysis without the Tears. In *Prog. Lang. Design and Impl. (PLDI'21)*. <https://doi.org/10.1145/3453483.3454110>

A A THEORY OF REGULAR HYPER-PATHS

Our development in this section is inspired by the theory on regular tree languages and regular tree expressions [Comon et al. 2007]. However, we have to tackle a technical challenge that a hyper-path can correspond to a tree with infinite depth. Induction is a well-established proof principle for reasoning about inductively defined datatypes (e.g., finite lists and finite trees), but does not work for infinite datatypes in general. Thus, in this section, we rely on a principle of *coinduction*, which, as shown in prior work (e.g. [Jeannin et al. 2017; Kozen and Silva 2017]), is indeed a well-founded principle for reasoning about coinductive datatypes (e.g., infinite streams and infinite trees).

A.1 Possibly-infinite Trees

A *ranked alphabet* is a pair $(\mathcal{F}, \text{Arity})$ where \mathcal{F} is a nonempty set and Arity is a mapping from \mathcal{F} to \mathbb{N} . The *arity* of a symbol $f \in \mathcal{F}$ is $\text{Arity}(f)$. The set of symbols of arity n is denoted by \mathcal{F}_n . For simplicity, we use parentheses and commas to specify symbols with their arity; for example, $f(,)$ specifies a binary symbol f .

Let \mathcal{K} be a set of constants (i.e., symbols with arity zero) called *holes*. We assume that the sets \mathcal{K} and \mathcal{F}_0 are disjoint, and there is a distinguished symbol $\cup \notin \mathcal{F}_0 \cup \mathcal{K}$ with arity zero. Intuitively, a hole symbol in \mathcal{K} represents a placeholder for later substitution with trees, and the \cup symbol indicates a “yet unknown subtree.” A *possibly-infinite tree* $t \in \text{Tree}^\infty(\mathcal{F}, \mathcal{K})$ over the ranked alphabet \mathcal{F} and the set of variables \mathcal{K} is a partial map $t : \mathbb{N}_{>0}^* \rightarrow \mathcal{F} \cup \mathcal{K} \cup \{\cup\}$ with domain $\text{dom}(t) \subseteq \mathbb{N}_{>0}^*$ satisfying the following properties:

- $\text{dom}(t)$ is non-empty and prefix-closed (thus $\epsilon \in \text{dom}(t)$);
- for any $p \in \text{dom}(t)$, if $t(p) \in \mathcal{F}_n$ for some $n > 0$, then $\{j \mid pj \in \text{dom}(t)\} = \{1, \dots, n\}$;
- for any $p \in \text{dom}(t)$, if $t(p) \in \mathcal{F}_0 \cup \mathcal{K} \cup \{\cup\}$, then $\{j \mid pj \in \text{dom}(t)\} = \emptyset$.

For brevity, we denote by a the finite tree $\{\epsilon \mapsto a\}$, for any $a \in \mathcal{F}_0 \cup \mathcal{K} \cup \{\cup\}$. We also denote by $f(s_1, \dots, s_n)$ the possibly-infinite tree

$$\{\epsilon \mapsto f\} \cup \bigcup_{j \in \{1, \dots, n\}} \{jp \mapsto s_j(p) \mid p \in \text{dom}(s_j)\},$$

for any $f \in \mathcal{F}_n$ with $n > 0$ where s_1, \dots, s_n are possibly-infinite trees.

Let $t \in \text{Tree}^\infty(\mathcal{F}, \mathcal{K})$ be a possibly-infinite tree. Every element in $\text{dom}(t)$ is called a *position*. A *leaf position* is a position p such that $pj \notin \text{dom}(t)$ for any $j \in \mathbb{N}$. We denote by $\text{root}(t)$ the *root symbol* of t , defined by $\text{root}(t) \triangleq t(\epsilon)$. A *subtree* $t|_p$ of a tree t at position p is the tree defined by the following:

- $\text{dom}(t|_p) = \{q \mid pq \in \text{dom}(t)\}$, and
- $\forall q \in \text{dom}(t|_p) : t|_p(q) = t(pq)$.

A tree t can then be decomposed as $f(t|_1, \dots, t|_n)$, where $f = \text{root}(t)$ with arity n .

To compare two possibly-infinite trees, we define a *refinement relation* $\sqsubseteq_{\mathcal{T}} \subseteq \text{Tree}^\infty(\mathcal{F}, \mathcal{K}) \times \text{Tree}^\infty(\mathcal{F}, \mathcal{K})$ as the *maximum* binary relation satisfying the following property: if $t_1 \sqsubseteq_{\mathcal{T}} t_2$, then either (i) $t_1 = \cup$, or (ii) $\text{root}(t_1) = f$ and $\text{root}(t_2) = f$ for some $f \in \mathcal{F} \cup \mathcal{K}$, and for all $i \in \{1, \dots, \text{Arity}(f)\}$, it holds that $t_1|_i \sqsubseteq_{\mathcal{T}} t_2|_i$. This property formulates the idea that the symbol \cup indicates a “yet unknown subtree.” Because the relation $\sqsubseteq_{\mathcal{T}}$ is maximum, it can be characterized by the *greatest* fixed point of the operator below.

$$T_{\sqsubseteq_{\mathcal{T}}}(R) \triangleq \{\langle \cup, t \rangle \mid t \in \text{Tree}^\infty(\mathcal{F}, \mathcal{K})\} \cup \{\langle f(s_1, \dots, s_n), f(t_1, \dots, t_n) \rangle \mid f \in \mathcal{F} \cup \mathcal{K}, \text{Arity}(f) = n, \forall j \in \{1, \dots, n\} : \langle s_j, t_j \rangle \in R\}.$$

LEMMA A.1. *The relation $\sqsubseteq_{\mathcal{T}}$ is a partial order on $\text{Tree}^\infty(\mathcal{F}, \mathcal{K})$.*

PROOF. The relation $\sqsubseteq_{\mathcal{T}}$ is reflexive because $\{\langle t, t \rangle \mid t \in \text{Tree}^\infty(\mathcal{F}, \mathcal{K})\}$ satisfies properties of the refinement relation, and the relation $\sqsubseteq_{\mathcal{T}}$ is maximum by definition.

We claim that $\sqsubseteq_{\mathcal{T}}$ is antisymmetric, i.e., if $t_1 \sqsubseteq_{\mathcal{T}} t_2$ and $t_2 \sqsubseteq_{\mathcal{T}} t_1$, then $t_1 = t_2$. If $\text{root}(t_1) = \cup$, then by $t_2 \sqsubseteq_{\mathcal{T}} t_1$ we also have $\text{root}(t_2) = \cup$, thus $t_1 = t_2$. Otherwise, we know that $\text{root}(t_1) = f$ for some $f \in \mathcal{F} \cup \mathcal{K}$, then by $t_1 \sqsubseteq_{\mathcal{T}} t_2$ we have $\text{root}(t_2) = f$, and for all $i \in \{1, \dots, \text{Arity}(f)\}$, it holds that $t_1|_i \sqsubseteq_{\mathcal{T}} t_2|_i$. By $t_2 \sqsubseteq_{\mathcal{T}} t_1$, we also have $t_2|_i \sqsubseteq_{\mathcal{T}} t_1|_i$ for all $i \in \{1, \dots, \text{Arity}(f)\}$. Therefore, by coinduction hypothesis⁹, we have $t_1|_i = t_2|_i$ for all $i \in \{1, \dots, \text{Arity}(f)\}$, thus $t_1 = t_2$.

We claim that $\sqsubseteq_{\mathcal{T}}$ is transitive, i.e., if $t_1 \sqsubseteq_{\mathcal{T}} t_2$ and $t_2 \sqsubseteq_{\mathcal{T}} t_3$, then $t_1 \sqsubseteq_{\mathcal{T}} t_3$. If $\text{root}(t_1) = \cup$, then we have $t_1 \sqsubseteq_{\mathcal{T}} t_3$ by definition. Otherwise, we know that $\text{root}(t_1) = f$ for some $f \in \mathcal{F} \cup \mathcal{K}$. By $t_1 \sqsubseteq_{\mathcal{T}} t_2$, we know that $\text{root}(t_2) = f$ and for all $i \in \{1, \dots, \text{Arity}(f)\}$, it holds that $t_1|_i \sqsubseteq_{\mathcal{T}} t_2|_i$. By $t_2 \sqsubseteq_{\mathcal{T}} t_3$, we know that $\text{root}(t_3) = f$ and for all $i \in \{1, \dots, \text{Arity}(f)\}$, it holds that $t_2|_i \sqsubseteq_{\mathcal{T}} t_3|_i$. Thus, by coinduction hypothesis, for all $i \in \{1, \dots, \text{Arity}(f)\}$, it holds that $t_1|_i \sqsubseteq_{\mathcal{T}} t_3|_i$. Therefore, by definition, we conclude that $t_1 \sqsubseteq_{\mathcal{T}} t_3$. \square

The lemma below justifies the relation $\sqsubseteq_{\mathcal{T}}$ in the sense that if $t_1 \sqsubseteq_{\mathcal{T}} t_2$, it must hold that t_2 is obtained by substituting some \cup symbols in t_1 with trees.

LEMMA A.2. *If $t_1 \sqsubseteq_{\mathcal{T}} t_2$, then $\text{dom}(t_1) \subseteq \text{dom}(t_2)$ and for all $p \in \text{dom}(t_1)$, $t_1(p) = \cup$ or $t_1(p) = t_2(p)$.*

PROOF. If $\text{root}(t_1) = \cup$, then $\text{dom}(t_1) = \{\epsilon\}$ and the lemma holds obviously. Otherwise, we know that $\text{root}(t_1) = f$ for some $f \in \mathcal{F} \cup \mathcal{K}$, and by $t_1 \sqsubseteq_{\mathcal{T}} t_2$, we have $\text{root}(t_2) = f$ and for all $i \in \{1, \dots, \text{Arity}(f)\}$, it holds that $t_1|_i \sqsubseteq_{\mathcal{T}} t_2|_i$. By coinduction hypothesis, for all $i \in \{1, \dots, \text{Arity}(f)\}$, we know that the lemma holds for $t_1|_i \sqsubseteq_{\mathcal{T}} t_2|_i$. Thus, we have

$$\begin{aligned} \text{dom}(t_1) &= \bigcup_{i \in \{1, \dots, \text{Arity}(f)\}} \{ip \mid p \in \text{dom}(t_1|_i)\} \\ &\subseteq \bigcup_{i \in \{1, \dots, \text{Arity}(f)\}} \{ip \mid p \in \text{dom}(t_2|_i)\} \\ &= \text{dom}(t_2). \end{aligned}$$

For any $p \in \text{dom}(t_1)$, either $p = \epsilon$, or there exists i such that $p = ip'$ and $p' \in \text{dom}(t_1|_i)$. In the former case, we have $t_1(\epsilon) = f = t_2(\epsilon)$. In the latter case, we know that either $t_1|_i(p') = \cup$ or $t_1|_i(p') = t_2|_i(p')$. Therefore, we conclude that either $t_1(p) = t_1(ip') = t_1|_i(p') = \cup$, or $t_1(p) = t_1(ip') = t_1|_i(p') = t_2|_i(p') = t_2(ip') = t_2(p)$. \square

The set $\text{Tree}^\omega(\mathcal{F}, \mathcal{K})$ of possibly-infinite trees forms an ω -cpo with respect to $\sqsubseteq_{\mathcal{T}}$.

LEMMA A.3. *The relation $\sqsubseteq_{\mathcal{T}}$ is an ω -continuous partial order with \cup as the least element.*

PROOF. Fix an ω -chain $\{t_k\}_{k \in \mathbb{N}}$ of possibly-infinite trees with respect to the partial order $\sqsubseteq_{\mathcal{T}}$ (Lem. A.1). We then try to construct the least upper bound of the chain, and call it t' . We set $\text{dom}(t')$ to be $\bigcup_{k \in \mathbb{N}} \text{dom}(t_k)$. By Lem. A.2, we know that $\{\text{dom}(t_k)\}_{k \in \mathbb{N}}$ is a \subseteq -chain. For a position $p \in \text{dom}(t')$, we set $t'(p)$ to be

- \cup , if there exists $K \in \mathbb{N}$ such that $t_k(p) = \cup$ for all $k \geq K$;
- $t_{k_p}(p)$, if there exists $k_p \in \mathbb{N}$ satisfying $p \in \text{dom}(t_{k_p})$ and $t_{k_p}(p) \neq \cup$.

The well-definedness of t' is guaranteed by Lem. A.2. If t' is an upper bound on $\{t_k\}_{k \in \mathbb{N}}$, then t' is the least one as $\text{dom}(t')$ is the least upper bound on $\{\text{dom}(t_k)\}_{k \in \mathbb{N}}$. Thus, it remains to prove that t' is indeed an upper bound.

We then proceed the proof by coinduction. For any $\ell \in \mathbb{N}$, if $\text{root}(t_\ell) = \cup$, then $t_\ell \sqsubseteq_{\mathcal{T}} t'$. Thus, without loss of generality, we can assume that $t_1 \neq \cup$. Let $f = \text{root}(t_1) \in \mathcal{F} \cup \mathcal{K}$. Thus, by definition, we know that $\text{root}(t_k) = f$ for all $k \in \mathbb{N}$, and also $\text{root}(t') = f$. Let $n = \text{Arity}(f)$. Then for each $j \in \{1, \dots, n\}$, it holds that $\{t_k|_j\}_{k \in \mathbb{N}}$ is an ω -chain, and by coinduction hypothesis, we know that $t'|_j$ is an upper bound on $\{t_k|_j\}_{k \in \mathbb{N}}$. Therefore, $f(t'|_1, \dots, t'|_n)$ is an upper bound of $\{f(t_k|_1, \dots, t_k|_n)\}_{k \in \mathbb{N}}$. \square

⁹As pointed out by [Kozen and Silva \[2017\]](#), we can use the coinduction hypothesis as long as there is progress in observing the roots of trees, and there is no further investigation of the children of the trees.

We now formulate a mechanism to define functions on possibly-infinite trees. Note that the standard definition of inductive functions does *not* work because of the existence of infinite trees. Let (D, \sqsubseteq_D) be an ω -cpo with a least element \perp_D . To define a function from possibly-infinite trees in $\text{Tree}^\omega(\mathcal{F}, \mathcal{K})$ to D , we require a mapping base : $\mathcal{F}_0 \cup \mathcal{K} \rightarrow D$ and a family of mappings into ω -continuous functions $\text{ind}_n : \mathcal{F}_n \rightarrow [D^n \rightarrow D]$ for all $n > 0$ such that $\mathcal{F}_n \neq \emptyset$. We then introduce an ω -chain of functions $\{h_i\}_{i \geq 0}$ as follows, and define the target function by $h \triangleq \bigsqcup_{i \geq 0} h_i$ with respect to the pointwise extension of \sqsubseteq_D .

$$h_0 \triangleq \lambda t. \perp_D,$$

$$h_{i+1} \triangleq \lambda t. \begin{cases} \perp_D & \text{if } \text{root}(t) = \cup \\ \text{base}(\text{root}(t)) & \text{if } \text{root}(t) \in \mathcal{F}_0 \cup \mathcal{K} \\ \text{ind}_n(\text{root}(t))(h_i(t|_1), \dots, h_i(t|_n)) & \text{if } \text{root}(t) \in \mathcal{F}_n \text{ for some } n > 0 \end{cases}.$$

LEMMA A.4. *Let $h : \text{Tree}^\omega(\mathcal{F}, \mathcal{K}) \rightarrow D$ be a function from possibly-infinite trees to an ω -cpo (D, \sqsubseteq_D) with a least element \perp_D . Then $h(\cup) = \perp_D$.*

PROOF. By definition, we know that there exists an ω -chain of functions $\{h_j\}_{j \geq 0}$ defined as above such that $h = \bigsqcup_{j \geq 0} h_j$. Also, it is obvious that for each $j \geq 0$, $h_j(\cup) = \perp_D$. Thus, we conclude that $h(\cup) = \bigsqcup_{j \geq 0} h_j(\cup) = \bigsqcup_{j \geq 0} \perp_D = \perp_D$. \square

LEMMA A.5. *Let $h : \text{Tree}^\omega(\mathcal{F}, \mathcal{K}) \rightarrow D$ be a function from possibly-infinite trees to an ω -cpo (D, \sqsubseteq_D) with a least element \perp_D , induced by mappings base and ind_n for $n > 0$. Then h is ω -continuous with respect to the pointwise extension of \sqsubseteq_D .*

PROOF. We know that there exists an ω -chain of functions $\{h_j\}_{j \geq 0}$ defined as above such that $h = \bigsqcup_{j \geq 0} h_j$. We claim that for each $j \geq 0$, the function h_j is ω -continuous. We proceed by induction on j .

When $j = 0$:

By definition, we have $h_0 \triangleq \lambda t. \perp$, which is obviously ω -continuous.

When $j = k + 1$:

By definition, we have

$$h_j \triangleq \lambda t. \begin{cases} \perp & \text{if } \text{root}(t) = \cup, \\ \text{base}(\text{root}(t)) & \text{if } \text{root}(t) \in \mathcal{F}_0 \cup \mathcal{K}, \\ \text{ind}_n(\text{root}(t))(h_k(t|_1), \dots, h_k(t|_n)) & \text{if } \text{root}(t) \in \mathcal{F}_n \text{ for some } n > 0. \end{cases}.$$

Consider an ω -chain of trees $\{t_i\}_{i \geq 0}$. Without loss of generality, we assume that $\text{root}(t_0) \neq \cup$ (because $h_j(\{\epsilon \mapsto \cup\}) = \perp$).

Then by the definition of the refinement order $\sqsubseteq_{\mathcal{T}}$, we know that $\{\text{root}(t_i) \mid i \geq 0\}$ is a singleton set.

If the singleton set contains a symbol $a \in \mathcal{F}_0 \cup \mathcal{K}$, we know that $h_j(t_i) = \text{base}(a)$ for all $i \geq 0$, thus $h_j(\bigsqcup_{i \geq 0} t_i) = h_j(a) = \bigsqcup_{i \geq 0} h_j(a) = \bigsqcup_{i \geq 0} h_j(t_i)$.

If the singleton set contains a symbol $f \in \mathcal{F}_n$ for some $n > 0$, then by the assumption that $\text{ind}_n(f)$ is ω -continuous and by induction hypothesis that h_k is ω -continuous, we can derive

$$\begin{aligned}
 h_j(\bigsqcup_{i \geq 0} t_i) &= \text{ind}_n(f)(h_k((\bigsqcup_{i \geq 0} t_i)|_1), \dots, h_k((\bigsqcup_{i \geq 0} t_i)|_n)) \\
 &= \text{ind}_n(f)(h_k(\bigsqcup_{i \geq 0} (t_i|_1)), \dots, h_k(\bigsqcup_{i \geq 0} (t_i|_n))) \\
 &= \text{ind}_n(f)(\bigsqcup_{i \geq 0} h_k(t_i|_1), \dots, \bigsqcup_{i \geq 0} h_k(t_i|_n)) \\
 &= \bigsqcup_{i_1 \geq 0} \dots \bigsqcup_{i_n \geq 0} \text{ind}_n(f)(h_k(t_{i_1}|_1), \dots, h_k(t_{i_n}|_n)) \\
 &\quad (\exists: \text{obvious}) \\
 &\quad (\sqsubseteq: \forall i_1, \dots, i_n, \text{ the LHS item is bounded by the } \max(i_1, \dots, i_n)\text{-th RHS item}) \\
 &= \bigsqcup_{i \geq 0} \text{ind}_n(f)(h_k(t_i|_1), \dots, h_k(t_i|_n)) \\
 &= \bigsqcup_{i \geq 0} h_j(t_i).
 \end{aligned}$$

Thus, we conclude the proof of the claim that h_j is ω -continuous for all $j \geq 0$.

Let us now consider an ω -chain of trees $\{t_i\}_{i \geq 0}$. Then we can conclude the proof by

$$\begin{aligned}
 h(\bigsqcup_{i \geq 0} t_i) &= (\bigsqcup_{j \geq 0} h_j)(\bigsqcup_{i \geq 0} t_i) \\
 &= \bigsqcup_{j \geq 0} h_j(\bigsqcup_{i \geq 0} t_i) \\
 &= \bigsqcup_{j \geq 0} \bigsqcup_{i \geq 0} h_j(t_i) \\
 &= \bigsqcup_{i \geq 0} \bigsqcup_{j \geq 0} h_j(t_i) \\
 &= \bigsqcup_{i \geq 0} (\bigsqcup_{j \geq 0} h_j)(t_i) \\
 &= \bigsqcup_{i \geq 0} h(t_i).
 \end{aligned}$$

□

We end this section by demonstrating a function that maps possibly-infinite trees to collections of *rooted paths* on trees. We define $\text{paths}(t) \subseteq \text{Path}^\omega(\mathcal{F}, \mathcal{K}) \triangleq (\mathcal{F} \cup \mathcal{K}) \cdot (\mathbb{N} \cdot (\mathcal{F} \cup \mathcal{K}))^\omega$ with the following base and induction steps:

$$\begin{aligned}
 \text{paths} : \text{Tree}^\omega(\mathcal{F}, \mathcal{K}) &\rightarrow \text{Path}^\omega(\mathcal{F}, \mathcal{K}) \\
 \text{paths}(a) &\triangleq \{a\} \quad \text{for } a \in \mathcal{F}_0 \cup \mathcal{K},
 \end{aligned}$$

$$\text{paths}(f(s_1, \dots, s_n)) \triangleq \{f j w \mid j = 1, \dots, n \wedge w \in \text{paths}(s_j)\} \quad \text{for } f \in \mathcal{F}_n \text{ for some } n > 0.$$

Note that the set $\text{Path}^\omega(\mathcal{F}, \mathcal{K})$ of possibly-infinite paths admits an ω -cpo with the ordering

$A \sqsubseteq_P B \triangleq (A \cap \text{Path}^+(\mathcal{F}, \mathcal{K}) \subseteq B \cap \text{Path}^+(\mathcal{F}, \mathcal{K})) \wedge (A \cap \text{Path}^\omega(\mathcal{F}, \mathcal{K}) \supseteq B \cap \text{Path}^\omega(\mathcal{F}, \mathcal{K}))$, and the least element $\perp_P \triangleq \text{Path}^\omega(\mathcal{F}, \mathcal{K})$, where $\text{Path}^+(\mathcal{F}, \mathcal{K}) \triangleq (\mathcal{F} \cup \mathcal{K}) \cdot (\mathbb{N} \cdot (\mathcal{F} \cup \mathcal{K}))^*$ is the set of finite paths, and $\text{Path}^\omega(\mathcal{F}, \mathcal{K}) \triangleq (\mathcal{F} \cup \mathcal{K}) \cdot (\mathbb{N} \cdot (\mathcal{F} \cup \mathcal{K}))^\omega$ is the set of infinite paths. We can again see that the \cup symbol indicates a “yet unknown tree,” because $\text{paths}(\cup)$, by Lem. A.4, equals \perp_P , i.e., the set of all infinite paths (and no finite paths).

A.2 Hyper-path Interpretation of Regular Hyper-path Expressions

To formally interpret regular hyper-path expressions as possibly-infinite trees, we first introduce a *substitution* operator, defined as a function on possibly-infinite trees with the following base and induction steps, where $-_1$ and $-_2$ indicate the two arguments of the substitution operator, and the function is defined coinductively on the first argument:

$$\begin{aligned}
 (-_1)\{\Box \rightsquigarrow -_2\} : \text{Tree}^\omega(\mathcal{F}, \mathcal{K} \cup \{\Box\}) \times \text{Tree}^\omega(\mathcal{F}, \mathcal{K}) &\rightarrow \text{Tree}^\omega(\mathcal{F}, \mathcal{K}) \\
 \Box\{\Box \rightsquigarrow t\} &\triangleq t, \\
 a\{\Box \rightsquigarrow t\} &\triangleq a \quad \text{for } a \neq \Box, \\
 f(s_1, \dots, s_n)\{\Box \rightsquigarrow t\} &\triangleq f(s_1\{\Box \rightsquigarrow t\}, \dots, s_n\{\Box \rightsquigarrow t\}).
 \end{aligned}$$

Example A.6. Let $\mathcal{F} = \{seq[x \sim \text{BER}(0.5)](\cdot), cond[x](\cdot, \epsilon)\}$ and $\mathcal{K} = \{\square_1, \square_2\}$. Let $t = cond[x](\square_1, \square_2)$ and $s = seq[x \sim \text{BER}(0.5)](\epsilon)$. Then

$$t\{\square_1 \rightsquigarrow s\} = \begin{array}{c} cond[x] \\ \swarrow \quad \searrow \\ seq[x \sim \text{BER}(0.5)] \quad \square_2 \\ \downarrow \quad \quad \quad \downarrow \\ \epsilon \quad \quad \quad \epsilon \end{array} ; \quad t\{\square_2 \rightsquigarrow s\} = \begin{array}{c} cond[x] \\ \swarrow \quad \searrow \\ \square_1 \quad seq[x \sim \text{BER}(0.5)] \\ \quad \quad \downarrow \\ \quad \quad \epsilon \end{array} .$$

PROPOSITION A.7. *The substitution operator is ω -continuous in its second argument.*

We now define an interpretation from regular hyper-path expressions to possibly-infinite trees via the mapping $\mathcal{T}_{\mathcal{K}}[\cdot]$, which is inductively defined on the structure of regular hyper-path expressions as follows, where “lfp” denotes the least-fixed-point operator.

$$\begin{aligned} \mathcal{T}_{\mathcal{K}}[\cdot] &: \text{RegExp}^\infty(\mathcal{F}, \mathcal{K}) \rightarrow \text{Tree}^\infty(\mathcal{F}, \mathcal{K}) \\ \mathcal{T}_{\mathcal{K}}[a] &\triangleq a, \\ \mathcal{T}_{\mathcal{K}}[f(E_1, \dots, E_n)] &\triangleq f(\mathcal{T}_{\mathcal{K}}[s_1], \dots, \mathcal{T}_{\mathcal{K}}[s_n]), \\ \mathcal{T}_{\mathcal{K}}[E_1 \cdot_\square E_2] &\triangleq \mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1]\{\square \rightsquigarrow \mathcal{T}_{\mathcal{K}}[E_2]\}, \\ \mathcal{T}_{\mathcal{K}}[E^{\infty\square}] &\triangleq \text{lfp}_{\subseteq}^{\mathcal{T}_{\mathcal{K} \cup \{\square\}}} \lambda t. (\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E]\{\square \rightsquigarrow t\}). \end{aligned}$$

Example A.8. The regular hyper-path expression $E \triangleq (cond[x](seq[x \sim \text{BER}(0.5)](\square, \epsilon))^{\infty\square}$ gives a finite representation of the infinite hyper-path of the loop “**while** $x \text{ do } x \sim \text{BER}(0.5) \text{ od}$.” Intuitively, the map $\mathcal{T}_{\mathcal{K}}[E]$ constructs the following ω -chain of trees and obtains the least upper bound as the interpretation of E :

$$\begin{array}{c} \bigcup \quad \subseteq_{\mathcal{T}} \quad \begin{array}{c} cond[x < 0] \\ \swarrow \quad \searrow \\ seq[x := x + 1] \quad \epsilon \\ \downarrow \\ \bigcup \end{array} \quad \subseteq_{\mathcal{T}} \quad \begin{array}{c} cond[x < 0] \\ \swarrow \quad \searrow \\ seq[x := x + 1] \quad \epsilon \\ \downarrow \\ cond[x < 0] \\ \swarrow \quad \searrow \\ seq[x := x + 1] \quad \epsilon \\ \downarrow \\ \bigcup \end{array} \quad \subseteq_{\mathcal{T}} \dots \end{array}$$

A.3 Hyper-path Interpretation is Sound for MA Interpretation

The key theoretical result we will establish in this section is that the tree-based interpretation (developed in appendix A.2) is *sound* for any MA interpretation, i.e., regular hyper-path expressions with the same tree-based interpretation also yield the same interpretation under any MA. (Recall the definition of Markov algebras reviewed in §4.5.)

THEOREM A.9. *Let $E, F \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$. If $\mathcal{T}_{\mathcal{K}}[E] = \mathcal{T}_{\mathcal{K}}[F]$, then for any MA interpretation $\mathcal{M} = (\mathcal{M}, [\cdot]^\mathcal{M})$ and any hole-valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$, it holds that $\mathcal{M}_\gamma[E] = \mathcal{M}_\gamma[F]$.*

Before proving the theorem, we define an interpretation map from possibly-infinite trees $\text{Tree}^\infty(\mathcal{F}, \mathcal{K})$ into a semantic algebra \mathcal{M} equipped with an interpretation $\mathcal{M} = (\mathcal{M}, [\cdot]^\mathcal{M})$. Because \mathcal{M} admits a dcpo, thus it also admits an ω -cpo, I can follow the coinductive principle for function definitions on trees (developed in appendix A.1). Below gives the base and induction steps for an interpretation map $r_\gamma^\mathcal{M}$, parameterized by a hole-valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$. The interpretation is well-defined because the operators $\otimes_M, \varphi_{M'}, p \oplus_M$ are all Scott-continuous, thus they are also

ω -continuous.

$$\begin{aligned}
 r_Y^{\mathcal{M}}(\varepsilon) &\triangleq 1_M \\
 r_Y^{\mathcal{M}}(\text{seq}[\text{act}](s_1)) &\triangleq [\text{act}]^{\mathcal{M}} \otimes_M r_Y^{\mathcal{M}}(s_1) \\
 r_Y^{\mathcal{M}}(\text{cond}[\varphi](s_1, s_2)) &\triangleq r_Y^{\mathcal{M}}(s_1) \diamond_{\varphi} r_Y^{\mathcal{M}}(s_2) \\
 r_Y^{\mathcal{M}}(\text{prob}[p](s_1, s_2)) &\triangleq r_Y^{\mathcal{M}}(s_1) \oplus_p r_Y^{\mathcal{M}}(s_2) \\
 r_Y^{\mathcal{M}}(\text{ndet}(s_1, s_2)) &\triangleq r_Y^{\mathcal{M}}(s_1) \uplus_M r_Y^{\mathcal{M}}(s_2) \\
 r_Y^{\mathcal{M}}(\square) &\triangleq \gamma(\square)
 \end{aligned}$$

We then prove the following property of $r_Y^{\mathcal{M}}$ about substitutions.

LEMMA A.10. *For any $t \in \text{Tree}^{\infty}(\mathcal{F}, \mathcal{K} \cup \{\square\})$, $u \in \text{Tree}^{\infty}(\mathcal{F}, \mathcal{K})$, and $\gamma : \mathcal{K} \rightarrow \mathcal{M}$, it holds that $r_{\gamma[\square \mapsto r_Y^{\mathcal{M}}(u)]}^{\mathcal{M}}(t) = r_Y^{\mathcal{M}}(t\{\square \leftarrow u\})$.*

PROOF. Let $\gamma' \triangleq \gamma[\square \mapsto r_Y^{\mathcal{M}}(u)]$. Then $\gamma' : (\mathcal{K} \cup \{\square\}) \rightarrow \mathcal{M}$.

We proceed by induction on the structure of t .

Case $t = \varepsilon$:

By definition, we have $r_{\gamma'}^{\mathcal{M}}(\varepsilon) = 1_M$.

By definition, we have $\varepsilon\{\square \leftarrow u\} = \varepsilon$, and also $r_Y^{\mathcal{M}}(\varepsilon) = 1_M$.

Thus, we conclude that $r_{\gamma'}^{\mathcal{M}}(\varepsilon) = r_Y^{\mathcal{M}}(\varepsilon\{\square \leftarrow u\})$.

Case $t = \text{seq}[\text{act}](s)$:

By induction hypothesis, we have $r_{\gamma'}^{\mathcal{M}}(s) = r_Y^{\mathcal{M}}(s\{\square \leftarrow u\})$.

By definition, we have $r_{\gamma'}^{\mathcal{M}}(\text{seq}[\text{act}](s)) = [\text{act}]^{\mathcal{M}} \otimes_M r_{\gamma'}^{\mathcal{M}}(s)$.

By definition, we have $\text{seq}[\text{act}](s)\{\square \leftarrow u\} = \text{seq}[\text{act}](s\{\square \leftarrow u\})$, and also $r_Y^{\mathcal{M}}(\text{seq}[\text{act}](s\{\square \leftarrow u\})) = [\text{act}]^{\mathcal{M}} \otimes_M r_Y^{\mathcal{M}}(s\{\square \leftarrow u\})$.

Thus, we conclude that $r_{\gamma'}^{\mathcal{M}}(\text{seq}[\text{act}](s)) = r_Y^{\mathcal{M}}(\text{seq}[\text{act}](s)\{\square \leftarrow u\})$.

Case $t = \text{cond}[\varphi](s_1, s_2)$:

By induction hypothesis on s_1 and s_2 , respectively, we have $r_{\gamma'}^{\mathcal{M}}(s_1) = r_Y^{\mathcal{M}}(s_1\{\square \leftarrow u\})$ and $r_{\gamma'}^{\mathcal{M}}(s_2) = r_Y^{\mathcal{M}}(s_2\{\square \leftarrow u\})$, respectively.

By definition, we have $r_{\gamma'}^{\mathcal{M}}(\text{cond}[\varphi](s_1, s_2)) = r_{\gamma'}^{\mathcal{M}}(s_1) \diamond_{\varphi} r_{\gamma'}^{\mathcal{M}}(s_2)$.

By definition, we have $\text{cond}[\varphi](s_1, s_2)\{\square \leftarrow u\} = \text{cond}[\varphi](s_1\{\square \leftarrow u\}, s_2\{\square \leftarrow u\})$, and also

$$r_Y^{\mathcal{M}}(\text{cond}[\varphi](s_1\{\square \leftarrow u\}, s_2\{\square \leftarrow u\})) = r_Y^{\mathcal{M}}(s_1\{\square \leftarrow u\}) \diamond_{\varphi} r_Y^{\mathcal{M}}(s_2\{\square \leftarrow u\}).$$

Thus, we conclude that $r_{\gamma'}^{\mathcal{M}}(\text{cond}[\varphi](s_1, s_2)) = r_Y^{\mathcal{M}}(\text{cond}[\varphi](s_1, s_2)\{\square \leftarrow u\})$.

Case $t = \text{prob}[p](s_1, s_2)$:

By induction hypothesis on s_1 and s_2 , respectively, we have $r_{\gamma'}^{\mathcal{M}}(s_1) = r_Y^{\mathcal{M}}(s_1\{\square \leftarrow u\})$ and $r_{\gamma'}^{\mathcal{M}}(s_2) = r_Y^{\mathcal{M}}(s_2\{\square \leftarrow u\})$, respectively.

By definition, we have $r_{\gamma'}^{\mathcal{M}}(\text{prob}[p](s_1, s_2)) = r_{\gamma'}^{\mathcal{M}}(s_1) \oplus_p r_{\gamma'}^{\mathcal{M}}(s_2)$.

By definition, we have $\text{prob}[p](s_1, s_2)\{\square \leftarrow u\} = \text{prob}[p](s_1\{\square \leftarrow u\}, s_2\{\square \leftarrow u\})$, and also

$$r_Y^{\mathcal{M}}(\text{prob}[p](s_1\{\square \leftarrow u\}, s_2\{\square \leftarrow u\})) = r_Y^{\mathcal{M}}(s_1\{\square \leftarrow u\}) \oplus_p r_Y^{\mathcal{M}}(s_2\{\square \leftarrow u\}).$$

Thus, we conclude that $r_{\gamma'}^{\mathcal{M}}(\text{prob}[p](s_1, s_2)) = r_Y^{\mathcal{M}}(\text{prob}[p](s_1, s_2)\{\square \leftarrow u\})$.

Case $t = \text{ndet}(s_1, s_2)$:

By induction hypothesis on s_1 and s_2 , respectively, we have $r_{\gamma'}^{\mathcal{M}}(s_1) = r_Y^{\mathcal{M}}(s_1\{\square \leftarrow u\})$ and $r_{\gamma'}^{\mathcal{M}}(s_2) = r_Y^{\mathcal{M}}(s_2\{\square \leftarrow u\})$, respectively.

By definition, we have $r_{\gamma'}^{\mathcal{M}}(\text{ndet}(s_1, s_2)) = r_{\gamma'}^{\mathcal{M}}(s_1) \uplus_M r_{\gamma'}^{\mathcal{M}}(s_2)$.

By definition, we have $ndet(s_1, s_2)\{\Box \rightsquigarrow u\} = ndet(s_1\{\Box \rightsquigarrow u\}, s_2\{\Box \rightsquigarrow u\})$, and also

$$r_Y^{\mathcal{M}}(ndet(s_1\{\Box \rightsquigarrow u\}, s_2\{\Box \rightsquigarrow u\})) = r_Y^{\mathcal{M}}(s_1\{\Box \rightsquigarrow u\}) \uplus_M r_Y^{\mathcal{M}}(s_2\{\Box \rightsquigarrow u\}).$$

Thus, we conclude that $r_{Y'}^{\mathcal{M}}(ndet(s_1, s_2)) = r_Y^{\mathcal{M}}(ndet(s_1, s_2)\{\Box \rightsquigarrow u\})$.

Case $t = \Box$:

By definition, we have $r_{Y'}^{\mathcal{M}}(\Box) = \gamma'(\Box) = r_Y^{\mathcal{M}}(u)$.

By definition, we have $\Box\{\Box \rightsquigarrow u\} = u$.

Thus, we conclude that $r_{Y'}^{\mathcal{M}}(\Box) = r_Y^{\mathcal{M}}(\Box\{\Box \rightsquigarrow u\})$.

Case $t = \Box'$ where $\Box' \neq \Box$:

By definition, we have $r_{Y'}^{\mathcal{M}}(\Box') = \gamma'(\Box') = \gamma(\Box')$.

By definition, we have $\Box'\{\Box \rightsquigarrow u\} = \Box'$, and also $r_Y^{\mathcal{M}}(\Box') = \gamma(\Box')$.

Thus, we conclude that $r_{Y'}^{\mathcal{M}}(\Box') = r_Y^{\mathcal{M}}(\Box'\{\Box \rightsquigarrow u\})$.

□

We can now present the proof of Thm. A.9.

PROOF OF THM. A.9. Fix $E, F \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$ such that $\mathcal{T}_{\mathcal{K}}[E] = \mathcal{T}_{\mathcal{K}}[F]$. Fix an interpretation $\mathcal{M} = (\mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}})$ and a hole-valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$. The theorem requires us to show that $\mathcal{M}_Y[E] = \mathcal{M}_Y[F]$. We claim that for any expression E , it holds that $\mathcal{M}_Y[E] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[E])$. If this is true, we can conclude the proof by

$$\mathcal{M}_Y[E] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[E]) = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[F]) = \mathcal{M}_Y[F].$$

To prove the claim, we fix an expression E and proceed by induction on the structure of E .

Case $E = \varepsilon$:

By definition, we have $\mathcal{M}_Y[\varepsilon] = 1_M$.

By definition, we have $\mathcal{T}_{\mathcal{K}}[\varepsilon] = \varepsilon$, and also $r_Y^{\mathcal{M}}(\varepsilon) = 1_M$.

Thus, we conclude that $\mathcal{M}_Y[\varepsilon] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[\varepsilon])$.

Case $E = \text{seq}[\text{act}](E_1)$:

By induction hypothesis, we have $\mathcal{M}_Y[E_1] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[E_1])$.

By definition, we have $\mathcal{M}_Y[\text{seq}[\text{act}](E_1)] = \llbracket \text{act} \rrbracket^{\mathcal{M}} \otimes_M \mathcal{M}_Y[E_1]$.

By definition, we have $\mathcal{T}_{\mathcal{K}}[\text{seq}[\text{act}](E_1)] = \text{seq}[\text{act}](s_1)$, where $s_1 \triangleq \mathcal{T}_{\mathcal{K}}[E_1]$, and also $r_Y^{\mathcal{M}}(\text{seq}[\text{act}](s_1)) = \llbracket \text{act} \rrbracket^{\mathcal{M}} \otimes_M r_Y^{\mathcal{M}}(s_1)$.

Thus, we conclude that $\mathcal{M}_Y[\text{seq}[\text{act}](E_1)] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[\text{seq}[\text{act}](E_1)])$.

Case $E = \text{cond}[\varphi](E_1, E_2)$:

By induction hypothesis on E_1 and E_2 , respectively, we have $\mathcal{M}_Y[E_1] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[E_1])$ and $\mathcal{M}_Y[E_2] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[E_2])$, respectively.

By definition, we have $\mathcal{M}_Y[\text{cond}[\varphi](E_1, E_2)] = \mathcal{M}_Y[E_1] \varphi \diamond_M \mathcal{M}_Y[E_2]$.

By definition, we have $\mathcal{T}_{\mathcal{K}}[\text{cond}[\varphi](E_1, E_2)] = \text{cond}[\varphi](s_1, s_2)$, where $s_1 \triangleq \mathcal{T}_{\mathcal{K}}[E_1]$, $s_2 \triangleq \mathcal{T}_{\mathcal{K}}[E_2]$, and also $r_Y^{\mathcal{M}}(\text{cond}[\varphi](s_1, s_2)) = r_Y^{\mathcal{M}}(s_1) \varphi \diamond_M r_Y^{\mathcal{M}}(s_2)$.

Thus, we conclude that $\mathcal{M}_Y[\text{cond}[\varphi](E_1, E_2)] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[\text{cond}[\varphi](E_1, E_2)])$.

Case $E = \text{prob}[p](E_1, E_2)$:

By induction hypothesis on E_1 and E_2 , respectively, we have $\mathcal{M}_Y[E_1] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[E_1])$ and $\mathcal{M}_Y[E_2] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[E_2])$, respectively.

By definition, we have $\mathcal{M}_Y[\text{prob}[p](E_1, E_2)] = \mathcal{M}_Y[E_1] \rho \oplus_M \mathcal{M}_Y[E_2]$.

By definition, we have $\mathcal{T}_{\mathcal{K}}[\text{prob}[p](E_1, E_2)] = \text{prob}[p](s_1, s_2)$, where $s_1 \triangleq \mathcal{T}_{\mathcal{K}}[E_1]$, $s_2 \triangleq \mathcal{T}_{\mathcal{K}}[E_2]$, and also $r_Y^{\mathcal{M}}(\text{prob}[p](s_1, s_2)) = r_Y^{\mathcal{M}}(s_1) \rho \oplus_M r_Y^{\mathcal{M}}(s_2)$.

Thus, we conclude that $\mathcal{M}_Y[\text{prob}[p](E_1, E_2)] = r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K}}[\text{prob}[p](E_1, E_2)])$.

Case $E = \text{ndet}(E_1, E_2)$:

By induction hypothesis on E_1 and E_2 , respectively, we have $\mathcal{M}_Y[E_1] = r_Y^{\mathcal{M}}(\mathcal{T}_K[E_1])$ and $\mathcal{M}_Y[E_2] = r_Y^{\mathcal{M}}(\mathcal{T}_K[E_2])$, respectively.

By definition, we have $\mathcal{M}_Y[\text{ndet}(E_1, E_2)] = \mathcal{M}_Y[E_1] \uplus_M \mathcal{M}_Y[E_2]$.

By definition, we have $\mathcal{T}_K[\text{ndet}(E_1, E_2)] = \text{ndet}(s_1, s_2)$, where $s_1 \triangleq \mathcal{T}_K[E_1]$, $s_2 \triangleq \mathcal{T}_K[E_2]$, and also $r_Y^{\mathcal{M}}(\text{ndet}(s_1, s_2)) = r_Y^{\mathcal{M}}(s_1) \uplus_M r_Y^{\mathcal{M}}(s_2)$.

Thus, we conclude that $\mathcal{M}_Y[\text{ndet}(E_1, E_2)] = r_Y^{\mathcal{M}}(\mathcal{T}_K[\text{ndet}(E_1, E_2)])$.

Case $E = \square$ for some $\square \in \mathcal{K}$:

By definition, we have $\mathcal{M}_Y[\square] = \gamma(\square)$.

By definition, we have $\mathcal{T}_K[\square] = \square$, and also $r_Y^{\mathcal{M}}(\square) = \gamma(\square)$.

Thus, we conclude that $\mathcal{M}_Y[\square] = r_Y^{\mathcal{M}}(\mathcal{T}_K[\square])$.

Case $E = E_1 \cdot \square E_2$:

By induction hypothesis on E_2 , we have $\mathcal{M}_Y[E_2] = r_Y^{\mathcal{M}}(\mathcal{T}_K[E_2])$.

Let $\gamma' \triangleq \gamma[\square \mapsto \mathcal{M}_Y[E_2]]$. Then $\gamma' : (\mathcal{K} \cup \{\square\}) \rightarrow \mathcal{M}$.

By induction hypothesis on E_1 (which is an expression in $\text{RegExp}^\infty(\mathcal{F}, \mathcal{K} \cup \{\square\})$), we have

$\mathcal{M}_{\gamma'}[E_1] = r_{\gamma'}^{\mathcal{M}}(\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1])$.

By definition, we have $\mathcal{M}_Y[E_1 \cdot \square E_2] = \mathcal{M}_{\gamma'}[E_1]$.

By definition, we have $\mathcal{T}_K[E_1 \cdot \square E_2] = \mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1]\{\square \leftarrow \mathcal{T}_K[E_2]\}$.

By Lem. A.10, we know that for any $s \in \text{Tree}^\infty(\mathcal{F}, \mathcal{K} \cup \{\square\})$, it holds that $r_{\gamma'}^{\mathcal{M}}(s) = r_Y(s\{\square \leftarrow \mathcal{T}_K[E_2]\})$. Therefore, we have

$$r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1]\{\square \leftarrow \mathcal{T}_K[E_2]\}) = r_{\gamma'}^{\mathcal{M}}(\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1]).$$

Thus, we conclude that $\mathcal{M}_Y[E_1 \cdot \square E_2] = r_Y^{\mathcal{M}}(\mathcal{T}_K[E_1 \cdot \square E_2])$.

Case $E = (E_1)^\infty \square$:

By definition, we have $\mathcal{M}_Y[(E_1)^\infty \square] = \text{lfp}_{\perp_M}^{\square} \lambda X. \mathcal{M}_{\gamma[\square \mapsto X]}[E_1]$.

By definition, we have $\mathcal{T}_K[(E_1)^\infty \square] = \text{lfp}_{\cup}^{\square} \lambda X. (\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1]\{\square \leftarrow X\})$.

Let $F \triangleq \lambda X. \mathcal{M}_{\gamma[\square \mapsto X]}[E_1]$ and $G \triangleq \lambda X. (\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1]\{\square \leftarrow X\})$. By the Kleene fixed-point theorem, it suffices to show that for any $n \geq 0$, $F^n(\perp_M) = r_Y^{\mathcal{M}}(G^n(\cup))$. We prove the claim by induction on n .

When $n = 0$:

By definition, we have $F^0(\perp_M) = \perp_M$.

By definition, we have $G^0(\cup) = \cup$, and also $r_Y^{\mathcal{M}}(\cup) = \perp_M$.

Thus, we conclude this that $F^0(\perp_M) = r_Y^{\mathcal{M}}(G^0(\cup))$.

When $n = k + 1$:

By induction hypothesis (on n), we have $F^k(\perp_M) = r_Y^{\mathcal{M}}(G^k(\cup))$.

Let $\gamma' \triangleq \gamma[\square \mapsto F^k(\perp_M)]$. Then $\gamma' : (\mathcal{K} \cup \{\square\}) \rightarrow \mathcal{M}$.

By definition, we have $F^n(\perp_M) = \mathcal{M}_{\gamma'}[E_1]$.

By definition, we have $G^n(\cup) = \mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1]\{\square \leftarrow G^k(\cup)\}$, and also by Lem. A.10, we have $r_Y^{\mathcal{M}}(\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1]\{\square \leftarrow G^k(\cup)\}) = r_{\gamma'}^{\mathcal{M}}(\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1])$.

By induction hypothesis (on E_1), we have $\mathcal{M}_{\gamma'}[E_1] = r_{\gamma'}^{\mathcal{M}}(\mathcal{T}_{\mathcal{K} \cup \{\square\}}[E_1])$.

Thus, we conclude that $F^n(\perp_M) = r_Y^{\mathcal{M}}(G^n(\cup))$.

□

$$\begin{array}{c}
\text{(CONST)} \\
\frac{c \in \mathcal{M}}{\Gamma \vdash c \Downarrow c \mid \emptyset} \\
\\
\text{(SEQ)} \\
\frac{c \in \mathcal{M} \quad \Gamma \vdash E \Downarrow F \mid \Theta}{\Gamma \vdash \text{seq}[c](E) \Downarrow \text{seq}[c](F) \mid \Theta} \\
\\
\text{(COND)} \\
\frac{\Gamma \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_2 \mid \Theta_2}{\Gamma \vdash \text{cond}[\varphi](E_1, E_2) \Downarrow \text{cond}[\varphi](F_1, F_2) \mid \Theta_1 \cup \Theta_2} \\
\\
\text{(PROB)} \\
\frac{\Gamma \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_2 \mid \Theta_2}{\Gamma \vdash \text{prob}[p](E_1, E_2) \Downarrow \text{prob}[p](F_1, F_2) \mid \Theta_1 \cup \Theta_2} \\
\\
\text{(NDET)} \\
\frac{\Gamma \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_2 \mid \Theta_2}{\Gamma \vdash \text{ndet}(E_1, E_2) \Downarrow \text{ndet}(F_1, F_2) \mid \Theta_1 \cup \Theta_2} \\
\\
\text{(HOLE)} \\
\frac{}{\Gamma \vdash \square \Downarrow \Gamma(\square) \mid \emptyset} \\
\\
\text{(CLOSURE)} \\
\frac{Z \text{ fresh} \quad \Gamma[\square \mapsto Z] \vdash E \Downarrow F \mid \Theta}{\Gamma \vdash E^{\infty \square} \Downarrow Z \mid \Theta \cup \{Z = F\}} \\
\\
\text{(SUB)} \\
\frac{\Gamma \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_1 \mid \Theta_2}{\Gamma \vdash \ominus(E_1, E_2) \Downarrow \ominus(F_1, F_2) \mid \Theta_1 \cup \Theta_2} \\
\\
\text{(CONCATENATION)} \\
\frac{\Gamma[\square \mapsto \square] \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_2 \mid \Theta_2}{\Gamma \vdash E_1 \cdot_{\square} E_2 \Downarrow F_1 \cdot_{\square} F_2 \mid \Theta_1 \cup \Theta_2} \\
\\
\text{(ADD)} \\
\frac{\Gamma \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_1 \mid \Theta_2}{\Gamma \vdash \oplus(E_1, E_2) \Downarrow \oplus(F_1, F_2) \mid \Theta_1 \cup \Theta_2} \\
\\
\text{(CALL-LIN)} \\
\frac{c \in \mathcal{M}}{\Gamma \vdash \text{call}_{\text{lin}}[Y_i; c] \Downarrow \text{call}_{\text{lin}}[Y_i; c] \mid \emptyset}
\end{array}$$

Fig. 13. Rules for extracting a closure-free linear regular hyper-path expression and an associated equation system from a linear regular hyper-path expression.

B PROOFS FOR NPA-PMA

Definition B.1 (Defn. 4.8). Let $f(\vec{X}) \in \text{RegExp}^{\infty}(\mathcal{F}^{\alpha}, \mathcal{K})$. The *differential* of $f(\vec{X})$ with respect to X_j at $\vec{v} \in \mathcal{M}^n$ under a hole-valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$, denoted by $\mathcal{D}_{X_j} f_{\gamma} |_{\vec{v}}(\vec{Y})$, is defined as follows:

$$\mathcal{D}_{X_j} f_{\gamma} |_{\vec{v}}(\vec{Y}) \triangleq \begin{cases} 0_M & \text{if } f = c \in \mathcal{M} \\ \text{seq}[c](\mathcal{D}_{X_j} g_{\gamma} |_{\vec{v}}(\vec{Y})) & \text{if } f = \text{seq}[c](g) \\ \text{seq}[v_k](\mathcal{D}_{X_j} g_{\gamma} |_{\vec{v}}(\vec{Y})) & \text{if } f = \text{call}[X_k](g) \text{ and } k \neq j \\ \oplus(\text{call}_{\text{lin}}[Y_j; g_{\gamma}(\vec{v})], \text{seq}[v_j](\mathcal{D}_{X_j} g_{\gamma} |_{\vec{v}}(\vec{Y}))) & \text{if } f = \text{call}[X_j](g) \\ \text{cond}[\varphi](\mathcal{D}_{X_j} g_{\gamma} |_{\vec{v}}(\vec{Y}), \mathcal{D}_{X_j} h_{\gamma} |_{\vec{v}}(\vec{Y})) & \text{if } f = \text{cond}[\varphi](g, h) \\ \text{prob}[p](\mathcal{D}_{X_j} g_{\gamma} |_{\vec{v}}(\vec{Y}), \mathcal{D}_{X_j} h_{\gamma} |_{\vec{v}}(\vec{Y})) & \text{if } f = \text{prob}[p](g, h) \\ \ominus(\text{ndet}(\oplus(g_{\gamma}(\vec{v}), \mathcal{D}_{X_j} g_{\gamma} |_{\vec{v}}(\vec{Y})), \oplus(h_{\gamma}(\vec{v}), \mathcal{D}_{X_j} h_{\gamma} |_{\vec{v}}(\vec{Y}))), f_{\gamma}(\vec{v})) & \text{if } f = \text{ndet}(g, h) \\ \square & \text{if } f = \square \in \mathcal{K} \\ \mathcal{D}_{X_j} g_{\gamma} |_{\square \mapsto h_{\gamma}(\vec{v})} |_{\vec{v}}(\vec{Y}) \cdot_{\square} \mathcal{D}_{X_j} h_{\gamma} |_{\vec{v}}(\vec{Y}) & \text{if } f = g \cdot_{\square} h \\ (\mathcal{D}_{X_j} g_{\gamma} |_{\square \mapsto f_{\gamma}(\vec{v})} |_{\vec{v}}(\vec{Y}))^{\infty \square} & \text{if } f = g^{\infty \square} \end{cases}$$

LEMMA (LEM. 4.3). For any expression $E \in \text{RegExp}^{\infty}(\mathcal{F}, \mathcal{K})$, there exists an expression $E' \in \text{RegExp}^{\infty}(\mathcal{F}^{\alpha}, \mathcal{K})$, such that for any $\gamma : \mathcal{K} \rightarrow \mathcal{M}$ and $\vec{v} \in \mathcal{M}^n$, it holds that $\mathcal{M}_{\gamma} \llbracket E \rrbracket(\vec{v}) = \mathcal{M}_{\gamma} \llbracket E' \rrbracket(\vec{v})$.

PROOF. By induction on the structure of E . Below shows the only two non-trivial cases.

Case $E = \varepsilon$: We set E' to $\underline{1}_M$.

Case $E = \text{seq}[\text{act}](E_1)$: By induction hypothesis, there exists E'_1 such that $\mathcal{M}_{\gamma} \llbracket E_1 \rrbracket(\vec{v}) = \mathcal{M}_{\gamma} \llbracket E'_1 \rrbracket(\vec{v})$. We then set E' to $\text{seq}[\llbracket \text{act} \rrbracket^{\mathcal{M}}](E'_1)$.

□

LEMMA B.2. If $E \in \text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \mathcal{K})$, $\text{dom}(\Theta) = \mathcal{Z}$, $\Gamma \vdash E \Downarrow F \mid \Theta$, $\gamma : \mathcal{K} \rightarrow \mathcal{M}$, $\eta : (\text{ran}(\Gamma) \setminus \mathcal{K}) \rightarrow \mathcal{M}$, $\vec{v} \in \mathcal{M}^n$, $\iota : \mathcal{Z} \rightarrow \mathcal{M}$ is the least solution of Θ with respect to $\gamma \cup \eta$ and \vec{v} , and $(\gamma \cup \eta) \circ \Gamma = \gamma$, then $\mathcal{M}_\gamma \llbracket E \rrbracket(\vec{v}) = \mathcal{M}_{\gamma \cup \eta \cup \iota} \llbracket F \rrbracket(\vec{v})$.

PROOF. We proceed by induction on the derivation of $\Gamma \vdash E \Downarrow F \mid \Theta$.

Case (CONST):

$$\frac{c \in \mathcal{M}}{\Gamma \vdash c \Downarrow c \mid \emptyset}$$

We have $\mathcal{M}_\gamma \llbracket c \rrbracket(\vec{v}) = c$.

We have $\mathcal{M}_{\gamma \cup \eta \cup \iota} \llbracket c \rrbracket(\vec{v}) = c$.

Then we conclude this case.

Case (SEQ):

$$\frac{c \in \mathcal{M} \quad \Gamma \vdash E_1 \Downarrow F_1 \mid \Theta}{\Gamma \vdash \text{seq}[c](E_1) \Downarrow \text{seq}[c](F_1) \mid \Theta}$$

By induction hypothesis, we know $\mathcal{M}_\gamma \llbracket E_1 \rrbracket(\vec{v}) = \mathcal{M}_{\gamma \cup \eta \cup \iota} \llbracket F_1 \rrbracket(\vec{v})$.

We have $\mathcal{M}_\gamma \llbracket \text{seq}[c](E_1) \rrbracket(\vec{v}) = c \otimes_M \mathcal{M}_\gamma \llbracket E_1 \rrbracket(\vec{v})$.

We have $\mathcal{M}_{\gamma \cup \eta \cup \iota} \llbracket \text{seq}[c](F_1) \rrbracket(\vec{v}) = c \otimes_M \mathcal{M}_{\gamma \cup \eta \cup \iota} \llbracket F_1 \rrbracket(\vec{v})$.

Then we conclude this case.

Case (COND):

$$\frac{\Gamma \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_2 \mid \Theta_2}{\Gamma \vdash \text{cond}[\varphi](E_1, E_2) \Downarrow \text{cond}[\varphi](F_1, F_2) \mid \Theta_1 \cup \Theta_2}$$

Let $\iota_1 \triangleq \iota|_{\text{dom}(\Theta_1)}$ and $\iota_2 \triangleq \iota|_{\text{dom}(\Theta_2)}$. Then $\iota = \iota_1 \cup \iota_2$.

By the premise, we know that Θ_1 and Θ_2 are two independent sets of equations, given $\gamma \cup \eta$ and \vec{v} .

Thus, ι_i is the least solution of Θ_i ($i \in \{1, 2\}$).

By induction hypothesis, we know $\mathcal{M}_\gamma \llbracket E_1 \rrbracket(\vec{v}) = \mathcal{M}_{\gamma \cup \eta \cup \iota_1} \llbracket F_1 \rrbracket(\vec{v})$.

By induction hypothesis, we know $\mathcal{M}_\gamma \llbracket E_2 \rrbracket(\vec{v}) = \mathcal{M}_{\gamma \cup \eta \cup \iota_2} \llbracket F_2 \rrbracket(\vec{v})$.

We have $\mathcal{M}_\gamma \llbracket \text{cond}[\varphi](E_1, E_2) \rrbracket(\vec{v}) = \mathcal{M}_\gamma \llbracket E_1 \rrbracket(\vec{v}) \varphi \diamond_M \mathcal{M}_\gamma \llbracket E_2 \rrbracket(\vec{v})$.

We have $\mathcal{M}_{\gamma \cup \eta \cup \iota} \llbracket \text{cond}[\varphi](F_1, F_2) \rrbracket(\vec{v}) = \mathcal{M}_{\gamma \cup \eta \cup \iota_1} \llbracket F_1 \rrbracket(\vec{v}) \varphi \diamond_M \mathcal{M}_{\gamma \cup \eta \cup \iota_2} \llbracket F_2 \rrbracket(\vec{v})$.

Then we conclude this case.

(Cases (PROB), (NDET), (ADD), and (SUB) are similar to this case.)

Case (HOLE):

$$\frac{}{\Gamma \vdash \square \Downarrow \Gamma(\square) \mid \emptyset}$$

We have $\mathcal{M}_\gamma \llbracket \square \rrbracket(\vec{v}) = \gamma(\square)$.

We have $\mathcal{M}_{\gamma \cup \eta \cup \iota} \llbracket \Gamma(\square) \rrbracket(\vec{v}) = ((\gamma \cup \eta) \circ \Gamma)(\square)$.

We conclude by the assumption $(\gamma \cup \eta) \circ \Gamma = \gamma$.

Case (CONCATENATION):

$$\frac{\Gamma[\square \mapsto \square] \vdash E_1 \Downarrow F_1 \mid \Theta_1 \quad \Gamma \vdash E_2 \Downarrow F_2 \mid \Theta_2}{\Gamma \vdash E_1 \cdot_\square E_2 \Downarrow F_1 \cdot_\square F_2 \mid \Theta_1 \cup \Theta_2}$$

Let $\iota_1 \triangleq \iota|_{\text{dom}(\Theta_1)}$ and $\iota_2 \triangleq \iota|_{\text{dom}(\Theta_2)}$. Then $\iota = \iota_1 \cup \iota_2$.

By the premise, we know that Θ_1 and Θ_2 are two independent sets of equations, given $\gamma \cup \eta$ and \vec{v} .

Thus, ι_i is the least solution of Θ_i ($i \in \{1, 2\}$).

By induction hypothesis, we know $\mathcal{M}_\gamma \llbracket E_2 \rrbracket(\vec{v}) = \mathcal{M}_{\gamma \cup \eta \cup \iota_2} \llbracket F_2 \rrbracket(\vec{v})$.

Let $\gamma' \triangleq \gamma[\square \mapsto \mathcal{M}_\gamma \llbracket E_2 \rrbracket(\vec{v})]$. We still have $((\gamma' \cup \eta) \circ (\Gamma[\square \mapsto \square])) = \gamma'$.

By induction hypothesis, we know $\mathcal{M}_{\gamma'} \llbracket E_1 \rrbracket(\vec{v}) = \mathcal{M}_{\gamma' \cup \eta \cup \iota_1} \llbracket F_1 \rrbracket(\vec{v})$.

We have $\mathcal{M}_Y \llbracket E_1 \cdot_{\square} E_2 \rrbracket(\vec{v}) = \mathcal{M}_{Y'} \llbracket E_1 \rrbracket(\vec{v})$.

We have $\mathcal{M}_{Y \cup \eta \cup \iota_1} \llbracket F_1 \cdot_{\square} F_2 \rrbracket(\vec{v}) = \mathcal{M}_{(Y \cup \eta \cup \iota_1)[\square \mapsto \mathcal{M}_{Y \cup \eta \cup \iota_2} \llbracket F_2 \rrbracket(\vec{v})]} \llbracket F_1 \rrbracket(\vec{v}) = \mathcal{M}_{Y' \cup \eta \cup \iota_1} \llbracket F_1 \rrbracket(\vec{v})$.

Then we conclude this case.

Case (CLOSURE):

$$\frac{Z \text{ fresh} \quad \Gamma[\square \mapsto Z] \vdash E_1 \Downarrow F_1 \mid \Theta_1}{\Gamma \vdash (E_1)^{\infty \square} \Downarrow Z \mid \Theta_1 \cup \{Z = F_1\}}$$

Let $\theta \triangleq \iota(Z)$. Let $\iota_1 \triangleq \iota|_{\text{dom}(\Theta_1)}$.

By assumption, ι is the least solution of $\Theta_1 \cup \{Z = F_1\}$, given $\gamma \cup \eta$ and \vec{v} .

Thus, $\theta = \mathcal{M}_{Y \cup \eta \cup \iota} \llbracket F_1 \rrbracket(\vec{v})$. Let $\eta' \triangleq \eta[Z \mapsto \theta]$. Then $\theta = \mathcal{M}_{Y' \cup \eta' \cup \iota_1} \llbracket F_1 \rrbracket(\vec{v})$.

Let $\gamma' \triangleq \gamma[\square \mapsto \theta]$. We have $(\gamma' \cup \eta') \circ \Gamma[\square \mapsto Z] = \gamma'$.

Also, ι_1 is the least solution of Θ_1 , given $\gamma' \cup \eta'$ and \vec{v} .

By induction hypothesis, we know $\mathcal{M}_{Y'} \llbracket E_1 \rrbracket(\vec{v}) = \mathcal{M}_{Y' \cup \eta' \cup \iota_1} \llbracket F_1 \rrbracket(\vec{v}) = \theta$.

We have $\mathcal{M}_Y \llbracket (E_1)^{\infty \square} \rrbracket(\vec{v}) = \rho$ where ρ is the least fixed-point of $\lambda \rho. \mathcal{M}_{Y[\square \mapsto \rho]} \llbracket E_1 \rrbracket(\vec{v})$.

Indeed, θ is a fixed-point of $\lambda \rho. \mathcal{M}_{Y[\square \mapsto \rho]} \llbracket E_1 \rrbracket(\vec{v})$. It remains to show θ is the least one.

Again, by assumption, θ is the least solution of $Z = F_1$, given $\gamma[\square \mapsto \rho] \cup \eta \cup \iota_1$ and \vec{v} .

Suppose that $\rho \sqsubset_M \theta$. Let ι'_1 be the least solution of Θ_1 given $\gamma' \cup \eta'[Z \mapsto \rho]$ and \vec{v} . By ω -continuity, we know that $\iota'_1 \sqsubset_M \iota_1$.

Thus, by induction hypothesis, we know $\mathcal{M}_{Y[\square \mapsto \rho]} \llbracket E_1 \rrbracket(\vec{v}) = \mathcal{M}_{Y[\square \mapsto \rho] \cup \eta[Z \mapsto \rho] \cup \iota'_1} \llbracket F_1 \rrbracket(\vec{v})$.

But $\rho = \mathcal{M}_{Y[\square \mapsto \rho]} \llbracket E_1 \rrbracket(\vec{v})$. Thus ρ is a solution of $Z = F_1$ given $\rho[\square \mapsto \rho] \cup \eta \cup \iota'_1$ and \vec{v} .

Therefore, let $\iota' \triangleq \iota'_1 \cup \{Z \mapsto \rho\}$, then ι' is a solution of $\Theta_1 \cup \{Z = F_1\}$ given $\gamma \cup \eta$ and \vec{v} .

But then we have $\iota' \sqsubset_M \iota$, which contradicts the assumption that ι is the least solution.

Then we conclude this case.

Case (CALL-LIN):

$$\frac{c \in \mathcal{M}}{\Gamma \vdash \text{call}_{\text{lin}}[Y_i; c] \Downarrow \text{call}_{\text{lin}}[Y_i; c] \mid \emptyset}$$

We have $\mathcal{M}_Y \llbracket \text{call}_{\text{lin}}[Y_i; c] \rrbracket(\vec{v}) = v_i \otimes_M c$.

We have $\mathcal{M}_{Y \cup \eta \cup \iota} \llbracket \text{call}_{\text{lin}}[Y_i; c] \rrbracket(\vec{v}) = v_i \otimes_M c$.

Then we conclude this case. \square

THEOREM (THM. 4.6). *The linear-equation-solving method presented in §4.3 computes $\text{lfp}_{\frac{\square}{M}}^M \lambda \vec{\theta}. \langle \mathcal{M}_Y \llbracket E_{Y_1} \rrbracket(\vec{\theta}), \dots, \mathcal{M}_Y \llbracket E_{Y_n} \rrbracket(\vec{\theta}) \rangle$.*

PROOF. By assumption, we have $\{\square \mapsto \square\}_{\square \in \mathcal{K}} \vdash E_{Y_i} \Downarrow F_{Y_i} \mid \Theta_i$ for each i .

Let $\text{solve}_{\mathcal{K}}(\{Y_i = F_{Y_i}\}_{i=1}^n, \bigcup_{i=1}^n \Theta_i, \gamma)$ return $\iota : \text{dom}(\bigcup_{i=1}^n \Theta_i) \rightarrow \mathcal{M}$ and $\vec{v} \in \mathcal{M}^n$, i.e., the least solution of $\{Y_i = F_{Y_i}\}_{i=1}^n \cup \bigcup_{i=1}^n \Theta_i$.

The method then returns $\mathcal{M}_{Y \cup \iota} \llbracket F_{Y_i} \rrbracket(\vec{v})$ for the solution of Y_i , but because solve returns the least solution, the value coincides with v_i .

Thus, here it is sufficient to prove \vec{v} is the least fixed-point of $\lambda \vec{\theta}. \langle \mathcal{M}_Y \llbracket E_{Y_1} \rrbracket(\vec{\theta}), \dots, \mathcal{M}_Y \llbracket E_{Y_n} \rrbracket(\vec{\theta}) \rangle$.

If we fix $\vec{\theta} \in \mathcal{M}^n$, then the sets Θ_i 's are independent sets of equations under γ and $\vec{\theta}$. Therefore, for each i , by Lem. B.2 (set η to $\{\}$), we know that $\mathcal{M}_Y \llbracket E_{Y_i} \rrbracket(\vec{\theta}) = \mathcal{M}_{Y \cup \iota_i} \llbracket F_{Y_i} \rrbracket(\vec{\theta})$.

Therefore, \vec{v} is a fixed point. It remains to show it is the least one.

Suppose there is another fixed-point $\vec{\rho}$ that satisfies $\vec{\rho} \sqsubset_M \vec{v}$.

For each i , Let ι'_i be the least solution of Θ_i given γ and $\vec{\rho}$. By ω -continuity, we know that $\iota'_i \sqsubseteq \iota_i \triangleq \iota|_{\text{dom}(\Theta_i)}$.

Define $\iota' \triangleq \bigcup_{i=1}^n \iota'_i$. Then $\iota' \sqsubset_M \iota$.

Then, again, by Lem. B.2 (set η to $\{\}$), for each i , we know that $\mathcal{M}_Y \llbracket E_{Y_i} \rrbracket(\vec{\rho}) = \mathcal{M}_{Y \cup \iota'} \llbracket F_{Y_i} \rrbracket(\vec{\rho})$.

But $\rho_i = \mathcal{M}_Y \llbracket E_{Y_i} \rrbracket(\vec{\rho})$ for each i . Thus ι' and $\vec{\rho}$ is a solution to $\{Y_i = F_{Y_i}\}_{i=1}^n \cup \bigcup_{i=1}^n \Theta_i$ under γ .

But then we obtain $(\vec{\rho}, \iota') \sqsubset (\vec{\nu}, \iota)$, which contradicts the assumption that $(\vec{\nu}, \iota)$ is the least solution.

Then we conclude the proof. \square

LEMMA (LEM. 4.10). *If \vec{f} is a vector of regular hyper-path expressions in $\text{RegExp}^\infty(\mathcal{F}^\alpha, \mathcal{K})$, it holds that $\mathcal{D}\vec{f}_Y|_{\vec{\nu}}(\vec{Y})$ is a vector of regular hyper-path expressions in $\text{RegExp}^\infty(\mathcal{F}_{\text{lin}}^\alpha, \mathcal{K})$.*

PROOF. Straightforward because we only use the alphabet $\mathcal{F}_{\text{lin}}^\alpha$ in the definition of differentials (see Defn. 4.8). \square

LEMMA B.3. *Let $\vec{f}(\vec{X})$ be a vector of regular hyper-path expression in $\text{RegExp}^\infty(\mathcal{F}^\alpha, \mathcal{K})$, $\gamma, \gamma' : \mathcal{K} \rightarrow \mathcal{M}$ be two hole-valuations, and \vec{u}, \vec{v} be two vectors in \mathcal{M}^n . We have*

$$\vec{f}_Y(\vec{u}) \oplus_M (\mathcal{D}\vec{f}_Y|_{\vec{u}}(\vec{v}))_{Y'} \sqsubseteq_M f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}).$$

PROOF. It suffices to show the inequality for each component separately, so W.L.O.G. let $\vec{f}(\vec{X}) = f(\vec{X})$ be a regular hyper-path expression. We then proceed by induction on the structure of f .

Case $f = c$: We have $\mathcal{D}f_Y|_{\vec{u}}(\vec{v}) = \underline{0}_M$ and thus $c \oplus_M \underline{0}_M \sqsubseteq_M c$.

Case $f = \text{seq}[c](g)$: We have $f_Y(\vec{x}) = c \otimes_M g_Y(\vec{x})$ and $(\mathcal{D}f_Y|_{\vec{x}}(\vec{y}))_{Y'} = c \otimes_M (\mathcal{D}g_Y|_{\vec{x}}(\vec{y}))_{Y'}$. Thus

$$\begin{aligned} f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) &= c \otimes_M g_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) \\ &\sqsubseteq_M c \otimes_M (g_Y(\vec{u}) \oplus_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'}) \quad (\text{induction hypothesis}) \\ &= (c \otimes_M g_Y(\vec{u})) \oplus_M (c \otimes_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'}) \\ &= f_Y(\vec{u}) \oplus_M (\mathcal{D}f_Y|_{\vec{u}}(\vec{v}))_{Y'}. \end{aligned}$$

Case $f = \text{call}[X_i](g)$: We have $f_Y(\vec{x}) = x_i \otimes_M g_Y(\vec{x})$ and

$$\begin{aligned} \mathcal{D}f_Y|_{\vec{x}}(\vec{Y}) &= D_{X_i} f_Y|_{\vec{x}}(\vec{Y}) \oplus \bigoplus_{j \neq i} D_{X_j} f_Y|_{\vec{x}}(\vec{Y}) \\ &= (\text{call}_{\text{lin}}[Y_i; g_Y(\vec{x})] \oplus \text{seq}[x_j](D_{X_i} g_Y|_{\vec{x}}(\vec{Y}))) \oplus \bigoplus_{j \neq i} \text{seq}[x_i](D_{X_j} g_Y|_{\vec{x}}(\vec{Y})), \\ (\mathcal{D}f_Y|_{\vec{x}}(\vec{y}))_{Y'} &= ((y_i \otimes_M g_Y(\vec{x})) \oplus_M (x_j \otimes_M (D_{X_i} g_Y|_{\vec{x}}(\vec{y}))_{Y'})) \oplus_M \bigoplus_{M, j \neq i} (x_j \otimes_M (D_{X_j} g_Y|_{\vec{x}}(\vec{y}))_{Y'}) \\ &= (y_i \otimes_M g_Y(\vec{x})) \oplus_M (x_j \otimes_M (\mathcal{D}g_Y|_{\vec{x}}(\vec{y}))_{Y'}). \end{aligned}$$

Thus

$$\begin{aligned} f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) &= (u_i \oplus_M v_i) \otimes_M g_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) \\ &\sqsubseteq_M (u_i \oplus_M v_i) \otimes_M (g_Y(\vec{u}) \oplus_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'}) \quad (\text{induction hypothesis}) \\ &= (u_i \otimes_M g_Y(\vec{u})) \oplus_M (v_i \otimes_M g_Y(\vec{u})) \oplus_M ((u_i \oplus_M v_i) \otimes_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'}) \\ &\sqsubseteq_M f_Y(\vec{u}) \oplus_M ((v_i \otimes_M g_Y(\vec{u})) \oplus_M (u_i \otimes_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'})) \\ &= f_Y(\vec{u}) \oplus_M (\mathcal{D}f_Y|_{\vec{u}}(\vec{v}))_{Y'}. \end{aligned}$$

Case $f = \text{cond}[\varphi](g, h)$: We have $f_Y(\vec{x}) = g_Y(\vec{x}) \varphi \diamond_M h_Y(\vec{x})$ and $(\mathcal{D}f_Y|_{\vec{x}}(\vec{y}))_{Y'} = (\mathcal{D}g_Y|_{\vec{x}}(\vec{y}))_{Y'} \varphi \diamond_M (\mathcal{D}h_Y|_{\vec{x}}(\vec{y}))_{Y'}$. Thus

$$\begin{aligned} f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) &= g_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) \varphi \diamond_M h_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) \\ &\sqsubseteq_M (g_Y(\vec{u}) \oplus_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'}) \varphi \diamond_M (h_Y(\vec{u}) \oplus_M (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'}) \quad (\text{induction hypothesis}) \\ &= (g_Y(\vec{u}) \varphi \diamond_M h_Y(\vec{u})) \oplus_M ((\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'} \varphi \diamond_M (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'}) \quad (\text{Defn. 4.2}) \\ &= f_Y(\vec{u}) \oplus_M (\mathcal{D}f_Y|_{\vec{u}}(\vec{v}))_{Y'}. \end{aligned}$$

Case $f = \text{prob}[p](g, h)$: We have $f_Y(\vec{x}) = g_Y(\vec{x}) \oplus_M h_Y(\vec{x})$ and $(\mathcal{D}f_Y|_{\vec{x}}(\vec{y}))_{Y'} = (\mathcal{D}g_Y|_{\vec{x}}(\vec{y}))_{Y'} \oplus_M (\mathcal{D}h_Y|_{\vec{x}}(\vec{y}))_{Y'}$. Thus

$$\begin{aligned} f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) &= g_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) \oplus_M h_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) \\ &\supseteq_M (g_Y(\vec{u}) \oplus_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'}) \oplus_M (h_Y(\vec{u}) \oplus_M (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'}) \quad (\text{induction hypothesis}) \\ &= (g_Y(\vec{u}) \oplus_M h_Y(\vec{u})) \oplus_M ((\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'} \oplus_M (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'}) \quad (\text{Defn. 4.2}) \\ &= f_Y(\vec{u}) \oplus_M (\mathcal{D}f_Y|_{\vec{u}}(\vec{v}))_{Y'}. \end{aligned}$$

Case $f = \text{ndet}(g, h)$: We have $f_Y(\vec{x}) = g_Y(\vec{x}) \uplus_M h_Y(\vec{x})$ and $(\mathcal{D}f_Y|_{\vec{x}}(\vec{y}))_{Y'} = ((g_Y(\vec{x}) \oplus_M (\mathcal{D}g_Y|_{\vec{x}}(\vec{y}))_{Y'}) \uplus_M (h_Y(\vec{x}) \oplus_M (\mathcal{D}h_Y|_{\vec{x}}(\vec{y}))_{Y'})) \ominus_M f_Y(\vec{x})$. Thus

$$\begin{aligned} f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) &= g_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) \uplus_M h_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) \\ &\supseteq_M ((g_Y(\vec{u}) \oplus_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'}) \uplus_M (h_Y(\vec{u}) \oplus_M (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'})) \quad (\text{induction hypothesis}) \\ &= f_Y(\vec{u}) \oplus_M (((g_Y(\vec{u}) \oplus_M (\mathcal{D}g_Y|_{\vec{u}}(\vec{v}))_{Y'}) \uplus_M (h_Y(\vec{u}) \oplus_M (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'})) \ominus_M f_Y(\vec{u})) \\ &= f_Y(\vec{u}) \oplus_M (\mathcal{D}f_Y|_{\vec{u}}(\vec{v}))_{Y'}. \end{aligned}$$

Case $f = \square$: We have $f_Y(\vec{x}) = \gamma(\square)$ and $(\mathcal{D}f_Y|_{\vec{x}}(\vec{y}))_{Y'} = \gamma'(\square)$. Thus

$$\begin{aligned} f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) &= \gamma(\square) \oplus_M \gamma'(\square) \\ &= f_Y(\vec{u}) \oplus_M (\mathcal{D}f_Y|_{\vec{u}}(\vec{v}))_{Y'}. \end{aligned}$$

Case $f = g \cdot_{\square} h$: We have $f_Y(\vec{x}) = g_{Y[\square \mapsto h_Y(\vec{x})]}(\vec{x})$ and $(\mathcal{D}f_Y|_{\vec{x}}(\vec{y}))_{Y'} = (\mathcal{D}g_{Y[\square \mapsto h_Y(\vec{x})]}|_{\vec{x}}(\vec{y}))_{Y'[\square \mapsto (\mathcal{D}h_Y|_{\vec{x}}(\vec{y}))_{Y'}]}$. Thus

$$\begin{aligned} f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) &= g_{(Y \oplus_M Y')[\square \mapsto h_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v})]}(\vec{u} \oplus_M \vec{v}) \\ &\supseteq_M g_{(Y \oplus_M Y')[\square \mapsto h_Y(\vec{u}) \oplus_M (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'}]}(\vec{u} \oplus_M \vec{v}) \quad (\text{induction hypothesis}) \\ &= g_{Y[\square \mapsto h_Y(\vec{u})] \oplus_M Y'[\square \mapsto (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'}]}(\vec{u} \oplus_M \vec{v}) \\ &\supseteq_M g_{Y[\square \mapsto h_Y(\vec{u})]}(\vec{u}) \oplus_M (\mathcal{D}g_{Y[\square \mapsto h_Y(\vec{u})]}|_{\vec{u}}(\vec{v}))_{Y'[\square \mapsto (\mathcal{D}h_Y|_{\vec{u}}(\vec{v}))_{Y'}]} \quad (\text{induction hypothesis}) \\ &= f_Y(\vec{u}) \oplus_M (\mathcal{D}f_Y|_{\vec{u}}(\vec{v}))_{Y'}. \end{aligned}$$

Case $f = (g)^{\omega_{\square}}$: We have $f_Y(\vec{x}) = \text{lfp}_{\underline{0}_M} \lambda \theta. g_{Y[\square \mapsto \theta]}(\vec{x})$ and $(\mathcal{D}f_Y|_{\vec{x}}(\vec{y}))_{Y'} = \text{lfp}_{\underline{0}_M} \lambda \theta. (\mathcal{D}g_{Y[\square \mapsto f_Y(\vec{x})]}|_{\vec{x}}(\vec{y}))_{Y'[\square \mapsto \theta]}$. Let $\rho \triangleq \text{lfp}_{\underline{0}_M} \lambda \rho. g_{Y[\square \mapsto \rho]}(\vec{u})$. Then by ω -continuity, we have $\rho \sqsubseteq_M \text{lfp}_{\underline{0}_M} \lambda \theta. g_{(Y \oplus_M Y')[\square \mapsto \theta]}(\vec{u} \oplus \vec{v})$. Thus

$$\begin{aligned} f_{Y \oplus_M Y'}(\vec{u} \oplus_M \vec{v}) &= \text{lfp}_{\underline{0}_M} \lambda \theta. g_{(Y \oplus_M Y')[\square \mapsto \theta]}(\vec{u} \oplus \vec{v}) \\ &= \text{lfp}_{\rho} \lambda \theta. g_{Y[\square \mapsto \rho] \oplus_M Y'[\square \mapsto \theta \oplus_M \rho]}(\vec{u} \oplus \vec{v}) \\ &\supseteq_M \text{lfp}_{\rho} \lambda \theta. (g_{Y[\square \mapsto \rho]}(\vec{u}) \oplus_M (\mathcal{D}g_{Y[\square \mapsto \rho]}|_{\vec{u}}(\vec{v}))_{Y'[\square \mapsto \theta \oplus_M \rho]}) \quad (\text{induction hypothesis}) \\ &= \text{lfp}_{\rho} \lambda \theta. (\rho \oplus_M (\mathcal{D}g_{Y[\square \mapsto \rho]}|_{\vec{u}}(\vec{v}))_{Y'[\square \mapsto \theta \oplus_M \rho]}) \\ &= \bigsqcup_{n \in \mathbb{N}} L^n(\rho), \end{aligned}$$

where $L \triangleq \lambda \theta. (\rho \oplus_M H(\theta \oplus_M \rho))$, $H \triangleq \lambda \theta. (\mathcal{D}g_{Y[\square \mapsto \rho]}|_{\vec{u}}(\vec{v}))_{Y'[\square \mapsto \theta]}$.

On the other hand, we have

$$\begin{aligned} f_Y(\vec{u}) \oplus_M (\mathcal{D}f_Y|_{\vec{u}}(\vec{v}))_{Y'} &= \rho \oplus_M (\text{lfp}_{\underline{0}_M} \lambda \theta. (\mathcal{D}g_{Y[\square \mapsto f_Y(\vec{u})]}|_{\vec{u}}(\vec{v}))_{Y'[\square \mapsto \theta]}) \\ &= \rho \oplus_M (\text{lfp}_{\underline{0}_M} \lambda \theta. (\mathcal{D}g_{Y[\square \mapsto \rho]}|_{\vec{u}}(\vec{v}))_{Y'[\square \mapsto \theta]}) \\ &= \rho \oplus_M \bigsqcup_{n \in \mathbb{N}} H^n(\underline{0}_M). \end{aligned}$$

We then prove by induction on n that $L^n(\rho) = \rho \oplus_M H^n(\underline{0}_M)$.

When $n = 0$, we have $L^0(\rho) = \rho$ and $\rho \oplus_M H^0(\rho) = \rho \oplus_M \underline{0}_M = \rho$.

For the induction step, we have

$$\begin{aligned}
 L^{n+1}(\rho) &= L(L^n(\rho)) \\
 &= \rho \oplus_M H(L^n(\rho) \ominus_M \rho) \\
 &= \rho \oplus_M H((\rho \oplus_M H^n(\underline{0}_M)) \ominus_M \rho) && \text{(induction hypothesis)} \\
 &= \rho \oplus_M H(H^n(\underline{0}_M)) \\
 &= \rho \oplus_M H^{n+1}(\underline{0}_M).
 \end{aligned}$$

□

LEMMA B.4. Let $\vec{f}_{\{\}}(\vec{x}) \sqsupseteq_M \vec{x}$. For all $d \geq 0$, there exists a vector $\vec{e}^{(d)}(\vec{x})$ such that $\vec{f}_{\{\}}^d(\vec{x}) \oplus_M \vec{e}^{(d)}(\vec{x}) = \vec{f}_{\{\}}^{d+1}(\vec{x})$ and $\vec{e}^{(d)}(\vec{x}) \sqsupseteq_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{f}_{\{\}}^{d-1}(\vec{x})}((\mathcal{D}\vec{f}_{\{\}}|_{\vec{f}_{\{\}}^{d-2}(\vec{x})}(\cdots(\mathcal{D}\vec{f}_{\{\}}|_{\vec{x}}(\vec{e}^{(0)}(\vec{x}))_{\{\}})\cdots))_{\{\}}))_{\{\}} \sqsupseteq_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{x}}^d(\vec{e}^{(0)}(\vec{x})))_{\{\}}.$

PROOF. By induction on d . For $d = 0$, we set $\vec{e}^{(0)}(\vec{x})$ to be $\vec{f}_{\{\}}(\vec{x}) \ominus_M \vec{x}$. Let $d \geq 0$.

$$\begin{aligned}
 \vec{f}_{\{\}}^{d+2}(\vec{x}) &= \vec{f}_{\{\}}(\vec{f}_{\{\}}^{d+1}(\vec{x})) \\
 &= \vec{f}_{\{\}}(\vec{f}_{\{\}}^d(\vec{x}) \oplus_M \vec{e}^{(d)}(\vec{x})) && \text{(induction hypothesis)} \\
 &\sqsupseteq_M \vec{f}_{\{\}}(\vec{f}_{\{\}}^d(\vec{x})) \oplus_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{f}_{\{\}}^d(\vec{x})}(\vec{e}^{(d)}(\vec{x})))_{\{\}} && \text{(Lem. B.3)} \\
 &\sqsupseteq_M \vec{f}_{\{\}}^{d+1}(\vec{x}) \\
 &\quad \oplus_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{f}_{\{\}}^d(\vec{x})}((\mathcal{D}\vec{f}_{\{\}}|_{\vec{f}_{\{\}}^{d-1}(\vec{x})}(\cdots(\mathcal{D}\vec{f}_{\{\}}|_{\vec{x}}(\vec{e}^{(0)}(\vec{x}))_{\{\}})\cdots))_{\{\}}))_{\{\}}. && \text{(induction hypothesis)}
 \end{aligned}$$

Therefore, there exists an $\vec{e}^{(d+1)}(\vec{x}) \sqsupseteq_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{f}_{\{\}}^d(\vec{x})}((\mathcal{D}\vec{f}_{\{\}}|_{\vec{f}_{\{\}}^{d-1}(\vec{x})}(\cdots(\mathcal{D}\vec{f}_{\{\}}|_{\vec{x}}(\vec{e}^{(0)}(\vec{x}))_{\{\}})\cdots))_{\{\}}))_{\{\}}.$ Since $(\mathcal{D}\vec{f}_{\{\}}|_{\vec{y}}(\cdot))_{\{\}}$ is monotone in \vec{y} and $\vec{x} \sqsubseteq_M \vec{f}(\vec{x}) \sqsubseteq_M \vec{f}^2(\vec{x}) \sqsubseteq_M \cdots$, the second inequality also holds. □

LEMMA B.5. The least solution of $\mathcal{D}\vec{f}_{\vec{y}}|_{\vec{u}}(\vec{Y}) \oplus \vec{v} = \vec{Y}$ under γ' is $((\mathcal{D}\vec{f}_{\vec{y}}|_{\vec{u}}(\cdot))_{\gamma'})^{\otimes}(\vec{v})$, where $\vec{g}^{\otimes}(\vec{v}) \triangleq \bigoplus_{M, n \in \mathbb{N}} \vec{g}^n(\vec{v})$.

PROOF. Set $\vec{g}(\vec{Y}) \triangleq \mathcal{D}\vec{f}_{\vec{y}}|_{\vec{u}}(\vec{Y}) \oplus \vec{v}$. Thus $\vec{g}_{\gamma'}(\vec{x}) = (\mathcal{D}\vec{f}_{\vec{y}}|_{\vec{u}}(\vec{x}))_{\gamma'} \oplus_M \vec{v}$. By Kleene's fixed-point theorem, the least solution of $\vec{g}(\vec{Y}) = \vec{Y}$ under γ' is given by $\bigsqcup_{n \in \mathbb{N}} \vec{g}_{\gamma'}^n(\vec{0}_M) = ((\mathcal{D}\vec{f}_{\vec{y}}|_{\vec{u}}(\cdot))_{\gamma'})^{\otimes}(\vec{v})$. □

THEOREM (THM. 4.12). Let \vec{f} be a vector of regular hyper-path expressions in $\text{RegExp}^{\infty}(\mathcal{F}^{\alpha}, \emptyset)$. Then the Newton sequence is monotonically increasing, and it converges to the least fixed-point as least as fast as the Kleene sequence, i.e., for all $i \in \mathbb{N}$, we have

$$\vec{\kappa}^{(i)} \sqsubseteq_M \vec{v}^{(i)} \sqsubseteq_M \vec{f}_{\{\}}^{(i)}(\vec{v}^{(i)}) \sqsubseteq_M \vec{v}^{(i+1)} \sqsubseteq_M \text{lfp}_{\underline{0}_M}^{\sqsubseteq_M} \vec{f}_{\{\}} = \bigsqcup_{j \in \mathbb{N}}^{\uparrow} \vec{\kappa}^{(j)},$$

where the Kleene sequence is defined as $\vec{\kappa}^{(j)} \triangleq \vec{f}_{\{\}}^j(\vec{0}_M)$ for $j \in \mathbb{N}$.

PROOF. We proceed by induction on i . The base case $i = 0$ is straightforward. For the induction step, let $i \geq 0$. Then we have

$$\begin{aligned}
\vec{\kappa}^{(i+1)} &= \vec{f}_{\{\}}(\vec{\kappa}^{(i)}) \\
&\sqsubseteq_M \vec{f}_{\{\}}(\vec{v}^{(i)}) && \text{(induction: } \vec{\kappa}^{(i)} \sqsubseteq_M \vec{v}^{(i)}) \\
&= \vec{v}^{(i)} \oplus_M \vec{\delta}^{(i)} && (\vec{\delta}^{(i)} \triangleq \vec{f}_{\{\}}(\vec{v}^{(i)}) \ominus_M \vec{v}^{(i)}) \\
&\sqsubseteq_M \vec{v}^{(i)} \oplus_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))_{\{\}}^{\otimes}(\vec{\delta}^{(i)}) && (\vec{\delta} \sqsubseteq_M \vec{g}^{\otimes}(\vec{\delta})) \\
&= \vec{v}^{(i+1)} && \text{(Lem. B.5)} \\
&= \vec{v}^{(i)} \oplus_M \vec{\delta}^{(i)} \oplus (\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))_{\{\}}((\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))_{\{\}}^{\otimes}(\vec{\delta}^{(i)})) && (\vec{g}^{\otimes}(\vec{\delta}) = \vec{\delta} \oplus_M \vec{g}(\vec{g}^{\otimes}(\vec{\delta}))) \\
&= \vec{f}_{\{\}}(\vec{v}^{(i)}) \oplus_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))_{\{\}}((\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))_{\{\}}^{\otimes}(\vec{\delta}^{(i)})) \\
&\sqsubseteq_M \vec{f}_{\{\}}(\vec{v}^{(i)} \oplus_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))_{\{\}}^{\otimes}(\vec{\delta}^{(i)})) && \text{(Lem. B.3)} \\
&= \vec{f}_{\{\}}(\vec{v}^{(i+1)}). && \text{(Lem. B.5)}
\end{aligned}$$

Next, we prove $\vec{f}_{\{\}}(\vec{v}^{(i+1)}) \sqsubseteq_M \vec{v}^{(i+2)}$:

$$\begin{aligned}
\vec{f}_{\{\}}(\vec{v}^{(i+1)}) &= \vec{v}^{(i+1)} \oplus_M \vec{\delta}^{(i+1)} && (\vec{\delta}^{(i+1)} \triangleq \vec{f}_{\{\}}(\vec{v}^{(i+1)}) \ominus_M \vec{v}^{(i+1)}) \\
&\sqsubseteq_M \vec{v}^{(i+1)} \oplus_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))_{\{\}}^{\otimes}(\vec{\delta}^{(i+1)}) && (\vec{\delta} \sqsubseteq_M \vec{g}^{\otimes}(\vec{\delta})) \\
&= \vec{v}^{(i+2)}. && \text{(Lem. B.5)}
\end{aligned}$$

By Kleene's fixed-point theorem, we know that $\text{lfp}_{\vec{0}_M}^{\sqsubseteq_M} \vec{f}_{\{\}} = \bigsqcup_{j \in \mathbb{N}} \vec{\kappa}^{(j)}$. It remains to show that $\vec{v}^{(i)} \sqsubseteq_M \text{lfp}_{\vec{0}_M}^{\sqsubseteq_M} \vec{f}_{\{\}}$ for all i . We proceed by induction on i . The base case $i = 0$ is trivial. Let $i \geq 0$. We have

$$\begin{aligned}
\vec{v}^{(i+1)} &= \vec{v}^{(i)} \oplus_M (\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))_{\{\}}^{\otimes}(\vec{\delta}^{(i)}) && \text{(Lem. B.5)} \\
&= \vec{v}^{(i)} \oplus_M \bigoplus_{M, d \in \mathbb{N}} (\mathcal{D}\vec{f}_{\{\}}|_{\vec{v}^{(i)}}(\cdot))^d(\vec{\delta}^{(i)}) \\
&\sqsubseteq_M \vec{v}^{(i)} \oplus_M \bigoplus_{M, d \in \mathbb{N}} \vec{e}^{(d)}(\vec{v}^{(i)}) && \text{(Lem. B.4)} \\
&= \bigsqcup_{d \in \mathbb{N}} \vec{f}_{\{\}}^d(\vec{v}^{(i)}) \\
&\sqsubseteq_M \text{lfp}_{\vec{0}_M}^{\sqsubseteq_M} \vec{f}_{\{\}},
\end{aligned}$$

where the last step follows from the induction hypothesis $\vec{v}^{(i)} \sqsubseteq_M \text{lfp}_{\vec{0}_M}^{\sqsubseteq_M} \vec{f}_{\{\}}$, thus $\vec{f}^d(\vec{v}^{(i)}) \sqsubseteq_M \text{lfp}_{\vec{0}_M}^{\sqsubseteq_M} \vec{f}_{\{\}}$ for any $d \in \mathbb{N}$. \square

THEOREM (THM. 4.16). *Let $\mathcal{C} = \langle \mathcal{C}, \llbracket \cdot \rrbracket^{\mathcal{C}} \rangle$ be an MA interpretation. Let $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$ be an ω PMA interpretation. Let $\Vdash \subseteq \mathcal{C} \times \mathcal{M}$ be a soundness relation. Then for any regular hyper-path expression $E \in \text{RegExp}^{\infty}(\mathcal{F}, \mathcal{K})$, $\gamma : \mathcal{K} \rightarrow \mathcal{C}$, $\gamma^{\#} : \mathcal{K} \rightarrow \mathcal{M}$ such that $\gamma(\square) \Vdash \gamma^{\#}(\square)$ for all $\square \in \mathcal{K}$, $\vec{v} : \mathcal{C}^n$, and $\vec{v}^{\#} : \mathcal{M}^n$ such that $v_i \Vdash v_i^{\#}$ for $i = 1, \dots, n$, we have $\mathcal{C}_{\gamma} \llbracket E \rrbracket(\vec{v}) \Vdash \mathcal{M}_{\gamma^{\#}} \llbracket E \rrbracket(\vec{v}^{\#})$.*

PROOF. By induction on the structure of E .

Case $E = \varepsilon$:

We have $\mathcal{C}_{\gamma} \llbracket E \rrbracket(\vec{v}) = \underline{1}_{\mathcal{C}}$.

We have $\mathcal{M}_{Y^\#}[E](\vec{v}^\#) = \underline{1}_M$.

By Defn. C.2, we conclude that $\underline{1}_C \Vdash \underline{1}_M$.

Case $E = \text{seq}[\text{act}](E_1)$:

We have $\mathcal{C}_Y[E](\vec{v}) = \llbracket \text{act} \rrbracket^{\mathcal{C}} \otimes_C \mathcal{C}_Y[E_1](\vec{v})$.

We have $\mathcal{M}_{Y^\#}[E](\vec{v}^\#) = \llbracket \text{act} \rrbracket^{\mathcal{M}} \otimes_M \mathcal{M}_{Y^\#}[E_1](\vec{v}^\#)$.

By induction hypothesis, we have $\mathcal{C}_Y[E_1](\vec{v}) \Vdash \mathcal{M}_{Y^\#}[E_1](\vec{v}^\#)$.

By Defn. C.2, we have $\llbracket \text{act} \rrbracket^{\mathcal{C}} \Vdash \llbracket \text{act} \rrbracket^{\mathcal{M}}$ and conclude by the fact that \Vdash preserves extend operation.

Case $E = \text{call}[X_i](E_1)$:

We have $\mathcal{C}_Y[E](\vec{v}) = v_i \otimes_C \mathcal{C}_Y[E_1](\vec{v})$.

We have $\mathcal{M}_{Y^\#}[E](\vec{v}^\#) = v_i^\# \otimes_M \mathcal{M}_{Y^\#}[E_1](\vec{v}^\#)$.

By induction hypothesis, we have $\mathcal{C}_Y[E_1](\vec{v}) \Vdash \mathcal{M}_{Y^\#}[E_1](\vec{v}^\#)$.

By assumption, we have $v_i \Vdash v_i^\#$ and conclude by the fact that \Vdash preserves extend operation.

Case $E = \text{cond}[\varphi](E_1, E_2)$:

We have $\mathcal{C}_Y[E](\vec{v}) = \mathcal{C}_Y[E_1](\vec{v}) \varphi \diamond_C \mathcal{C}_Y[E_2](\vec{v})$.

We have $\mathcal{M}_{Y^\#}[E](\vec{v}^\#) = \mathcal{M}_{Y^\#}[E_1](\vec{v}^\#) \varphi \diamond_M \mathcal{M}_{Y^\#}[E_2](\vec{v}^\#)$.

By induction hypothesis, we have $\mathcal{C}_Y[E_1](\vec{v}) \Vdash \mathcal{M}_{Y^\#}[E_1](\vec{v}^\#)$ and $\mathcal{C}_Y[E_2](\vec{v}) \Vdash \mathcal{M}_{Y^\#}[E_2](\vec{v}^\#)$.

We then conclude by the fact that \Vdash preserves conditional-choice operation.

Cases $\text{prob}[p](E_1, E_2)$ and $\text{ndet}(E_1, E_2)$ are similar to this case.

Case $E = \square$:

We have $\mathcal{C}_Y[E](\vec{v}) = \gamma(\square)$.

We have $\mathcal{M}_{Y^\#}[E](\vec{v}) = \gamma^\#(\square)$.

We conclude by the assumption $\gamma(\square) \Vdash \gamma^\#(\square)$.

Case $E = E_1 \cdot_\square E_2$:

We have $\mathcal{C}_Y[E](\vec{v}) = \mathcal{C}_{Y[\square \mapsto \mathcal{C}_Y[E_2](\vec{v})]}[E_1](\vec{v})$.

We have $\mathcal{M}_{Y^\#}[E](\vec{v}^\#) = \mathcal{M}_{Y^\#[\square \mapsto \mathcal{M}_{Y^\#}[E_2](\vec{v}^\#)]}[E_1](\vec{v}^\#)$.

By induction hypothesis, we have $\mathcal{C}_Y[E_2](\vec{v}) \Vdash \mathcal{M}_{Y^\#}[E_2](\vec{v}^\#)$.

Thus, $\gamma[\square \mapsto \mathcal{C}_Y[E_2](\vec{v})] \Vdash \gamma^\#[\square \mapsto \mathcal{M}_{Y^\#}[E_2](\vec{v}^\#)]$.

By induction hypothesis, we have $\mathcal{C}_{Y[\square \mapsto \mathcal{C}_Y[E_2](\vec{v})]}[E_1](\vec{v}) \Vdash \mathcal{M}_{Y^\#[\square \mapsto \mathcal{M}_{Y^\#}[E_2](\vec{v}^\#)]}[E_1](\vec{v}^\#)$ and then conclude this case.

Case $E = (E_1)^{\infty_\square}$:

We have $\mathcal{C}_Y[E](\vec{v}) = \text{lfp}_{0_C}^{\text{EC}} \lambda \theta. \mathcal{C}_{Y[\square \mapsto \theta]}[E_1](\vec{v})$. Define $f(\theta) \triangleq \mathcal{C}_{Y[\square \mapsto \theta]}[E_1](\vec{v})$.

We have $\mathcal{M}_{Y^\#}[E](\vec{v}^\#) = \text{lfp}_{0_M}^{\text{EM}} \lambda \theta^\#. \mathcal{M}_{Y^\#[\square \mapsto \theta^\#]}[E_1](\vec{v}^\#)$. Define $g(\theta^\#) \triangleq \mathcal{M}_{Y^\#[\square \mapsto \theta^\#]}[E_1](\vec{v}^\#)$.

For any $\theta \Vdash \theta^\#$, by induction hypothesis, we have $f(\theta) \Vdash g(\theta^\#)$.

Thus, by Defn. C.2, we have $\text{lfp}_{0_C}^{\text{EC}} f \Vdash \text{lfp}_{0_M}^{\text{EM}} g$ and conclude this case.

□

C SOUNDNESS OF NPA-PMA

In this section, we prove the soundness of NPA-PMA. We use a recently proposed family of algebraic structures, namely *Markov algebras* (MAs) [Wang et al. 2018], to specify concrete semantics of probabilistic programs. We then introduce *soundness relations* between MAs and ω PMAs and show those relations guarantee the soundness of program analyses in NPA-PMA. We use some standard notions from domain theory: directed-complete partial order (dcpo) and Scott-continuous function.

Semantic foundation. We use the interpretation of regular infinite-tree expressions over MAs to define concrete semantics of probabilistic programs. A *Markov algebra* (MA) $\mathcal{M} = \langle M, \sqsubseteq_M, \otimes_M, \varphi \diamond_M, p \oplus_M, \uplus_M, \underline{0}_M, \underline{1}_M \rangle$ over a set \mathcal{L} of logical conditions is almost the same as an ω PMA, except that it does not contain the \oplus_M operation but has an explicit partial order \sqsubseteq_M , as well as it satisfies a different set of algebraic laws: $\langle M, \sqsubseteq_M \rangle$ forms a dcpo with $\underline{0}_M$ as its least element; $\langle M, \otimes_M, \underline{1}_M \rangle$ forms a monoid; \uplus_M is idempotent, commutative, associative and for all $a, b \in M$ and $\varphi \in \mathcal{L}, p \in [0, 1]$ it holds that $a \varphi \diamond_M b, a p \oplus_M b \leq_M a \uplus_M b$, where \leq_M is the semilattice ordering induced by \uplus_M (i.e., $a \leq_M b$ if $a \uplus_M b = b$); and $\otimes_M, \varphi \diamond_M, p \oplus_M, \uplus_M$ are Scott-continuous. An MA interpretation is then a pair $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$ where $\llbracket \cdot \rrbracket^{\mathcal{M}}$ maps data actions to \mathcal{M} .

We consider probabilistic programs represented as an equation system $\{X_i = E_{X_i}\}_{i=1}^n$, where the i^{th} procedure has name X_i , $E_{X_i} \in \text{RegExp}^\infty(\mathcal{F}, \emptyset)$ encodes the regular infinite-tree expression through the CFHG of X_i . Given a regular infinite-tree expression $E \in \text{RegExp}^\infty(\mathcal{F}, \mathcal{K})$, an MA interpretation $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$, a valuation $\gamma : \mathcal{K} \rightarrow \mathcal{M}$, and a procedure-summary vector $\vec{v} \in \mathcal{M}^n$, the interpretation of E under γ and \vec{v} , denoted by $\mathcal{M} \llbracket E \rrbracket(\vec{v})$, can be defined in the same way as the ω PMA interpretations presented in §4.2. We still define the interpretation for μ -binders using least fixed-points, because \mathcal{M} is a dcpo.

Example C.1. Consider probabilistic Boolean programs with a set Var of program variables. Let $S \triangleq 2^{\text{Var}}$ denote the state space of such programs. We formulate a demonic denotational semantics [McIver and Morgan 2005] by defining an MA interpretation. Let $\bar{S} \triangleq \{\mu : S \rightarrow [0, 1] \mid \sum_{s \in S} \mu(s) \leq 1\}$ denote the set of *sub-distributions* on S . We define a partial order on \bar{S} by $\mu_1 \leq \mu_2 \triangleq \forall s \in S : \mu_1(s) \leq \mu_2(s)$. We write \mathbb{CS} to be the set of non-empty, up-closed, convex, and Cauchy-closed subsets of \bar{S} : a set \mathcal{O} of \bar{S} is said to be *up-closed* if $\mu \in \mathcal{O}$ and $\mu \leq \mu'$ imply $\mu' \in \mathcal{O}$, *convex* if $\mu_1, \mu_2 \in \mathcal{O}$ implies $p \cdot \mu_1 + (1-p) \cdot \mu_2 \in \mathcal{O}$ for any $p \in [0, 1]$, and *Cauchy-closed* if \mathcal{O} is a closed subset of some N -dimensional Euclidean space \mathbb{R}^N (in this example, we have $N = |S|$). McIver and Morgan proved that $\mathbb{HS} \triangleq S \rightarrow \mathbb{CS}$ forms a dcpo with respect to the ordering $r_1 \sqsubseteq_C r_2 \triangleq \forall s \in S : r_1(s) \supseteq r_2(s)$.

For any $r \in \mathbb{HS}$, we can lift r to a Scott-continuous mapping $\widehat{r} : \mathbb{CS} \rightarrow \mathbb{CS}$ as $\widehat{r} \triangleq \lambda \mathcal{O}. \{\bar{f}(\mu) \mid f \in \mathbb{DS}, r \leq f, \mu \in \mathcal{O}\}$, where $\mathbb{DS} \triangleq S \rightarrow \bar{S}$ is the set of state-to-distribution mappings, for any $f \in \mathbb{DS}$ we can lift f to a Scott-continuous mapping $\bar{f} : \bar{S} \rightarrow \bar{S}$ as $\bar{f} \triangleq \lambda \mu. \lambda s'. \sum_{s \in S} f(s)(s') \cdot \mu(s)$, and by $r \leq f$ we mean $\forall s \in S : f(s) \in r(s)$.

We now formulate an MA $\mathcal{C} = \langle \mathbb{HS}, \sqsubseteq_C, \otimes_C, \varphi \diamond_C, p \oplus_C, \uplus_C, \underline{0}_C, \underline{1}_C \rangle$ on \mathbb{HS} as follows, where we write $\varphi(s)$ for the truth value of φ in state s and ite for the if-then-else operator.

$$\begin{aligned} r_1 \otimes_C r_2 &\triangleq \lambda s. \widehat{r}_2(r_1(s)), & r_1 p \oplus_C r_2 &\triangleq \lambda s. \{p \cdot \mu_1 + (1-p) \cdot \mu_2 \mid \mu_1 \in r_1(s), \mu_2 \in r_2(s)\}, \\ r_1 \varphi \diamond_C r_2 &\triangleq \lambda s. \text{ite}(\varphi(s), r_1(s), r_2(s)), & r_1 \uplus_C r_2 &\triangleq \lambda s. \{q \cdot \mu_1 + (1-q) \cdot \mu_2 \mid q \in [0, 1], \mu_1 \in r_1(s), \mu_2 \in r_2(s)\}, \\ \underline{0}_C &\triangleq \lambda s. \bar{S}, & \underline{1}_C &\triangleq \lambda s. \{\lambda s'. \text{ite}(s = s', 1, 0)\}. \end{aligned}$$

Soundness relations. We adapt abstract interpretation [Cousot and Cousot 1977] to justify NPA-PMA by establishing a relation between the concrete and abstract semantics. We express the relation via a *soundness relation*, which is a binary relation between an MA interpretation and an ω PMA interpretation that is preserved by the algebraic operations. Intuitively, a (concrete, abstract) pair in the relation should be read as “the concrete element is approximated by the abstract element.”

Definition C.2. Let $\mathcal{C} = \langle \mathcal{C}, \llbracket \cdot \rrbracket^{\mathcal{C}} \rangle$ be an MA interpretation and $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$ be an ω PMA interpretation. We say $- \Vdash - \subseteq \mathcal{C} \times \mathcal{M}$ is a *soundness relation*, if $\llbracket \text{act} \rrbracket^{\mathcal{C}} \Vdash \llbracket \text{act} \rrbracket^{\mathcal{M}}$ for all data actions $\text{act} \in \mathcal{A}$, $\underline{0}_C \Vdash \underline{0}_M$, $\underline{1}_C \Vdash \underline{1}_M$, and for all $x_1 \Vdash y_1$ and $x_2 \Vdash y_2$ we have

- $x_1 \otimes_C x_2 \Vdash y_1 \otimes_M y_2$, $x_1 \varphi \diamond_C x_2 \Vdash y_1 \varphi \diamond_M y_2$ for all $\varphi \in \mathcal{L}$, $x_1 p \oplus_C x_2 \Vdash y_1 p \oplus_M y_2$ for all $p \in [0, 1]$, $x_1 \uplus_C x_2 \Vdash y_1 \uplus_M y_2$, and

- for any Scott-continuous $\vec{f} : \vec{C} \rightarrow \vec{C}$ and Scott-continuous $\vec{g} : \vec{D} \rightarrow \vec{D}$, the property “ $\vec{x} \Vdash \vec{y}$ implies $\vec{f}(\vec{x}) \Vdash \vec{g}(\vec{y})$ for all \vec{x}, \vec{y} ” implies $\text{lfp}_{\vec{C}}^{\vec{f}} \Vdash \text{lfp}_{\vec{D}}^{\vec{g}}$.

Example C.3. Let $\mathcal{C} = \langle \mathcal{C}, \llbracket \cdot \rrbracket^{\mathcal{C}} \rangle$ be an MA interpretation where \mathcal{C} is an MA for probabilistic Boolean programs, which is introduced in Ex. C.1. Let $\mathcal{B} = \langle \mathcal{B}, \llbracket \cdot \rrbracket^{\mathcal{B}} \rangle$ be an ω PMA interpretation where \mathcal{B} is the ω PMA over matrices described in §3.3. We define the following approximation relation to indicate that the program analysis reasons about lower bounds:

$$r \Vdash \mathbf{A} \iff \forall s \in S : \forall \mu \in r(s) : \forall s' \in S : \mu(s') \geq \mathbf{A}(s, s').$$

The correctness of an interpretation is then justified by the following soundness theorem, which followed by induction on the structure of regular infinite-tree expressions.

THEOREM (THM. 4.16). *Let $\mathcal{C} = \langle \mathcal{C}, \llbracket \cdot \rrbracket^{\mathcal{C}} \rangle$ be an MA interpretation. Let $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$ be an ω PMA interpretation. Let $\Vdash \subseteq \mathcal{C} \times \mathcal{M}$ be a soundness relation. Then for any regular infinite-tree expression $E \in \text{RegExp}^{\infty}(\mathcal{F}, \mathcal{K})$, $\gamma : \mathcal{K} \rightarrow \mathcal{C}$, $\gamma^{\#} : \mathcal{K} \rightarrow \mathcal{M}$ such that $\gamma(Z) \Vdash \gamma^{\#}(Z)$ for all $Z \in \mathcal{K}$, $\vec{v} : \mathcal{C}^n$, and $\vec{v}^{\#} : \mathcal{M}^n$ such that $v_i \Vdash v_i^{\#}$ for $i = 1, \dots, n$, we have $\mathcal{C}_{\gamma}[E](\vec{v}) \Vdash \mathcal{M}_{\gamma^{\#}}[E](\vec{v}^{\#})$.*

Let $\mathcal{C} = \langle \mathcal{C}, \llbracket \cdot \rrbracket^{\mathcal{C}} \rangle$ be an MA interpretation and $\mathcal{M} = \langle \mathcal{M}, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$ be an ω PMA interpretation. By Thm. 4.16 and the last property of Defn. C.2, we can show that if $P \triangleq \{X_i = E_{X_i}\}_{i=1}^n$ is an interprocedural equation system where $E_{X_i} \in \text{RegExp}^{\infty}(\mathcal{F}, \emptyset)$ for each $i = 1, \dots, n$, it holds that

$$\text{lfp}_{\vec{C}}^{\vec{f}} \lambda \vec{\theta}. \langle \mathcal{C}_{\{\}}[E_{X_i}](\vec{\theta}) \rangle_{i=1, \dots, n} \Vdash \text{lfp}_{\vec{M}}^{\vec{f}} \lambda \vec{\theta}^{\#}. \langle \mathcal{M}_{\{\}}[E_{X_i}](\vec{\theta}^{\#}) \rangle_{i=1, \dots, n}, \quad (9)$$

where the left-hand side of Eqn. (9) stands for the concrete semantics of the procedures and the right-hand side of Eqn. (9) is their corresponding abstractions, i.e., $\mathcal{C}[P] \Vdash \mathcal{M}[P]$.

Our NPA-PMA framework first translates each E_{X_i} to an equivalent algebraic expression E'_{X_i} by Lem. 4.3, and then applies Newton’s method (Defn. 4.11 and Cor. 4.13) to approximate $\text{lfp}_{\vec{M}}^{\vec{f}} \lambda \vec{\theta}^{\#}. \langle \mathcal{M}_{\{\}}[E'_{X_i}](\vec{\theta}^{\#}) \rangle_{i=1, \dots, n}$, which is equivalent to the right-hand side of Eqn. (9).

D CASE STUDY: BAYESIAN-INFERENC ANALYSIS VIA ALGEBRAIC DECISION DIAGRAMS

In §3.3 and §5.1, we discussed an instantiation of NPA-PMA for Bayesian-inference analysis via matrices that encode distribution transformers. The matrix-based encoding leads to a very high computational complexity because the instantiation needs 2^n -by- 2^n matrices to analyze programs with n Boolean program variables. Recall that in this article, we consider the style of Bayesian-inference analysis from prior work by Claret et al. [2013]. Their implementation uses *Algebraic Decision Diagrams* (ADDs) [Bahar et al. 1997] to represent distributions compactly. In this section, we develop an instantiation that uses ADDs to represent distribution transformers and carry out Bayesian-inference analysis of probabilistic programs *without* nondeterminism.

Let Var be a set of Boolean program variables. As discussed in §3.3, distribution transformers are mappings from states (2^{Var}) to state-distributions ($2^{\text{Var}} \rightarrow [0, 1]$) and can be encoded as matrices of type $2^{\text{Var}} \times 2^{\text{Var}} \rightarrow [0, 1]$. In this section, we introduce a set Var' that consists of primed copies of the variables in Var and change the type of matrices to

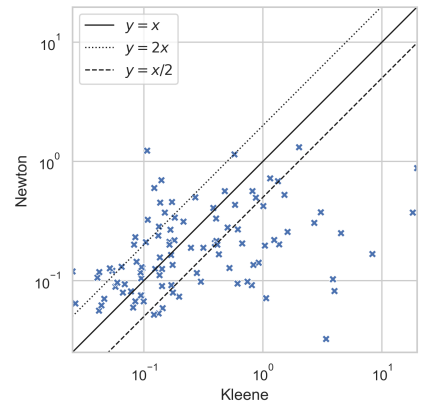


Fig. 14. Log-log scatter plots for running times (seconds) of Kleene iteration and NPA-PMA.

$2^{\text{Var}} \times 2^{\text{Var}'} \rightarrow [0, 1]$. The primed copies in Var' represent the values of the variables in the *post*-state of transformers and the unprimed original ones represent the *pre*-state. We then use ADDs as a compact representation of mappings in $2^{\text{Var} \cup \text{Var}'} \rightarrow [0, 1]$. ADDs have been shown capable of performing many matrix operations including pointwise addition/subtraction, matrix multiplication, and matrix inversion (via Gaussian elimination) [Bahar et al. 1997]. Therefore, to simplify the presentation in this section, we still use matrices to describe our development and the all the matrix operations are realizable using ADDs.

Recall that without nondeterminism, the linear-recursion-solving strategy for Bayesian-inference analysis essentially solves a system of linear matrix equations, i.e., each equation has the form $Z_j = C_j + \sum_{i,k} (A_{i,j,k} \cdot Z_i \cdot B_{i,j,k})$, where $A_{i,j,k}, B_{i,j,k}, C_j$ are known matrices. The matrices of type $2^{\text{Var}} \times 2^{\text{Var}'} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ admit an ω -continuous semiring with pointwise addition and matrix multiplication as the combine and extend operations, respectively. In addition, the Kleene-star of a matrix A satisfying that $\|A\| < 1$ can be computed effectively via matrix inversion, where $\|\cdot\|$ denotes the Frobenius norm and I is the identity matrix:

$$A^{\otimes} \triangleq \sum_{i \in \mathbb{N}} A^i = (I - A)^{-1}.$$

We then adopt a technique proposed by Repts et al. [2016] for solving linear equation systems over ω -continuous semirings. Their technique transforms a linear equation system to a *regular* one, i.e., each equation has the form $W_j = E_j + \sum_i (D_{i,j} \cdot W_i)$, via a *tensor-product* operation \odot and a *detensor-transpose* operation ζ . The obtained regular system is then solved by Tarjan [1981b]'s path-expression algorithm. In our development, the tensor-product and the detensor-transpose operations are defined as follows, where R and S are N -by- N matrices and T is an N^2 -by- N^2 matrix:

$$R \odot S \triangleq \begin{pmatrix} r_{1,1}S & \dots & r_{1,N}S \\ \vdots & \ddots & \vdots \\ r_{N,1}S & \dots & r_{N,N}S \end{pmatrix}, \quad \zeta(T) \triangleq \left(\sum_{k=1}^N T_{(i-1)N+j, (k-1)N+k} \right)_{i=1, \dots, N; j=1, \dots, N}.$$

The detensor-transpose operation satisfies $\zeta(A \odot B) = A \cdot B^T$ for matrices A and B . For a linear matrix equation $Z_j = C_j + \sum_{i,k} (A_{i,j,k} \cdot Z_i \cdot B_{i,j,k})$, we transform it to the regular equation

$$W_j = (I \odot C_j^T + \sum_{i,k} ((A_{i,j,k} \odot B_{i,j,k}^T) \cdot W_i),$$

and then the solution to the original linear system can be recovered from the solution to the obtained regular system via $Z_j = \zeta(W_j)$.

Example D.1. Consider the following linear matrix equation where Z is a 2-by-2 matrix:

$$Z = \begin{pmatrix} 0.2 & 0.3 \\ 0.1 & 0.4 \end{pmatrix} + \begin{pmatrix} 0.2 & 0.3 \\ 0.4 & 0.1 \end{pmatrix} \cdot Z \cdot \begin{pmatrix} 0.1 & 0.9 \\ 0.8 & 0.2 \end{pmatrix}.$$

Using the tensor-product operation, we transform it to a regular equation:

$$W = \begin{pmatrix} 0.2 & 0.1 & 0 & 0 \\ 0.3 & 0.4 & 0 & 0 \\ 0 & 0 & 0.2 & 0.1 \\ 0 & 0 & 0.3 & 0.4 \end{pmatrix} + \begin{pmatrix} 0.02 & 0.16 & 0.03 & 0.24 \\ 0.18 & 0.04 & 0.27 & 0.06 \\ 0.04 & 0.32 & 0.01 & 0.08 \\ 0.36 & 0.08 & 0.09 & 0.02 \end{pmatrix} \cdot W.$$

The solution to the regular equation on W can be obtained by

$$W = \begin{pmatrix} 0.02 & 0.16 & 0.03 & 0.24 \\ 0.18 & 0.04 & 0.27 & 0.06 \\ 0.04 & 0.32 & 0.01 & 0.08 \\ 0.36 & 0.08 & 0.09 & 0.02 \end{pmatrix}^{\otimes} \cdot \begin{pmatrix} 0.2 & 0.1 & 0 & 0 \\ 0.3 & 0.4 & 0 & 0 \\ 0 & 0 & 0.2 & 0.1 \\ 0 & 0 & 0.3 & 0.4 \end{pmatrix} = \left(I - \begin{pmatrix} 0.02 & 0.16 & 0.03 & 0.24 \\ 0.18 & 0.04 & 0.27 & 0.06 \\ 0.04 & 0.32 & 0.01 & 0.08 \\ 0.36 & 0.08 & 0.09 & 0.02 \end{pmatrix} \right)^{-1} \cdot \begin{pmatrix} 0.2 & 0.1 & 0 & 0 \\ 0.3 & 0.4 & 0 & 0 \\ 0 & 0 & 0.2 & 0.1 \\ 0 & 0 & 0.3 & 0.4 \end{pmatrix} \approx \begin{pmatrix} 0.32 & 0.23 & 0.12 & 0.14 \\ 0.43 & 0.52 & 0.13 & 0.11 \\ 0.17 & 0.19 & 0.28 & 0.18 \\ 0.17 & 0.14 & 0.39 & 0.49 \end{pmatrix}.$$

Finally, we use the detensor-transpose operation to obtain the solution to the linear equation on Z :

$$Z = \zeta(W) = \begin{pmatrix} W_{1,1}+W_{1,4} & W_{2,1}+W_{2,4} \\ W_{3,1}+W_{3,4} & W_{4,1}+W_{4,4} \end{pmatrix} \approx \begin{pmatrix} 0.46 & 0.54 \\ 0.35 & 0.65 \end{pmatrix}.$$

Our implementation of the abstract domain for ADD-based Bayesian-inference analysis consists of about 500 lines of code (excluding the CUDD package, which implements basic ADD operations). We evaluated the performance of NPA-PMA against Kleene iteration on the same benchmark suite of 100 randomly generated programs as the suite mentioned in §5.1. The average number of Newton rounds performed by NPA-PMA is 9.58, whereas the average number of Kleene rounds is 3,071.08. Fig. 14 presents a scatter plot that compares the running time of Kleene iteration against

NPA-PMA. In terms of running time, NPA-PMA achieves an overall 1.54x speedup compared with Kleene iteration. Note that although the linear-recursion-solving strategy involves computationally heavy tensor-product and detensor-transpose operations, the significant reduction in the number of iterations ($3,071.08 \rightarrow 9.58$) leads to an actual speedup. This experimental result is much better than that of prior work by Reps et al. [2016], from which we adopt the technique of solving linear equation systems via tensor products. Reps et al. used tensor products in the NPA framework for analyzing non-probabilistic programs and developed an extension of NPA named NPA-TP. However, in their experimental evaluation, NPA-TP turns out to be slower than chaotic iteration on a BDD-based predicate-abstraction domain. In our work, we think it is the quantitative nature of probabilistic programs that unleashes the potential of Newton's method.

E CASE STUDY: HIGHER-MOMENT ANALYSIS OF ACCUMULATED REWARDS

In §2.2, we demonstrated the interprocedural part of NPA-PMA using the termination-probability analysis of finite-state probabilistic programs with nondeterminism. In this section, we augment the programs with a global reward accumulator, which can be incremented via the data action **reward**(c), where $c \in \mathbb{N}$ is a nonnegative number. Reasoning about accumulated rewards of probabilistic programs has many applications (e.g., expected runtime analysis [Esparza et al. 2005; Kaminski et al. 2016; Ngo et al. 2018], expected accumulated-cost analysis [Etessami et al. 2008; Wang et al. 2019a], and tail-bound analysis [Kura et al. 2019; Wang et al. 2021]). We consider the *higher-moment* analysis of accumulated rewards, where the k^{th} moment of a random variable X is $\mathbb{E}[X^k]$. Note that this analysis is an extension of termination-probability analysis because the zeroth moment of the accumulated reward corresponds to the termination probability.

For analyzing k^{th} moments, we define an ω PMA $\mathcal{R}_k = \langle \overline{\mathbb{R}}_{\geq 0}^{k+1}, \oplus_R, \otimes_R, \varphi_{\Diamond_R}, p \oplus_R, \cup_R, \underline{0}_R, \underline{1}_R \rangle$ as an extension of *expectation semirings* [Li and Eisner 2009] and *moment semirings* [Wang et al. 2021], where $\mathbb{R}_{\geq 0} \triangleq \mathbb{R}_{\geq 0} \cup \{\infty\}$, and the operations and constants are defined as follows.

$$\begin{aligned} \vec{u} \oplus_R \vec{v} &\triangleq \langle u_i + v_i \rangle_{i=0, \dots, k}, & \vec{u} p \oplus_R \vec{v} &\triangleq \langle p \cdot u_i + (1-p) \cdot v_i \rangle_{i=0, \dots, k}, & \underline{0}_R &\triangleq \langle 0, 0, \dots, 0 \rangle, \\ \vec{u} \otimes_R \vec{v} &\triangleq \langle \sum_{j=0}^i \binom{i}{j} \cdot u_j \cdot v_{i-j} \rangle_{i=0, \dots, k}, & \vec{u} \cup_R \vec{v} &\triangleq \langle \max(u_i, v_i) \rangle_{i=0, \dots, k}, & \underline{1}_R &\triangleq \langle 1, 0, \dots, 0 \rangle. \end{aligned}$$

We do not define φ_{\Diamond_R} because the considered finite-state probabilistic programs do not contain conditional branching. The ω -continuous semiring $\langle \overline{\mathbb{R}}_{\geq 0}^{k+1}, \oplus_R, \otimes_R, \underline{0}_R, \underline{1}_R \rangle$ admits the partial order $\vec{u} \sqsubseteq_R \vec{v} \triangleq \vec{u} \leq \vec{v}$ (i.e., pointwise comparison) and the subtraction operation $\vec{u} \ominus_R \vec{v} \triangleq \langle u_i - v_i \rangle_{i=0, \dots, k}$.

We can then formulate an ω PMA interpretation $\mathcal{R} = \langle \mathcal{R}, \llbracket \cdot \rrbracket^{\mathcal{R}} \rangle$, where $\llbracket \text{reward}(c) \rrbracket^{\mathcal{R}} \triangleq \langle c^i \rangle_{i=0, \dots, k}$ for $c \in \mathbb{N}$. To prove soundness, we again use the concrete semantics presented in Ex. C.1 with the state space $S \triangleq \mathbb{N}$. We define the following approximation relation to indicate that the program analysis reasons about upper bounds on the moments of the accumulated reward:

$$r \Vdash \vec{u} \iff \forall n \in \mathbb{N}: \forall \mu \in r(n): \bigwedge_{i=0}^k \sum_{n' \in \mathbb{N}} \mu(n') \cdot (n' - n)^i \leq u_i.$$

The equation system obtained from a linearly recursive program is a max-linear numeric equation system; thus, a linear-recursion-solving strategy solve can be obtained via *linear programming* (LP). Consider a numeric equation system $\vec{Z} = \vec{f}(\vec{Z})$ where each $f_i(\vec{Z})$ is either a linear expression over \vec{Z} or the maximum of two variables $\max(Z_j, Z_k)$ for some j, k . The least solution of $\vec{Z} = \vec{f}(\vec{Z})$ can

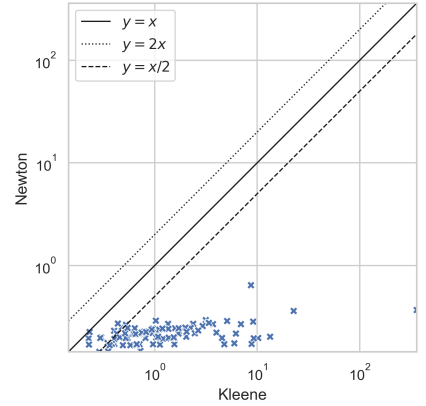


Fig. 15. Log-log scatter plots for running times (seconds) of Kleene iteration and NPA-PMA.

Table 4. Expectation-invariant analysis. (Time is in seconds.)

#	Program	Time	Invariants (selected)
1	2d-walk	0.085399	$\mathbb{E}[dist1'] \leq dist1 + 0.5, \mathbb{E}[x1'] \leq x1 + 0.25$
2	aggregate-rv	0.000688	$\mathbb{E}[i'] \leq i + 1, \mathbb{E}[r'] \leq r + 0.5, \mathbb{E}[x'] \leq x + 0.5$
3	biased-coin	0.001104	$\mathbb{E}[x1'] \leq x1 + 0.5, \mathbb{E}[x2'] \leq x2 + 0.5$
4	binom-update	0.000674	$\mathbb{E}[n'] \leq n + 1, \mathbb{E}[x'] \leq x + 0.25$
5	coupon5	0.003276	$\mathbb{E}[count'] \leq count + 1, \mathbb{E}[i'] \leq i + 0.2$
6	dist	0.000499	$\mathbb{E}[x'] \leq x, \mathbb{E}[y'] \leq y, \mathbb{E}[z'] \leq 0.5x + 0.5y$
7	eg	0.002978	$\mathbb{E}[x'] \leq x + 3, \mathbb{E}[y'] \leq y + 3, \mathbb{E}[z'] \leq 0.25z + 0.75$
8	eg-tail	0.008772	$\mathbb{E}[x'] \leq x + 3, \mathbb{E}[y'] \leq y + 3, \mathbb{E}[z'] \leq 0.25z + 0.75$
9	hare-turtle	0.000657	$\mathbb{E}[h'] \leq h + 2.5, \mathbb{E}[r'] \leq r + 2.5, \mathbb{E}[t'] \leq t + 1$
10	hawk-dove	0.003477	$\mathbb{E}[p1bal1'] \leq p1bal1 + 2, \mathbb{E}[p1bal2'] \leq p1bal2 + 1$
11	mot-ex	0.001039	$\mathbb{E}[r'] \leq r + 0.75, \mathbb{E}[x'] \leq x + 0.75, \mathbb{E}[y'] \leq y + 1.5$
12	recursive	0.001473	$\mathbb{E}[x'] \leq x + 9$
13	uniform-dist	0.000422	$\mathbb{E}[g'] \leq 2g + 0.5, \mathbb{E}[n'] \leq 2n$
14	coupon-five	0.001739	$\mathbb{E}[coupons'] \leq 5, \mathbb{E}[t'] \leq t + 25$
15	coupon-five-fsm	0.005259	$\mathbb{E}[t'] \leq t + 11.4167$
16	dice	0.001575	$\mathbb{E}[r'] \leq 3.5, \mathbb{E}[t'] \leq t + 1.33333$
17	exponential	0.040215	$\mathbb{E}[n'] \leq 1, \mathbb{E}[r'] \leq 0.2$
18	geometric	0.001840	$\mathbb{E}[i'] \leq n, \mathbb{E}[t'] \leq -2560i + 2560n + t$
19	non-linear-recursion	0.001642	$\mathbb{E}[t'] \leq t + 0.333333x + 2.33333, \mathbb{E}[x'] \leq 0.666667x + 0.166667$
20	random-walk	0.001842	$\mathbb{E}[t'] \leq 2n + t + 2x_{neg} - 2x_{pos}, \mathbb{E}[x_{neg}'] \leq 0.5n + 1.5x_{neg} - 0.5x_{pos}$
21	random-walk-uneven	0.001664	$\mathbb{E}[t'] \leq 2n + t + 2x_{neg} - 2x_{pos} + 2, \mathbb{E}[x_{neg}'] \leq n + 2x_{neg} - x_{pos} + 1$
22	unbiased	0.000624	$\mathbb{E}[r'] \leq 1.5, \mathbb{E}[t'] \leq t + 11.1111$

then be obtained via the following LP:

$$\begin{aligned}
 \text{minimize} \quad & \sum_i Z_i, \quad \text{s.t.} \quad Z_i = b_i + \sum_j a_{i,j} \cdot Z_j \quad \text{for each } i \text{ such that } f_i(\vec{Z}) = c_i + \sum_j a_{i,j} \cdot Z_j, \\
 & Z_i \geq Z_j, Z_i \geq Z_k \quad \text{for each } i \text{ such that } f_i(\vec{Z}) = \max(Z_j, Z_k), \\
 & Z_i \geq 0 \quad \text{for each } i.
 \end{aligned}$$

Our implementation of the abstract domain for the higher-moment analysis consists of about 300 lines of code. We evaluated the performance of NPA-PMA for second-moment analysis against Kleene iteration on 100 randomly generated finite-state probabilistic programs, each of which consists of 500 procedures such that each procedure takes one of the following forms: (i) $\text{prob}[p](\text{call}[X_i](\epsilon), \text{call}[X_j](\epsilon))$ for some probability p and procedures X_i, X_j ; (ii) $\text{prob}[p_1](\text{prob}[p_2](\text{call}[X_i](\epsilon), \text{call}[X_j](\epsilon)), \text{seq}[\text{reward}(1)](\epsilon))$ for some probabilities p_1, p_2 and procedures X_i, X_j ; or (iii) $\text{call}[X_i](\text{call}[X_j](\epsilon))$ for some procedures X_i, X_j . On this benchmark suite of 100 programs, the average number of Newton rounds performed by NPA-PMA is 8.99, whereas the average number of Kleene rounds is 9,579.07. Fig. 15 presents a scatter plot that compares the running time (in seconds) of Kleene iteration against NPA-PMA for moment-of-accumulated-reward analysis. We observe that the overall performance of NPA-PMA is better than Kleene iteration with an average speedup of 5.77x.

F CASE STUDY: EXPECTATION-INVARIANT ANALYSIS

Tab. 4 presents the results of the evaluation for the expectation-invariant analysis described in §5.2.

G CASE STUDY: EXPECTATION-RECURRENCE ANALYSIS

Tab. 5 presents the evaluation results for the expectation-recurrence analysis described in §5.3.

Table 5. Expectation-recurrence analysis. (Time is in seconds and “T/O” means “time out after 3 minutes.”)

#	Program	Time (our work / POLAR)	Invariants (selected) derived by our work
1	50coinflips	108.920 / 0.448	$\mathbb{E}[r0'] = 1 - (\frac{1}{2})^n, \mathbb{E}[total'] = 50(1 - (\frac{1}{2})^{n-1})$
2	bimodal-x	0.069 / 0.191	$\mathbb{E}[xlow'] = -\frac{1}{2}n, \mathbb{E}[xup'] = \frac{1}{4}n, \mathbb{E}[x'] = -\frac{1}{4}n$
3	dbn-component-health	0.109 / 0.181	$\mathbb{E}[health'] = (\frac{9}{100})^n, \mathbb{E}[obs'] = \frac{2683}{1000}(\frac{9}{100})^n + \frac{701}{1000}$
4	dbn-umbrella	T/O / 0.456	-
5	gambling	0.054 / 0.079	$\mathbb{E}[bet'] = \frac{1}{2}n + 1, \mathbb{E}[money'] = 0$
6	geometric	0.151 / 0.153	$\mathbb{E}[stop'] = \frac{1}{2}, \mathbb{E}[x'] = 2^n$
7	hawk-dove	0.063 / 0.148	$\mathbb{E}[p1bal'] = n, \mathbb{E}[p2bal'] = n$
8	hermann3	T/O / 0.294	-
9	las-vegas-search	0.059 / 0.288	$\mathbb{E}[found'] = 1 - (\frac{20}{21})^n, \mathbb{E}[attempts'] = \frac{20}{21}n$
10	random-walk-1d	0.039 / 0.077	$\mathbb{E}[x'] = 1$
11	randomized-response	0.093 / 0.168	$\mathbb{E}[n1'] = \frac{5}{4}n - pn, \mathbb{E}[p1'] = \frac{1}{4}n + \frac{1}{2}pn,$ $\mathbb{E}[ntrue'] = \frac{3}{4}n, \mathbb{E}[nfalse'] = \frac{1}{4}n$
12	rock-paper-scissors	0.110 / 0.215	$\mathbb{E}[p1bal'] = n(p2(-q2 - 2q3 + 1) + p3(q2 - q3) + q3),$ $\mathbb{E}[p2bal'] = n(-(p2 - 1)q2 + p2q3 - p3(2q2 + q3 - 1))$
13	lane-keeping	0.087 / 0.193	$\mathbb{E}[z'] = 0, \mathbb{E}[x'] = (\frac{4}{5})^n x, \mathbb{E}[y'] = y + \frac{1}{2}x - \frac{1}{2}(\frac{4}{5})^n x$
14	running-example	0.273 / 0.182	$\mathbb{E}[x'] = \frac{3}{2}n + 1, \mathbb{E}[y'] = -9n - \frac{37}{5}(\frac{2}{3})^n + \frac{67}{5}(\frac{3}{2})^n - 5,$ $\mathbb{E}[z'] = \frac{9}{4}n^2 + \frac{19}{4}n - \frac{201}{10}(\frac{3}{2})^n - \frac{37}{5}(\frac{2}{3})^n + \frac{57}{2}$