CARNEGIE MELLON UNIVERSITY

Static Analysis of Probabilistic Programs: An Algebraic Approach

Thesis Proposal

by

Di Wang

Thesis proposal submitted in partial fulfillment
for the degree of Doctor of Philosophy

Thesis committee:
Jan Hoffmann (chair)
Matt Fredrikson
Stephen Brookes
Thomas Reps (University of Wisconsin-Madison)
Hongfei Fu (Shanghai Jiao Tong University)

August 2021

Abstract

Probabilistic programs are programs that can draw random samples from probability distributions and involve random control flows. They are becoming increasingly popular and have been applied in algorithm design, cryptographic protocols, uncertainty modeling, and statistical inference. Formal reasoning about probabilistic programs comes with unique challenges, because it is usually not tractable to obtain the exact result distributions of probabilistic programs. This thesis focuses on an algebraic approach for static analysis of probabilistic programs. The thesis first provides a brief background on measure theory and introduces an imperative arithmetic probabilistic programming language Appl with a novel hyper-graph program model. Second, the thesis presents an algebraic denotational semantics for Appl with a new model of nondeterminism that involves nondeterminacy among state transformers. The thesis then proposes a general algebraic framework PMAF for designing, implementing, and proving the correctness of static analyses of probabilistic programs. The thesis also includes a concrete static analysis: central-moment analysis for cost accumulators in probabilistic programs. As proposed work, I plan to (i) enhance the implementation for central-moment analysis, and (ii) instantiate the analysis framework PMAF for central-moment reasoning. Furthermore, I will investigate static-analysis techniques for programs used for probabilistic inference to help improve efficiency and detect bugs.

Contents

Chapter 1

Introduction

Probabilistic systems are becoming increasingly popular in computer science, for their capability of improving efficiency (e.g., randomized Quicksort [64]), protecting confidential information (e.g., optimal asymmetric encryption padding [11]), modeling peripheral uncertainty (e.g., airborne collision-avoidance systems [99]), and describing statistical models (e.g., probabilistic graphical models [89]). *Probabilistic programming* provides a framework for implementing and analyzing probabilistic systems, such as randomized algorithms [7], cryptographic protocols [8], cyber-physical systems [17], and machine-learning algorithms [55].

A *probabilistic program* is any program that can draw random samples from probability distributions and involve random control flows. In practice, the source of randomness is usually a *pseudo-random number generator* (e.g., the `rand()` function in C language); however, in this thesis, I focus on *true randomness*, in the sense that given an initial program state, a probabilistic program produces a *distribution* that describes the probability of its computation results. The two reasons why I assume true randomness in my development are that (i) some high-quality random number generators are shown to be sufficiently good approximations of true randomness [72], so the behavior of a program under pseudo-randomness would be conceptually close to its behavior under true randomness, and (ii) compared to pseudo-randomness, there are plenty of well-studied mathematical theories for true randomness, such as measure theory.

As the interest in probabilistic programming is increasing, there has been a corresponding increase in the study of *formal reasoning* about probabilistic programs: *What is the probability that an assertion holds after a program terminates? Is there any expression an invariant under expectation for a probabilistic loop? What is the expected time complexity of a program?* In general, these kinds of reasoning problems are challenging, because it is usually not tractable to compute the result distributions of probabilistic programs precisely: composing simple distributions can quickly complicate the result distribution, and randomness in the control flow can easily lead to state-space explosion. Monte-Carlo simulation [120] is a common approach to analyze probabilistic programs, but the technique does not provide

formal guarantees on the accuracy of analysis results, and can sometimes be inefficient [10].

Static analysis is a longstanding area about (usually automated) techniques for analyzing and proving program properties *without* actually executing the programs. Static analysis of probabilistic programs has also received a lot of attention [19, 20, 25–30, 39, 45–47, 49, 54, 80–82, 114, 122]. In this thesis, I have conducted research on *algebraic* approaches for *compositional* static analysis of probabilistic programs, i.e., designing and implementing static analyses based on *semantic algebra*, which consists of a space of program properties and composition operators that correspond to program constructs.

> **Thesis Statement** Algebraic static analysis helps people reason about probabilistic programs at compile time in a compositional and versatile way. Markov algebras provide a natural way to describe the algebraic structure of probabilistic programs. Based on Markov algebras, a denotational semantics for combining nondeterminism and probabilities lays the foundation for an algebraic framework for static analysis of probabilistic programs.

Denotational Semantics with Nondeterminism The first step in my development of static-analysis techniques is to provide a suitable formal semantics for probabilistic programs. Despite the fact that lots of existing work focuses on *high*-level probabilistic programs, e.g., lambda calculus [16], higher-order functions [44, 63], and recursive types [129], I observe that *low*-level features could arise naturally. For example, when developing a compiler for a probabilistic programming language [52, 115], we need a semantics for the imperative target language to prove compiler correctness. Also, static analysis of low-level code becomes desirable for verifying cross-language programs [136] and detecting security vulnerabilities [1]. There have been studies on *denotational* semantics for *well-structured* imperative programs [13, 73, 80, 91, 92, 101, 102, 114, 127], as well as *operational* semantics for *control-flow graphs* (CFGs) based on Markov chains and Markov decision processes ([28, 29, 49]). On the one hand, I prefer CFGs as program representations because they enable rich low-level features such as *unstructured* flows, e.g., those introduced by **break** and **continue**. On the other hand, from the perspective of formal reasoning, a denotational semantics (i) abstracts from details about program executions and focuses on program *effects*, and (ii) is *compositional* in the sense that the semantics of a program fragment is established from the semantics of the fragment's proper constituents.

Therefore, I devise a denotational semantics for low-level probabilistic programs with nondeterminism. This semantics is published as a standalone paper at the 35^{th} Conference on the *Mathematical Foundations of Programming Semantics* [131]. In that work, I make three main contributions:

- I use *hyper-graphs* as the representation for low-level probabilistic programs with unstructured control-flow, general recursion, and nondeterminism.
- I develop a domain-theoretic characterization of a new model of nondeterminism for probabilistic programming, which involves nondeterminacy among *state transformers*,

opposed to a common model that involves nondeterminacy among *program states*.
- I propose *Markov algebras* and devise an *algebraic* framework for denotational semantics. One advantage of the framework is that it can be instantiated with different models of nondeterminism. I also prove that for programs without procedure calls and nondeterminism, the resulting denotational semantics is equivalent to a standard distribution-based operational semantics.

An Algebraic Framework for Static Analysis Despite the fact that there have been many static analyses for probabilistic programs, these analyses are usually standalone developments, and it is not immediately clear how different techniques relate. For example, Claret et al. [30] propose a syntax-directed data-flow analysis to perform Bayesian inference, Chakarov and Sankaranarayanan [26] develop a martingale-based abstract interpretation to construct expectation invariants, and Sankaranarayanan et al. [122] combine symbolic execution with probabilistic volume-bound computation to estimate the probability that an assertion holds at a program point. Thus, I develop a framework, which I call *PMAF* (for Pre-Markov Algebra Framework), for designing, implementing, and proving the correctness of static analyses of probabilistic programs. The framework is published as a standalone paper at the 39$^{\text{th}}$ Conference on *Programming Language Design and Implementation* [130]. Using PMAF, I can formulate and generalize several analyses that may appear to be quite different. Examples include Bayesian inference [30, 46, 47], Markov decision problem with rewards [116], and probabilistic-invariant generation [26, 81].

PMAF is based on the algebraic denotational semantics that I described earlier in this chapter. To formulate a static analysis, I introduce a new algebraic structure, called a *pre-Markov algebra*, which is equipped with operations corresponding to control-flow actions in probabilistic programs: *sequencing*, *conditional-choice*, *probabilistic-choice*, and *nondeterministic-choice*. To establish correctness, I introduce *probabilistic abstractions* between a pre-Markov algebra—which represents the abstract semantics—and the concrete semantics. This work shows how, with suitable extensions, a blending of ideas from previous work on (i) static analysis of single-procedure probabilistic programs, and (ii) interprocedural dataflow analysis of standard (non-probabilistic) programs can be used to create a framework for interprocedural analysis of probabilistic programs. In particular,

- The semantics on which PMAF is based is an interpretation of the CFGs for a program's procedures. One key idea is to treat each CFG as a *hyper-graph* rather than a standard graph.
- The abstract semantics is formulated so that the analyzer can obtain *procedure summaries*.

The main advantage of PMAF is that instead of starting from scratch to create a new analysis, one only needs to instantiate PMAF with the implementation of a new pre-Markov algebra. To establish soundness, one has to establish some well-defined algebraic properties, and can then rely on the soundness proof of the framework. To implement the analysis, one can rely on PMAF to perform sound interprocedural analysis, with

respect to the provided abstraction. The PMAF implementation supplies common parts of different static analyses of probabilistic programs, e.g., efficient iteration strategies with widenings and interprocedural summarization. Moreover, improvements made to the PMAF implementation could immediately translate into improvements to *all of its instantiations*.

Central Moment Analysis for Cost Accumulators As a concrete static analysis of probabilistic programs, I study a specific yet important kind of uncertain quantity: *cost accumulators*, which are program variables that can only be incremented or decremented through the program execution and do not influence the control flow. Examples of cost accumulators include termination time [10, 27, 28, 80, 114], rewards in Markov decision processes [116], position information in control systems [9, 17, 122], and cash flow during bitcoin mining [137]. Recent work [17, 94, 112, 137] has proposed successful static-analysis approaches that leverage *aggregate* information of a cost accumulator X, such as X's *expected* value $\mathbb{E}[X]$ (i.e., X's "first moment"). The intuition why it is beneficial to compute aggregate information—in lieu of distributions—is that aggregate measures like expectations *abstract* distributions to a single number, while still indicating non-trivial properties. Moreover, expectations are transformed by statements in a probabilistic program in a manner similar to the weakest-precondition transformation of formulas in a non-probabilistic program [102].

One important kind of aggregate information is *moments*. Whereas most previous work focused on *raw* moments (i.e., $\mathbb{E}[X^k]$ for any $k \geq 1$) I find out that *central* moments (i.e., $\mathbb{E}[(X - \mathbb{E}[X])^k]$ for any $k \geq 2$) can usually provide more information about distributions. For example, the *variance* $\mathbb{V}[X]$ (i.e., $\mathbb{E}[(X - \mathbb{E}[X])^2]$, X's "second central moment") indicates how X can deviate from its mean, the *skewness* (i.e., $\frac{\mathbb{E}[(X-\mathbb{E}[X])^3]}{(\mathbb{V}[X])^{\frac{3}{2}}}$, X's "third standardized moment") indicates how lopsided the distribution of X is, and the *kurtosis* (i.e., $\frac{\mathbb{E}[(X-\mathbb{E}[X])^4]}{(\mathbb{V}[X])^2}$, X's "fourth standardized moment") measures the heaviness of the tails of the distribution of X. One application of moments is to answer queries about *tail bounds*, e.g., the assertions about probabilities of the form $\mathbb{P}[X \geq d]$, via *concentration-of-measure* inequalities from probability theory [43]. With central moments, we find an opportunity to obtain more precise tail bounds of the form $\mathbb{P}[X \geq d]$, and become able to derive bounds on tail probabilities of the form $\mathbb{P}[|X - \mathbb{E}[X]| \geq d]$.

In this thesis, I have proposed and implemented the first fully automatic analysis for deriving symbolic *interval* bounds on higher central moments for cost accumulators in probabilistic programs with general recursion and continuous distributions. This analysis is published as a standalone paper at the 42nd Conference on *Programming Language Design and Implementation* [132]. One challenge is to support *inter*procedural reasoning to reuse analysis results for procedures. My solution makes use of a "lifting" technique from the natural-language-processing community. That technique derives an algebra for second moments from an algebra for first moments [97]. I follow the *algebraic* approach and develop *moment semirings*, and use them to carry out interprocedural analysis, as well as derive a novel *frame* rule to handle procedure calls with *moment-polymorphic recursion*.

1.1 Proposed Work

Enhance the Implementation of Central Moment Analysis A successful static analyzer should be both effective, usable, and efficient. I have proved the soundness of my central-moment analysis framework and implemented a working prototype. Despite the effectiveness of the prototype implementation, it lacks satisfactory usability or efficiency.

My current implementation only supports arithmetic imperative programs with real-valued program variables. Towards good usability, I plan to add support for (i) common programming features such as heap manipulation, and (ii) common data types such as integers and Booleans. There are interesting design decisions to consider in supporting heap manipulation. One possible approach is to integrate my implementation with abstract interpreters that deal with heap allocations (e.g., [59, 74]). Such abstract interpreters may be able to derive nontrivial program invariants that involve the contents of the heap, and as a consequence help my central-moment analysis tool derives nontrivial bounds that depend on the heap. Another possibility is to support *semi-automatic* reasoning by allowing the user to provide logical or quantitative program invariants (e.g., [23]). This approach may be more flexible and lightweight than using a full-fledged abstract interpreter. One challenge is that most of the existing approaches to supporting heap target non-probabilistic programs, so I may need to tweak them to effectively handle probabilistic constructs such as probabilistic branching. Meanwhile, to support more data types, I plan to adapt *resource polynomials* from multivariate resource analysis of functional programs (e.g., [66, 67]). The idea is to introduce for each data type τ a set of *base monomials* that map values of type τ to real numbers, and then use those monomials to construct polynomial cost bounds. I also plan to add a data type that encapsulates probability to allow branching with *non-constant* probabilities (e.g., [135]).

The time complexity of my current implementation increases *exponentially* in the number of program variables; such behavior affects the scalability of the central-moment analysis tool on large programs. The exponential blowup is caused by the use of polynomial bounds over program variables: given a fixed maximal degree, the number of possible monomials grows exponentially in the number of program variables. I have observed that in many probabilistic programs, many program variables are only used locally to save random values drawn from probability distributions. Therefore, I plan to use only *needed* program variables to specify the resource bound at each program point. Besides, it is also possible to use compilation techniques such as unused-variable detection and dead-code elimination to reduce code size and speed up the analysis.

Instantiate PMAF for Central Moment Reasoning Although I have developed a general framework PMAF for designing and implementing static analysis of probabilistic programs, my central-moment analysis tool is implemented as a standalone tool. One reason why I do not implement central-moment analysis as a PMAF abstract domain is that the underlying algorithm of PMAF is *iteration* based, i.e., the algorithm starts with an initial solution to

the analysis problem and then iterates until convergence, but my central-moment analysis framework is *constraint* based, i.e., the analysis walks through the analyzed program to collect constraints and then discharges those constraints using an off-the-shelf constraint solver. However, it has been shown that iteration-based and constraint-based analyses can be integrated together to create a more powerful analysis [21]. I plan to instantiate PMAF with a new abstract domain that tracks central moments of cost accumulators in probabilistic programs. There are two possible approaches I will investigate: (i) integrate my constraint-based approach to central-moment analysis as a *loop-summarization* technique and then apply it in each round of an iteration-based algorithm, or (ii) develop recurrence relations for expectations and moments and then use recurrence solvers (e.g., [85, 86]) to derive moment bounds. With an implementation of one of the approaches, I will evaluate if the PMAF instantiation is competitive with my current central-moment analysis tool in terms of analysis precision and efficiency.

Further Proposed Work Probabilistic programs are also used to express and perform *probabilistic inference*, a method of inferring a statistical model from observed data, which is good at capturing uncertainty in model parameters and integrating expert domain knowledge (e.g., *a priori* distributions), but requires considerable expertise from the practitioner [56]. There are a lot of probabilistic programming languages (PPLs) in active development, such as WebPPL [58], Stan [24], Edward [128], Venture [100], Pyro [15], and HackPPL [3]. These PPLs not only provide language features to reduce the difficulty of expressing various models, but also implement efficient inference engines to handle enormous real-world data. However, balancing between *language expressibility* and *inference efficiency* remains a challenge for the design and implementation of PPLs. If time permits, I will develop static-analysis techniques to improve the inference efficiency *without* restricting language expressibility or compromising on high-level language clarity.

1.2 Outline

This thesis is based on three published articles:

- Di Wang, Jan Hoffmann, and Thomas Reps. 2019. A Denotational Semantics for Low-Level Probabilistic Programs with Nondeterminism. In *the 35th Conference on the Mathematical Foundations of Programming Semantics (MFPS'19)*. Chapter 3 presents the denotational semantics.
- Di Wang, Jan Hoffmann, and Thomas Reps. 2018. PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs. In *the 39th Conference on Programming Language Design and Implementation (PLDI'18)*. Chapter 4 presents the algebraic framework.
- Di Wang, Jan Hoffmann, and Thomas Reps. 2021. Central Moment Analysis for Cost Accumulators in Probabilistic Programs. In *the 42nd Conference on Programming*

Language Design and Implementation (PLDI'21). Chapter 5 presents the central moment analysis.

Chapter 2 provides necessary background for understanding this thesis. It first reviews standard notions from measure theory. Then, it introduces an imperative probabilistic programming language Appl that I use in this thesis, and a basic program execution model for the language.

Chapter 3 presents a *denotational* semantics for Appl with a novel *nondeterminism-first* resolution. This chapter develops a domain-theoretic characterization of nondeterminism-first and proposes an algebraic denotation semantics for probabilistic programs.

Chapter 4 designs an algebraic analysis framework, called PMAF, for designing, implementing, and proving the correctness of static analyses of probabilistic programs. The framework is based the algebraic denotational semantics presented in Chapter 3. This chapter also includes three instantiations of PMAF to solve different analysis problems.

Chapter 5 develops a novel static analysis for deriving symbolic interval bounds on higher *central moments* for *cost accumulators* in probabilistic programs. The analysis is based on a novel algebraic structure called *moment semirings*. This chapter proves the soundness of the moment-bound analysis with respect to a Markov-chain cost semantics.

Chapter 6 focuses on proposed work and also describes further proposed work that I will address if time permits. I will enhance the implementation of the central-moment analysis tool to improve its usability and efficiency. I will also instantiate the PMAF analysis framework with a new abstract domain to reason about central moments of cost accumulators. Furthermore, I would like to explore the possibility to apply static analysis to reason about and detect bugs in programs used for probabilistic inference.

Chapter 2

Setting the Stage: An Imperative Probabilistic Programming Language

In this thesis, I use an imperative arithmetic probabilistic programming language Appl that supports unstructured control-flow, general recursion, and nondeterminism, where program variables have numeric values. I use the following notational conventions. Natural numbers $\mathbb{N}$ *exclude* 0, i.e., $\mathbb{N} \stackrel{\text{def}}{=} \{1, 2, 3, \cdots\} \subseteq \mathbb{Z}^+ \stackrel{\text{def}}{=} \{0, 1, 2, \cdots\}$. Reals, nonnegative reals, and extended nonnegative reals are denoted by $\mathbb{R}$, $\mathbb{R}^+$, and $\mathbb{R}^+_\infty$, respectively. Let $\mathbb{2}$ denote the set of Boolean values, i.e., $\mathbb{2} \stackrel{\text{def}}{=} \{\top, \bot\}$. The *Iverson brackets* $[\cdot]$ are defined by $[\varphi] = 1$ if φ is true and otherwise $[\varphi] = 0$. Finite partial maps from A to B can be represented by finite sets of bindings, e.g., $\{a_1 \mapsto b_1, a_2 \mapsto b_2, \cdots, a_n \mapsto b_n\}$. Updating an existing binding of x in a finite map f to v is denoted by $f[x \mapsto v]$.

2.1 Preliminaries on Measure Theory

This section reviews the following standard notions from measure theory: measurable spaces, measurable functions, random variables, probability measures, expectations, and kernels. Interested readers can refer to textbooks in the literature [14, 140] for details about measure theory.

A *measurable space* is a pair $(S, \mathcal{S})$, where S is a nonempty set, and $\mathcal{S}$ is a σ-algebra on S, i.e., a family of subsets of S that contains $\emptyset$ and is closed under complement and countable unions. The smallest σ-algebra that contains a family $\mathcal{A}$ of subsets of S is said to be *generated* by $\mathcal{A}$, denoted by $\sigma(\mathcal{A})$. Every topological space (S, τ) (i.e., $\tau \subseteq \wp(S)$ is a collection of open sets that is closed under arbitrary unions and finite intersections) admits a *Borel σ-algebra*, given by $\sigma(\tau)$. This gives canonical σ-algebras on $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{N}$, etc.

Example 2.1. One of the most important σ-algebras is the Borel σ-algebra on $\mathbb{R}$, and it is a standard shorthand to denote this σ-algebra by $\mathcal{B}$. Elements of $\mathcal{B}$ can be very complicated, so

people have come up with simpler constructions for $\mathcal{B}$, e.g., $\mathcal{B} = \sigma(\{(-\infty, x] \mid x \in \mathbb{R}\})$.

A *measure* μ on a measurable space $(S, \mathcal{S})$ is a mapping from $\mathcal{S}$ to $[0, \infty]$ such that (i) $\mu(\emptyset) = 0$, and (ii) for all pairwise-disjoint $\{A_n\}_{n\in\mathbb{N}}$ in $\mathcal{S}$, it holds that $\mu(\bigcup_{n\in\mathbb{N}} A_n) = \sum_{n\in\mathbb{N}} \mu(A_n)$. The triple $(S, \mathcal{S}, \mu)$ is called a *measure space*. A measure μ is called a *probability* (resp., *sub-probability*) measure, if $\mu(S) = 1$ (resp., $\mu(S) \leq 1$). We denote the collection of probability measures on $(S, \mathcal{S})$ by $\mathbb{D}(S, \mathcal{S})$. The set of sub-probability measures on $(S, \mathcal{S})$ with the pointwise order forms an *ω-complete partial order* (ω-cpo), i.e., every ω-chain of sub-probability measures $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_n \leq \cdots$ has a supremum that is also a sub-probability measure. The *zero measure* $\mathbf{0}$ is defined as $\lambda A.0$. For each $x \in S$, the *Dirac measure* $\delta(x)$ is defined as $\lambda A.[x \in A]$. For measures μ and ν, we write $\mu + \nu$ for the measure $\lambda A.\mu(A) + \nu(A)$. For measure μ and scalar $c \geq 0$, we write $c \cdot \mu$ for the measure $\lambda A.c \cdot \mu(A)$.

Example 2.2. The Lebesgue measure Leb *on $(\mathbb{R}, \mathcal{B})$ makes precise the concept of* length *of a measurable subset of $\mathbb{R}$.* Leb *is defined as the unique measure such that for any A that can be written as a finite union $A = (a_1, b_1] \cup \cdots \cup (a_m, b_m]$ where $m \in \mathbb{N}$ and $a_1 \leq b_1 \leq \cdots \leq a_m \leq b_m$, it holds that* $\mathrm{Leb}(A) = \sum_{i=1}^{m}(b_i - a_i)$.

The counting measure *is commonly used on a measurable space $(S, \mathcal{S})$ where S is finite or countable. The measure μ is defined by $\mu \stackrel{\text{def}}{=} \lambda A.|A|$, where $|A|$ denotes the cardinality of the set A, with the understanding that $\mu(A) = \infty$ if A is infinite.*

A function $f : S \to T$, where $(S, \mathcal{S})$ and $(T, \mathcal{T})$ are measurable spaces, is said to be *$(\mathcal{S}, \mathcal{T})$-measurable*, if $f^{-1}(B) \in \mathcal{S}$ for each $B \in \mathcal{T}$. If $T = \mathbb{R}$, we tacitly assume that the Borel σ-algebra $\mathcal{B}$ is defined on $\mathbb{R}$, and we simply call f *measurable*, or a *random variable*. Measurable functions form a vector space, and products and maxima preserve measurability; that is, for any measurable functions f_1, f_2 and real numbers c_1, c_2, the functions $(c_1 \cdot f_1 + c_2 \cdot f_2)$, $(f_1 \cdot f_2)$, and $\max(f_1, f_2)$ are also measurable.

Example 2.3. Consider an experiment where one tosses a coin infinitely often. We can take $\Omega \stackrel{\text{def}}{=} \{\mathsf{H}, \mathsf{T}\}^{\mathbb{N}}$, so an element ω of Ω is a coin-toss sequence $\omega = \{\omega_n\}_{n\in\mathbb{N}}$, where $\omega_n \in \{\mathsf{H}, \mathsf{T}\}$. We now define a σ-algebra $\mathcal{F}$ on Ω as follows:

$$\mathcal{F} \stackrel{\text{def}}{=} \sigma(\{\{\omega \in \Omega \mid \omega_n = \mathscr{C}\} \mid \mathscr{C} \in \{\mathsf{H}, \mathsf{T}\}, n \in \mathbb{N}\}),$$

which allows us to reason about events such as "the n-th toss shows heads." Then for any $n \in \mathbb{N}$, $X_n \stackrel{\text{def}}{=} \lambda\omega.[\omega_n = \mathsf{H}]$ is a random variable on the measurable space $(\Omega, \mathcal{F})$. Because measurable functions form a vector space, the number of heads in first n tosses $S_n \stackrel{\text{def}}{=} \sum_{i=1}^{n} X_i$ is also a random variable.

The *integral* of a measurable function f on $A \in \mathcal{S}$ with respect to a measure μ on $(S, \mathcal{S})$ is defined following Lebesgue's theory and is denoted by $\mu(f; A)$, $\int_A f d\mu$, or $\int_A f(x)\mu(dx)$. If μ is a probability measure, we call the integral as the *expectation* of f, written $\mathbb{E}_{x\sim\mu}[f; A]$,

or simply $\mathbb{E}[f; A]$ when the scope is clear in the context. If $A = S$, we tacitly omit A from the notations. For each $A \in \mathcal{S}$, it holds that $\mu(f; A) = \mu(f\mathrm{I}_A)$, where $\mathrm{I}_A \stackrel{\text{def}}{=} \lambda x.[x \in A]$ is the indicator function for A. If f is nonnegative, then $\mu(f)$ is well-defined with the understanding that the integral can be infinite. If $\mu(|f|) < \infty$, then f is said to be *integrable*, written $f \in \mathcal{L}^1(S, \mathcal{S}, \mu)$, and its integral is well-defined and $\mu(f) = \mu(f^+) - \mu(f^-)$, where $f^+ \stackrel{\text{def}}{=} \lambda x. \max(f(x), 0)$ and $f^- \stackrel{\text{def}}{=} \lambda x. \max(-f(x), 0)$. Integration is *linear*, in the sense that for any $c, d \in \mathbb{R}$ and integrable functions f, g, $c \cdot f + d \cdot g$ is integrable and $\mu(c \cdot f + d \cdot g) = c \cdot \mu(f) + d \cdot \mu(g)$.

Example 2.4. Recall the coin-toss experiment in Ex. 2.3. If the tossed coin is fair, we can assign a probability measure μ on the measurable space $(\Omega, \mathcal{F})$ such that

$$\mu(\{\omega \in \Omega \mid \omega_n = \mathscr{C}\}) = \frac{1}{2}, \quad \forall \mathscr{C} \in \{\mathsf{H}, \mathsf{T}\}, n \in \mathbb{N}.$$

Then the expectation of the random variable X_n for any n is $\mathbb{E}[X_n] = \sum_{\mathscr{C} \in \{\mathsf{H},\mathsf{T}\}} \frac{1}{2} \cdot [\mathscr{C} = \mathsf{H}] = \frac{1}{2}$ and by linearity, we obtain that for any n, the expectation of S_n is $\mathbb{E}[S_n] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \frac{1}{2}n$.

A *kernel* from a measurable space $(S, \mathcal{S})$ to another $(T, \mathcal{T})$ is a mapping from $S \times \mathcal{T}$ to $[0, \infty]$ such that (i) for each $x \in S$, the set function $\lambda B.\kappa(x, B)$ is a measure on $(T, \mathcal{T})$, and (ii) for each $B \in \mathcal{T}$, the function $\lambda x.\kappa(x, B)$ is measurable. We write $\kappa : (S, \mathcal{S}) \rightsquigarrow (T, \mathcal{T})$ to declare that κ is a kernel from $(S, \mathcal{S})$ to $(T, \mathcal{T})$. Intuitively, kernels describe measure transformers from one measurable space to another. A kernel κ is called a *probability* (resp., *sub-probability*) kernel, if $\kappa(x, T) = 1$ (resp., $\kappa(x, T) \leq 1$) for all $x \in S$. We denote the collection of probability kernels from $(S, \mathcal{S})$ to $(T, \mathcal{T})$ by $\mathbb{K}((S, \mathcal{S}), (T, \mathcal{T}))$. If the two measurable spaces coincide, we simply write $\mathbb{K}(S, \mathcal{S})$. The set of sub-probability kernels from $(S, \mathcal{S})$ to $(T, \mathcal{T})$ with the pointwise order forms an ω-cpo. We can *push-forward* a measure μ on $(S, \mathcal{S})$ to a measure on $(T, \mathcal{T})$ through a kernel $\kappa : (S, \mathcal{S}) \rightsquigarrow (T, \mathcal{T})$ by integration: $\mu \gg\!= \kappa \stackrel{\text{def}}{=} \lambda B. \int_S \kappa(x, B)\mu(dx)$.

Example 2.5. As explained by Kozen [92], for a measurable space $(S, \mathcal{S})$ where S is finite or countable, a probability kernel $\kappa \in \mathbb{K}(S, \mathcal{S})$ has a representation as a Markov transition matrix M_κ, in which each entry $M_\kappa(x, x')$ for a pair of elements $(x, x') \in S^2$ gives the probability that x transitions to x' under the kernel κ, and for any $x \in S$ and $A \in \mathcal{S}$, it holds that $\kappa(x, A) = \sum_{x' \in A} M_\kappa(x, x')$.

2.2 A Hyper-Graph Program Model

In contrast to program models—such as standard control-flow graphs (CFGs)—for deterministic programming languages, I use *control-flow hyper-graphs* (CFHGs) to model probabilistic programs. Hyper-graphs consist of *hyper-edges*, each of which connects one source node and possibly several destination nodes. For example, probabilistic choices are represented by weighted hyper-edges with *two* destinations. Nondeterminism is then represented by

multiple hyper-edges starting in the same node. The interpretation of hyper-edges is also different from standard edges. If the CFHG were treated as a standard graph, the subpaths from each successor of a branching node would be analyzed *independently*. In contrast, the hyper-graph approach interprets a probabilistic-choice hyper-edge with probability p as a function $\lambda a.\lambda b.a \ {}_p\!\oplus\ b$, where ${}_p\!\oplus$ is an operation that weights the subpaths through the two successors by p and $1-p$. In other words, we do not reason about subpaths starting from a node *individually*, instead we analyze these subpaths *jointly* as a probability distribution. If a node has two outgoing probabilistic-choice hyper-edges, it represents two "worlds" of subpaths, each of which carries a probability distribution with respect to the probabilistic choice made in this "world."

To define CFHGs of probabilistic programs, I adopt a common approach for standard CFGs in which the nodes represent program locations, and edges labeled with instructions describe transitions among program locations (e.g., [48, 96, 111]). Instead of standard directed graphs, I make use of *hyper-graphs* [53].

Definition 2.6. A *hyper-graph* H is a quadruple $\langle V, E, v^{\mathsf{entry}}, v^{\mathsf{exit}}\rangle$, where V is a finite set of nodes, E is a set of hyper-edges, $v^{\mathsf{entry}} \in V$ is a distinguished *entry node*, and $v^{\mathsf{exit}} \in V$ is a distinguished *exit node*. A *hyper-edge* is an ordered pair $\langle x, Y\rangle$, where $x \in V$ is a node and $Y \subseteq V$ is an ordered, non-empty set of nodes. For a hyper-edge $e = \langle x, Y\rangle$ in E, we use $src(e)$ to denote x and $Dst(e)$ to denote Y. Following the terminology from graphs, we say that e is an *outgoing edge* of x and an *incoming edge* of each of the nodes $y \in Y$. We assume v^{entry} does not have incoming edges, and v^{exit} has no outgoing edges.

Definition 2.7. A *probabilistic program* contains a finite set of procedures $\{H_i\}_{1\le i\le n}$, where each procedure $H_i = \langle V_i, E_i, v_i^{\mathsf{entry}}, v_i^{\mathsf{exit}}\rangle$ is a *control-flow hyper-graph* (CFHG) in which each node except v_i^{exit} has *at least* one outgoing hyper-edge, and v_i^{exit} has no outgoing hyper-edge. Define $V \stackrel{\text{def}}{=} \bigcup_{1\le i\le n} V_i$. To assign meanings to probabilistic programs modulo *data actions* Act and *deterministic conditions* Cond that *can* be probabilistic, we associate with each hyper-edge $e \in E = \bigcup_{1\le i\le n} E_i$ a *control-flow action* $Ctrl(e)$ that has one of the following three forms:

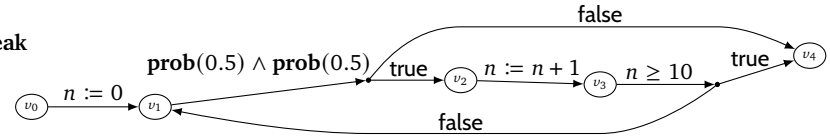
$$\begin{aligned}\mathsf{Ctrl} ::= {} & seq[\mathsf{act}], \text{ where } \mathsf{act} \in \mathsf{Act}\\ \mid {} & cond[\varphi], \text{ where } \varphi \in \mathsf{Cond}\\ \mid {} & call[i \to j], \text{ where } 1 \le i, j \le n\end{aligned}$$

where the number of destination nodes $|Dst(e)|$ of a hyper-edge e is 1 if $Ctrl(e)$ is $seq[\mathsf{act}]$ or $call[i \to j]$, and 2 otherwise.

Example 2.8. Fig. 2.1(b) shows the CFHG of the program in Fig. 2.1(a), where v_0 is the entry and v_4 is the exit. The hyper-edge $\langle v_2, \{v_3\}\rangle$ is associated with a sequencing action $seq[n := n+1]$, while $\langle v_1, \{v_2, v_4\}\rangle$ is assigned a deterministic-choice action $cond[\mathbf{prob}(0.5) \wedge \mathbf{prob}(0.5)]$, i.e., an event where two coin flips both show heads.

```
1 n := 0;
2 while prob(0.5) ∧ prob(0.5) do
3   n := n + 1;
4   if n ≥ 10 then break
5   else continue
6   fi
7 od
```

(a)



(b)

Fig. 2.1: (a) An example of probabilistic programs; (b) The corresponding CFHG.

Note that **break**, **continue** (and also **goto**) are not data actions, and are encoded directly as edges in CFHGs in a standard way. The grammar below defines data actions Act and deterministic conditions Cond for Appl, where $p \in [0,1]$, $a, b, c \in \mathbb{R}$, $a < b$, and $n \in \mathbb{N}$.

$$\begin{aligned} \mathsf{Act} &::= x := e \mid x \sim D \mid \mathbf{observe}(\varphi) \mid \mathbf{skip} \\ \varphi \in \mathsf{Cond} &::= \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid e_1 \le e_2 \mid \mathbf{prob}(p) \\ e \in \mathsf{Exp} &::= x \mid c \mid e_1 + e_2 \mid e_1 \times e_2 \\ D \in \mathsf{Dist} &::= \textsc{Binomial}(n, p) \mid \textsc{Uniform}(a, b) \mid \cdots \end{aligned}$$

Dist stands for a collection of primitive probability distributions. For example, $\textsc{Binomial}(n, p)$ with $n \in \mathbb{N}$ and $p \in [0,1]$ describes the distribution of the number of successes in n independent experiments, each of which succeeds with probability p; $\textsc{Uniform}(a, b)$ represents a uniform distribution on the interval $[a, b]$. Each distribution D is associated with a probability measure $\mu_D \in \mathbb{D}(\mathbb{R})$. For example, the probability measure for $\textsc{Uniform}(a, b)$ is the integration of its density function $\mu_{\textsc{Uniform}(a,b)}(A) \stackrel{\text{def}}{=} \int_A \frac{[a \le x \le b]}{b-a} dx$.

2.3 A Distribution-Based Small-Step Dynamics

The next step is to define a dynamics for Appl based on CFHGs. I adopt Borgström et al. [16]'s distribution-based small-step dynamics for lambda calculus to the hyper-graph setting, while suppressing the features of multiple procedures and nondeterminism for now. In Chapter 3, I will develop a denotational semantics that supports multiple procedures and nondeterminism, and justify the semantics by proving an equivalence result with respect to the small-step dynamics in this section.

Three components are used to define the dynamics:

- A measurable space $(\Omega, \mathcal{F})$ on program states. For Appl, we define $\Omega \stackrel{\text{def}}{=} \mathsf{Var} \to_{\text{fin}} \mathbb{R}$, i.e., a set of finite partial maps from program variables to their values, and $\mathcal{F}$ be the Borel σ-algebra on the finite-dimensional space Ω.
- A (sub-)probability kernel $[\![\mathsf{act}]\!]$ on program states for each data action act. Intuitively,

$[\![x := e]\!] \stackrel{\text{def}}{=} \lambda\omega.\delta(\omega[x \mapsto \omega(e)])$	$[\![\top]\!] \stackrel{\text{def}}{=} \lambda\omega.1$
$[\![x \sim D]\!] \stackrel{\text{def}}{=} \lambda\omega.\mu_D \gg\!= \lambda v.\delta(\omega[x \mapsto v])$	$[\![\neg\varphi]\!] \stackrel{\text{def}}{=} \lambda\omega.1 - [\![\varphi]\!](\omega)$
$[\![\textbf{observe}(\varphi)]\!] \stackrel{\text{def}}{=} \lambda\omega.[\![\varphi]\!](\omega) \cdot \delta(\omega)$	$[\![\varphi_1 \wedge \varphi_2]\!] \stackrel{\text{def}}{=} \lambda\omega.[\![\varphi_1]\!](\omega) \cdot [\![\varphi_2]\!](\omega)$
$[\![\textbf{skip}]\!] \stackrel{\text{def}}{=} \lambda\omega.\delta(\omega)$	$[\![e_1 \leq e_2]\!] \stackrel{\text{def}}{=} \lambda\omega.[\omega(e_1) \leq \omega(e_2)]$
	$[\![\textbf{prob}(p)]\!] \stackrel{\text{def}}{=} \lambda\omega.p$

Fig. 2.2: Interpretation of data actions and deterministic conditions

$[\![\mathsf{act}]\!](\omega, F)$ is the probability that the action act, starting in state $\omega \in \Omega$, halts in a state that satisfies $F \in \mathcal{F}$.

- A $[0,1]$-valued measurable function $[\![\varphi]\!]$ from program states for each deterministic condition φ. Intuitively, $[\![\varphi]\!](\omega)$ is the probability that the condition φ holds in state $\omega \in \Omega$.

Fig. 2.2 shows interpretation of the data actions and deterministic conditions given in §2.2, where $\omega(e)$ evaluates expression e in state ω. If φ does not contain any probabilistic choices $\textbf{prob}(p)$, then $[\![\varphi]\!](\omega)$ is either 0 or 1. Intuitively, $[\![\varphi]\!](\omega)$ is the probability that φ is true in the state ω, with respect to a probability space specified by all the $\textbf{prob}(p)$'s in φ. Then the probability of $\varphi_1 \wedge \varphi_2$ is defined as the product of the individual probabilities of φ_1 and φ_2, because φ_1 and φ_2 are interpreted with respect to probabilistic choices in φ_1 and φ_2, respectively, and these two sets of choices are disjoint, thus independent.

Suppose that $P = \langle V, E, v^{\mathsf{entry}}, v^{\mathsf{exit}} \rangle$ is a single-procedure deterministic program. Therefore, each node in P except v^{exit} is associated with *exactly* one hyper-edge. The *program configurations* $T = V \times \Omega$ are pairs of the form $\langle v, \omega \rangle$, where $v \in V$ is a node in the CFHG, and $\omega \in \Omega$ is a program state. To construct a measurable space of program configurations, my approach is to use the *product measurable space* of $(V, \wp(V))$ and $(\Omega, \mathcal{F})$. The product of two measurable spaces $(S_1, \mathcal{S}_1)$ and $(S_1, \mathcal{S}_2)$ is defined as $(S_1, \mathcal{S}_1) \otimes (S_2, \mathcal{S}_2) \stackrel{\text{def}}{=} (S_1 \times S_2, \mathcal{S}_1 \otimes \mathcal{S}_2)$, where $\mathcal{S}_1 \otimes \mathcal{S}_2$ is the smallest σ-algebra that makes coordinate maps measurable, i.e., $\sigma(\{\pi_1^{-1}(A_1) \mid A_1 \in \mathcal{S}_1\} \cup \{\pi_2^{-1}(A_2) \mid A_2 \in \mathcal{S}_2\})$, where π_i is the i-th coordinate map.

I define *one-step evaluation* as a relation $\langle v, \omega \rangle \longrightarrow \mu$ between configurations $\langle v, \omega \rangle$ and sub-probability measures μ on configurations, as shown in Fig. 2.3.

Example 2.9. For the program in Fig. 2.1, some one-step evaluations are $\langle v_0, \{n \mapsto 233\} \rangle \longrightarrow \delta(\langle v_1, \{n \mapsto 0\} \rangle)$, $\langle v_1, \{n \mapsto 1\} \rangle \longrightarrow 0.25 \cdot \delta(\langle v_2, \{n \mapsto 1\} \rangle) + 0.75 \cdot \delta(\langle v_4, \{n \mapsto 1\} \rangle)$, *and* $\langle v_3, n \mapsto 9 \rangle \longrightarrow \delta(\langle v_1, \{n \mapsto 9\} \rangle)$.

Lemma 2.10. *The one-step evaluation relation* $\longrightarrow$ *defines a sub-probability kernel on program configurations.*

$$\begin{aligned}
\langle v, \omega\rangle &\longrightarrow \lambda F.[\![\mathsf{act}]\!](\omega, \{\omega' \mid \langle u, \omega'\rangle \in F\}) \\
&\quad \text{where } e = \langle v, \{u\}\rangle \in E, Ctrl(e) = seq[\mathsf{act}] \\
\langle v, \omega\rangle &\longrightarrow [\![\varphi]\!](\omega) \cdot \delta(\langle u_1, \omega\rangle) + (1 - [\![\varphi]\!](\omega)) \cdot \delta(\langle u_2, \omega\rangle) \\
&\quad \text{where } e = \langle v, \{u_1, u_2\}\rangle \in E, Ctrl(e) = cond[\varphi]
\end{aligned}$$

Fig. 2.3: One-step evaluation relation

$$\begin{aligned}
\langle v, \omega\rangle &\longrightarrow_0 \mathbf{0} && \\
\langle v^{\mathsf{exit}}, \omega\rangle &\longrightarrow_n \delta(\omega) && \text{if } n > 0 \\
\langle v, \omega\rangle &\longrightarrow_{n+1} \lambda F. \int_{\mathrm{supp}(\mu)} \mu(d\tau) \cdot \mu'_\tau(F) && \text{where} \langle v, \omega\rangle \longrightarrow \mu \\
& && \text{and } \tau \longrightarrow_n \mu'_\tau \text{ for any } \tau \in \mathrm{supp}(\mu)
\end{aligned}$$

Fig. 2.4: Step-indexed evaluation relation

Proof. The evaluation relation $\longrightarrow$ can be seen as a function $\hat{\longrightarrow}$ defined as follows:

$$\hat{\longrightarrow}(x)(A) \stackrel{\text{def}}{=} \begin{cases} \mu(A) & \text{if } x \longrightarrow \mu \\ 0 & \text{otherwise} \end{cases}$$

- For any x, it is straightforward to show that $\hat{\longrightarrow}(x)$ is a distribution.
- For any A, we want to show that $\lambda x.\hat{\longrightarrow}(x)(A)$ is a measurable function. Let $f \stackrel{\text{def}}{=} \lambda x.\hat{\longrightarrow}(x)(A)$. It is sufficient to show that for all $t \in (0,1)$, $\{x \mid f(x) < t\}$ is a measurable set. Observe that $\{x \mid f(x) < t\}$ is equal to $\bigcup_v \{x = \langle v, \omega\rangle \mid f(x) < t\}$, We need to show that for all v, $\{x = \langle v, \omega\rangle \mid f(x) < t\}$ is measurable. Let me denote the set by $O(t, v)$.
  - If $e = \langle v, \{u\}\rangle \in E$ and $Ctrl(e) = seq[\mathsf{act}]$, then $O(t, v) = \{v\} \times \{\omega \mid [\![\mathsf{act}]\!](\omega, \{\omega' \mid \langle u, \omega'\rangle \in A\}) < t\}$. Because $[\![\mathsf{act}]\!]$ is a kernel, we conclude that $O(t, v)$ is measurable.
  - If $e = \langle v, \{u_1, u_2\}\rangle \in E$ and $Ctrl(e) = cond[\varphi]$, then $O(t, v) = \bigcup_{r \in \mathbb{Q} \cap (0,t)} \{\langle v, \omega\rangle \mid \langle u_1, \omega\rangle \in A \wedge [\![\varphi]\!](\omega) < r\} \cap \{\langle v, \omega\rangle \mid \langle u_2, \omega\rangle \in A \wedge 1 - [\![\varphi]\!](\omega) < t - r\}$. Because $[\![\varphi]\!]$ is a measurable function, and measurable sets are closed under countable union and intersection, we conclude that $O(t, v)$ is measurable.

□

I now define *step-indexed evaluation* as the family of n-indexed relations $\langle v, \omega\rangle \longrightarrow_n \mu$ between configurations $\langle v, \omega\rangle$ and sub-probability measures μ on program states inductively, as shown in Fig. 2.4.

Example 2.11. For the program in Fig. 2.1, some step-indexed evaluations are $\langle v_4, \{n \mapsto$

$10\}\rangle \longrightarrow_1 \delta(\{n \mapsto 10\})$, $\langle v_1, \{n \mapsto 0\}\rangle \longrightarrow_2 0.75 \cdot \delta(\{n \mapsto 0\})$, *and* $\langle v_1, \{n \mapsto 0\}\rangle \longrightarrow_5 0.75 \cdot \delta(\{n \mapsto 0\}) + 0.1875 \cdot \delta(\{n \mapsto 1\})$.

LEMMA 2.12. *The step-indexed evaluation relation* $\longrightarrow_n$ *defines a sub-probability kernel from program configurations to program states for any* $n \in \mathbb{Z}^+$. *Moreover, if* $\langle v, \omega\rangle \longrightarrow_n \mu_1$, $\langle v, \omega\rangle \longrightarrow_m \mu_2$, *and* $n \leq m$, *then* $\mu_1 \leq \mu_2$ *pointwise.*

PROOF. By induction on n:

- $\longrightarrow_0$ can be seen as the everywhere-zero function $\xrightarrow{\wedge}_0$ which is trivially a kernel.
- $\longrightarrow_{n+1}$ can be seen as the function defined as follows:

$$\xrightarrow{\wedge}_{n+1}(\langle v, \omega\rangle)(F) \stackrel{\text{def}}{=} \begin{cases} 1 & v = v^{\mathsf{exit}} \wedge \omega \in F \\ 0 & v = v^{\mathsf{exit}} \wedge \omega \notin F \\ \int \xrightarrow{\wedge}_n(\tau)(F)\mu(d\tau) & \langle v, \omega\rangle \longrightarrow \mu \end{cases}$$

  - For any x, it is straightforward to show that $\xrightarrow{\wedge}_{n+1}(x)$ is a distribution.
  - For any F, we want to show that $\lambda x.\xrightarrow{\wedge}_{n+1}(x)(F)$ is a measurable function. Let $f \stackrel{\text{def}}{=} \lambda x.\xrightarrow{\wedge}_{n+1}(x)(F)$. It is sufficient to show that for all $t \in (0,1)$, $\{x \mid f(x) < t\}$ is a measurable set. Let $O = \{v^{\mathsf{exit}}\} \times \Omega$. Observe that $\{x \mid f(x) < t\} = (\{x \mid f(x) < t\} \cap O) \cap (\{x \mid f(x) < t\} \cap (T \setminus O))$.
    - $\{x \mid f(x) < t\} \cap O = \{\langle v^{\mathsf{exit}}, \omega\rangle \mid [\omega \in F] < t\}$ is trivially measurable.
    - $\{x \mid f(x) < t\} \cap (T \setminus O) = \{x \mid \int \xrightarrow{\wedge}_n(y)(F)\xrightarrow{\wedge}(x)(dy) < t\} = \{x \mid \kappa(x)(F) < t\}$, where κ is the composition of $\xrightarrow{\wedge}$ and $\xrightarrow{\wedge}_n$. Hence $\{x \mid f(x) < t\} \cap (T \setminus O)$ is measurable.

  Meanwhile, we want to show that $\xrightarrow{\wedge}_{n+1} \geq \xrightarrow{\wedge}_n$ pointwise. If $n = 0$, then the inequality holds trivially. For $n > 0$, consider any v, ω, F,
  - If $v = v^{\mathsf{exit}}$ and $\omega \in F$, then $\xrightarrow{\wedge}_{n+1}(\langle v, \omega\rangle)(F) = \xrightarrow{\wedge}_n(\langle v, \omega\rangle)(F) = 1$.
  - If $v = v^{\mathsf{exit}}$ and $\omega \notin F$, then $\xrightarrow{\wedge}_{n+1}(\langle v, \omega\rangle)(F) = \xrightarrow{\wedge}_n(\langle v, \omega\rangle)(F) = 0$.
  - Otherwise, suppose that $\langle v, \omega\rangle \longrightarrow \mu$, then $\xrightarrow{\wedge}_{n+1}(\langle v, \omega\rangle)(F) = \int \xrightarrow{\wedge}_n(\tau)(F)\mu(d\tau)$ and $\xrightarrow{\wedge}_n(\langle v, \omega\rangle)(F) = \int \xrightarrow{\wedge}_{n-1}(\tau)(F)\mu(d\tau)$. By induction hypothesis, we know that $\xrightarrow{\wedge}_{n-1} \leq \xrightarrow{\wedge}_n$ pointwise, thus $\xrightarrow{\wedge}_n(\langle v, \omega\rangle)(F) \leq \xrightarrow{\wedge}_{n+1}(\langle v, \omega\rangle)(F)$.

□

For the program $P = \langle V, E, v^{\mathsf{entry}}, v^{\mathsf{exit}}\rangle$, by Lem. 2.12, I define its semantics as $[\![P]\!]_{\mathsf{os}}(\omega, F) \stackrel{\text{def}}{=} \sup_{n \in \mathbb{Z}^+}\{\mu(F) \mid \langle v^{\mathsf{entry}}, \omega\rangle \longrightarrow_n \mu\}$.

Example 2.13. For the program P in Fig. 2.1, the sub-probability measure $\lambda F.[\![P]\!]_{\mathsf{os}}(\omega, F)$ *for any initial state* ω *that contains the program variable n is given by* $\sum_{k=0}^{9}(0.75 \times 0.25^k) \cdot \delta(\omega[n \mapsto k]) + 0.00000095367431640625 \cdot \delta(\omega[n \mapsto 10])$.

LEMMA 2.14. *For any program P,* $[\![P]\!]_{\mathsf{os}}$ *defines a sub-probability kernel on program states.*

PROOF. By definition, we have $[\![P]\!]_{\mathsf{os}} = \sup_{n \in \mathbb{Z}^+} \xrightarrow{\wedge}_n$. Then we can conclude by the fact that the set of sub-probability kernels forms an ω-cpo. □

In general, the measure $\lambda F.[\![P]\!]_{\mathsf{os}}(\omega, F)$ that describes the evaluation results of a program P with an initial state ω is *not* always a probability measure. In the case that P *diverges* with probability $p > 0$ from the initial state ω, the measure $\lambda F.[\![P]\!]_{\mathsf{os}}(\omega, F)$ is a sub-probability measure that satisfies $[\![P]\!]_{\mathsf{os}}(\omega, \Omega) = 1 - p$, i.e., the probabilities of the resulting states "sum up" to $1 - p$.

Chapter 3

A Denotational Semantics for Probabilistic Programs with Nondeterminism-First Resolution

In §2.3, I have shown how probabilistic programs execute *operationally*. In this chapter, I focus on developing a *denotational* semantics, which concentrates on the *effects* of programs and abstracts from how the program executes. This characterization of denotational semantics is beneficial for *rigorous reasoning* about programs, such as static analysis and model checking, because one usually only cares whether programs satisfy certain properties, e.g., if they terminate on all possible inputs. Even better, a denotational semantics is often *compositional*—that is, the property of a whole program can be established from properties of its proper constituents. In other words, one can develop *local*—and thus *scalable*—reasoning techniques based on a denotational semantics. In contrast, the operational semantics in §2.3 is not compositional—it takes into account the whole program P to define $[\![P]\!]_{\mathsf{os}}$.

Another benefit of a denotational semantics is that it is often easier to extend than an operational one. As an example, let me briefly compare the complexity of adding procedure calls and nondeterminism to an operational semantics versus a denotational semantics. To support multiple procedures and procedure calls in the semantics proposed in §2.3, one needs to introduce a notion of *stacks* to keep track of procedure calls, as in [46, 47, 114]. Then the program configurations become triples of call stacks, control-flow-graph nodes, and program states. As a consequence, the one-step and step-indexed evaluation relations in Figs. 2.3 and 2.4 would become more complex. However, such an extension is almost trivial for a denotational semantics. Suppose we are able to *compose* semantic objects, e.g, $[\![C_1; C_2]\!]_{\mathsf{ds}} = [\![C_2]\!]_{\mathsf{ds}} \circ [\![C_1]\!]_{\mathsf{ds}}$, where C_1, C_2 are program fragments, $\circ$ denotes a composition operation, and $[\![C]\!]_{\mathsf{ds}}$ gives the denotation of C. For example, consider that C_1 is a procedure call **call** Q, where Q is a procedure. Because we can obtain the denotation $[\![Q]\!]_{\mathsf{ds}}$ of Q, we can interpret $[\![\textbf{call } Q; C_2]\!]_{\mathsf{ds}}$ merely as $[\![C_2]\!]_{\mathsf{ds}} \circ [\![Q]\!]_{\mathsf{ds}}$. By this means we do not need to

reason about stacks explicitly.

Another important programming feature is nondeterminism. For operational semantics of probabilistic programs, nondeterminism is often formalized using the notion of a *scheduler*, which resolves a nondeterministic choice from the computation that leads up to it (e.g., [28, 29, 49]). When the scheduler is fixed, a program can be executed deterministically (as shown in §2.3). To reason about nondeterministic programs with respect to an operational semantics, one needs to take all possible schedulers into consideration. However, if one only cares about the effects of a program, it is possible to sidestep these schedulers by switching to a denotational semantics. For example, let C_1, C_2 be two program fragments and $[\![C_1]\!]_{\mathsf{ds}}, [\![C_2]\!]_{\mathsf{ds}}$ be their denotations, which could be maps from initial states to a collection of possible final states. Then the denotation $[\![\textbf{if } \star \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ fi}]\!]_{\mathsf{ds}}$ of a nondeterministic-choice between C_1 and C_2 could be something like $\lambda\omega.[\![C_1]\!]_{\mathsf{ds}}(\omega) \cup [\![C_2]\!]_{\mathsf{ds}}(\omega)$. Note that this approach does not need to consider schedulers explicitly.

Some high-level decision choices about *nondeterminism* arise when we develop the semantics for Appl. Nondeterminism itself is an important feature from two perspectives: (i) it arises naturally from probabilistic models, such as the agent for a Markov decision process [12], or the unknown input distribution for modeling *fault tolerance* [82], and (ii) it is required by the common paradigm of *abstraction* and *refinement*[1] on programs [42, 102]. While nondeterminism has been well studied for standard programming languages, the combination of probabilities and nondeterminism turns out to be tricky. One substantial question is *when* the nondeterminism is resolved. A well-studied model for nondeterminism in probabilistic programming is to resolve program inputs *prior to* nondeterminism [41, 101, 102, 105, 106, 127]. This model follows a commonplace principle of semantics research that represents a nondeterministic function as a set-valued function that maps an input to a collection of possible outputs, i.e., an element in $X \to \wp(X)$, where X is a program state space and $\wp(\cdot)$ is the powerset operator. However, it is sometimes desirable to resolve nondeterminism *prior to* program inputs, i.e., a nondeterministic program should represent a collection of elements in $\wp(X \to X)$. For example, one may want to show for every refined version of a nondeterministic program with each nondeterministic choice replaced by a conditional, its behavior on all *inputs* are indistinguishable. I call the common model *nondeterminism-last* and the other *nondeterminism-first*.

Example 3.1. Consider the following program P where $\star$ represents nondeterminism.

$$\textbf{if } \textbf{prob}(\star) \textbf{ then } t \coloneqq t + 1 \textbf{ else } t \coloneqq t - 1 \textbf{ fi}$$

Fig. 3.1 illustrates the nondeterminism-last model: given an input $t = 1$, $\star$ is resolved as 0.5 in the left box, whereas it is resolved as 0.8 in the right box. Fig. 3.2 then demonstrates the novel nondeterminism-first model: $\star$ is resolved prior to the input, i.e., each resolution leads to a function that maps an input to a probability distribution.

[1] Abstraction enables reasoning about a program through its high-level specifications, and refinement allows stepwise software development, where programs are "refined" from specifications to low-level implementations.

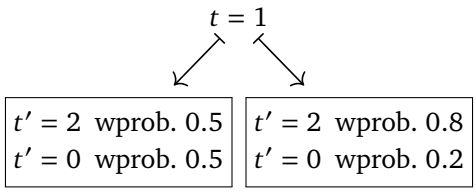


Fig. 3.1: An example where $\star$ resolved **after** t is given. A box represents a probability distribution.

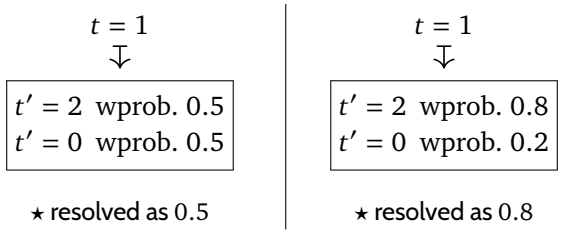


Fig. 3.2: An example where $\star$ resolved **before** t is given. A box represents a probability distribution.

In §3.2, I present a domain-theoretic study of nondeterminism-first for probabilistic programs with a *countable* state space. Technically, I propose a notion of *generalized convexity* (*g-convexity*, for short), which expresses that a set of *state transformers* is stable under refinements (while standard convexity describes that a set of *states* is stable under refinements), as well as devise a g-convex *powerdomain* that characterizes expressible semantic objects.

To achieve my ultimate goal of developing a denotational semantics, instead of restricting myself to one specific model for nondeterminism, I propose a general *algebraic* denotational semantics in §3.3, which can be instantiated with different treatments of nondeterminism. The semantics is algebraic in the sense that it performs reasoning in some space of program states and state transformers, while the transformers should obey some algebraic laws. For instance, the program command **skip** should be interpreted as the *identity* element for sequencing in an algebra of program-state transformers. In addition, the algebraic approach is a good fit for static analysis of probabilistic programs.

The *algebraic* approach I take in this thesis is challenging in the setting of probabilistic programming. In contrast, for standard, non-probabilistic programming languages, it is almost trivial to derive a low-level denotational semantics *once* one has a semantics for well-structured programs at hand. The trick is to first define the semantic operations as a *Kleene algebra* [33, 87, 90, 93], which admits an *extend* operation, used for sequencing, a *combine* operation, used for branching, and a *closure* operation, used for looping; then extract from the CFG a *regular expression* that captures all execution paths by Tarjan [125]'s path-expression algorithm; and finally use the Kleene algebra to *reinterpret* the regular expression to obtain the semantics for the CFG. However, this approach fails when both probabilities and nondeterminism come into the picture. Consider the probabilistic program

$$\begin{array}{lll} \textbf{if} \star & \textbf{then} & \textbf{if } \textbf{prob}(1/2) \textbf{ then } t \coloneqq 0 \textbf{ else } t \coloneqq 1 \textbf{ fi} \\ & \textbf{else} & \textbf{if } \textbf{prob}(1/3) \textbf{ then } t \coloneqq 0 \textbf{ else } t \coloneqq 1 \textbf{ fi} \quad \textbf{fi} \end{array}$$

Fig. 3.3: A nondeterministic, probabilistic program

with a *nondeterministic* choice $\star$ in Fig. 3.3. The program is intended to draw a random value t from either a fair coin flip or a biased one. If one adopts the path-expression approach, one ends up with a regular expression that describes a *single* collection of four program executions: (i) $t \coloneqq 0$ with probability $1/2$, (ii) $t \coloneqq 1$ with probability $1/2$, (iii) $t \coloneqq 0$ with probability $1/3$, and (iv) $t \coloneqq 1$ with probability $2/3$. The collection does *not* describe the intended meaning, and does *not* even form a well-defined probability distribution—all the probabilities sum up to 2 instead of 1. Intuitively, the path-expression approach fails for probabilistic programs because it can only express the semantics as a collection of executions with probabilities, whereas probabilistic programs actually specify collections of *distributions* over executions.

3.1 A Summary of Existing Domain-Theoretic Developments

To present the technical development of nondeterminism-first for probabilistic programs with a *countable* state space, I will use a simplified notion of sub-probability measures. Let X be a nonempty countable set. A *distribution* on X is a function $\Delta : X \to [0, 1]$ such that $\sum_{x \in X} \Delta(x) \leq 1$, and the *Dirac distribution* $\delta(x)$ for some $x \in X$ is defined as $\lambda x'.[x = x']$. The set of all distributions on X is denoted by $\underline{\mathcal{D}}(X)$.

My development of models for nondeterminism makes great use of existing domain-theoretic studies of powerdomains, thus in this section, I present a brief summary of them. I review some standard notions from domain theory [2, 69, 104], as well as some results on probabilistic powerdomains [76, 77] and nondeterministic powerdomains [41, 101, 102, 105, 106, 127].

3.1.1 Background from Domain Theory

Let P be a nonempty set with a partial order $\sqsubseteq$, i.e., a *poset*. The *lower closure* of a subset A is defined as $\downarrow A \stackrel{\text{def}}{=} \{x \in P \mid \exists a \in A : x \sqsubseteq a\}$. The *upper closure* of a subset A is defined as $\uparrow A \stackrel{\text{def}}{=} \{x \in P \mid \exists a \in A : a \sqsubseteq x\}$. A subset A satisfying $\downarrow A = A$ is called a *lower set*. A subset A satisfying $\uparrow A = A$ is called an *upper set*. If all elements of P are above a single element $x \in P$, then x is called the *least element*, denoted commonly by $\bot$. A function $f : P \to Q$ between two posets P and Q is *monotone* if for all $x, y \in P$ such that $x \sqsubseteq y$, we have $f(x) \sqsubseteq f(y)$. A subset A of P is *directed* if it is nonempty and each pair of elements in A has an upper bound in A. If A is totally ordered and isomorphic to natural numbers, then A is called an *ω-chain*. If a directed set A has a supremum, then it is denoted by $\bigsqcup^{\uparrow} A$.

A poset D is called *directed complete* or a dcpo if each directed subset A of D has a supremum $\bigsqcup^{\uparrow} A$ in D. A function $f : D \to E$ between two dcpos D and E is *Scott-continuous* if it is monotone and preserves directed suprema, i.e., $f(\bigsqcup^{\uparrow} A) = \bigsqcup^{\uparrow} f(A)$ for all directed subsets A of D.

Let D be a dcpo. For two elements x, y of D, we say that x *approximates* y, denoted by $x \ll y$, if for all directed subsets A of D, we have $y \sqsubseteq \bigsqcup^{\uparrow} A$ implies $x \sqsubseteq a$ for some $a \in A$. We define $\mathop{\downarrow} A \stackrel{\text{def}}{=} \{x \in D \mid \exists a \in A : x \ll a\}$ and $\mathop{\Uparrow} A \stackrel{\text{def}}{=} \{x \in D \mid \exists a \in A : a \ll x\}$. The dcpo D is called *continuous* if there exists a subset B of D such that for every element x of D, the set $\mathop{\downarrow} x \cap B$ is directed and $x = \bigsqcup^{\uparrow}(\mathop{\downarrow} x \cap B)$. The set B is called a *basis* of D.

Let D be a dcpo. A subset A is *Scott-closed* if A is a lower set and is closed under directed suprema. The complement $D \setminus A$ of a Scott-closed subset A is called *Scott-open*. These Scott-open subsets form the *Scott-topology* on D. The *closure* of a subset A is the smallest Scott-closed set containing A as a subset, denoted by $\overline{A}$.

Let X be a topological space whose open sets are denoted by $O(X)$. A *cover* C of a subset A of X is a collection of subsets whose union contains A as a subset. A *sub-cover* of C is a subset of C that still covers A. The cover C is called an *open-cover* if each of its members is an open set. A subset A is *compact* if every open-cover of A contains a finite sub-cover. A subset A is *saturated* if A is an intersection of its neighborhoods. The *saturation* of a subset A is the intersection of its neighborhoods. In dcpo's equipped with the Scott-topology, saturated sets are precisely the upper sets, and the saturation of a subset A is given by $\uparrow A$. The *Lawson-topology* on a dcpo D is generated by Scott-open sets and sets of the form $D \setminus \uparrow x$. A *lens* is a nonempty subset that is the intersection of a Scott-closed subset and a Scott-compact saturated subset. Lenses are always Lawson-closed sets. A continuous dcpo D is called *coherent* if the intersection of any two Scott-compact saturated subsets is also Scott-compact. The Lawson-topology on a coherent dcpo is compact.

I am going to use the following theorems in my technical development.

Proposition 3.2 (Kleene fixed-point theorem). *Suppose $\langle D, \sqsubseteq\rangle$ is a dcpo with a least element $\bot$, and let $f : D \to D$ be a Scott-continuous function. Then f has a least fixed point which is the supremum of the ascending Kleene chain of f (i.e., the ω-chain $\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq \cdots \sqsubseteq f^n(\bot) \sqsubseteq \cdots$), denoted by $\mathrm{lfp}^{\sqsubseteq}_{\bot} f$.*

Proposition 3.3 (Cor. of [69, Hofmann-Mislove theorem]). *Let X be a sober space, i.e., a T_0-space where every nonempty closed set is either the closure of a point or the union of two proper closed subsets. The intersection of a filtered family $\{A_i\}_{i\in\mathcal{I}}$ (i.e., the intersection of any two subsets is in the family) of nonempty compact saturated subsets is compact and nonempty. If such a filtered intersection is contained in an open set U, then $A_i \subseteq U$ for some $i \in \mathcal{I}$. Specifically, continuous dcpos equipped with the Scott-topology and coherent dcpos equipped with the Lawson-topology are sober.*

3.1.2 Probabilistic Powerdomains

Jones and Plotkin [77]'s pioneer work on probabilistic powerdomains extends the complete partially ordered sets, which are pervasively used in computer science, to model probabilistic computations. Let X be a nonempty countable set. We say that the set of all distributions on X, denoted by $\underline{\mathcal{D}}(X)$, is a *probabilistic powerdomain* over X. Distributions are ordered pointwise, i.e., $\Delta_1 \sqsubseteq_D \Delta_2 \stackrel{\text{def}}{=} \forall x \in X\colon \Delta_1(x) \leq \Delta_2(x)$. We define the *probabilistic-choice* of distributions Δ_1, Δ_2 with respect to a weight $p \in [0,1]$, written $\Delta_1 \,{}_p\!\oplus \Delta_2$, as $p \cdot \Delta_1 + (1-p) \cdot \Delta_2$. The operation ${}_p\!\oplus$ corresponds to the program construct "**if prob**(p) **then** $\cdots$ **else** $\cdots$ **fi**."

The following theorems provide a characterization of the probabilistic powerdomains.

Proposition 3.4 ([76, 77, 101, 127]). *The poset $\langle \underline{\mathcal{D}}(X), \sqsubseteq_D \rangle$ forms a coherent dcpo with a countable basis $\{\sum_{i=1}^{n} r_i \cdot \delta(x_i) \mid n \in \mathbb{Z}^+ \wedge r_i \in \mathbb{Q}^+ \wedge \sum_{i=1}^{n} r_i \leq 1 \wedge x_i \in X\}$. It admits a least element $\bot_D \stackrel{\text{def}}{=} \lambda x.0$. Moreover, ${}_p\!\oplus$ is Scott-continuous for all $p \in [0,1]$.*

Proposition 3.5 ([76, 127]). *Every function $f : X \to \underline{\mathcal{D}}(X)$ can be lifted to a unique Scott-continuous linear (in the sense that it preserves probabilistic-choice) map $\widehat{f} : \underline{\mathcal{D}}(X) \to \underline{\mathcal{D}}(X)$.*

3.1.3 Nondeterministic Powerdomains

When nondeterminism comes into the picture, as I discussed at the beginning of this chapter, existing studies usually resolve program inputs *prior to* nondeterminism [41, 79, 101, 102, 105, 106, 127]. I call such a model *nondeterminism-last*, which interprets nondeterministic functions as maps from inputs to sets of outputs. Let X be a nonempty countable set. A subset A of $\underline{\mathcal{D}}(X)$ is called *convex* if for all $\Delta_1, \Delta_2 \in A$ and all $p \in [0,1]$, we have $\Delta_1 \,{}_p\!\oplus \Delta_2 \in A$. The *convex hull* of an arbitrary subset A is the smallest convex set containing A as a subset, denoted by $conv(A)$. The convexity condition ensures that from the perspective of programming, nondeterministic choices can always be *refined* by probabilistic choices. The *convex powerdomain* $\mathcal{P}\underline{\mathcal{D}}(X)$ over the probabilistic powerdomain $\underline{\mathcal{D}}(X)$ is then defined as convex lenses in $\underline{\mathcal{D}}(X)$ with the *Egli-Milner order* $A \sqsubseteq_P B \stackrel{\text{def}}{=} A \subseteq \downarrow B \wedge \uparrow A \supseteq B$.

The following theorems provide a characterization of the convex powerdomains.

Proposition 3.6 ([101, 127]). *The poset $\langle \mathcal{P}\underline{\mathcal{D}}(X), \sqsubseteq_P \rangle$ forms a coherent dcpo. It admits a least element $\bot_P \stackrel{\text{def}}{=} \{\bot_D\}$. For $r_1, r_2 \in [0,1]$ satisfying $r_1 + r_2 \leq 1$, we define $r_1 \cdot A + r_2 \cdot B \stackrel{\text{def}}{=} \overline{C} \cap \uparrow C$ where C is $\{r_1 \cdot \Delta_1 + r_2 \cdot \Delta_2 \mid \Delta_1 \in A \wedge \Delta_2 \in B\}$. Then the probabilistic-choice operation is lifted to a Scott-continuous operation as $A \,{}_p\!\oplus_P B \stackrel{\text{def}}{=} p \cdot A + (1-p) \cdot B$. Moreover, it carries a Scott-continuous semilattice operation, called* formal union*, defined as $A \uplus_P B \stackrel{\text{def}}{=} \overline{C} \cap \uparrow C$ where C is $conv(A \cup B)$.*

The formal-union operation $\uplus$ corresponds to the program construct "**if** $\star$ **then** $\cdots$ **else** $\cdots$ **fi**" for nondeterministic choices.

Proposition 3.7 ([127]). *Every function $g : X \to \mathcal{P}\underline{\mathcal{D}}(X)$ can be lifted to a unique Scott-continuous linear (in the sense that it preserves lifted probabilistic-choice) map $\widehat{\widehat{g}} : \mathcal{P}\underline{\mathcal{D}}(X) \to \mathcal{P}\underline{\mathcal{D}}(X)$ preserving formal unions.*

Example 3.8. Consider the following program P where $\star$ can be refined by any deterministic condition involving the program variable t:

$$\textbf{if } \star \textbf{ then } t \coloneqq t + 1 \textbf{ else } t \coloneqq t - 1 \textbf{ fi}$$

and we want to assign a denotation to it from $X \to \mathcal{P}\underline{\mathcal{D}}(X)$, where the state space $X = \mathbb{Q}$ represents the value of t. Fix an input $t \in \mathbb{Q}$. The data actions $t \coloneqq t + 1$ and $t \coloneqq t - 1$ then take the input to singletons $\{\delta(t+1)\}$ and $\{\delta(t-1)\}$, respectively, in the powerdomain $\mathcal{P}\underline{\mathcal{D}}(\mathbb{Q})$. Thus the nondeterministic-choice is interpreted as $\{\delta(t+1)\} \uplus_P \{\delta(t-1)\}$, which is $\{r \cdot \delta(t+1) + (1-r) \cdot \delta(t-1) \mid r \in [0,1]\}$, for a given $t \in \mathbb{Q}$.

3.2 Nondeterminism-First

In this section, I develop a new model of nondeterminism—the *nondeterminism-first* approach, which resolves nondeterministic choices *prior to* program inputs—in a domain-theoretic way. This model is inspired by reasoning about a program's behavior on different inputs (as mentioned in the beginning of this chapter), which requires nondeterministic functions to be treated as a family of *transformers* (i.e., an element of $\wp(X \to X)$) instead of a set-valued map (i.e., an element of $X \to \wp(X)$). As will be shown in this section, with nondeterminism-first, $t \coloneqq t + 1$ and $t \coloneqq t - 1$ are assigned semantic objects $\{\lambda t.\delta(t+1)\}$ and $\{\lambda t.\delta(t-1)\}$, respectively.

I first introduce a simplified notion of *kernels* on a countable state space, then propose a new notion of *generalized convexity* (*g-convexity*, for short), and finally develop a powerdomain for nondeterminism-first.

3.2.1 A Powerdomain for Sub-Probability Kernels

Let X be a nonempty countable set. A function $\kappa : X \to \underline{\mathcal{D}}(X)$ is called a *(sub-probability) kernel*. Intuitively, a kernel maps an input state to a distribution over output states. The set of all such kernels is denoted by $\underline{\mathcal{K}}(X) \stackrel{\text{def}}{=} X \to \underline{\mathcal{D}}(X)$. Kernels are ordered pointwise, i.e., $\kappa_1 \sqsubseteq_K \kappa_2 \stackrel{\text{def}}{=} \forall x \in X : \kappa_1(x) \sqsubseteq_D \kappa_2(x)$.

Theorem 3.9. *The poset $\langle \underline{\mathcal{K}}(X), \sqsubseteq_K \rangle$ forms a coherent dcpo, with $\bot_K \stackrel{\text{def}}{=} \lambda x.\bot_D$ as its least element.*

Proof. We equip X with the discrete topology. We then define $X_\bot = X \cup \{\bot\}$ with a distinguished least element $\bot$ and thus $X_\bot$ is a flat domain. Then $X_\bot$ is a bounded-complete

domain. The Scott-compact subsets of $X_\perp$ are precisely finite subsets of X and all subsets that contain $\perp$. Thus $X_\perp$ is coherent. By [2, Ex. 4.3.11.14], we know that $X_\perp$ is an FS-domain.

By Prop. 3.4 we know that $\underline{\mathcal{D}}(X)$ is coherent. Moreover, $\underline{\mathcal{D}}(X)$ is also bounded-complete. Thus $\underline{\mathcal{D}}(X)$ is an FS-domain. By [2, Thm. 4.2.11], we know that $[X_\perp \to \underline{\mathcal{D}}(X)]$ is an FS-domain.

Let $s \stackrel{\text{def}}{=} \lambda f.f$ and $r \stackrel{\text{def}}{=} \lambda g.\lambda x.\textbf{if } x = \perp \textbf{ then } \perp_D \textbf{ else } g(x)$. Then $s : [X_\perp \xrightarrow{\perp!} \underline{\mathcal{D}}(X)] \to [X_\perp \to \underline{\mathcal{D}}(X)]$, $r : [X_\perp \to \underline{\mathcal{D}}(X)] \to [X_\perp \xrightarrow{\perp!} \underline{\mathcal{D}}(X)]$, and $r \circ s$ is the identity on $[X_\perp \xrightarrow{\perp!} \underline{\mathcal{D}}(X)]$, where $[A \xrightarrow{\perp!} B]$ stands for continuous functions from a dcpo A to a dcpo B that preserve the least element. Hence $[X_\perp \xrightarrow{\perp!} \underline{\mathcal{D}}(X)]$ is a retract of $[X_\perp \to \underline{\mathcal{D}}(X)]$. By [2, Prop. 4.2.12], we know that $[X_\perp \xrightarrow{\perp!} \underline{\mathcal{D}}(X)]$ is also an FS-domain.

For any f in $[X \to \underline{\mathcal{D}}(X)]$, we define a function $g \stackrel{\text{def}}{=} \lambda x.\textbf{if } x = \perp \textbf{ then } \perp_D \textbf{ else } f(x)$. For any g in $[X_\perp \xrightarrow{\perp!} \underline{\mathcal{D}}(X)]$, we define a function $f \stackrel{\text{def}}{=} \lambda x.g(x)$. Thus $[X \to \underline{\mathcal{D}}(X)]$ is homeomorphic to $[X_\perp \xrightarrow{\perp!} \underline{\mathcal{D}}(X)]$, and we know that $[X \to \underline{\mathcal{D}}(X)]$ is also an FS-domain. By [2, Thm. 4.2.18], we know that $[X \to \underline{\mathcal{D}}(X)]$ is coherent. Because the topology on X is discrete, $[X \to \underline{\mathcal{D}}(X)]$ is precisely $X \to \underline{\mathcal{D}}(X)$. Thus we conclude that $\underline{\mathcal{K}}(X)$ is coherent. □

Let $\mathbb{W}(X) \stackrel{\text{def}}{=} X \to [0,1]$ be the set of functions from X to the interval $[0,1]$. We denote the pointwise comparison by $\dot{\leq}$ and the constant function by $\dot{r}$ for any $r \in [0,1]$. If κ is a kernel and $\phi \in \mathbb{W}(X)$, I write $\phi \cdot \kappa$ for the kernel $\lambda x.\phi(x) \cdot \kappa(x)$. If κ_1, κ_2 are kernels and $\phi_1, \phi_2 \in \mathbb{W}(X)$ such that $\phi_1 + \phi_2 \dot{\leq} \dot{1}$, we write $\phi_1 \cdot \kappa_1 + \phi_2 \cdot \kappa_2$ for the kernel $\lambda x.\phi_1(x) \cdot \kappa_1(x) + \phi_2(x) \cdot \kappa_2(x)$. More generally, if $\{\kappa_i\}_{i\in\mathbb{N}}$ is a sequence of kernels, and $\{\phi_i\}_{i\in\mathbb{N}}$ is a sequence of functions in $\mathbb{W}(X)$ such that $\sum_{i=1}^{\infty} \phi_i \dot{\leq} \dot{1}$, we write $\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i$ for the kernel $\bigsqcup^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \kappa_i$. Then we define *conditional-choice* of kernels κ_1, κ_2 conditioning on a function $\phi \in \mathbb{W}(X)$ as $\kappa_1 \,{}_\phi\!\diamond\, \kappa_2 \stackrel{\text{def}}{=} \phi \cdot \kappa_1 + (\dot{1} - \phi) \cdot \kappa_2$. We define the *composition* of kernels κ_1, κ_2 as $\kappa_1 \otimes \kappa_2 \stackrel{\text{def}}{=} \lambda x.\lambda x''.\, \sum_{x'\in X} \kappa_1(x)(x') \cdot \kappa_2(x')(x'')$.

Lemma 3.10.

1. *The conditional-choice operation* ${}_\phi\!\diamond$ *is Scott-continuous for all* $\phi \in \mathbb{W}(X)$.
2. *The composition operation* $\otimes$ *is Scott-continuous.*

Proof.

1. Monotonicity is trivial. It then suffices to show that for all directed set $A \subseteq \underline{\mathcal{K}}(X)$, $\phi \cdot (\bigsqcup^{\uparrow} A) = \bigsqcup^{\uparrow}_{\kappa\in A} \phi \cdot \kappa$. Let $\kappa' \stackrel{\text{def}}{=} \bigsqcup^{\uparrow} A$. We conclude the proof by $\bigsqcup^{\uparrow}_{\kappa\in A} \phi(x) \cdot \kappa(x) = \phi(x) \cdot \bigsqcup^{\uparrow}_{\kappa\in A} \kappa(x) = \phi(x) \cdot (\bigsqcup^{\uparrow} A)(x) = \phi(x) \cdot \kappa'(x)$ for any x.
2. Monotonicity is trivial.
   Left-Scott-continuity. For all directed set $A \subseteq \underline{\mathcal{K}}(X)$ and all $\rho \in \underline{\mathcal{K}}(X)$, we want to show that $(\bigsqcup^{\uparrow} A) \otimes \rho = \bigsqcup^{\uparrow}_{\kappa\in A} \kappa \otimes \rho$. Let $\kappa' \stackrel{\text{def}}{=} \bigsqcup^{\uparrow} A$. Then it is sufficient to show that

for all x and x'', $\int \kappa'(x)(dx')\rho(x')(x'') = \bigsqcup^{\uparrow}_{\kappa\in A} \int \kappa(x)(dx')\rho(x')(x'')$. Because A is directed and $\underline{\mathcal{K}}(X)$ is ordered pointwise, $\{\kappa(x) \mid \kappa \in A\}$ is also directed in $\underline{\mathcal{D}}(X)$. By [77, Thm. 3.3] , the right-hand-side is equal to $\int (\bigsqcup^{\uparrow}_{\kappa\in A} \kappa(x))(dx')\rho(x')(x'')$. We conclude the proof by $\kappa'(x) = \bigsqcup^{\uparrow}_{\kappa\in A} \kappa(x)$ by the definition of κ'.
Right-Scott-continuity. For all directed set $A \subseteq \underline{\mathcal{K}}(X)$ and all $\rho \in \underline{\mathcal{K}}(X)$, we want to show that $\rho \otimes (\bigsqcup^{\uparrow} A) = \bigsqcup^{\uparrow}_{\kappa\in A} \rho \otimes \kappa$. Let $\kappa' \stackrel{\text{def}}{=} \bigsqcup^{\uparrow} A$. Then it is sufficient to show that for all x and x'', $\int \rho(x)(dx')\kappa'(x')(x'') = \bigsqcup^{\uparrow}_{\kappa\in A} \int \rho(x)(dx')\kappa(x')(x'')$. Because A is directed and $\underline{\mathcal{K}}(X)$ as well as $\underline{\mathcal{D}}(X)$ are ordered pointwise, $\{\lambda x'.\kappa(x')(x'') \mid \kappa \in A\}$ is directed and bounded. By [77, Thm. 3.1], the right-hand-side is equal to $\int \rho(x)(dx')(\bigsqcup^{\uparrow}_{\kappa\in A} \lambda x'.\kappa(x')(x''))(x')$. We conclude the proof by $\lambda x'.\kappa'(x')(x'') = \bigsqcup^{\uparrow}_{\kappa\in A} \lambda x'.\kappa(x')(x'')$ by the definition of κ'.

□

3.2.2 Generalized Convexity

As shown in §3.1.3, nondeterminism-*last* is captured by convex sets of distributions. However, a more complicated notion of convexity is needed to develop nondeterminism-*first* semantics over kernels. Let X be a nonempty countable set. Every semantic object should be closed under the conditional-choice ${}_{\phi}\diamond$ for every function $\phi \in \mathbb{W}(X)$. The operation ${}_{\phi}\diamond$ corresponds to the program construct "**if** ϕ **then** $\cdots$ **else** $\cdots$ **fi**." Recall that the definition $\kappa_1 \,{}_{\phi}\diamond\, \kappa_2 \stackrel{\text{def}}{=} \phi \cdot \kappa_1 + (\dot{1} - \phi) \cdot \kappa_2$ is similar to a convex combination, except that the coefficients might not only be constants, but can also depend on the state. I formalize the idea by defining a notion of *g-convexity*.

Definition 3.11. A subset A of $\underline{\mathcal{K}}(X)$ is called *g-convex*, if for all sequences $\{\kappa_i\}_{i\in\mathbb{N}} \subseteq A$ and $\{\phi_i\}_{i\in\mathbb{N}} \subseteq \mathbb{W}(X)$ such that $\sum_{i=1}^{\infty} \phi_i = \dot{1}$, then $\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i$ is contained in A.

I now show that some domain-theoretic operations preserve g-convexity.

Lemma 3.12. *Let A be a g-convex subset of $\underline{\mathcal{K}}(X)$. Then*

1. *The saturation $\uparrow A$ and the lower closure $\downarrow A$ are g-convex.*
2. *The closure $\overline{A}$ is g-convex.*

Proof.

1. Straightforward by the fact that if $\kappa_i \sqsubseteq_K \rho_i$ for all $i \in \mathbb{Z}^+$, then $\sum_{i=0}^{\infty} \phi_i \cdot \kappa_i \sqsubseteq_K \sum_{i=0}^{\infty} \phi_i \cdot \rho_i$.
2. The Scott-closure of A can be obtained by $\overline{A} = \{\bigsqcup^{\uparrow} B \mid B \subseteq \downarrow A, B \text{ directed}\}$ [127]. For any $\{\kappa_i\}_{i\in\mathbb{N}} \subseteq \overline{A}$, there are directed subsets B_i of $\downarrow A$ such that $\kappa_i = \bigsqcup^{\uparrow} B_i$ for all

$i \in \mathbb{N}$. For any $\{\phi_i\}_{i\in\mathbb{N}} \subseteq \mathbb{W}(X)$ such that $\sum_{i=1}^{\infty} \phi_i = \dot{1}$, we have

$$\begin{aligned}
\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i &= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \kappa_i \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \left(\bigsqcup\nolimits^{\uparrow} B_i\right) \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \bigsqcup\nolimits^{\uparrow}_{\rho_i\in B_i} \phi_i \cdot \rho_i \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \bigsqcup\nolimits^{\uparrow}_{\forall i\, \rho_i\in B_i} \sum_{i=1}^{n} \phi_i \cdot \rho_i \\
&= \bigsqcup\nolimits^{\uparrow}_{\forall i\, \rho_i\in B_i} \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \rho_i \\
&= \bigsqcup\nolimits^{\uparrow}_{\forall i\, \rho_i\in B_i} \sum_{i=1}^{\infty} \phi_i \cdot \rho_i
\end{aligned}$$

where $\sum_{i=1}^{\infty} \phi_i \cdot \rho_i$ is indeed contained in $\downarrow A$ by its g-convexity and hence $\{\sum_{i=1}^{\infty} \phi_i \cdot \rho_i \mid \forall i\colon \rho_i \in B_i\}$ is a directed subset of $\downarrow A$, thus $\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i$ is contained in $\overline{A}$.

□

The *g-convex hull* of a subset A of $\underline{\mathcal{K}}(X)$ is the smallest g-convex set containing A as a subset, denoted by $gconv(A)$. Intuitively, $gconv(A)$ enriches A to become a reasonable semantic object that is closed under arbitrary conditional-choice.

Below are some properties of the $gconv(\cdot)$ operator.

Lemma 3.13. *Suppose that A and B are g-convex subsets of $\underline{\mathcal{K}}(X)$. Then $\{\kappa \,{}_{\phi}\!\diamond\, \rho \mid \kappa \in A \wedge \rho \in B\}$ is g-convex for all functions $\phi \in \mathbb{W}(X)$.*

Proof. Let $\{\eta_i\}_{i\in\mathbb{N}}$ be any sequence in $\{\kappa \,{}_{\phi}\!\diamond\, \rho \mid \kappa \in A \wedge \rho \in B\}$, and $\eta_i = \kappa_i \,{}_{\phi}\!\diamond\, \rho_i$ such

that $\kappa_i \in A, \rho_i \in B$ for all $i \in \mathbb{N}$. For any $\{\psi_i\}_{i\in\mathbb{N}} \subseteq \mathbb{W}(X)$ such that $\sum_{i=1}^{\infty} \psi_i = \dot{1}$, I have

$$\begin{aligned}
\sum_{i=1}^{\infty} \psi_i \cdot \eta_i &= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \psi_i \cdot \eta_i \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \psi_i \cdot (\kappa_i \,{}_{\phi}\!\diamond\, \rho_i) \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \psi_i \cdot (\phi \cdot \kappa_i + (\dot{1} - \phi) \cdot \rho_i) \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} ((\psi_i\phi) \cdot \kappa_i + (\psi_i - \psi_i\phi) \cdot \rho_i) \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \left(\sum_{i=1}^{n} (\psi_i\phi) \cdot \kappa_i + \sum_{i=1}^{n} (\psi_i - \psi_i\phi) \cdot \rho_i \right) \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} (\psi_i\phi) \cdot \kappa_i + \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} (\psi_i - \psi_i\phi) \cdot \rho_i \\
&= \phi \cdot \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \psi_i \cdot \kappa_i + (\dot{1} - \phi) \cdot \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \psi_i \cdot \rho_i \\
&= \left(\sum_{i=1}^{\infty} \psi_i \cdot \kappa_i \right) {}_{\phi}\!\diamond \left(\sum_{i=1}^{\infty} \psi_i \cdot \rho_i \right).
\end{aligned}$$

Because A and B are g-convex, we know that $\sum_{i=0}^{\infty} \psi_i \cdot \kappa_i \in A$ and $\sum_{i=1}^{\infty} \psi_i \cdot \rho_i \in B$. Hence $\sum_{i=1}^{\infty} \psi_i \cdot \eta_i$ is contained in $\{\kappa \,{}_{\phi}\!\diamond\, \rho \mid \kappa \in A \wedge \rho \in B\}$. □

Corollary 3.14. *If A and B are g-convex, then $gconv(A \cup B)$ is given by $\{\kappa_1 \,{}_{\phi}\!\diamond\, \kappa_2 \mid \kappa_1 \in A \wedge \kappa_2 \in B \wedge \phi \in \mathbb{W}(X)\}$.*

Proof. It is straightforward to show that $gconv(A \cup B)$ is a superset of $\{\kappa_1 \,{}_{\phi}\!\diamond\, \kappa_2 \mid \kappa_1 \in A \wedge \kappa_2 \in B \wedge \phi \in \mathbb{W}(X)\}$. Then it suffices to show this set is indeed g-convex. We conclude the proof by Lem. 3.13. □

For a finite subset F of $\underline{\mathcal{K}}(X)$, as an immediate corollary of Cor. 3.14, by a simple induction we know that $gconv(F) = \{\sum_{\kappa\in F} \phi_\kappa \cdot \kappa \mid \{\phi_\kappa\}_{\kappa\in F} \subseteq \mathbb{W}(X) \wedge \sum_{\kappa\in F} \phi_\kappa = \dot{1}\}$.

Lemma 3.15. *For an arbitrary $A \subseteq \mathcal{K}(X)$, it holds that*

$$gconv(A) = \left\{ \sum_{i=1}^{\infty} \phi_i \cdot \kappa_i \mid \{\kappa_i\}_{i\in\mathbb{N}} \subseteq A \wedge \{\phi_i\}_{i\in\mathbb{N}} \subseteq \mathbb{W}(X) \wedge \sum_{i=1}^{\infty} \phi_i = \dot{1} \right\}.$$

Proof. It is straightforward to show that $gconv(A)$ is a superset of the right-hand-side. Then we want to show the right-hand-side is indeed g-convex, which indicates the desired equality by the definition of $gconv(A)$.

Suppose $\{\kappa_i\}_{i\in\mathbb{N}}$ are contained in the right-hand-side. Then for all $i \in \mathbb{N}$, there exists $\{\kappa_{i,j}\}_{j\in\mathbb{N}} \subseteq A$ and $\{\phi_{i,j}\}_{j\in\mathbb{N}}$ such that $\sum_{j=1}^{\infty} \phi_{i,j} = \dot{1}$ and $\kappa_i = \sum_{j=1}^{\infty} \phi_{i,j} \cdot \kappa_{i,j}$. It is sufficient to show that for all $\{\phi_i\}_{i\in\mathbb{N}}$, $\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i$ is contained in the right-hand-side. We have

$$\begin{aligned}
\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i &= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \kappa_i \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \sum_{j=1}^{\infty} \phi_{i,j} \cdot \kappa_{i,j} \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \bigsqcup\nolimits^{\uparrow}_{m\in\mathbb{Z}^+} \sum_{j=1}^{m} \phi_{i,j} \cdot \kappa_{i,j} \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+,m\in\mathbb{Z}^+} \sum_{1\le i\le n,1\le j\le m} (\phi_i\phi_{i,j}) \cdot \kappa_{i,j}.
\end{aligned}$$

Let $\theta : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ be a bijection. Let $\rho_k \stackrel{\text{def}}{=} \kappa_{i,j}$ and $\psi_k \stackrel{\text{def}}{=} \phi_i\phi_{i,j}$ such that $(i,j) = \theta^{-1}(k)$. Then $\sum_{k=1}^{\infty} \psi_k = \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} \psi_{\theta(i,j)} = \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} \phi_i\phi_{i,j} = \sum_{i=1}^{\infty} \phi_i \sum_{j=1}^{\infty} \phi_{i,j} = \sum_{i=1}^{\infty} \phi_i \cdot \dot{1} = \sum_{i=1}^{\infty} \phi_i = \dot{1}$. I now have

$$\begin{aligned}
\bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+,m\in\mathbb{Z}^+} \sum_{1\le i\le n,1\le j\le m} (\phi_i\phi_{i,j}) \cdot \kappa_{i,j} &= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+,m\in\mathbb{Z}^+} \sum_{1\le i\le n,1\le j\le m} \psi_{\theta(i,j)} \cdot \rho_{\theta(i,j)} \\
&= \bigsqcup\nolimits^{\uparrow}_{l\in\mathbb{Z}^+} \sum_{k=1}^{l} \psi_k \cdot \rho_k \\
&= \sum_{k=1}^{\infty} \psi_l \cdot \rho_l
\end{aligned}$$

that is indeed contained in the right-hand-side. The second last equation is established as follows:

- To show $\bigsqcup^{\uparrow}_{n\in\mathbb{Z}^+,m\in\mathbb{Z}^+} \sum_{1\le i\le n,1\le j\le m} \psi_{\theta(i,j)} \cdot \rho_{\theta(i,j)} \sqsubseteq_K \bigsqcup^{\uparrow}_{l\in\mathbb{Z}^+} \sum_{k=1}^{l} \psi_k \cdot \rho_k$: Fix $n_o \in \mathbb{Z}^+$ and $m_o \in \mathbb{Z}^+$. Let $l_o \stackrel{\text{def}}{=} \max_{1\le i\le n_o,1\le j\le m_o} \theta(i,j)$. Then we conclude by $\sum_{1\le i\le n_o,1\le j\le m_o} \psi_{\theta(i,j)} \cdot \rho_{\theta(i,j)} \sqsubseteq_K \sum_{k=1}^{l_o} \psi_k \cdot \rho_k$.
- To show $\bigsqcup^{\uparrow}_{l\in\mathbb{Z}^+} \sum_{k=1}^{l} \psi_k \cdot \rho_k \sqsubseteq_K \bigsqcup^{\uparrow}_{n\in\mathbb{Z}^+,m\in\mathbb{Z}^+} \sum_{1\le i\le n,1\le j\le m} \psi_{\theta(i,j)} \cdot \rho_{\theta(i,j)}$: Fix $l_o \in \mathbb{Z}^+$. Let $n_o \stackrel{\text{def}}{=} \max_{1\le k\le l_o} \theta^{-1}(k).\mathbf{fst}$ and $m_o \stackrel{\text{def}}{=} \max_{1\le k\le l_o} \theta^{-1}(k).\mathbf{snd}$. Then we conclude by $\sum_{k=1}^{l_o} \psi_k \cdot \rho_k \sqsubseteq_K \sum_{1\le i\le n_o,1\le j\le m_o} \psi_{\theta(i,j)} \cdot \rho_{\theta(i,j)}$.

□

LEMMA 3.16.

1. *For an arbitrary $A \subseteq \underline{\mathcal{K}}(X)$, it holds that $\overline{gconv(A)} = \overline{gconv\left(\overline{A}\right)}$.*
2. *If $\{A_i\}_{i\in I}$ is a directed collection of Scott-closed subsets of $\underline{\mathcal{K}}(X)$ ordered by set inclusion, then $\overline{gconv\left(\bigcup A_i\right)} = \bigcup \overline{gconv(A_i)}$.*

PROOF.

1. The $\subseteq$-direction is straightforward. For the $\supseteq$-direction, we have

$$gconv\left(\overline{A}\right) = \left\{\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i \mid \{\kappa_i\}_{i\in\mathbb{N}} \subseteq \overline{A} \wedge \{\phi_i\}_{i\in\mathbb{N}} \subseteq \mathbb{W}(X) \wedge \sum_{i=1}^{\infty} \phi_i = \dot{1}\right\}$$

by Lem. 3.15 and $\overline{A} = \{\bigsqcup^{\uparrow} B \mid B \subseteq \downarrow A, B \text{ directed}\}$. Let $\kappa \stackrel{\text{def}}{=} \sum_{i=1}^{\infty} \phi_i \cdot \kappa_i$ be an element of $gconv\left(\overline{A}\right)$ where $\{\kappa_i\}_{i\in\mathbb{N}} \subseteq \overline{A}$. Then for all $i \in \mathbb{N}$, there exists a directed $B_i \subseteq \downarrow A$ satisfying $\kappa_i = \bigsqcup^{\uparrow} B_i$. Then we have

$$\begin{aligned}
\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i &= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \kappa_i \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \bigsqcup^{\uparrow} B_i \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \bigsqcup\nolimits^{\uparrow}_{\rho_i\in B_i} (\phi_i \cdot \rho_i) \\
&= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \bigsqcup\nolimits^{\uparrow}_{\forall i:\rho_i\in B_i} \sum_{i=1}^{n} \phi_i \cdot \rho_i \\
&= \bigsqcup\nolimits^{\uparrow}_{\forall i:\rho_i\in B_i} \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{i=1}^{n} \phi_i \cdot \rho_i \\
&= \bigsqcup\nolimits^{\uparrow}_{\forall i:\rho_i\in B_i} \sum_{i=1}^{\infty} \phi_i \cdot \rho_i.
\end{aligned}$$

Because $\rho_i \in B_i \subseteq \downarrow A$, there exists $\eta_i \in A$ satisfying $\rho_i \sqsubseteq_K \eta_i$ for all $i \in \mathbb{N}$, and thus $\sum_{i=1}^{\infty} \phi_i \cdot \eta_i \in gconv(A)$. We also know that $\sum_{i=1}^{\infty} \phi_i \cdot \rho_i \sqsubseteq_K \sum_{i=1}^{\infty} \phi_i \cdot \eta_i$, thus $\sum_{i=1}^{\infty} \phi_i \cdot \rho_i \in \downarrow gconv(A)$. Therefore $\sum_{i=1}^{\infty} \phi_i \cdot \kappa_i \in \overline{gconv(A)}$. By $gconv\left(\overline{A}\right) \subseteq \overline{gconv(A)}$ we conclude that $\overline{gconv\left(\overline{A}\right)} \subseteq \overline{gconv(A)}$.

2. For the $\supseteq$-direction, we have

$$\begin{aligned}
& gconv\left(\bigcup A_i\right) \supseteq gconv(A_i) \\
\Longrightarrow\ & \overline{gconv\left(\bigcup A_i\right)} \supseteq \overline{gconv(A_i)} \\
\Longrightarrow\ & \overline{gconv\left(\bigcup A_i\right)} \supseteq \bigcup \overline{gconv(A_i)} \\
\Longrightarrow\ & \overline{gconv\left(\bigcup A_i\right)} \supseteq \overline{\bigcup \overline{gconv(A_i)}}.
\end{aligned}$$

For the $\subseteq$-direction, I know that

$$gconv\left(\bigcup A_i\right) = \left\{\sum_{j=1}^{\infty} \phi_j \cdot \kappa_j \mid \{\kappa_j\}_{j\in\mathbb{N}} \subseteq \bigcup A_i \wedge \{\phi_j\}_{j\in\mathbb{N}} \subseteq \mathbb{W}(X) \wedge \sum_{j=1}^{\infty} \phi_j = \dot{1}\right\}$$

by Lem. 3.15. Let $\kappa \stackrel{\text{def}}{=} \sum_{j=1}^{\infty} \phi_j \cdot \kappa_j$ be an element of $gconv\,(\bigcup A_i)$ where $\{\kappa_j\}_{j\in\mathbb{Z}^+} \subseteq \bigcup A_i$. For all $n \in \mathbb{Z}^+$, because $\{A_i\}_{i\in\mathcal{I}}$ is directed, there exists $A_{o(n)}$ satisfying $\{\kappa_1, \cdots, \kappa_n\} \subseteq A_{o(n)}$. Thus $\sum_{j=1}^{n} \phi_j \cdot \kappa_j \in \overline{gconv\left(A_{o(n)}\right)}$. By the definition of Scott-closure, we know that $\bigsqcup^{\uparrow}_{n\in\mathbb{Z}^+} \sum_{j=1}^{n} \phi_j \cdot \kappa_j \in \bigcup \overline{\overline{gconv(A_i)}}$. Thus κ is contained in the right-hand-side and $gconv\,(\bigcup A_i) \subseteq \bigcup \overline{\overline{gconv(A_i)}}$. Hence we conclude that $\overline{gconv\,(\bigcup A_i)} \subseteq \bigcup \overline{\overline{gconv(A_i)}}$.

□

LEMMA 3.17. *Let A and B be Scott-compact g-convex subsets of $\underline{\mathcal{K}}(X)$. Then $gconv(A \cup B)$ is also Scott-compact.*

PROOF. $[0, 1]$ equipped with its usual linear order forms a Scott-compact topology. By Tychonoff's theorem we know that $X \to [0, 1]$ with the product topology is a Scott-compact space. Hence $\Gamma \stackrel{\text{def}}{=} \{(\phi, \dot{1} - \phi) \mid \phi \in \mathbb{W}(X)\}$ is also a Scott-compact space. The map from $\Gamma \times \underline{\mathcal{K}}(X) \times \underline{\mathcal{K}}(X)$ to $\underline{\mathcal{K}}(X)$ defined by $((\phi, \dot{1} - \phi), \kappa_1, \kappa_2) \mapsto \kappa_1 \,{}_{\phi}\!\diamond\, \kappa_2$ is Scott-continuous. By Cor. 3.14 we know that $gconv(A \cup B)$ is precisely the image of the Scott-compact set $\Gamma \times A \times B$. Because Scott-continuous functions preserve Scott-compactness, we conclude that $gconv(A \cup B)$ is also Scott-compact. □

I now turn to discuss some separation properties for g-convexity.

LEMMA 3.18.

1. *If $A \subseteq \underline{\mathcal{K}}(X)$ is g-convex, then for all x, $\{\kappa(x) \mid \kappa \in A\}$ is convex.*
2. *If $A \subseteq \underline{\mathcal{K}}(X)$ is Scott-compact, then for all x, $\{\kappa(x) \mid \kappa \in A\}$ is Scott-compact.*
3. *If $A \subseteq \underline{\mathcal{K}}(X)$ is Scott-closed, then for all x, $\{\kappa(x) \mid \kappa \in A\}$ is Scott-closed.*

PROOF.

1. Let $x \in X$, $\kappa_1, \kappa_2 \in A$, and $p \in [0, 1]$. We want to show that $p \cdot \kappa_1(x) + (1-p) \cdot \kappa_2(x) \in \{\kappa(x) \mid \kappa \in A\}$. Let $\phi \stackrel{\text{def}}{=} \lambda x.p$. Then $\kappa_1 \,{}_{\phi}\!\diamond\, \kappa_2 \in A$ because of g-convexity. We conclude the proof by $(\kappa_1 \,{}_{\phi}\!\diamond\, \kappa_2)(x) = \phi(x) \cdot \kappa_1(x) + (1 - \phi(x)) \cdot \kappa_2(x) = p \cdot \kappa_1(x) + (1-p) \cdot \kappa_2(x)$.
2. Let $x \in X$. Let $F(\kappa) \stackrel{\text{def}}{=} \kappa(x)$ be a map from $\underline{\mathcal{K}}(X)$ to $\underline{\mathcal{D}}(X)$. Because F is Scott-continuous and Scott-continuous functions preserve Scott-compactness, we conclude that $F(A)$ is Scott-compact because A is Scott-compact.
3. Straightforward by the fact that $\underline{\mathcal{K}}(X) = X \to \underline{\mathcal{D}}(X)$ and $\underline{\mathcal{K}}(X)$ is ordered pointwise.

□

LEMMA 3.19. *Let us consider subsets of $\underline{\mathcal{K}}(X)$. Suppose that K is a Scott-compact g-convex set and A is a nonempty Scott-closed g-convex set that is disjoint from K. Then they can be separated by a g-convex Scott-open set, i.e., there is a g-convex Scott-open set V including K and disjoint from A.*

Proof. We claim that there exists $x \in X$ such that $K(x) \cap A(x) = \emptyset$.

If not, then for all $x \in X$ there is $K(x) \cap A(x) \neq \emptyset$. Hence we can define a kernel κ such that $\kappa(x) \in K(x) \cap A(x)$ for every x. We want to show that $\kappa \in A$ and $\kappa \in K$. This follows from g-convexity of A and K: suppose $\kappa(x) = \kappa_x(x)$ such that $\kappa_x \in K$ for all x, then $\kappa = \sum_{x \in X}(\lambda x'.[x = x']) \cdot \kappa_x$. This contradicts the fact that K and A are disjoint.

Let $x \in X$ such that $K(x) \cap A(x) = \emptyset$. By Lem. 3.18(ii)(iii) we know that $K(x)$ is Scott-compact and $A(x)$ is Scott-closed. By [127, Thm. 3.8] we know that there exist a Scott-continuous linear map F and an a in $\mathbb{R}^+_\infty$ such that $F(\mu) > a > 1 \geq F(\nu)$ for all μ in $K(x)$ and ν in $A(x)$. Let $V \stackrel{\text{def}}{=} \{\kappa \mid F(\kappa(x)) > a\}$ be a Scott-open subset of $\underline{\mathcal{K}}(X)$. Then we know that $K \subseteq V$ and $A \cap V = \emptyset$. Then it suffices to show that V is g-convex. For any $\{\kappa_i\}_{i \in \mathbb{N}} \subseteq V$ and $\{\phi_i\}_{i \in \mathbb{N}} \subseteq \mathbb{W}(X)$ such that $\sum_{i=1}^\infty \phi_i = \dot{1}$. Then

$$\begin{aligned}
F\left(\left(\sum_{i=1}^\infty \phi_i \cdot \kappa_i\right)(x)\right) &= F\left(\sum_{i=1}^\infty \phi_i(x) \cdot \kappa_i(x)\right) \\
&= F\left(\bigsqcup\nolimits^{\uparrow}_{n \in \mathbb{Z}^+} \sum_{i=1}^n \phi_i(x) \cdot \kappa_i(x)\right) \\
&= \bigsqcup\nolimits^{\uparrow}_{n \in \mathbb{Z}^+} F\left(\sum_{i=1}^n \phi_i(x) \cdot \kappa_i(x)\right) \\
&= \bigsqcup\nolimits^{\uparrow}_{n \in \mathbb{Z}^+} \sum_{i=1}^n \phi_i(x) \cdot F(\kappa_i(x)) \\
&> a
\end{aligned}$$

hence $\sum_{i=1}^\infty \phi_i \cdot \kappa_i \in V$. □

Lemma 3.20. *If $K \subseteq \underline{\mathcal{K}}(X)$ is nonempty and Scott-compact, then $gconv(K)$ is Scott-compact.*

Proof. It suffices to show that any open-cover of K is an open-cover of $gconv(K)$. Let C be an open-cover of K. Let $U = \bigcup C$. If $gconv(K)$ is not contained in U, then by Lem. 3.15, there exist $\{\kappa_i\}_{i \in \mathbb{N}} \subseteq K$ and $\{\phi_i\}_{i \in \mathbb{N}} \subseteq \mathbb{W}(X)$ such that $\sum_{i=1}^\infty \phi_i = \dot{1}$ and $\kappa \stackrel{\text{def}}{=} \sum_{i=1}^\infty \phi_i \cdot \kappa_i \in gconv(K) \setminus U$. Let $A = {\downarrow}\kappa$ be a Scott-closed set, then A is disjoint from U, and thus disjoint from K. Similar to the proof of Lem. 3.19, we claim that there exist $x \in X$ and a Scott-continuous linear map F and an $a \in \mathbb{R}^+_\infty$ such that $F(\mu) > a > 1 \geq F(\nu)$ for all μ in $K(x)$ and $\nu \in A(x)$. Then $F(\kappa(x)) = F((\sum_{i=1}^\infty \phi_i \cdot \kappa_i)(x)) = F(\sum_{i=1}^\infty \phi_i(x) \cdot \kappa_i(x)) = \bigsqcup^{\uparrow}_{n \in \mathbb{Z}^+} F(\sum_{i=1}^n \phi_i(x) \cdot \kappa_i(x)) = \bigsqcup^{\uparrow}_{n \in \mathbb{Z}^+} \phi_i(x) \cdot F(\kappa_i(x)) > a > 1$, but because $\kappa \in A$ we also know that $F(\kappa(x)) \leq 1$. We then conclude the proof by contradiction. □

3.2.3 A g-convex Powerdomain for Nondeterminism-First

From the literature, a *Plotkin powertheory* [2] is defined by one binary operation $\uplus$, called *formal union*, and the following laws: (i) $A \uplus B = B \uplus A$, (ii) $(A \uplus B) \uplus C = A \uplus (B \uplus C)$,

and (iii) $A \uplus A = A$, for all objects A, B, C in the powerdomain. Intuitively, the formal union $\uplus$ represents nondeterministic-choice. Moreover, the formal union induces a semilattice ordering: $A \leq B$ if $A \uplus B = B$. The semilattice ordering is usually not interesting from the perspective of domain theory, however, it is instrumental to describe the relation between conditional-choice and nondeterministic-choice—$A \,{}_{\phi}\!\diamond\, B \leq A \uplus B$ for all semantic objects A, B—a nondeterministic-choice should *abstract* an arbitrary (possibly probabilistic) conditional-choice.

Let X be a nonempty countable set. As nondeterminism-first interprets programs as collections of input-output transformers, I want to develop a powerdomain on $\underline{\mathcal{K}}(X)$, i.e., kernels on X. To achieve this goal, I need to (i) identify a collection of well-formed semantic objects in $\wp(\underline{\mathcal{K}}(X))$, which admits a formal-union operation described above, (ii) lift conditional-choice $\,{}_{\phi}\!\diamond\,$ and composition $\otimes$ on kernels to the powerdomain properly, and (iii) prove the powerdomain is a dcpo and the operations are Scott-continuous.

Inspired by studies on convex powerdomains [2, 101, 127], I start with the following collection

$$\mathcal{G}\underline{\mathcal{K}}(X) \stackrel{\text{def}}{=} \{S \subseteq \underline{\mathcal{K}}(X) \mid S \text{ a nonempty g-convex lens}\}$$

to be the set of all g-convex lenses of $\underline{\mathcal{K}}(X)$ ordered by Egli-Miler order $A \sqsubseteq_G B \stackrel{\text{def}}{=} A \subseteq \downarrow B \wedge \uparrow A \supseteq B$. I call $\mathcal{G}\underline{\mathcal{K}}(X)$ a *g-convex powerdomain* over kernels on X.

The following theorem establishes a characterization of g-convex powerdomains.

Theorem 3.21. *$\langle \mathcal{G}\underline{\mathcal{K}}(X), \sqsubseteq_G \rangle$ forms a dcpo, with a least element $\bot_G \stackrel{\text{def}}{=} \{\bot_K\}$.*

Proof. It is straightforward to show that $\langle \mathcal{G}\underline{\mathcal{K}}(X), \sqsubseteq_G \rangle$ forms a poset and $\bot_G$ is the least element. Then it suffices to show the powerdomain admits directed suprema. For a directed collection $\mathcal{A} = \{A_i\}_{i \in I} \subseteq \mathcal{G}\underline{\mathcal{K}}(X)$, we define $\bigsqcup_i^{\uparrow} A_i \stackrel{\text{def}}{=} \overline{\bigcup_i \downarrow A_i} \cap \bigcap_i \uparrow A_i$. We now show $\bigsqcup_i^{\uparrow} A_i$ is indeed the least upper bound of $\mathcal{A}$.

We already know $\underline{\mathcal{K}}(X)$ is coherent by Thm. 3.9. Observe that $\bigsqcup_i^{\uparrow} A_i = \overline{\bigcup_i \downarrow A_i} \cap \bigcap_i \uparrow A_i = \bigcap_i (\overline{\bigcup \uparrow A_i} \cap \uparrow A_i)$ and $\{\overline{\bigcup \uparrow A_i} \cap \uparrow A_i\}_{i \in I}$ is a filtered family of nonempty lenses, or more generally, nonempty Lawson-closed subsets thus nonempty Lawson-compact subsets because of the coherence of $\underline{\mathcal{K}}(X)$. By Prop. 3.3 we know the filtered family admits a nonempty intersection. Thus $\bigsqcup_i^{\uparrow} A_i$ is a nonempty lens that is indeed g-convex by Lem. 3.12 and the g-convexity of A_i's. In this way we show that $\bigsqcup_i^{\uparrow} A_i \in \mathcal{G}\underline{\mathcal{K}}(X)$.

Let $B \stackrel{\text{def}}{=} \bigsqcup_i^{\uparrow} A_i$. To show that B is the least upper bound of $\mathcal{A}$, we claim that $\downarrow B = \overline{\bigcup_i \downarrow A_i}$ and $\uparrow B = \bigcap_i \uparrow A_i$. If so, then B is obviously an upper bound of $\mathcal{A}$ and if $A_i \sqsubseteq_G B'$ for all $i \in \mathcal{I}$, then $\downarrow A_i \subseteq \downarrow B'$ and $\uparrow A_i \supseteq \uparrow B'$ for all $i \in \mathcal{I}$, thus $\downarrow B = \overline{\bigcup_i \downarrow A_i} \subseteq \downarrow B'$ and $\uparrow B = \bigcap_i \uparrow A_i \supseteq \uparrow B'$, or equivalently, $B \sqsubseteq_G B'$. Since B' is arbitrarily chosen, we can conclude that B is the least upper bound of $\mathcal{A}$. We adapt proofs from [127] as follows.

- To show $\downarrow B = \overline{\bigcup_i \downarrow A_i}$: Inclusion is trivial. For the reverse inclusion, it is sufficient to show $\downarrow B \supseteq \bigcup_i \downarrow A_i$ since $\downarrow B$ is Scott-closed. Fix $x \in \downarrow A_i$ for some $i \in \mathcal{I}$. Then

there exists $y \in A_i$ such that $x \sqsubseteq_K y$. For all $j \in \mathcal{I}$ satisfying $A_i \sqsubseteq_G A_j$, there exists $z \in A_j$ such that $y \sqsubseteq_K z$. Therefore $\uparrow x \cap \overline{\bigcup_i \downarrow A_i} \cap \uparrow A_j \neq \emptyset$. Again a filtered family of nonempty Lawson-compact sets admits a nonempty intersection by Prop. 3.3, we have $\uparrow x \cap \overline{\bigcup_i \downarrow A_i} \cap \bigcap_j \uparrow A_j \neq \emptyset$, i.e., $\uparrow x \cap B \neq \emptyset$, thus $x \in \downarrow B$.

- To show $\uparrow B = \bigcap_i \uparrow A_i$: Inclusion is trivial. For the reverse inclusion, fix $x \in \bigcap_i \uparrow A_i$. Then we have $\downarrow x \cap \overline{\bigcup_i \downarrow A_i} \cap \uparrow A_j \neq \emptyset$ for all $j \in \mathcal{I}$. By a similar reasoning to the previous case we have $\downarrow x \cap \overline{\bigcup_i \downarrow A_i} \cap \bigcap_j \uparrow A_j \neq \emptyset$, i.e., $\downarrow x \cap B \neq \emptyset$, thus $x \in \uparrow B$.

□

We now lift conditional-choice ${}_\phi\Diamond$ (where $\phi \in \mathbb{W}(X)$) and composition $\otimes$ for kernels to the powerdomain $\mathcal{GK}(X)$ as follows.

$$A \; {}_\phi\Diamond_G \; B \stackrel{\text{def}}{=} \overline{\{a \; {}_\phi\Diamond \; b \mid a \in A \wedge b \in B\}} \cap \uparrow \{a \; {}_\phi\Diamond \; b \mid a \in A \wedge b \in B\}$$
$$A \otimes_G B \stackrel{\text{def}}{=} \overline{gconv(\{a \otimes b \mid a \in A \wedge b \in B\})} \cap \uparrow gconv(\{a \otimes b \mid a \in A \wedge b \in B\})$$

The operations construct nonempty g-convex lenses by Lemmas 3.12 and 3.20. As conditional-choice and composition operations are Scott-continuous on kernels, the lifted operations are also Scott-continuous in the powerdomain.

Lemma 3.22. *The operations ${}_\phi\Diamond_G$ and $\otimes_G$ are Scott-continuous for all $\phi \in \mathbb{W}(X)$.*

Proof. The only nontrivial part of the proof is to show $\otimes_G$ preserves directed suprema. Firstly we claim that $\downarrow(A \otimes_G B) = \overline{gconv(\{a \otimes b \mid a \in \downarrow A \wedge b \in \downarrow B\})}$ and $\uparrow(A \otimes_G B) = \uparrow gconv(\{a \otimes b \mid a \in \uparrow A \wedge b \in \uparrow B\})$. Let's write $A \dot{\otimes} B$ for $\{a \otimes b \mid a \in A \wedge b \in B\}$.

- To show $\downarrow(A \otimes_G B) = \overline{gconv(\downarrow A \dot{\otimes} \downarrow B)}$: Inclusion is trivial. For the reverse inclusion, we have $\overline{gconv(\downarrow A \dot{\otimes} \downarrow B)} \subseteq \overline{gconv(\downarrow(A \dot{\otimes} B))} = \overline{gconv(\overline{\downarrow(A \dot{\otimes} B)})} = \overline{gconv(\overline{A \dot{\otimes} B})} = \overline{gconv(A \dot{\otimes} B)} \subseteq \downarrow(A \otimes_G B)$ by Lem. 3.16(i) and Lawson-compactness of $A \otimes_G B$.
- To show $\uparrow(A \otimes_G B) = \uparrow gconv(\uparrow A \dot{\otimes} \uparrow B)$: Inclusion is trivial. For the reverse inclusion, we have $\uparrow gconv(\uparrow A \dot{\otimes} \uparrow B) \subseteq \uparrow gconv(\uparrow(A \dot{\otimes} B)) \subseteq \uparrow gconv(A \dot{\otimes} B) \subseteq \uparrow(A \otimes_G B)$.

Then it suffices to show that $\otimes_G$ is Scott-continuous in the space of down-closures (i.e., $\{\downarrow A \mid A \in \mathcal{GK}(X)\}$), as well as in the space of up-closures (i.e., $\{\uparrow A \mid A \in \mathcal{GK}(X)\}$).

- Let a directed family $\{A_i\}_{i \in \mathcal{I}}$ (ordered by inclusion) and B be nonempty Scott-closed g-convex subsets of $\mathcal{K}(X)$. We want to show that $\overline{gconv(\overline{\bigcup A_i} \dot{\otimes} B)} = \overline{\bigcup \overline{gconv(A_i \dot{\otimes} B)}}$, i.e., the left-Scott-continuity. Indeed, we have $\overline{gconv(\overline{\bigcup A_i} \dot{\otimes} B)} = \overline{gconv(\overline{\bigcup A_i} \dot{\otimes} \overline{B})} = \overline{gconv(\overline{(\bigcup A_i) \dot{\otimes} B})} = \overline{gconv((\bigcup A_i) \dot{\otimes} B)} = \overline{gconv(\bigcup (A_i \dot{\otimes} B))} = \bigcup \overline{gconv(A_i \dot{\otimes} B)}$ by Lem. 3.16 and Scott-continuity of $\otimes$ from Lem. 3.10(ii). The right-Scott-continuity is proved in a similar way.
- Let a directed family $\{A_i\}_{i \in \mathcal{I}}$ (ordered by reverse inclusion) and B be nonempty Scott-compact saturated g-convex subsets of $\mathcal{K}(X)$. We want to show that

$\uparrow gconv((\bigcap A_i)\dot{\otimes}B) = \bigcap \uparrow gconv(A_i\dot{\otimes}B)$. Inclusion is trivial. For the reverse inclusion, choose any g-convex Scott-open set U containing $\uparrow gconv(\bigcap A_i\dot{\otimes}B)$. As every g-convex Scott-compact saturated subset of a dcpo is the intersection of its g-convex Scott-open neighborhoods (by Lem. 3.19), it suffices to prove that the right-hand-side is contained in U. Observe that $gconv((\bigcap A_i)\dot{\otimes}B) \subseteq U$ and also $(\bigcap A_i)\dot{\otimes}B \subseteq U$, as $\otimes$ is Scott-continuous by Lem. 3.10(ii) and $\bigcap A_i$ and B are Scott-compact saturated, we know that $\bigcap A_i$ and B have Scott-open neighborhoods V and W respectively such that $V\dot{\otimes}W \subseteq U$. Because $\bigcap A_i \subseteq V$, by Prop. 3.3 we know there is an i such that $A_i \subseteq V$. Therefore $A_i\dot{\otimes}B \subseteq V\dot{\otimes}W \subseteq U$, and because U is g-convex, we know $gconv(A_i\dot{\otimes}B) \subseteq U$. Recall that U is Scott-open, we conclude that $\bigcap \uparrow gconv(A_i\dot{\otimes}B) \subseteq U$. The right-Scott-continuity is proved in a similar way.

□

Finally, I define a *formal union* operation $\uplus_G$ as in Prop. 3.6 to interpret nondeterministic-choice as $A \uplus_G B \stackrel{\text{def}}{=} \overline{C} \cap \uparrow C$ where C is $gconv(A \cup B)$.

LEMMA 3.23. *The formal union $\uplus_G$ is a Scott-continuous semilattice operation on $\mathcal{GK}(X)$.*

PROOF. It is straightforward to show that $\uplus_G$ is idempotent, commutative, and associative, i.e., $\uplus_G$ is a semilattice operation. Similar to the argument in the proof of Lem. 3.22, it suffices to show the Scott-continuity of $\uplus_G$ with respect to lower closures as well as upper closures.

- Let a directed family $\{A_i\}_{i\in\mathcal{I}}$ (ordered by inclusion) and B be nonempty Scott-closed g-convex subsets of $\underline{\mathcal{K}}(X)$. We want to show $\overline{gconv(\overline{\bigcup A_i} \cup B)} = \bigcup \overline{gconv(A_i \cup B)}$. Indeed, we have $\overline{gconv(\overline{\bigcup A_i} \cup B)} = \overline{gconv(\overline{\bigcup A_i} \cup B)} = \overline{gconv(\overline{\bigcup A_i \cup B})} = \overline{gconv(\bigcup A_i \cup B)} = \overline{gconv(\bigcup (A_i \cup B))} = \bigcup \overline{gconv(A_i \cup B)}$ by Lem. 3.16.
- Let a directed family $\{A_i\}_{i\in\mathcal{I}}$ (ordered by reverse inclusion) and B be nonempty Scott-compact saturated g-convex subsets of $\underline{\mathcal{K}}(X)$. We want to show that $\uparrow gconv((\bigcap A_i) \cup B) = \bigcap \uparrow gconv(A_i \cup B)$. Inclusion is trivial. For reverse inclusion, it suffices to show that for every open set U that is a neighborhood of $\uparrow gconv((\bigcap A_i) \cup B)$, we have U contains the right-hand-side as a subset by Lem. 3.19. Observe that $gconv((\bigcap A_i) \cup B) \subseteq U$ thus $(\bigcap A_i) \cup B \subseteq U$. Since $\bigcap A_i$ and B are Scott-compact saturated, there exist Scott-open neighborhoods V and W of $\bigcap A_i$ and B, respectively, such that $V \cup W \subseteq U$. Then by Prop. 3.3 we know that there exists $i \in \mathcal{I}$ such that $A_i \subseteq V$ by the fact that $\bigcap A_i \subseteq V$. Thus $A_i \cup B \subseteq V \cup W \subseteq U$. Recall that U is g-convex, we have $gconv(A_i \cup B) \subseteq U$. Moreover, U is Scott-open, thus saturated, hence we conclude that $\bigcap \uparrow gconv(A_i \cup B) \subseteq U$.

□

Example 3.24. Recall the probabilistic program P in Ex. 3.8:

$$\textbf{if } \star \textbf{ then } t := t + 1 \textbf{ else } t := t - 1 \textbf{ fi}$$

the state space X is $\mathbb{Q}$, and we want to show that for any probabilistic refinement P_r of P (i.e., $\star$ is refined by $\mathbf{prob}(r)$*), for input values t_1, t_2 of t, we have $\mathbb{E}_{t_1' \sim \Delta_1, t_2' \sim \Delta_2}[t_1' - t_2'] = t_1 - t_2$, where the program P_r ends up with a distribution Δ_1 starting with $t = t_1$ and Δ_2 with $t = t_2$.*

With the g-convex powerdomain $\mathcal{GK}(X)$ for nondeterminism-first, $t := t + 1$ and $t := t - 1$ are assigned semantic objects $\{\lambda t.\delta(t+1)\}$ and $\{\lambda t.\delta(t-1)\}$, respectively. Thus the nondeterministic-choice is interpreted as a subset of $\{\lambda t.\delta(t+1)\} \uplus_G \{\lambda t.\delta(t-1)\}$, which is $\{\kappa_r \mid r \in [0,1]\}$, where $\kappa_r = \lambda t.r \cdot \delta(t+1) + (1-r) \cdot \delta(t-1)$ is the kernel for the deterministic refinement P_r of P. Therefore for every $r \in [0,1]$, we have $\mathbb{E}_{t_1' \sim \Delta_1, t_2' \sim \Delta_2}[t_1' - t_2'] = \mathbb{E}_{t_1' \sim \kappa_r(t_1), t_2' \sim \kappa_r(t_2)}[t_1'] - \mathbb{E}_{t_1' \sim \kappa_r(t_1), t_2' \sim \kappa_r(t_2)}[t_2'] = (r(t_1+1) + (1-r)(t_1-1)) - (r(t_2+1) + (1-r)(t_2-1)) = t_1 - t_2$.

In contrast, if one started with the convex powerdomain $\mathcal{P}\underline{\mathcal{D}}(X)$ reviewed in §3.1.3 for nondeterminism-last, we would obtain the semantic object $\lambda t.\{r \cdot \delta(t+1) + (1-r) \cdot \delta(t-1) \mid r \in [0,1]\}$ for the program P, as shown in Ex. 3.8. Now the refinements of P include some κ such that $\kappa(t_1) = 0.5 \cdot \delta(t_1+1) + 0.5 \cdot \delta(t_1-1)$ and $\kappa(t_2) = 0.3 \cdot \delta(t_2+1) + 0.7 \cdot \delta(t_2-1)$, thus we are not able to prove the claim $\mathbb{E}[t_1' - t_2'] = t_1 - t_2$.

3.3 Algebraic Denotational Semantics

The operational dynamics described in §2.3 presents a reasonable model for evaluating single-procedure probabilistic programs without nondeterminism. In this section, I develop a general denotational semantics for CFHGs (introduced in §2.2) of multi-procedure probabilistic programs with nondeterminism. The semantics is *algebraic* in the sense that it could be instantiated with different concrete models of nondeterminism, e.g., nondeterminism-last reviewed in §3.1.3, as well as nondeterminism-first developed in §3.2.3. I also show the denotational semantics is equivalent to the operational semantics in §2.3 if we suppress procedure calls and nondeterminism in the programming model.

3.3.1 A Fixpoint Semantics based on Markov Algebras

The algebraic denotational semantics is obtained by composing $Ctrl(e)$ operations along hyper-edges. The semantics of programs is determined by an *interpretation*, which consists of two parts: (i) a *semantic algebra*, which defines a set of possible program meanings, and which is equipped with sequencing, conditional-choice, and nondeterministic-choice operators to compose these meanings, and (ii) a *semantic function*, which assigns a meaning to each data action $\mathsf{act} \in \mathsf{Act}$. In my thesis, I propose *Markov algebras* as the semantic algebras:

Definition 3.25. A *Markov algebra* (MA) over a set Cond of deterministic conditions is a 7-tuple $\mathcal{M} = \langle M, \sqsubseteq_M, \otimes_M, {}_\phi\Diamond_M, \uplus_M, \bot_M, 1_M \rangle$, where $\langle M, \sqsubseteq_M \rangle$ forms a dcpo with $\bot_M$ as its least element; $\langle M, \otimes_M, 1_M \rangle$ forms a monoid (i.e., $\otimes_M$ is an associative binary operator with 1_M as its identity element); ${}_\varphi\Diamond_M$ is a binary operator parametrized by a condition

$\varphi \in \mathsf{Cond}$; $\uplus_M$ is idempotent, commutative, associative and for all $a, b \in M$ and $\varphi \in \mathsf{Cond}$ it holds that $a \ {}_\varphi\diamondsuit_M \ b \leq_M a \uplus_M b$ where $\leq_M$ is the semilattice ordering induced by $\uplus_M$ (i.e., $a \leq_M b$ if $a \uplus_M b = b$); and $\otimes_M$, ${}_\varphi\diamondsuit_M$, $\uplus_M$ are Scott-continuous.

Example 3.26. Let Ω be a nonempty countable set of program states and Cond *be a set of deterministic conditions, the definition and meaning of which are given in §2.2 and §2.3.*

1. *The convex powerdomain $\mathcal{P}\underline{\mathcal{D}}(\Omega)$ admits an MA $\langle \Omega \to \mathcal{P}\underline{\mathcal{D}}(\Omega), \dot{\sqsubseteq}_P, \otimes_P, {}_\varphi\diamondsuit_P, \dot{\uplus}_P, \dot{\bot}_P, 1_P\rangle$, where $\dot{\sqsubseteq}_P, \dot{\uplus}_P, \dot{\bot}_P$ are pointwise extensions of $\sqsubseteq_P, \uplus_P, \bot_P$, defined in §3.1.3, and $g \otimes_P h \stackrel{\text{def}}{=} \widehat{\widehat{h}} \circ g$ where $\widehat{\widehat{h}}$ is given by Prop. 3.7, $g \ {}_\varphi\diamondsuit_P \ h \stackrel{\text{def}}{=} \lambda\omega.g(\omega) \ {}^{\oplus_P}_{[\![\varphi]\!](\omega)} \ h(\omega)$, as well as $1_P \stackrel{\text{def}}{=} \lambda\omega.\{\delta(\omega)\}$.*
2. *The g-convex powerdomain $\mathcal{G}\underline{\mathcal{K}}(\Omega)$ admits an MA $\langle \mathcal{G}\underline{\mathcal{K}}(\Omega), \sqsubseteq_G, \otimes_G, {}_\varphi\diamondsuit_G, \uplus_G, \bot_G, 1_G\rangle$, where $\sqsubseteq_G, \otimes_G, {}_\varphi\diamondsuit_G, \uplus_G, \bot_G$ come from §3.2.3,[2] and $1_G \stackrel{\text{def}}{=} \{\lambda\omega.\delta(\omega)\}$.*

Definition 3.27. An *interpretation* is a pair $\mathcal{I} = \langle \mathcal{M}, [\![\cdot]\!]^{\mathcal{I}}\rangle$, where $\mathcal{M}$ is an MA and $[\![\cdot]\!]^{\mathcal{I}} : \mathsf{Act} \to \mathcal{M}$. We call $\mathcal{M}$ the *semantic algebra* of the interpretation and $[\![\cdot]\!]^{\mathcal{I}}$ the *semantic function*.

Example 3.28. We can lift the interpretation of data actions defined in Fig. 2.2 to semantic functions with respect to convex or g-convex powerdomains—$\mathcal{P} = \langle \mathcal{P}\underline{\mathcal{D}}(\Omega), [\![\cdot]\!]^{\mathcal{P}}\rangle$ with $[\![\mathsf{act}]\!]^{\mathcal{P}} \stackrel{\text{def}}{=} \lambda\omega.\{[\![\mathsf{act}]\!](\omega)\}$ and $\mathcal{G} = \langle \mathcal{G}\underline{\mathcal{K}}(\Omega), [\![\cdot]\!]^{\mathcal{G}}\rangle$ with $[\![\mathsf{act}]\!]^{\mathcal{G}} \stackrel{\text{def}}{=} \{[\![\mathsf{act}]\!]\}$.

Given a probabilistic program $P = \{H_i\}_{1\leq i\leq n}$ where each $H_i = \langle V_i, E_i, v_i^{\mathsf{entry}}, v_i^{\mathsf{exit}}\rangle$ is a CFHG, and an interpretation $\mathcal{I} = \langle \mathcal{M}, [\![\cdot]\!]^{\mathcal{I}}\rangle$, I define $\mathcal{I}[\![P]\!]$ to be the interpretation of the probabilistic program, as the least fixpoint of the function F_P, which is defined as

$$\lambda\mathbf{S}.\lambda v.\begin{cases} \bigsqcup_M \left\{ \widehat{Ctrl(e)}(\mathbf{S}(u_1), \cdots, \mathbf{S}(u_k)) \mid e = \langle v, \{u_1, \cdots, u_k\}\rangle \in E \right\} & v \neq v_i^{\mathsf{exit}} \text{ for all } i \\ 1_M & \text{otherwise} \end{cases}$$

where $\widehat{Ctrl(e)}$ for different kinds of control-flow actions is defined as follows:

$$\widehat{seq[\mathsf{act}]}(S_1) \stackrel{\text{def}}{=} [\![\mathsf{act}]\!]^{\mathcal{I}} \otimes_M S_1, \qquad \widehat{cond[\varphi]}(S_1, S_2) \stackrel{\text{def}}{=} S_1 \ {}_\varphi\diamondsuit_M \ S_2,$$
$$\widehat{call[i \to j]}(S_1) \stackrel{\text{def}}{=} \mathbf{S}(v_j^{\mathsf{entry}}) \otimes_M S_1.$$

The least fixpoint of F_P exists by Prop. 3.2 as well as the following lemma. Hence the semantics of the procedure H_i is given by $[\![H_i]\!]_{\mathsf{ds}} \stackrel{\text{def}}{=} (\mathrm{lfp}^{\dot{\sqsubseteq}_M}_{\dot{\bot}_M} F_P)(v_i^{\mathsf{entry}})$.

Lemma 3.29. *The function F_P is Scott-continuous on the dcpo $\langle V \to M, \dot{\sqsubseteq}_M\rangle$ with $\dot{\bot}_M \stackrel{\text{def}}{=} \lambda v.\bot_M$ as the least element, where $\dot{\sqsubseteq}_M$ is the pointwise extension of $\sqsubseteq_M$.*

Proof. Appeal to the Scott-continuity of the operations $\otimes_M$, ${}_\varphi\diamondsuit_M$, and $\uplus_M$. □

[2] The conditional-choice is actually interpreted as ${}_{[\![\varphi]\!]}\diamondsuit_G$ in the powerdomain.

3.3.2 An Equivalence Result

To justify the denotational semantics proposed in §3.3.1, I go back to the restricted programming language used to define the operational semantics in §2.3. If we suppress the features of multi-procedure and nondeterminism, we should end up with a semantics that is equivalent to the operational semantics $[\![\cdot]\!]_{\mathsf{os}}$.

LEMMA 3.30. *Let $P = \langle V, E, v^{\mathsf{entry}}, v^{\mathsf{exit}}\rangle$ be a deterministic single-procedure probabilistic program.*

1. *If we interpret P using $\mathscr{P} = \langle \mathcal{PD}(\Omega), [\![\cdot]\!]^{\mathscr{P}}\rangle$, we will have $[\![P]\!]_{\mathsf{ds}} = \lambda\omega.\{[\![P]\!]_{\mathsf{os}}(\omega)\}$.*
2. *If we interpret P using $\mathscr{G} = \langle \mathcal{GK}(\Omega), [\![\cdot]\!]^{\mathscr{G}}\rangle$, we will have $[\![P]\!]_{\mathsf{ds}} = \{[\![P]\!]_{\mathsf{os}}\}$.*

PROOF. It is sufficient to show that

$$\lambda\omega.\ \sup_{n\in\mathbb{Z}^+}\{\xrightarrow{\wedge}_n\}(\langle v^{\mathsf{entry}}, \omega\rangle) = (\mathrm{lfp}^{\dot{\sqsubseteq}_K}_{\lambda v.\perp_K} F_P)(v^{\mathsf{entry}})$$

and we are instead going to show for all $n \in \mathbb{Z}^+$ and $v \in V$ the following holds

$$\lambda\omega.\xrightarrow{\wedge}_n(\langle v, \omega\rangle) = F_P^n(\lambda v.\perp_K)(v).$$

By induction on n, the base case is trivial because both sides compute to $\perp_K$. Suppose that for some n, the equality holds for all $v \in V$. Then for all $v \in V$, we want to show that

$$\lambda\omega.\xrightarrow{\wedge}_{n+1}(\langle v, \omega\rangle) = F_P^{n+1}(\lambda v.\perp_K)(v).$$

- If v is not associated with any edges, then $\xrightarrow{\wedge}_{n+1}(\langle v, \omega\rangle)(\omega') = [\omega = \omega']$ for all ω and ω'. The right-hand-side computes to $F_P(F_P^n(\lambda v.\perp_K))(v)$ and by the definition of F_P we know it is equal to $\lambda\omega.\lambda\omega'.[\omega = \omega']$.
- If v is associated with $e = \langle v, \{u_1, \cdots, u_k\}\rangle$, then we know $\lambda\omega.\xrightarrow{\wedge}_n(\langle u_i, \omega\rangle) = F_P^n(\lambda v.\perp_K)(u_i)$ for all i by induction hypothesis.
  - If $Ctrl(e) = seq[\mathsf{act}]$, then the right-hand-side is equal to $[\![\mathsf{act}]\!] \otimes F_P^n(\lambda v.\perp_K)(u_1)$. The left-hand-side is

$$\begin{aligned}
&\lambda\omega.\lambda\omega'.\ \sum_{\tau} \xrightarrow{\wedge}(\langle v, \omega\rangle)(\tau) \cdot \xrightarrow{\wedge}_n(\tau)(\omega') \\
&= \lambda\omega.\lambda\omega'.\ \sum_{\omega''} [\![\mathsf{act}]\!](\omega)(\omega'') \cdot \xrightarrow{\wedge}_n(\langle u_1, \omega''\rangle)(\omega') \\
&= [\![\mathsf{act}]\!] \otimes F_P^n(\lambda v.\perp_K)(u_1).
\end{aligned}$$

  - If $Ctrl(e) = cond[\varphi]$, then the right-hand-side is equal to $F_P^n(\lambda v.\perp_K)(u_1)\ {}_{[\![\varphi]\!]}\Diamond$

$F_P^n(\lambda v.\bot_K)(u_2)$. The left-hand-side is

$$
\begin{aligned}
&\lambda\omega.\lambda\omega'.\sum_{\tau} \mathrel{\hat{\longrightarrow}}(\langle v,\omega\rangle)(\tau)\cdot \mathrel{\hat{\longrightarrow}}_n(\tau)(\omega') \\
&= \lambda\omega.\lambda\omega'.\Big(\sum_{\omega''}[\![\varphi]\!](\omega)\cdot\delta(\omega)(\omega'')\cdot \mathrel{\hat{\longrightarrow}}_n(\langle u_1,\omega''\rangle)(\omega') \\
&\qquad\qquad + \sum_{\omega''}(1-[\![\varphi]\!](\omega))\cdot\delta(\omega)(\omega'')\cdot \mathrel{\hat{\longrightarrow}}_n(\langle u_2,\omega''\rangle)(\omega')\Big) \\
&= \lambda\omega.\lambda\omega'.\left([\![\varphi]\!](\omega)\cdot \mathrel{\hat{\longrightarrow}}_n(\langle u_1,\omega\rangle)(\omega') + (1-[\![\varphi]\!](\omega))\cdot \mathrel{\hat{\longrightarrow}}_n(\langle u_2,\omega\rangle)(\omega')\right) \\
&= \lambda\omega.\lambda\omega'.\left([\![\varphi]\!](\omega)\cdot F_P^n(\lambda v.\bot_K)(u_1)(\omega)(\omega') + (1-[\![\varphi]\!](\omega))\cdot F_P^n(\lambda v.\bot_K)(u_2)(\omega)(\omega')\right) \\
&= F_P^n(\lambda v.\bot_K)(u_1)\ {}_{[\![\varphi]\!]}\!\diamond\ F_P^n(\lambda v.\bot_K)(u_2).
\end{aligned}
$$

Thus we conclude the proof.

□

Chapter 4

PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs

In this chapter, I present a framework, which I call *PMAF* (for Pre-Markov Algebra Framework), for designing, implementing, and proving the correctness of static analyses of probabilistic programs. PMAF is based on the algebraic denotational semantics introduced in §3.3, which is an interpretation of the control-flow *hyper-graphs* for a program's procedures. Recall that Hyper-graphs contain *hyper-edges*, each of which consists of one source node and possibly several destination nodes. Conditional-branching, probabilistic-branching, and nondeterministic-branching statements are represented by hyper-edges. In ordinary CFGs, nodes can also have several successors; however, the operator applied at a confluence point q when analyzing a CFG is join ($\sqcup$), and the paths leading up to q are analyzed *independently*. For reasons discussed in §4.1.3, PMAF is based on a *backward* analysis, so the confluence points represent the program's branch points (i.e., for if-statements and while-loops). If the CFG is treated as a graph, join would be applied at each branch-node, and the subpaths from each successor would be analyzed independently. In contrast, when the CFG is treated as a hyper-graph, the operator applied at a probabilistic-choice node with probability p is $\lambda a.\lambda b.a\ {}_p\!\oplus\ b$—where ${}_p\!\oplus$ is not join, but an operator that weights the two successor paths by p and $1-p$. For instance, in Fig. 4.2(b), the hyper-edge $\langle v_0, \{v_1, v_5\}\rangle$ generates the inequality $\mathscr{A}[v_0] \sqsupseteq \mathscr{A}[v_1]\ {}_{0.75}\!\oplus\ \mathscr{A}[v_5]$, for some analysis $\mathscr{A}$. This approach allows the (hyper-)subpaths from the successors to be analyzed *jointly*.

To perform interprocedural analyses of probabilistic programs, I adopt a common practice from interprocedural analysis of standard non-probabilistic programs: the abstract domain is a *two-vocabulary* domain (each value represents an abstraction of a state transformer) rather than a *one-vocabulary* domain (each value represents an abstraction of a state). In the algebraic approach, an element in the algebra represents a two-vocabulary transformer.

Elements can be "multiplied" by the algebra's formal multiplication operator, which is typically interpreted as (an abstraction of) the reversal of transformer composition. The transformer obtained for the set of hyper-paths from the entry of procedure P to the exit of P is the summary for P.

In the case of loops and recursive procedures, PMAF uses widening to ensure convergence. Here my approach is slightly non-standard: I found that for some instantiations of the framework, we could improve precision by using different widening operators for loops controlled by conditional, probabilistic, and nondeterministic branches.

To evaluate PMAF, I created a prototype implementation, and reformulated two existing intraprocedural probabilistic-program analyses—the Bayesian-inference algorithm proposed by Claret et al. [30], and Markov decision problem with rewards [116]—to fit into PMAF: Reformulation involved changing from the one-vocabulary abstract domains proposed in the original papers to appropriate two-vocabulary abstract domains. I also developed a new program analysis: *linear expectation-invariant analysis* (LEIA). Linear expectation-invariants are equalities involving expected values of linear expressions over program variables.

A related approach to static analysis of probabilistic programs is *probabilistic abstract interpretation* (PAI) [39, 107–109], which lifts standard program analysis to the probabilistic setting. PAI is both general and elegant, but the more concrete approach developed in my work on PMAF has a couple of advantages. First, PMAF is algebraic and provides a simple and well-defined interface for implementing new abstractions. We provide an actual implementation of PMAF that can be easily instantiated to specific abstract domains. Second, PMAF is based on a different semantic foundation, which follows the standard interpretation of non-deterministic probabilistic programs in domain theory [41, 76, 77, 105, 106, 127].

The concrete semantics of PAI isolates probabilistic choices from the non-probabilistic part of the semantics by interpreting programs as distributions $P : \Omega \to (D \to D)$, where Ω is a probability space and $D \to D$ is the space of non-probabilistic transformers. As a result, the PAI interpretation of the following non-deterministic program is that with probability $\frac{1}{2}$, we have a program that non-deterministically returns 1 or 2; with probability $\frac{1}{4}$, we have a program that returns 1; and with probability $\frac{1}{4}$, a program that returns 2.

$$\begin{array}{lll} \textbf{if} \star & \textbf{then} & \textbf{if prob}(\tfrac{1}{2}) \textbf{ then return } 1 \textbf{ else return } 2 \\ & \textbf{else} & \textbf{if prob}(\tfrac{1}{2}) \textbf{ then return } 1 \textbf{ else return } 2 \quad \textbf{fi} \end{array}$$

In contrast, the semantics used in PMAF resolves non-determinism on the outside, and thus the semantics of the program is that it returns 1 with probability $\frac{1}{2}$ and 2 with $\frac{1}{2}$. As a result, one can conclude that the expected return value r is 1.5. However, PAI—and every static analysis based on PAI—can only conclude $r \in \{1.25, 1.5, 1.75\}$.

$b_1 \sim \textsc{Bernoulli}(0.5)$;
$b_2 \sim \textsc{Bernoulli}(0.5)$;
while $(\neg b_1 \wedge \neg b_2)$ **do**
  $b_1 \sim \textsc{Bernoulli}(0.5)$;
  $b_2 \sim \textsc{Bernoulli}(0.5)$
od

(a)

while prob$(\frac{3}{4})$ **do**
  $z \sim \textsc{Uniform}(0, 2)$;
  if $\star$ **then** $x := x + z$
  else $y := y + z$
  fi
od

(b)

Fig. 4.1: (a) Boolean probabilistic program; (b) Arithmetic probabilistic program.

4.1 Overview

In this section, I briefly introduce two different static analyses of probabilistic programs: Bayesian inference and linear expectation invariant analysis. I then informally explain the main ideas behind my algebraic framework for analyzing probabilistic programs and show how it generalizes the aforementioned analyses.

4.1.1 Example Probabilistic Programs

In PMAF, we categorize the new constructs in probabilistic programs into two kinds of randomness: (i) *data randomness*, i.e., the ability to draw random values from distributions, and (ii) *control-flow randomness*, i.e., the ability to branch probabilistically.

I use the Boolean program in Fig. 4.1(a) to illustrate data randomness. In the program, b_1 and b_2 are two Boolean-valued variables. The *sampling statement* $x \sim \textsc{Dist}(\bar{\theta})$ draws a value from a distribution $\textsc{Dist}$ with a vector of parameters $\bar{\theta}$, and assigns it to the variable x, e.g., $b_1 \sim \textsc{Bernoulli}(0.5)$ assigns to b_1 a random value drawn from a Bernoulli distribution with mean 0.5. Intuitively, the program tosses two fair Boolean-valued coins repeatedly, until one coin is *true*.

I introduce control-flow randomness through the arithmetic program in Fig. 4.1(b). In the program, x, y, and z are real-valued variables. As in the previous example, we have sampling statements, and $\textsc{Uniform}(l, r)$ represents a uniform distribution on the interval $[l, r]$. The *probabilistic choice* **prob**(p) returns true with probability p and false with probability $1 - p$. Moreover, the program also exhibits *nondeterminism*, as the symbol $\star$ stands for a *nondeterministic choice* that can behave like standard nondeterminism, as well as an arbitrary probabilistic choice [102, §6.6]. Intuitively, the program describes two players x and y playing a round-based game that ends with probability $\frac{1}{4}$ after each round. In each round, either player x or player y gains some reward that is uniformly distributed on $[0, 2]$.

4.1.2 Two Static Analyses

Bayesian Inference (BI) Probabilistic programs can be seen as descriptions of probability distributions [24, 57, 138]. For a Boolean probabilistic program, such as the one in Fig. 4.1(a), *Bayesian-inference analysis* [30] calculates the distribution over variable valuations at the end of the program, conditioned on the program terminating. The inferred probability distribution is called the *posterior probability distribution*. The program in Fig. 4.1(a) specifies the posterior distribution over the variables (b_1, b_2) given by: $\mathbb{P}[b_1 = false, b_2 = false] = 0$, and $\mathbb{P}[b_1 = false, b_2 = true] = \mathbb{P}[b_1 = true, b_2 = false] = \mathbb{P}[b_1 = true, b_2 = true] = \frac{1}{3}$. This distribution also indicates that the program terminates *almost surely*, i.e., the probability that the program terminates is 1.

Linear Expectation Invariant Analysis (LEIA) Loop invariants are crucial to verification of imperative programs [42, 50, 65]. Although loop invariants for traditional programs are usually Boolean-valued expressions over program variables, real-valued invariants are needed to prove the correctness of probabilistic loops [91, 102]. Such *expectation invariants* are usually defined as random variables—specified as expressions over program variables—with some desirable properties [25, 26, 81]. In this chapter, I work on a more general kind of expectation invariant, defined as follows:

Definition 4.1. For a program P, $\mathbb{E}[\mathcal{E}_2] \bowtie \mathcal{E}_1$ is called an *expectation invariant* if $\mathcal{E}_1$ and $\mathcal{E}_2$ are real-valued expressions over P's program variables, $\bowtie$ is one of $\{=, <, >, \leq, \geq\}$, and the following property holds: For any initial valuation of the program variables, the expected value of $\mathcal{E}_2$ in the final valuation (i.e., after the execution of P) is related to the value of $\mathcal{E}_1$ in the initial valuation by $\bowtie$.

We typically use variables with primes in $\mathcal{E}_2$ to denote the values in the final valuation. For example, for the program in Fig. 4.1(b), $\mathbb{E}[x' + y'] = x + y + 3$, $\mathbb{E}[z'] = \frac{1}{4}z + \frac{3}{4}$, $\mathbb{E}[x'] \leq x + 3$, $\mathbb{E}[x'] \geq x$, $\mathbb{E}[y'] \leq y + 3$, and $\mathbb{E}[y'] \geq y$ are several linear expectation invariants, and my analysis can derive all of these automatically! The expectation invariant $\mathbb{E}[x' + y'] = x + y + 3$ indicates that the expected value of the total reward that the two players would gain is exactly 3.

4.1.3 The Algebraic Framework

This section explains the main ideas behind PMAF, which is general enough to encode the two analyses from §4.1.2.

Data Randomness vs. Control-Flow Randomness The first principle is to make an *explicit separation between data randomness and control-flow randomness*. This distinction is intended to make the framework more flexible for analysis designers by providing multiple ways to translate the constructs of their specific probabilistic programming language into the

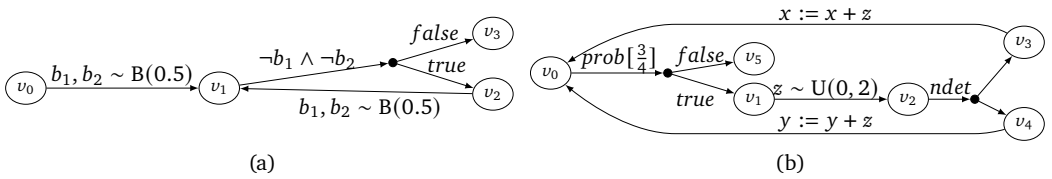


Fig. 4.2: (a) Control-flow hyper-graph of the program in Fig. 4.1(a); (b) Control-flow hyper-graph of the program in Fig. 4.1(b).

constructs of PMAF. Analysis designers may find it useful to use the control-flow-randomness construct directly (e.g., "**if prob**$(0.3)\cdots$"), rather than simulating control-flow randomness by data randomness (e.g., "$p \sim \textsc{Uniform}(0,1)$; **if** $(p < 0.3)\cdots$"). For program analysis, such a simulation can lead to suboptimal results if the constructs used in the simulation require properties to be tracked that are outside the class of properties that a particular analysis's abstract domain is capable of tracking. For example, if an analysis domain only keeps track of expectations, then analysis of "$p \sim \textsc{Uniform}(0,1)$" only indicates that $\mathbb{E}[p] = 0.5$, which does not provide enough information to establish that $\mathbb{P}[p < 0.3] = 0.3$ in the then-branch of "**if** $(p < 0.3)\cdots$". In contrast, when "**prob**$(0.3)\cdots$" is analyzed in the fragment with the explicit control-flow-randomness construct ("**if prob**$(0.3)\cdots$") the analyzer can directly assign the probabilities 0.3 and 0.7 to the outgoing branches, and use those probabilities to compute appropriate expectations in the respective branches.

I achieve the separation between data randomness and control-flow randomness by capturing the different types of randomness in the graphs that I use for representing programs. In contrast to traditional program analyses, which usually work on control-flow graphs (CFGs), I use *control-flow hyper-graphs* (CFHGs) to model probabilistic programs. Hyper-graphs are directed graphs, each edge of which (i) has one source and possibly multiple destinations, and (ii) has an associated *control-flow action*—either *sequencing*, *conditional-choice*, *probabilistic-choice*, or *nondeterministic-choice*. A traditional CFG represents a collection of execution paths, while in probabilistic programs, paths are no longer independent, and the program specifies probability distributions over the paths. It is natural to treat a collection of paths as a whole and define distributions over the collections. These kinds of collections can be precisely formalized as *hyper-paths* made up of *hyper-edges* in hyper-graphs.

Fig. 4.2 shows the CFHGs of the two programs in Fig. 4.1. Every edge has an associated action, e.g., the control-flow actions $cond[\neg b_1 \wedge \neg b_2]$, $prob[\frac{3}{4}]$, and $ndet$ are conditional-choice, probabilistic-choice, and nondeterministic-choice actions. Data actions, like $x := x+z$ and $b_1 \sim \textsc{Bernoulli}(0.5)$, also perform a trivial control-flow action to transfer control to their one destination node.

Just as the control-flow graph of a procedure typically has a single entry node and a single exit node, a procedure's control-flow hyper-graph also has a single entry node and a single exit node. In Fig. 4.2(a), the entry and exit nodes are v_0 and v_3, respectively; in

Fig. 4.2(b), the entry and exit nodes are v_0 and v_5, respectively.

Backward Analysis Traditional static analyses assign to a CFG node v either backward assertions—about the computations that can lead up to v—or forward assertions—about the computations that can continue from v [35, 37]. Backward assertions are computed via a forward analysis (in the same direction as CFG edges); forward assertions are computed via a backward analysis (counter to the flow of CFG edges).

Because we work with hyper-graphs rather than CFGs, from the perspective of a node v, there is a difference in how things "look" in the backward and forward direction: hyper-edges fan *out* in the forward direction. Hyper-edges can have two destination nodes, but only one source node.

The second principle of the framework is essentially dictated by this structural asymmetry: the framework *supports backward analyses that compute a particular kind of forward assertion*. In particular, the property to be computed for a node v in the control-flow hyper-graph for procedure P is (an abstraction of) a transformer that summarizes the transformation carried out by the hyper-graph fragment that extends from v to the exit node of P. It is possible to reason in the forward direction—i.e., about computations that lead up to v—but one would have to "break" hyper-paths into paths and "relocate" probabilities, which is more complicated than reasoning in the backward direction. The framework interprets an edge as a property transformer that computes properties of the edge's source node as a function of properties of the edge's destination node(s) and the edge's associated action. These property transformers propagate information in a *hypergraph-leaf-to-hypergraph-root* manner, which is natural in hyper-graph problems. For example, standard formulations of interprocedural dataflow analysis [88, 95, 111, 124] can be viewed as hyper-graph analyses, and propagation is performed in the leaf-to-root direction there as well.

Recall the Boolean program in Fig. 4.1(a). Suppose that we want to perform BI to analyze $\mathbb{P}[b_1 = \mathit{true}, b_2 = \mathit{true}]$ in the posterior distribution. The property to be computed for a node will be a mapping from variable valuations to probabilities, where the probability reflects the chance that a given state will cause the program to terminate in the post-state $(b_1 = \mathit{true}, b_2 = \mathit{true})$. For example, the property that we would hope to compute for node v_1 is the function $\lambda(b_1, b_2).[b_1 \wedge b_2] + [\neg b_1 \wedge \neg b_2] \cdot \frac{1}{3}$, where $[\varphi]$ is an *Iverson bracket*, which evaluates to 1 if φ is true, and 0 otherwise.

Two-Vocabulary Program Properties In the example of BI above, we can observe that the property transformation discussed above is not suitable for *interprocedural* analysis. Suppose that (i) we want analysis results to tell us something about $\mathbb{P}[b_1 = \mathit{true}, b_2 = \mathit{true}]$ in the posterior distribution of the main procedure, but (ii) to obtain the answer, the analysis must also analyze a call to some other procedure Q. In the main procedure, the analysis is driven by the posterior-probability query $\mathbb{P}[b_1 = \mathit{true}, b_2 = \mathit{true}]$; in general, however, Q will need to be analyzed with respect to some other posterior probability (obtained from the distribution of valuations at the point in main just after the call to Q). One might

try to solve this issue by analyzing each procedure multiple times with different posterior probabilities. However, in an infinite state space, this approach is no longer feasible.

Following common practice in interprocedural static analysis of traditional programs, the third principle of the framework is to work with *two-vocabulary program properties*. The property sketched in the BI example above is actually *one-vocabulary*, i.e., the property assigned to a control-flow node only involves the state at that node. In contrast, a two-vocabulary property at node v (in the control-flow hyper-graph for procedure P) should describe the state transformation carried out by the hyper-graph fragment that extends from v to the exit node of P.

For instance, LEIA assigns to each control-flow node a conjunction of expectation invariants, which relate the state at the node to the state at the exit node; consequently, LEIA deals with two-vocabulary properties. In §4.3, I reformulate BI to manipulate two-vocabulary properties. As in interprocedural dataflow analysis [36, 124], procedure summaries are used to interpret procedure calls.

Separation of Concerns The fourth principle—which is common to most analysis frameworks—is *separation of concerns*, by which I mean

> Provide a declarative interface for a client to specify the program properties to be tracked by a desired analysis, but leave it to the framework to furnish the analysis implementation by which the analysis is carried out.

I achieve this goal by adopting (and adapting) ideas from previous work on *algebraic program analysis* [48, 117, 125]. Algebraic program analysis is based on the following idea:

> Any static analysis method performs reasoning in some space of program properties and property transformers; such property transformers should obey algebraic laws.

For instance, the data action **skip**, which does nothing, can be interpreted as the *identity* element in an algebra of program-property transformers.

Concretely, the fourth principle has three aspects:

1. For an intended domain of probabilistic programs, identify an appropriate set of algebraic laws that hold for useful sets of property transformers.
2. Define a specific algebra $\mathcal{A}$ for a program-analysis problem by defining a specific set of property transformers that obey the laws identified in item 1. Give translations from data actions and control-flow actions to such property transformers. (When such a translation is applied to a specific program, it sets up an equation system to be solved over $\mathcal{A}$.)
3. Develop a generic analysis algorithm that solves an equation system over any algebra that satisfies the laws identified in item 1.

Items 1 and 3 are tasks for me, the framework designer; they are the subjects of §4.2. Item 2 is a task for a client of the framework: examples are given in §4.3.

$$\begin{array}{ll} \mathcal{S}(v_0) \sqsupseteq prob[\frac{3}{4}](\mathcal{S}(v_1), \mathcal{S}(v_5)) & \mathcal{S}(v_3) \sqsupseteq seq[x := x + z](\mathcal{S}(v_0)) \\ \mathcal{S}(v_1) \sqsupseteq seq[z \sim \textsc{Uniform}(0,2)](\mathcal{S}(v_2)) & \mathcal{S}(v_4) \sqsupseteq seq[y := y + z](\mathcal{S}(v_0)) \\ \mathcal{S}(v_2) \sqsupseteq ndet(\mathcal{S}(v_3), \mathcal{S}(v_4)) & \mathcal{S}(v_5) \sqsupseteq \underline{1} \end{array}$$

Fig. 4.3: The system of inequalities corresponding to Fig. 4.2(b).

A client of the framework must furnish an *interpretation*—which consists of a *semantic algebra* and a *semantic function*—and a program. The semantic algebra consists of a *universe*, which defines the space of possible program-property transformers, and sequencing, conditional-choice, probabilistic-choice, and nondeterministic-choice operators, corresponding to control-flow actions. The semantic function is a mapping from data actions to the universe. (An interpretation is also called a *domain*.)

To address item 3, my prototype implementation follows the standard *iterative* paradigm of static analysis [35, 83]: I first transform the control-flow hyper-graph into a system of inequalities, and then use a chaotic-iteration algorithm to compute a solution to it (e.g., [18]), which repeatedly applies the interpretation until a fixed point is reached (possibly using widening to ensure convergence). For example, the control-flow hyper-graph in Fig. 4.2(b) can be transformed into the system shown in Fig. 4.3, where $\mathcal{S}(v) \in \mathcal{M}$ are elements in the semantic algebra; $\sqsubseteq$ is the approximation order on $\mathcal{M}$; $[\![\cdot]\!]$ is the semantic function, which maps data actions to $\mathcal{M}$; and $\underline{1}$ is the transformer associated with the exit node.

The soundness of the analysis (with respect to a concrete semantics) is proved by (i) establishing an approximation relation between the concrete domain and the abstract domain; (ii) showing that the abstract semantic function approximates the concrete one; and (iii) showing that the abstract operators (sequencing, conditional-choice, probabilistic-choice, and nondeterministic-choice) approximate the concrete ones.

For BI, I instantiate the PMAF framework to give lower bounds on posterior distributions, using with an interpretation in which state transformers are probability matrices (see §4.3.1). For LEIA, I design an interpretation using a Cartesian product of polyhedra (see §4.3.3). Once the functions of the interpretations are implemented, and a program is translated into the appropriate hyper-graph, the framework handles the rest of the work, namely, solving the equation system.

4.2 Analysis Framework

To aid in creating abstractions of probabilistic programs, I first identify, in §4.2.1, some algebraic properties that underlie the mechanisms used in the semantics from §3.3. This algebra will aid my later definitions of abstractions in §4.2.2. I then discuss interprocedural analysis (in §4.2.3) and widening (§4.2.4).

4.2.1 An Algebraic Characterization of Fixpoint Semantics

In the denotational semantics, the concrete semantics is obtained by composing $\widehat{Ctrl(e)}$ operations along hyper-paths. Hence in the algebraic framework, the semantics of probabilistic programs is denoted by an *interpretation*, which consists of two parts: (i) a *semantic algebra*, which defines a set of possible program meanings, and which is equipped with sequencing, conditional-choice, probabilistic-choice, and nondeterministic-choice operators to compose these meanings, and (ii) a *semantic function*, which assigns a meaning to each basic program action.

Recall that in §3.3, I introduce Markov algebras (MAs) as the semantic algebras. The lattices used for abstract interpretation are *pre-Markov algebras*:

Definition 4.2 (Pre-Markov algebras). A *pre-Markov algebra* (PMA) over a set of logical conditions $\mathcal{L}$ is an 8-tuple $\mathcal{M} = \langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \uplus, \underline{\bot}, \underline{1}\rangle$, which is essentially an MA, *except* that $\langle M, \sqsubseteq\rangle$ forms a complete lattice; $\otimes$, ${}_{p}\oplus$, ${}_{\varphi}\Diamond$, and $\uplus$ are only required to be monotone; and the following properties hold:

$$a \sqsubseteq a\ {}_{\varphi}\Diamond\ a,\ a \sqsubseteq a\ {}_{true}\Diamond\ b, \qquad a\ {}_{\varphi}\Diamond\ b = b\ {}_{\neg\varphi}\Diamond\ a$$
$$a \sqsubseteq a\ {}_{p}\oplus\ a,\quad a \sqsubseteq a\ {}_{1}\oplus\ b\ , \qquad a\ {}_{p}\oplus\ b = b\ {}_{1-p}\oplus\ a$$
$$a\ {}_{\varphi}\Diamond\ (b\ {}_{\psi}\Diamond\ c) = (a\ {}_{\varphi'}\Diamond\ b)\ {}_{\psi'}\Diamond\ c \text{ where } \varphi = \varphi' \wedge \psi', \varphi \vee \psi = \psi'$$
$$a\ {}_{p}\oplus\ (b\ {}_{q}\oplus\ c) = (a\ {}_{p'}\oplus\ b)\ {}_{q'}\oplus\ c \text{ where } p = p'q', \overline{p} \cdot \overline{q} = \overline{q'}$$

The precedence of the operators is that $\otimes$ binds tightest, followed by ${}_{\varphi}\Diamond$, ${}_{p}\oplus$, and $\uplus$.

Remark 4.3. These algebraic laws are not needed to prove soundness of the framework (stated in Thm. 4.6). These laws helped us when designing the abstract domains. Exploiting these algebraic laws to design better algorithms is an interesting direction for future work.

As is standard in abstract interpretation, the order on the algebra should represent an approximation order: $a \sqsubseteq b$ iff a is approximated by b (i.e., if a represents a more precise property than b).

Definition 4.4 (Interpretations). An interpretation is a pair $\mathcal{I} = \langle \mathcal{M}, [\![\cdot]\!]\rangle$, where $\mathcal{M}$ is a pre-Markov algebra, and $[\![\cdot]\!] : \mathcal{A} \to \mathcal{M}$, where $\mathcal{A}$ is the set of data actions for probabilistic programs. We call $\mathcal{M}$ the *semantic algebra* of the interpretation and $[\![\cdot]\!]$ the *semantic function*.

Given a probabilistic program P and an interpretation $\mathcal{I} = \langle \mathcal{M}, [\![\cdot]\!]\rangle$, I define $\mathcal{I}[P]$ to be the interpretation of the probabilistic program. $\mathcal{I}[P]$ is then defined as the *least fixed point* of the function $F^{\sharp}_{P}$, which is defined as

$$\lambda\mathcal{S}^{\sharp}.\lambda v.\begin{cases}\widehat{Ctrl(e)}^{\sharp}(\mathcal{S}^{\sharp}(u_1), \ldots, \mathcal{S}^{\sharp}(u_k)) & e = \langle v, \{u_1, \ldots, u_k\}\rangle \in E \\ \underline{1} & \text{otherwise}\end{cases}$$

where

$\widehat{seq_{\mathsf{act}}}^\sharp(a_1) \stackrel{\text{def}}{=} [\![\mathsf{act}]\!] \otimes a_1$	$\widehat{cond[\varphi]}^\sharp(a_1, a_2) \stackrel{\text{def}}{=} a_1 \ {}_\varphi\diamondsuit\ a_2$
$\widehat{call[i]}^\sharp(a_1) \stackrel{\text{def}}{=} \mathcal{S}^\sharp(v_i^{\mathsf{entry}}) \otimes a_1$	$\widehat{prob[p]}^\sharp(a_1, a_2) \stackrel{\text{def}}{=} a_1 \ {}_p\oplus\ a_2$
	$\widehat{ndet}^\sharp(a_1, a_2) \stackrel{\text{def}}{=} a_1 \uplus a_2$

By the Knaster-Tarski theorem, I use the least fixed point of $F_P^\sharp$ to define the interpretation of a probabilistic program P as $\mathscr{I}[P] = \mathrm{lfp}^{\dot{\sqsubseteq}}_{\lambda v.\underline{\bot}} F_P^\sharp$. The interpretation of a control-flow node v is then defined as $\mathscr{I}[v] = \mathscr{I}[P](v)$.

4.2.2 Abstractions of Probabilistic Programs

Given an MA C and a PMA $\mathcal{A}$, a *probabilistic abstraction* is defined as follows:

Definition 4.5 (Probabilistic abstractions). A *probabilistic over-abstraction (or under-abstraction, resp.)* from a PMA C to a PMA $\mathcal{A}$ is a concretization mapping, $\gamma : \mathcal{A} \to C$, such that

- $\underline{1}_C \sqsubseteq_C \gamma(\underline{1}_{\mathcal{A}})$ (or $\gamma(\underline{1}_{\mathcal{A}}) \sqsubseteq_C \underline{1}_C$, resp.),
- for all $Q_1, Q_2 \in \mathcal{A}$, $\gamma(Q_1) \otimes_C \gamma(Q_2) \sqsubseteq_C \gamma(Q_1 \otimes_{\mathcal{A}} Q_2)$ (or $\gamma(Q_1 \otimes_{\mathcal{A}} Q_2) \sqsubseteq_C \gamma(Q_1) \otimes_C \gamma(Q_2)$, resp.),
- for all $Q_1, Q_2 \in \mathcal{A}$, $\gamma(Q_1) \ {}_\varphi\diamondsuit_C\ \gamma(Q_2) \sqsubseteq_C \gamma(Q_1 \ {}_\varphi\diamondsuit_{\mathcal{A}}\ Q_2)$ (or $\gamma(Q_1 \ {}_\varphi\diamondsuit_{\mathcal{A}}\ Q_2) \sqsubseteq_C \gamma(Q_1) \ {}_\varphi\diamondsuit_C\ \gamma(Q_2)$, resp.),
- for all $Q_1, Q_2 \in \mathcal{A}$, $\gamma(Q_1) \ {}_p\oplus_C\ \gamma(Q_2) \sqsubseteq_C \gamma(Q_1 \ {}_p\oplus_{\mathcal{A}}\ Q_2)$ (or $\gamma(Q_1 \ {}_p\oplus_{\mathcal{A}}\ Q_2) \sqsubseteq_C \gamma(Q_1) \ {}_p\oplus_C\ \gamma(Q_2)$, resp.), and
- for all $Q_1, Q_2 \in \mathcal{A}$, $\gamma(Q_1) \uplus_C \gamma(Q_2) \sqsubseteq_C \gamma(Q_1 \uplus_{\mathcal{A}} Q_2)$ (or $\gamma(Q_1 \uplus_{\mathcal{A}} Q_2) \sqsubseteq_C \gamma(Q_1) \uplus_C \gamma(Q_2)$, resp.).

A probabilistic abstraction leads to a sound analyses:

Theorem 4.6. *Let $\mathscr{C}$ and $\mathscr{A}$ be interpretations over the MA C and the PMA $\mathcal{A}$; let γ be a probabilistic over-abstraction (or under-abstraction, resp.) from C to $\mathcal{A}$; and let P be an arbitrary probabilistic program. If for all basic actions* act*, $[\![\mathsf{act}]\!]^{\mathscr{C}} \sqsubseteq_C \gamma([\![\mathsf{act}]\!]^{\mathscr{A}})$ (or $\gamma([\![\mathsf{act}]\!]^{\mathscr{A}}) \sqsubseteq_C [\![\mathsf{act}]\!]^{\mathscr{C}}$, resp.), then it holds that $\mathscr{C}[P] \dot{\sqsubseteq}_C \dot{\gamma}(\mathscr{A}[P])$ (or $\dot{\gamma}(\mathscr{A}[P]) \dot{\sqsubseteq}_C \mathscr{C}[P]$, resp.).*

Proof. Without loss of generality, I present the proof for the over-approximations. By definition, $\mathscr{C}[P] = \mathrm{lfp}^{\dot{\sqsubseteq}_C}_{\lambda v.\underline{\bot}_C} F_P^{\mathscr{C}} = \sup\{(F_P^{\mathscr{C}})^n(\lambda v.\underline{\bot}_C)\}$ by Kleene, and $\mathscr{A}[P] = \mathrm{lfp}^{\dot{\sqsubseteq}_{\mathcal{A}}}_{\lambda v.\underline{\bot}_{\mathcal{A}}} F_P^{\mathscr{A}}$ obtained by Knaster-Tarski. We want to show that for all n it holds that $(F_P^{\mathscr{C}})^n(\lambda v.\underline{\bot}_C) \dot{\sqsubseteq}_C \dot{\gamma}(\mathscr{A}[P])$. Let's prove by induction on n. The base case follows directly from the fact that $\underline{\bot}_C$ is the least element in C. Suppose we know $(F_P^{\mathscr{C}})^n(\lambda v.\underline{\bot}_C) \dot{\sqsubseteq}_C \dot{\gamma}(\mathscr{A}[P])$ for some n. Let me denote the left hand side by LHS and $\mathscr{A}[P]$ by SOL. We want to show that $F_P^{\mathscr{C}}(LHS) \dot{\sqsubseteq}_C \dot{\gamma}(SOL)$. This expands to $F_P^{\mathscr{C}}(LHS)(v) \sqsubseteq_C \gamma(SOL(v))$ for all $v \in V$. We proceed by a case analysis on the kind of edges leaving v.

1. If $v = v_i^{\mathsf{exit}}$ for some i, then $F_P^{\mathscr{C}}(LHS)(v) = \underline{1}_C$. Then we can conclude this case by showing that $SOL(v) = \underline{1}_{\mathscr{A}}$. By definition of SOL, we know that $F_P^{\mathscr{A}}(SOL) = SOL$, thus $F_P^{\mathscr{A}}(SOL)(v) = SOL(v)$. By definition of $F_P^{\mathscr{A}}$, we know that $F_P^{\mathscr{A}}(SOL)(v) = \underline{1}_{\mathscr{A}}$.
2. If $v \neq v_i^{\mathsf{exit}}$ for all i, then v is associated with some $e = \langle v, \{u_1, \cdots, u_k\}\rangle \in E$, and we have

$$\begin{aligned} F_P^{\mathscr{C}}(LHS)(v) &= \widehat{Ctrl(e)}(LHS(u_1), \cdots, LHS(u_k)) \\ &\sqsubseteq_C \widehat{Ctrl(e)}(\gamma(SOL(u_1)), \cdots, \gamma(SOL(u_k))). \end{aligned}$$

If we can prove that for any kind of $Ctrl(e)$ it holds that $\widehat{Ctrl(e)}(\gamma(x_1), \cdots, \gamma(x_k)) \sqsubseteq_C \gamma(\widehat{Ctrl(e)}^\sharp(x_1, \cdots, x_k))$, then we can conclude the case by the following argument:

$$\begin{aligned} F_P^{\mathscr{C}}(LHS)(v) &\sqsubseteq_C \gamma(\widehat{Ctrl(e)}^\sharp(SOL(u_1), \cdots, SOL(u_k))) \\ &= \gamma(F_P^{\mathscr{A}}(SOL)(v)) \\ &= \gamma(SOL(v)). \end{aligned}$$

Now consider the form of $Ctrl(e)$.

- $Ctrl(e) = seq[\mathsf{act}]$: We want to show that $\widehat{seq[\mathsf{act}]}(\gamma(x_1)) \sqsubseteq_C \gamma(\widehat{seq[\mathsf{act}]}^\sharp(x_1))$. It is equivalent to $[\![\mathsf{act}]\!]^{\mathscr{C}} \otimes_C \gamma(x_1) \sqsubseteq_C \gamma([\![\mathsf{act}]\!]^{\mathscr{A}} \otimes_{\mathscr{A}} x_1)$. Indeed, we have

$$[\![\mathsf{act}]\!]^{\mathscr{C}} \otimes_C \gamma(x_1) \sqsubseteq_C \gamma([\![\mathsf{act}]\!]^{\mathscr{A}}) \otimes_C \gamma(x_1) \sqsubseteq_C \gamma([\![\mathsf{act}]\!]^{\mathscr{A}} \otimes_{\mathscr{A}} x_1)$$

  by assumption, monotonicity of $\otimes_C$, and properties of γ.
- $Ctrl(e) = call[i \to j]$: We want to show that $\widehat{call[i \to j]}(\gamma(x_1)) \sqsubseteq_C \gamma(\widehat{call[i \to j]}^\sharp(x_1))$. It is equivalent to $LHS(v_j^{\mathsf{entry}}) \otimes_C \gamma(x_1) \sqsubseteq_C \gamma(SOL(v_j^{\mathsf{entry}}) \otimes_{\mathscr{A}} x_1)$. Indeed, we have

$$LHS(v_j^{\mathsf{entry}}) \otimes_C \gamma(x_1) \sqsubseteq_C \gamma(SOL(v_j^{\mathsf{entry}})) \otimes_C \gamma(x_1) \sqsubseteq_C \gamma(SOL(v_j^{\mathsf{entry}}) \otimes_{\mathscr{A}} x_1)$$

  by induction hypothesis, monotonicity of $\otimes_C$, and properties of γ.
- $Ctrl(e) = cond[\varphi]$: We want to show that $\widehat{cond[\varphi]}(\gamma(x_1), \gamma(x_2)) \sqsubseteq_C \gamma(\widehat{cond[\varphi]}^\sharp(x_1, x_2))$. It is equivalent to $\gamma(x_1)\ {}_\varphi\Diamond_C\ \gamma(x_2) \sqsubseteq_C \gamma(x_1\ {}_\varphi\Diamond_{\mathscr{A}}\ x_2)$. Appeal to properties of γ.
- $Ctrl(e) = prob[p]$: We want to show that $\widehat{prob[p]}(\gamma(x_1), \gamma(x_2)) \sqsubseteq_C \gamma(\widehat{prob[p]}^\sharp(x_1, x_2))$. It is equivalent to $\gamma(x_1)\ {}_p\oplus_C\ \gamma(x_2) \sqsubseteq_C \gamma(x_1\ {}_p\oplus_{\mathscr{A}}\ x_2)$. Appeal to properties of γ.
- $Ctrl(e) = ndet$: We want to show that $\widehat{ndet}(\gamma(x_1), \gamma(x_2)) \sqsubseteq_C \gamma(\widehat{ndet}^\sharp(x_1, x_2))$. It is equivalent to $\gamma(x_1) \uplus_C \gamma(x_2) \sqsubseteq_C \gamma(x_1 \uplus_{\mathscr{A}} x_2)$. Appeal to properties of γ.

□

4.2.3 Interprocedural Analysis Algorithm

We are given a probabilistic program P and an interpretation $\mathscr{A} = \langle \mathcal{A}, [\![\cdot]\!]^{\mathscr{A}} \rangle$, where $\mathcal{A} = \langle M_{\mathcal{A}}, \sqsubseteq_{\mathcal{A}}, \otimes_{\mathcal{A}}, {}_{\varphi}\diamond_{\mathcal{A}}, {}_{p}\oplus_{\mathcal{A}}, \uplus_{\mathcal{A}}, \underline{\bot}_{\mathcal{A}}, \underline{1}_{\mathcal{A}} \rangle$ is a PMA and $[\![\cdot]\!]^{\mathscr{A}}$ is a semantic function. The goal is to compute (an overapproximation of) $\mathscr{A}[P] = \mathrm{lfp}^{\dot{\sqsubseteq}_{\mathcal{A}}}_{\lambda v.\underline{\bot}_{\mathcal{A}}} F^{\sharp}_{P}$. An equivalent way to define $\mathscr{A}[P]$ is to specify it as the least solution to a system of inequalities on $\{\mathscr{A}[v] \mid v \in V\}$ (where $e \in E$ in each case):

	e	$Ctrl(e)$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} [\![\mathsf{act}]\!]^{\mathscr{A}} \otimes_{\mathcal{A}} \mathscr{A}[u_1]$	$\langle v, \{u_1\}\rangle$	$seq[\mathsf{act}]$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[u_1] \; {}_{\varphi}\diamond_{\mathcal{A}} \; \mathscr{A}[u_2]$	$\langle v, \{u_1, u_2\}\rangle$	$cond[\varphi]$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[u_1] \; {}_{p}\oplus_{\mathcal{A}} \; \mathscr{A}[u_2]$	$\langle v, \{u_1, u_2\}\rangle$	$prob[p]$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[u_1] \uplus_{\mathcal{A}} \mathscr{A}[u_2]$	$\langle v, \{u_1, u_2\}\rangle$	$ndet$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[v_i^{\mathsf{entry}}] \otimes_{\mathcal{A}} \mathscr{A}[u_1]$	$\langle v, \{u_1\}\rangle$	$call[i]$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \underline{1}_{\mathcal{A}}$	if $v = v_i^{\mathsf{exit}}$	

Note that in line 5 a call is treated as a hyper-edge with the action $\lambda(entry, succ).entry \otimes_{\mathcal{A}} succ$. There is no explicit return edge to match a call (as in many multi-procedure program representations, e.g., [118]); instead, each exit node is initialized with the constant $\underline{1}_{\mathcal{A}}$ (line 6).

I mainly use known techniques from previous work on interprocedural dataflow analysis, with some adaptations to my setting, which uses hyper-graphs instead of ordinary graphs (i.e., CFGs).[1] The analysis direction is backward, and the algorithm is similar to methods for computing summary edges in demand interprocedural-dataflow-analysis algorithms ([71, Fig. 4], [121, Fig. 10]). The algorithm uses a standard chaotic-iteration strategy [18] (except that propagation is performed along hyper-edges instead of edges); it uses a fair iteration strategy for selecting the next edge to consider.

4.2.4 Widening

Widening is a general technique in static analysis to ensure and speed up convergence [34, 36]. To choose the nodes at which widening is to be applied, I treat the hyper-graph as a graph—i.e., each hyper-edge (including calls) contributes one or two ordinary edges. More precisely, I construct a *dependence graph* $G(H) = \langle N, A \rangle$ from hyper-graph $H = \{\langle V_i, E_i, v_i^{\mathsf{entry}}, v_i^{\mathsf{exit}} \rangle\}_{1 \le i \le n}$ by defining $N \stackrel{\text{def}}{=} \bigcup_{1 \le i \le n} V_i$, and

$$\begin{aligned} A \stackrel{\text{def}}{=} & \{\langle u, v\rangle \mid \exists e \in E.(v = src(e) \wedge u \in Dst(e))\} \\ & \cup \{\langle v_i^{\mathsf{entry}}, v\rangle \mid \exists e \in E.(v = src(e) \wedge Ctrl(e) = call[i])\}. \end{aligned} \tag{4.1}$$

[1] As mentioned in §4.1.3, standard formulations of interprocedural dataflow analysis [88, 95, 111, 124] can be viewed as hyper-graph analyses. In that setting, one deals with hyper-graphs with constituent control-flow graphs. With PMAF, because each procedure is represented as a hyper-graph, one has hyper-graphs of constituent hyper-graphs. Fortunately, each procedure's hyper-graph is a *single-entry/single-exit* hyper-graph, so the basic ideas and algorithms from standard interprocedural dataflow analysis carry over to PMAF.

I then compute a set W of widening points for $G(H)$ via the algorithm of Bourdoncle [18, Fig. 4]. Because of the second set-former in (4.1), W contains widening points that cut each cycle caused by recursion.

While traditional programs exhibit only one sort of choice operator, probabilistic programs can have three different kinds of choice operators, and hence loops can exhibit three different kinds of behavior. I found that if we used the same widening operator for all widening nodes, there could be a substantial loss in precision. Thus, I equip the framework with three separate widening operators: $\triangledown_c$, $\triangledown_p$, and $\triangledown_n$. Let $v \in W$ be the source of edge $e \in E$. Then the inequalities become

	e	$Ctrl(e)$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[v] \triangledown_n (\llbracket \mathsf{act} \rrbracket^{\mathscr{A}} \otimes_{\mathcal{A}} \mathscr{A}[u_1])$	$\langle v, \{u_1\}\rangle$	$seq[\mathsf{act}]$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[v] \triangledown_c (\mathscr{A}[u_1] \ {}_{\varphi}\Diamond_{\mathcal{A}} \ \mathscr{A}[u_2])$	$\langle v, \{u_1, u_2\}\rangle$	$cond[\varphi]$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[v] \triangledown_p (\mathscr{A}[u_1] \ {}_{p}\oplus_{\mathcal{A}} \ \mathscr{A}[u_2])$	$\langle v, \{u_1, u_2\}\rangle$	$prob[p]$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[v] \triangledown_n (\mathscr{A}[u_1] \uplus_{\mathcal{A}} \mathscr{A}[u_2])$	$\langle v, \{u_1, u_2\}\rangle$	$ndet$
$\mathscr{A}[v] \sqsupseteq_{\mathcal{A}} \mathscr{A}[v] \triangledown_n (\mathscr{A}[v_i^{\mathsf{entry}}] \otimes_{\mathcal{A}} \mathscr{A}[u_1])$	$\langle v, \{u_1\}\rangle$	$call[i]$

Observation 4.7. Recall from Defn. 2.7 that in a probabilistic program each non-exit node has exactly one outgoing hyper-edge. In each right-hand side above, the second argument to the widening operator re-evaluates the action of the (one outgoing) hyper-edge. Consequently, during an analysis, there is an invariant that whenever a widening operation $a \triangledown b$ is performed, the property $a \sqsubseteq_{\mathcal{A}} b$ holds.

The safety properties for the three widening operators are adaptations of the standard stabilization condition: For every pair of ascending chains $\{a_k\}_{k\in\mathbb{N}}$ and $\{b_k\}_{k\in\mathbb{N}}$,

- the chain $\{c_k\}_{k\in\mathbb{N}}$ defined by $c_0 = a_0 \ {}_{\varphi}\Diamond_{\mathcal{A}} \ b_0$ and $c_{k+1} = c_k \triangledown_c (a_{k+1} \ {}_{\varphi}\Diamond_{\mathcal{A}} \ b_{k+1})$ is eventually stable;
- the chain $\{c_k\}_{k\in\mathbb{N}}$ defined by $c_0 = a_0 \ {}_{p}\oplus_{\mathcal{A}} \ b_0$ and $c_{k+1} = c_k \triangledown_p (a_{k+1} \ {}_{p}\oplus_{\mathcal{A}} \ b_{k+1})$ is eventually stable; and
- the chain $\{c_k\}_{k\in\mathbb{N}}$ defined by $c_0 = a_0 \uplus_{\mathcal{A}} b_0$ and $c_{k+1} = c_k \triangledown_n (a_{k+1} \uplus_{\mathcal{A}} b_{k+1})$ is eventually stable.

4.3 Instantiations

In this section, I instantiate the framework to derive three important analyses: Bayesian inference (BI) (§4.3.1), computing rewards in Markov decision processes (§4.3.2), and linear expectation-invariant analysis (LEIA) (§4.3.3).

4.3.1 Bayesian Inference

Claret et al. [30] proposed a technique to perform Bayesian inference on Boolean programs using dataflow analysis. They use a forward analysis to compute the posterior distribution

of a single-procedure, well-structured, probabilistic program. Their analysis is similar to an intraprocedural dataflow analysis: they use discrete joint-probability distributions as dataflow facts, merge these facts at join points, and compute fixpoints in the presence of loops. Let Var be the set of program variables; the set of program states is $\Omega = \mathsf{Var} \to \mathbb{B}$. Note that Ω is isomorphic to $\mathbb{B}^{|\mathsf{Var}|}$, and consequently, a distribution can be represented by a vector of length $2^{|\mathsf{Var}|}$ of reals in $\mathbb{R}_{[0,1]}$. (Their implementation uses Algebraic Decision Diagrams [4] to represent distributions compactly.)

The algorithm by Claret et al. [30, Alg. 2] is defined inductively on the structure of programs—for example, the output distribution of $x \sim \textsc{Bernoulli}(r)$ from an input distribution μ, denoted by $Post(\mu, x \sim \textsc{Bernoulli}(r))$, is computed as $\lambda\sigma'.\left(r \cdot \sum_{\{\sigma|\sigma'=\sigma[x\leftarrow true]\}} \mu(\sigma) + (1-r) \cdot \sum_{\{\sigma|\sigma'=\sigma[x\leftarrow false]\}} \mu(\sigma)\right)$.

I have used PMAF to extend their work in two dimensions, creating (i) an *interprocedural* version of Bayesian inference with (ii) *nondeterminism*. Because of nondeterminism, for a given input state the posterior distribution is not unique; consequently, my goal is to compute procedure summaries that gives *lower bounds* on posterior distributions.

To reformulate the domain in the two-vocabulary setting needed for computing procedure summaries, we introduce Var′, primed versions of the variables in Var. Var and Var′ denote the variables in the pre-state and post-state of a state transformer. A distribution transformer (and therefore a procedure summary) is a matrix of size $2^{|\mathsf{Var}|} \times 2^{|\mathsf{Var}'|}$ of reals in $\mathbb{R}_{[0,1]}$. We define a PMA $\mathcal{B} = \langle M_{\mathcal{B}}, \sqsubseteq_{\mathcal{B}}, \otimes_{\mathcal{B}}, {}_{\varphi}\diamondsuit_{\mathcal{B}}, {}_{p}\oplus_{\mathcal{B}}, \uplus_{\mathcal{B}}, \underline{\bot}_{\mathcal{B}}, \underline{1}_{\mathcal{B}}\rangle$ as follows:

$M_{\mathcal{B}} \stackrel{\text{def}}{=} 2^{\lvert\mathsf{Var}\rvert} \times 2^{\lvert\mathsf{Var}'\rvert} \to \mathbb{R}_{[0,1]}$	
$a \sqsubseteq_{\mathcal{B}} b \stackrel{\text{def}}{=} a \dot{\leq} b$	$a \uplus_{\mathcal{B}} b \stackrel{\text{def}}{=} \dot{\min}(a, b)$
$a \otimes_{\mathcal{B}} b \stackrel{\text{def}}{=} a \times b$	$\underline{\bot}_{\mathcal{B}} \stackrel{\text{def}}{=} \lambda(s,t).0$
$a\ {}_{p}\oplus_{\mathcal{B}}\ b \stackrel{\text{def}}{=} p \cdot a + (1-p) \cdot b$	$\underline{1}_{\mathcal{B}} \stackrel{\text{def}}{=} \lambda(s,t).[s = t]$
$a\ {}_{\varphi}\diamondsuit_{\mathcal{B}}\ b \stackrel{\text{def}}{=} \lambda(s,t).$**if** $\widehat{\varphi}(s)$ **then** $a(s,t)$ **else** $b(s,t)$	

The use of pointwise min in the definition of $a \uplus_{\mathcal{B}} b$ causes the analysis to compute procedure summaries that provide lower bounds on the posterior distributions.

Let $\mathscr{B} = \langle \mathcal{B}, [\![\cdot]\!]^{\mathscr{B}} \rangle$ be the interpretation for Bayesian inference. I define the semantic function as $[\![x := \mathcal{E}]\!]^{\mathscr{C}} = \lambda(s, A).[s[x \leftarrow \mathcal{E}(s)] \in A]$ and $[\![x := \mathcal{E}]\!]^{\mathscr{B}} = \lambda(s,t).[s[x \leftarrow \mathcal{E}(s)] = t]$, as well as $[\![x \sim \textsc{Bernoulli}(p)]\!]^{\mathscr{C}} = \lambda(s, A).p \cdot [s[x \leftarrow true] \in A] + (1-p) \cdot [s[x \leftarrow false] \in A]$ and $[\![x \sim \textsc{Bernoulli}(p)]\!]^{\mathscr{B}} = \lambda(s,t).p \cdot [s[x \leftarrow true] = t] + (1-p) \cdot [s[x \leftarrow false] = t])$.

I define the concretization mapping $\gamma_{\mathcal{B}} : M_{\mathcal{B}} \to \mathbb{P}\Omega$ as $\gamma_{\mathcal{B}}(a) = \langle\!\langle \{\kappa \mid \forall s, s'.\kappa(s, \{s'\}) \geq a(s, s')\} \rangle\!\rangle$ where $\langle\!\langle C \rangle\!\rangle$ denotes the smallest element in $\mathbb{P}\Omega$ such that contains C.

Theorem 4.8. *$\gamma_{\mathcal{B}}$ is a prob. under-abstraction from C to $\mathcal{B}$.*

Proof. By the properties of the concrete semantics, we have $\gamma_{\mathcal{B}}(a) = \uparrow (\lambda(s, S').\sum_{s' \in S'} a(s, s'))$, i.e., the upper closure of $\{a\}$ in $\mathbb{P}\Omega$.

- We want to show $\gamma_{\mathcal{B}}(\underline{1}_{\mathcal{B}}) \sqsubseteq_C \underline{1}_C$. Appeal to the fact that $\uparrow (\lambda(s,S').[s \in S']) = \{\underline{1}_{\mathbb{K}}\} = \underline{1}_{\mathbb{P}}$.
- We want to show for all $Q_1, Q_2 \in \mathcal{B}$, $\gamma_{\mathcal{B}}(Q_1 \otimes_{\mathcal{B}} Q_2) \sqsubseteq_C \gamma_{\mathcal{B}}(Q_1) \otimes_C \gamma_{\mathcal{B}}(Q_2)$. It is sufficient to show $\gamma_{\mathcal{B}}(Q_1 \times Q_2) \supseteq\uparrow conv(\gamma_{\mathcal{B}}(Q_1) \otimes \gamma_{\mathcal{B}}(Q_2))$. Observe that $\gamma_{\mathcal{B}}(Q_1 \times Q_2)$ is saturated and convex, it is sufficient to show $\gamma_{\mathcal{B}}(Q_1) \otimes \gamma_{\mathcal{B}}(Q_2) \subseteq \gamma_{\mathcal{B}}(Q_1 \times Q_2)$. Suppose $\kappa_1 \in \gamma_{\mathcal{B}}(Q_1)$ and $\kappa_2 \in \gamma_{\mathcal{B}}(Q_2)$. Let $q_1 = \lambda(s,S').\sum_{s' \in S'} Q_1(s,s')$ and $q_2 = \lambda(s,S').\sum_{s' \in S'} Q_2(s,s')$. Then $\kappa_1 \sqsupseteq q_1$ and $\kappa_2 \sqsupseteq q_2$. Because $\otimes$ is monotone, we know that $\kappa_1 \otimes \kappa_2 \sqsupseteq q_1 \otimes q_2$. Observe that $q_1 \otimes q_2 = \lambda(s,S').\int q_1(s,dy) q_2(y,S') = \lambda(s,S').\sum_{s' \in S'} (Q_1 \times Q_2)(s,s')$, thus $q_1 \otimes q_2 \in \gamma_{\mathcal{B}}(Q_1 \times Q_2)$. Hence $\kappa_1 \otimes \kappa_2 \in \gamma_{\mathcal{B}}(Q_1 \times Q_2)$.
- We want to show for all $Q_1, Q_2 \in \mathcal{B}$ and $\varphi \in \mathcal{L}$, $\gamma_{\mathcal{B}}(Q_1 \ {}_{\varphi}\Diamond_{\mathcal{B}} \ Q_2) \sqsubseteq_C \gamma_{\mathcal{B}}(Q_1) \ {}_{\varphi}\Diamond_C \ \gamma_{\mathcal{B}}(Q_2)$. It is sufficient to show $\gamma_{\mathcal{B}}(\lambda(s,s').\textbf{if } \widehat{\varphi}(s) \textbf{ then } Q_1(s,s') \textbf{ else } Q_2(s,s')) \supseteq\uparrow \gamma_{\mathcal{B}}(Q_1) \ {}_{\varphi}\Diamond \ \gamma_{\mathcal{B}}(Q_2)$. Observe that the left-hand-side is saturated, it is sufficient to show $\gamma_{\mathcal{B}}(Q_1) \ {}_{\varphi}\Diamond \ \gamma_{\mathcal{B}}(Q_2) \subseteq \gamma_{\mathcal{B}}(\lambda(s,s').\textbf{if } \widehat{\varphi}(s) \textbf{ then } Q_1(s,s') \textbf{ else } Q_2(s,s'))$. Suppose $\kappa_1 \in \gamma_{\mathcal{B}}(Q_1)$ and $\kappa_2 \in \gamma_{\mathcal{B}}(Q_2)$. Let $q_1 = \lambda(s,S').\sum_{s' \in S'} Q_1(s,s')$ and $q_2 = \lambda(s,S').\sum_{s' \in S'} Q_2(s,s')$. Then $\kappa_1 \sqsupseteq q_1$ and $\kappa_2 \sqsupseteq q_2$. Because ${}_{\varphi}\Diamond$ is monotone, we know that $\kappa_1 \ {}_{\varphi}\Diamond \ \kappa_2 \sqsupseteq q_1 \ {}_{\varphi}\Diamond \ q_2$. Observe that $q_1 \ {}_{\varphi}\Diamond \ q_2 = \lambda(s,S').\textbf{if } \widehat{\varphi}(s) \textbf{ then } q_1(s,S') \textbf{ else } q_2(s,S') = \lambda(s,S').\sum_{s' \in S'} \textbf{if } \widehat{\varphi}(s) \textbf{ then } Q_1(s,s') \textbf{ else } Q_2(s,s')$, thus $q_1 \ {}_{\varphi}\Diamond \ q_2 \in \gamma_{\mathcal{B}}(\lambda(s,s').\textbf{if } \widehat{\varphi}(s) \textbf{ then } Q_1(s,s') \textbf{ else } Q_2(s,s'))$. Hence $\kappa_1 \ {}_{\varphi}\Diamond \ \kappa_2 \in \gamma_{\mathcal{B}}(\lambda(s,s').\textbf{if } \widehat{\varphi}(s) \textbf{ then } Q_1(s,s') \textbf{ else } Q_2(s,s'))$.
- We want to show for all $Q_1, Q_2 \in \mathcal{B}$ and $p \in [0,1]$, $\gamma_{\mathcal{B}}(Q_1 \ {}_p\oplus_{\mathcal{B}} \ Q_2) \sqsubseteq_C \gamma_{\mathcal{B}}(Q_1) \ {}_p\oplus_C \gamma_{\mathcal{B}}(Q_2)$. It is sufficient to show $\gamma_{\mathcal{B}}(pQ_1 + (1-p)Q_2) \supseteq\uparrow \gamma_{\mathcal{B}}(Q_1) \ {}_p\oplus \gamma_{\mathcal{B}}(Q_2)$. Observe that the left-hand-side is saturated, it is sufficient to show $\gamma_{\mathcal{B}}(Q_1) \ {}_p\oplus \gamma_{\mathcal{B}}(Q_2) \subseteq \gamma_{\mathcal{B}}(pQ_1 + (1-p)Q_2)$. Suppose $\kappa_1 \in \gamma_{\mathcal{B}}(Q_1)$ and $\kappa_2 \in \gamma_{\mathcal{B}}(Q_2)$. Let $q_1 = \lambda(s,S').\sum_{s' \in S'} Q_1(s,s')$ and $q_2 = \lambda(s,S').\sum_{s' \in S'} Q_2(s,s')$. Then $\kappa_1 \sqsupseteq q_1$ and $\kappa_2 \sqsupseteq q_2$. Because ${}_p\oplus$ is monotone, we know that $\kappa_1 \ {}_p\oplus \ \kappa_2 \sqsupseteq q_1 \ {}_p\oplus \ q_2$. Observe that $q_1 \ {}_p\oplus \ q_2 = \lambda(s,S').p \cdot q_1(s,S') + (1-p) \cdot q_2(s,S') = \lambda(s,S').\sum_{s' \in S'} p \cdot Q_1(s,s') + (1-p) \cdot Q_2(s,s')$, thus $q_1 \ {}_p\oplus \ q_2 \in \gamma_{\mathcal{B}}(pQ_1 + (1-p)Q_2)$. Hence $\kappa_1 \ {}_p\oplus \ \kappa_2 \in \gamma_{\mathcal{B}}(pQ_1 + (1-p)Q_2)$.
- We want to show that for all $Q_1, Q_2 \in \mathcal{B}$, $\gamma_{\mathcal{B}}(Q_1 \uplus_{\mathcal{B}} Q_2) \sqsubseteq_C \gamma_{\mathcal{B}}(Q_1) \uplus_C \gamma_{\mathcal{B}}(Q_2)$. It is sufficient to show $\gamma_{\mathcal{B}}(\dot{\min}(Q_1, Q_2)) \supseteq\uparrow conv(\gamma_{\mathcal{B}}(Q_1) \cup \gamma_{\mathcal{B}}(Q_2))$. Observe that the left-hand-side is saturated and convex, it is sufficient to show $\gamma_{\mathcal{B}}(Q_1) \cup \gamma_{\mathcal{B}}(Q_2) \subseteq \gamma_{\mathcal{B}}(\dot{\min}(Q_1, Q_2))$. It follows directly from the fact that $\dot{\min}(Q_1, Q_2) \dot{\leq} Q_1$ and $\dot{\min}(Q_1, Q_2) \dot{\leq} Q_2$, as well as the definition of $\gamma_{\mathcal{B}}$.

□

I do not define widening operators for BI, because $\gamma_{\mathcal{B}}$ is an under-abstraction and my algorithm starts from the bottom element in the abstract domain, the intermediate result at any iteration is a sound answer.

4.3.2 Markov Decision Process with Rewards

Analyses of finite-state Markov decision processes were originally developed in the fields of operational research and finance mathematics [116]. Originally, Markov decision processes were defined as finite-state machines with actions that exhibit probabilistic transitions. In this section, I use a slightly different formalization, using hyper-graphs.

Definition 4.9 (Markov decision processes). A *Markov decision process* (MDP) is a hyper-graph $H = \langle V, E, v^{\mathsf{entry}}, v^{\mathsf{exit}} \rangle$, where every node except v^{exit} has exactly one outgoing hyper-edge; each hyper-edge with just a single destination has an associated *reward*, $seq[\mathbf{reward}(r)]$, where r is a positive real number; and each hyper-edge with two destinations has either $prob[p]$, where $0 \leq p \leq 1$, or $ndet$. Note that MDPs are a specialization of single-procedure probabilistic programs without conditional-choice.

We can also treat the hyper-graph as a graph: each hyper-edge contributes one or two graph edges. A path through the graph has a *reward*, which is the sum of the rewards that label the edges of the path. (Edges from hyper-edges with the actions $prob[p]$ or $ndet$ are considered to have reward 0.) The analysis problem that I wish to solve is to determine, for each node v, the greatest expected reward that one can gain by executing the program from v.

It is natural to extend MDPs with procedure calls and multiple procedures, to obtain *recursive Markov decision processes*. The set of program states is defined to be the set of nonnegative real numbers: $\Omega = [0, \infty]$. To address the maximum-expected-reward problem for a recursive Markov decision process, I define a PMA $\mathcal{R} = \langle M_{\mathcal{R}}, \sqsubseteq_{\mathcal{R}}, \otimes_{\mathcal{R}}, {}_{\varphi}\Diamond_{\mathcal{R}}, {}_{p}\oplus_{\mathcal{R}}, \uplus_{\mathcal{R}}, \underline{\bot}_{\mathcal{R}}, \underline{1}_{\mathcal{R}} \rangle$ as follows:

$M_{\mathcal{R}} \stackrel{\text{def}}{=} [0, \infty]$	${}_{\varphi}\Diamond_{\mathcal{R}} \stackrel{\text{def}}{=} \max$	$\underline{\bot}_{\mathcal{R}} \stackrel{\text{def}}{=} 0$
$\sqsubseteq_{\mathcal{R}} \stackrel{\text{def}}{=} \leq$	$a \, {}_{p}\oplus_{\mathcal{R}} \, b \stackrel{\text{def}}{=} p \cdot a + (1-p) \cdot b$	$\underline{1}_{\mathcal{R}} \stackrel{\text{def}}{=} 0$
$\otimes_{\mathcal{R}} \stackrel{\text{def}}{=} +$	$\uplus_{\mathcal{R}} \stackrel{\text{def}}{=} \max$	

Let $\mathscr{R} = \langle \mathcal{R}, [\![\cdot]\!]^{\mathscr{R}} \rangle$ be the interpretation for a Markov decision process with rewards. I define the semantic function as $[\![\mathbf{reward}(r)]\!]^{\mathscr{C}} = \lambda(s, A).[s + r \in A]$ and $[\![\mathbf{reward}(r)]\!]^{\mathscr{R}} = r$.

I define the concretization mapping $\gamma_{\mathcal{R}} : M_{\mathcal{R}} \to \mathbb{P}[0, \infty]$ as follows: $\gamma_{\mathcal{R}}(a) = \langle\!\langle \{\kappa \mid \forall s. \int y \cdot \kappa(s, dy) \leq s + a\} \rangle\!\rangle$.

THEOREM 4.10. *$\gamma_{\mathcal{R}}$ is a prob. over-abstraction from C to $\mathcal{R}$.*

PROOF. By the properties of the concrete semantics, we have $\gamma_{\mathcal{R}}(a) = \{\kappa \mid \forall s. \int y \cdot \kappa(s, dy) \leq s + a\}$, where $\overline{C}$ is the Scott closure of C in $\mathbb{P}\Omega$.

- We want to show $\underline{1}_C \sqsubseteq_C \gamma_{\mathcal{R}}(\underline{1}_{\mathcal{R}})$. Appeal to the fact that $\underline{1}_{\mathbb{K}} \in \gamma_{\mathcal{R}}(0)$.
- We want to show for all $Q_1, Q_2 \in \mathcal{R}$, $\gamma_{\mathcal{R}}(Q_1) \otimes_C \gamma_{\mathcal{R}}(Q_2) \sqsubseteq_C \gamma_{\mathcal{R}}(Q_1 \otimes_{\mathcal{R}} Q_2)$. It is sufficient to show that $\overline{conv(\gamma_{\mathcal{R}}(Q_1) \otimes \gamma_{\mathcal{R}}(Q_2))} \subseteq \gamma_{\mathcal{R}}(Q_1 + Q_2)$. Observe that the right-hand-side is Scott-closed and convex, it is sufficient to show that $\gamma_{\mathcal{R}}(Q_1) \otimes \gamma_{\mathcal{R}}(Q_2) \subseteq \gamma_{\mathcal{R}}(Q_1 + Q_2)$. Suppose $\kappa_1 \in \gamma_{\mathcal{R}}(Q_1)$ and $\kappa_2 \in \gamma_{\mathcal{R}}(Q_2)$. Observe that $\int y \cdot \kappa_1(s, dy) \leq s + Q_1$ and

$\int y \cdot \kappa_2(s, dy) \leq s + Q_2$. Then $\int y \cdot (\kappa_1 \otimes \kappa_2)(s, dy) = \int_y y(\int_z \kappa_1(s, dz) \cdot \kappa_2(z, dy)) = \int_z (\int_y y \cdot \kappa_2(z, dy)) \cdot \kappa_1(s, dz) \leq \int (z + Q_2) \cdot \kappa_1(s, dz) = \int z \cdot \kappa_1(s, dz) + \int Q_2 \cdot \kappa_1(s, dz) \leq s + Q_1 + Q_2$. Hence $\kappa_1 \otimes \kappa_2 \in \gamma_{\mathcal{R}}(Q_1 + Q_2)$.

- We want to show for all $Q_1, Q_2 \in \mathcal{R}$ and $p \in [0, 1]$, $\gamma_{\mathcal{R}}(Q_1) \,{}_p\!\oplus_C \gamma_{\mathcal{R}}(Q_2) \sqsubseteq_C \gamma_{\mathcal{R}}(Q_1 \,{}_p\!\oplus_{\mathcal{R}} Q_2)$. It is sufficient to show that $\overline{\gamma_{\mathcal{R}}(Q_1) \,{}_p\!\oplus \gamma_{\mathcal{R}}(Q_2)} \subseteq \gamma_{\mathcal{R}}(pQ_1 + (1-p)Q_2)$. Observe that the right-hand-side is Scott-closed, it is sufficient to show that $\gamma_{\mathcal{R}}(Q_1) \,{}_p\!\oplus \gamma_{\mathcal{R}}(Q_2) \subseteq \gamma_{\mathcal{R}}(pQ_1 + (1-p)Q_2)$. Suppose $\kappa_1 \in \gamma_{\mathcal{R}}(Q_1)$ and $\kappa_2 \in \gamma_{\mathcal{R}}(Q_2)$. Observe that $\int y \cdot \kappa_1(s, dy) \leq s + Q_1$ and $\int y \cdot \kappa_2(s, dy) \leq s + Q_2$. Then $\int y \cdot (\kappa_1 \,{}_p\!\oplus \kappa_2)(s, dy) = \int y \cdot (p\kappa_1 + (1-p)\kappa_2)(s, dy) = \int y \cdot p \cdot \kappa_1(s, dy) + \int y \cdot (1-p) \cdot \kappa_2(s, dy) \leq pQ_1 + (1-p)Q_2$. Hence $\kappa_1 \,{}_p\!\oplus \kappa_2 \in \gamma_{\mathcal{R}}(pQ_1 + (1-p)Q_2)$.
- We want to show for all $Q_1, Q_2 \in \mathcal{R}$, $\gamma_{\mathcal{R}}(Q_1) \uplus_C \gamma_{\mathcal{R}}(Q_2) \sqsubseteq_C \gamma_{\mathcal{R}}(Q_1 \uplus_{\mathcal{R}} Q_2)$. It is sufficient to show that $\overline{conv(\gamma_{\mathcal{R}}(Q_1) \cup \gamma_{\mathcal{R}}(Q_2))} \subseteq \gamma_{\mathcal{R}}(\max(Q_1, Q_2))$. Observe that the right-hand-side is Scott-closed and convex, it is sufficient to show that $\gamma_{\mathcal{R}}(Q_1) \cup \gamma_{\mathcal{R}}(Q_2) \subseteq \gamma_{\mathcal{R}}(\max(Q_1, Q_2))$. It follows directly from the fact that $Q_1 \leq \max(Q_1, Q_2)$ and $Q_2 \leq \max(Q_1, Q_2)$, as well as the definition of $\gamma_{\mathcal{R}}$.

□

I then use a trivial widening in this analysis: if after some fixed number of iterations the analysis does not converge, it returns ∞ as the result.

4.3.3 Linear Expectation-Invariant Analysis

Several examples of expectation invariants obtained via linear expectation-invariant analysis (LEIA) were given in §4.1.2. This section gives details of the abstract domain for LEIA.

I make use of an existing abstract domain, namely, the domain of *convex polyhedra* [38]. Elements of the polyhedral domain are defined by linear-inequality and linear-equality constraints among program variables. For LEIA, I use two-vocabulary polyhedra over nonnegative program variables. Let $x = (x_1, \cdots, x_n)^T$ be a column vector of nonnegative program variables and $x' = (x'_1, \cdots, x'_n)^T$ be a column vector of the "primed" versions of corresponding program variables. A polyhedron $P \subseteq \mathbb{R}^{2n}_{\geq 0}$ captures linear-inequality constraints among x and x', which can be interpreted as a relation between pre-state and post-state variable valuations.

A polyhedron $P = \{(x'^T x^T)^T \in \mathbb{R}^{2n}_{\geq 0} \mid A'x' + Ax \leq b \wedge D'x' + Dx = e\}$, can be encoded as the intersection of a finite number of closed half spaces and a finite number of subspaces, where A', A, D', D are matrices and b, e are vectors. The associated *constraint set* is defined as $C_P = \{A'x' + Ax \leq b, D'x' + Dx = e\}$. Let $\mathcal{P}$ be the set of polyhedra; $\mathcal{P}$ is equipped with meet, join, renaming, forgetting, and comparison operations.

LEIA uses *expectation polyhedra*. They are actually the same as polyhedra, except that the two vocabularies are $x = (x_1, \cdots, x_n)^T$ and $\mathbb{E}[x'] = (\mathbb{E}[x'_1], \cdots, \mathbb{E}[x'_n])^T$. An expectation

polyhedron represents a constraint set of the form

$$\{A'\mathbb{E}[x'] + Ax \leq b, D'\mathbb{E}[x'] + Dx = e\}. \tag{4.2}$$

Because of the linearity of the expectation operator $\mathbb{E}$, an equivalent way to express (4.2) is as follows:

$$\{\mathbb{E}[A'x'] + Ax \leq b, \mathbb{E}[D'x'] + Dx = e\}.$$

Let $\mathcal{EP}$ be the set of expectation polyhedra. $\mathcal{EP}$ is equipped with the same set of operations as $\mathcal{P}$.

I define the state space to be $\Omega = \mathbb{R}^n_{\geq 0}$. I then define a PMA $\mathcal{I}$ with a universe $M_{\mathcal{I}} \stackrel{\text{def}}{=} \mathcal{P} \times \mathcal{EP}$. An element $(P, EP) \in \mathcal{I}$ consists of

(i) a set of standard constraints $P \in \mathcal{P}$, and

(ii) a set of expectation constraints $EP \in \mathcal{EP}$, such that $\mathbf{0} \sqcup P[\mathbb{E}[x']/x'] \sqsupseteq EP$ holds, where $\mathbf{0} \stackrel{\text{def}}{=} \wedge_{i=1}^{n}(\mathbb{E}[x'_i] = 0)$.

The latter property means that, if necessary, one can always "rebuild" a pessimistic $\mathcal{EP}$ component from the $\mathcal{P}$ component as $\mathbf{0} \sqcup P[\mathbb{E}[x']/x']$.[2]

I define the concretization mapping $\gamma_{\mathcal{I}}$ as follows:

$$\gamma_{\mathcal{I}}(P, EP) = \langle\!\langle \left\{ \kappa \mid \forall s.\kappa \left(s, \left\{ s' \mid \begin{bmatrix} s' \\ s \end{bmatrix} \models \neg P \right\} \right) = 0 \ \wedge \begin{bmatrix} \int s'\kappa(s, ds') \\ s \end{bmatrix} \models EP \right\} \rangle\!\rangle.$$

Comparison The comparison operation on ordinary polyhedra can be defined as standard set inclusion. For expectation polyhedra, taking into account sub-probability distributions, I define $EP_1 \sqsubseteq EP_2$ to be $\mathbf{0} \sqcup EP_1 \subseteq \mathbf{0} \sqcup EP_2$, so that any element inside or below EP_1 should also be inside or below EP_2. Consequently, I define $(P_1, EP_1) \sqsubseteq_{\mathcal{I}} (P_2, EP_2) \stackrel{\text{def}}{=} P_1 \subseteq P_2 \wedge \mathbf{0} \sqcup EP_1 \subseteq \mathbf{0} \sqcup EP_2$.

Composition For ordinary polyhedra, the composition of P_1 and P_2 can be defined as

$$(\exists x''.C_{P_1}[x''/x'] \wedge C_{P_2}[x''/x]) \implies C_{P_1 \otimes P_2},$$

where I introduce an intermediate vocabulary $x'' = (x''_1, \cdots, x''_n)^T$, and use it to connect P_1 and P_2. Consequently, I define $P_1 \otimes P_2$ to be $\exists x''.C_{P_1}[x''/x'] \wedge C_{P_2}[x''/x]$. Operationally, composition involves first introducing a new vocabulary; renaming the variables properly; performing a meet, and finally forgetting the intermediate vocabulary.

Somewhat surprisingly, because of the *tower property* in probability theory, exactly the same steps can be used to compose expectation polyhedra. Informally, the tower property

[2]The intuition is that P represents a convex over-approximation to some desired set of points; the expected value has to lie somewhere inside $\vec{0} \sqcup P$, where "$\vec{0}\sqcup\ldots$" is needed to account for sub-probability distributions. For instance, for a nonnegative interval $[lo, hi]$, it must hold that $expected \in ([0,0] \sqcup [lo, hi])$; i.e., $0 \leq expected \leq hi$.

means that $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]]$, where X and Y are two random variables, and $\mathbb{E}[X \mid Y]$ is a conditional expectation. For instance, suppose that EP_1 and EP_2 are defined by the constraint sets $\{\mathbb{E}(x') = x + 2\}$ and $\{\mathbb{E}(x') = 7x\}$, respectively. Following the renaming recipe above, we have $\mathbb{E}(x'') = x + 2$ and $\mathbb{E}(x' \mid x'') = 7x''$. By the tower property, we have $\mathbb{E}(x') = \mathbb{E}(\mathbb{E}(x' \mid x'')) = \mathbb{E}(7x'') = 7\mathbb{E}(x'') = 7x + 14$. Operationally, the tower property allows me to compose linear expectation invariants, and eliminate the intermediate vocabulary x''. Consequently, I define

$$(P_1, EP_1) \otimes_{\mathcal{I}} (P_2, EP_2) \stackrel{\text{def}}{=} (P_1 \otimes P_2, EP_1 \otimes EP_2).$$

Conditional-choice For the ordinary-polyhedron component, a conditional-choice ${}_{\varphi}\diamondsuit$ is performed by first meeting each operand with the logical constraint φ, and then joining the results. However, for the expectation-polyhedron component, conditioning can split the probability space in almost arbitrary ways. Consequently, the constraints on post-state expectations as a function of pre-state valuations are not necessarily true after conditioning. Thus, I define

$$\begin{aligned}(P_1, EP_1)\ {}_{\varphi}\diamondsuit_{\mathcal{I}}\ (P_2, EP_2) \stackrel{\text{def}}{=}\ & \textbf{let } P = (\{\varphi\} \sqcap P_1) \sqcup (\{\neg\varphi\} \sqcap P_2) \\ & \textbf{in } (P, (EP_1 \sqcup EP_2) \sqcap (\mathbf{0} \sqcup P[\mathbb{E}[x']/x'])).\end{aligned}$$

The $\sqcap$ in the second component is performed to maintain the invariant that $\mathbf{0} \sqcup P[\mathbb{E}[x']/x'] \sqsupseteq$ the second component.

Probabilistic-choice For the ordinary-polyhedron component, I merely join the components of the two operands. For the expectation-polyhedron component, I introduce two more vocabularies and have

$$(\exists x'', x'''.C_{EP_1}[x''/\mathbb{E}[x']] \wedge C_{EP_2}[x'''/\mathbb{E}[x']] \wedge \bigwedge_{i=1}^{n} \mathbb{E}[x_i'] = p \cdot x_i'' + (1-p) \cdot x_i''') \implies C_{EP_1\ {}_p\oplus EP_2}.$$

Consequently, I define $EP_1\ {}_p\oplus\ EP_2$ to be

$$\exists x'', x'''. \left(C_{EP_1}[x''/\mathbb{E}[x']] \wedge C_{EP_2}[x'''/\mathbb{E}[x']] \wedge \bigwedge_{i=1}^{n} \mathbb{E}[x_i'] = p \cdot x_i'' + (1-p) \cdot x_i''' \right),$$

and $(P_1, EP_1)\ {}_p\oplus_{\mathcal{I}}\ (P_2, EP_2) \stackrel{\text{def}}{=} (P_1 \sqcup P_2, EP_1\ {}_p\oplus\ EP_2)$.

Nondeterministic-choice The nondeterministic-choice operations on both ordinary polyhedra and expectation polyhedra can be defined as join. Hence, I define $(P_1, EP_1) \uplus_{\mathcal{I}} (P_2, EP_2) \stackrel{\text{def}}{=} (P_1 \sqcup P_2, EP_1 \sqcup EP_2)$.

Bottom and Unit Element I define $\underline{\bot}_{\mathcal{I}} \stackrel{\text{def}}{=} (\mathit{false}, \mathbf{0})$, and $\underline{1}_{\mathcal{I}} \stackrel{\text{def}}{=} (\{x_i' = x_i \mid 1 \le i \le n\}, \{\mathbb{E}[x_i'] = x_i \mid 1 \le i \le n\})$.

Semantic Function Some examples of the semantic mapping $[\![\cdot]\!]^{\mathcal{I}}$ are as follows, where $\min(\mathscr{D})$ and $\max(\mathscr{D})$ represents the interval of the support of a distribution $\mathscr{D}$, while $\mathrm{mean}(\mathscr{D})$ stands for its average.

$$
\begin{aligned}
[\![x_i := \mathcal{E}]\!]^{\mathcal{I}} &\stackrel{\text{def}}{=} \Big(\{x_i' = \mathcal{E}(x)\} \cup \{x_j' = x_j \mid j \neq i\}, \{\mathbb{E}[x_i'] = \mathcal{E}(x)\} \cup \{\mathbb{E}[x_j'] = x_j \mid j \neq i\}\Big) \\
[\![x_i \sim \mathscr{D}]\!]^{\mathcal{I}} &\stackrel{\text{def}}{=} \Big(\{\min(\mathscr{D}) \le x_i' \le \max(\mathscr{D})\} \cup \{x_j' = x_j \mid j \neq i\}, \\
&\qquad\qquad \{\mathbb{E}[x_i'] = \mathrm{mean}(\mathscr{D})\} \cup \{\mathbb{E}[x_j'] = x_j \mid j \neq i\}\Big) \\
[\![\mathbf{skip}]\!]^{\mathcal{I}} &\stackrel{\text{def}}{=} \underline{1}_{\mathcal{I}}
\end{aligned}
$$

Note I assume all expressions in the program are linear. For nonlinear arithmetic programs, one can adopt some linearization techniques [48, 103].

THEOREM 4.11. *$\gamma_{\mathcal{I}}$ is a prob. over-abstraction from $\mathcal{C}$ to $\mathcal{I}$.*

PROOF. Let $\Omega \stackrel{\text{def}}{=} \mathbb{R}^n_{\ge 0}$ be the set of program states. (Note here we assume $\mathbb{R}_{\ge 0}$ admits a Borel-field generated by sets of compact intervals [101].) By the properties of the concrete semantics, we have

$$
\gamma_{\mathcal{I}}(P, EP) = \overline{\{\kappa \mid \forall s. \kappa(s)((P|_s)^c) = 0 \wedge \begin{bmatrix} \int y \cdot \kappa(s)(dy) \\ s \end{bmatrix} \in \overline{EP}\}}.
$$

- We want to show $\underline{1}_{\mathcal{C}} \sqsubseteq_{\mathcal{C}} \gamma_{\mathcal{I}}(\underline{\bot}_{\mathcal{I}})$. It is sufficient to show $\downarrow \underline{1}_{\mathbb{K}} \subseteq \gamma_{\mathcal{I}}(\{x_i' = x_i\}, \{\mathbb{E}[x_i'] = x_i\})$. Appeal to the fact that $\underline{1}_{\mathbb{K}} \in \gamma_{\mathcal{I}}(\{x_i' = x_i\}, \{\mathbb{E}[x_i'] = x_i\})$.
- We want to show for all $(P_1, EP_1), (P_2, EP_2) \in \mathcal{I}$, $\gamma_{\mathcal{I}}(P_1, EP_1) \otimes_{\mathcal{C}} \gamma_{\mathcal{I}}(P_2, EP_2) \sqsubseteq_{\mathcal{C}} \gamma_{\mathcal{I}}((P_1, EP_1) \otimes_{\mathcal{I}} (P_2, EP_2))$. It is sufficient to show that $\gamma_{\mathcal{I}}(P_1, EP_1) \otimes \gamma_{\mathcal{I}}(P_2, EP_2) \subseteq \gamma_{\mathcal{I}}(P_1 \otimes P_2, EP_1 \otimes EP_2)$. Suppose $\kappa_1 \in \gamma_{\mathcal{I}}(P_1, EP_1)$ and $\kappa_2 \in \gamma_{\mathcal{I}}(P_2, EP_2)$. Observe that $(\kappa_1 \otimes \kappa_2)(s, (P_1 \otimes P_2|_s)^c) = \int \kappa_1(s, dy) \cdot \kappa_2(y, (P_1 \otimes P_2|_s)^c)$. If $y \in (P_1|_s)^c$, then $\kappa_1(s, \{y\}) = 0$. If $y \in P_1|_s$, then by the definition of $P_1 \otimes P_2$, we know $\kappa_2(y, (P_1 \otimes P_2|_s)^c) = 0$. Hence $(\kappa_1 \otimes \kappa_2)(s, (P_1 \otimes P_2|_s)^c) = 0$. On the other hand, observe that $\int y \cdot (\kappa_1 \otimes \kappa_2)(s, dy) = \int_y y (\int_z \kappa_1(s, dz) \cdot \kappa_2(z, dy)) = \int_z (\int_y y \cdot \kappa_2(z, dy)) \cdot \kappa_1(s, dz)$, and $\begin{bmatrix} \int y \cdot \kappa_2(z, dy) \\ z \end{bmatrix} \in \overline{EP_2}$ for all z, by the fact that EP_2 is convex, we know that $\begin{bmatrix} \int_z (\int_y y \cdot \kappa_2(z, dy)) \cdot \kappa_1(s, dz) \\ \int_z z \cdot \kappa_1(s, dz) \end{bmatrix} \in \overline{EP_2}$. Because $\begin{bmatrix} \int_z z \cdot \kappa_1(s, dz) \\ s \end{bmatrix} \in \overline{EP_1}$, we know that $\begin{bmatrix} \int y \cdot (\kappa_1 \otimes \kappa_2)(s, dy) \\ y \end{bmatrix} \in \overline{EP_1} \otimes \overline{EP_2} \subseteq \overline{EP_1 \otimes EP_2}$.

- We want to show for all $(P_1, EP_1), (P_2, EP_2) \in \mathcal{I}$ and $\varphi \in \mathcal{L}$, $\gamma_{\mathcal{I}}(P_1, EP_1)\ {}_{\varphi}\Diamond_{C}\ \gamma_{\mathcal{I}}(P_2, EP_2) \sqsubseteq_C \gamma_{\mathcal{I}}((P_1, EP_1)\ {}_{\varphi}\Diamond_{\mathcal{I}}\ (P_2, EP_2))$. It is sufficient to show that $\gamma_{\mathcal{I}}(P_1, EP_1)\ {}_{\varphi}\Diamond\ \gamma_{\mathcal{I}}(P_2, EP_2) \subseteq \gamma_{\mathcal{I}}(P \stackrel{\text{def}}{=} (\{\varphi\} \sqcap P_1) \sqcup (\{\neg\varphi\} \sqcap P_2), (EP_1 \sqcup EP_2) \sqcap \overline{P[\mathbb{E}[x']/x']})$. Suppose $\kappa_1 \in \gamma_{\mathcal{I}}(P_1, EP_1)$ and $\kappa_2 \in \gamma_{\mathcal{I}}(P_2, EP_2)$. Observe that $(\kappa_1\ {}_{\varphi}\Diamond\ \kappa_2)(s, (P|_s)^c) = \textbf{if } \widehat{\varphi}(s) \textbf{ then } \kappa_1(s, (P|_s)^c) \textbf{ else } \kappa_2(s, (P|_s)^c)$. If $\widehat{\varphi}(s)$, then $P|_s = \{\varphi\} \sqcap P_1 \sqsubseteq P_1$, hence $(\kappa_1\ {}_{\varphi}\Diamond\ \kappa_2)(s, (P|_s)^c) = 0$. If $\neg\widehat{\varphi}(s)$, then $P|_s = \{\neg\varphi\} \sqcap P_2 \sqsubseteq P_2$, hence $(\kappa_1\ {}_{\varphi}\Diamond\ \kappa_2)(s, (P|_s)^c) = 0$. On the other hand, observe that $\int y \cdot (\kappa_1\ {}_{\varphi}\Diamond\ \kappa_2)(s, dy) = \textbf{if } \widehat{\varphi}(s) \textbf{ then } \int y \cdot \kappa_1(s, dy) \textbf{ else } \int y \cdot \kappa_2(s, dy)$. Hence $\begin{bmatrix} \int y \cdot (\kappa_1\ {}_{\varphi}\Diamond\ \kappa_2)(s, dy) \\ s \end{bmatrix} \in \overline{EP_1} \sqcup \overline{EP_2} \subseteq \overline{EP_1 \sqcup EP_2}$.
- We want to show for all $(P_1, EP_1), (P_2, EP_2) \in \mathcal{I}$ and $p \in [0, 1]$, $\gamma_{\mathcal{I}}(P_1, EP_1)\ {}_{p}\oplus_{C}\ \gamma_{\mathcal{I}}(P_2, EP_2) \sqsubseteq_C \gamma_{\mathcal{I}}((P_1, EP_1)\ {}_{p}\oplus_{\mathcal{I}}\ (P_2, EP_2))$. It is sufficient to show that $\gamma_{\mathcal{I}}(P_1, EP_1)\ {}_{p}\oplus\ \gamma_{\mathcal{I}}(P_2, EP_2) \subseteq \gamma_{\mathcal{I}}(P_1 \sqcup P_2, EP_1\ {}_{p}\oplus\ EP_2)$. Suppose $\kappa_1 \in \gamma_{\mathcal{I}}(P_1, EP_1)$ and $\kappa_2 \in \gamma_{\mathcal{I}}(P_2, EP_2)$. Observe that $(\kappa_1\ {}_{p}\oplus\ \kappa_2)(s, (P_1 \sqcup P_2|_s)^c) = p \cdot \kappa_1(s, (P_1 \sqcup P_2|_s)^c) + (1 - p) \cdot \kappa_2(s, (P_1 \sqcup P_2|_s)^c) = 0$. On the other hand, observe that $\int y \cdot (\kappa_1\ {}_{p}\oplus\ \kappa_2)(s, dy) = p \int y \cdot \kappa_1(s, dy) + (1 - p) \int y \cdot \kappa_2(s, dy)$. Hence $\kappa_1\ {}_{p}\oplus\ \kappa_2 \in \overline{EP_1\ {}_{p}\oplus\ EP_2}$.
- We want to show that for all $(P_1, EP_1), (P_2, EP_2) \in \mathcal{I}$, $\gamma_{\mathcal{I}}(P_1, EP_1) \uplus_C \gamma_{\mathcal{I}}(P_2, EP_2) \sqsubseteq_C \gamma_{\mathcal{I}}((P_1, EP_1) \uplus_{\mathcal{I}} (P_2, EP_2))$. Appeal to the fact that $\gamma_{\mathcal{I}}(P_1, EP_1) \cup \gamma_{\mathcal{I}}(P_2, EP_2) \subseteq \gamma_{\mathcal{I}}(P_1 \sqcup P_2, EP_1 \sqcup EP_2)$.

□

Widening Let $\triangledown$ be the standard widening operator on ordinary polyhedra [61]. Recall from Obs. 4.7 that whenever a widening operation $a \triangledown b$ is performed, the property $a \sqsubseteq_{\mathcal{A}} b$ holds. There is a subtle issue with expectation invariants when dealing with conditional or nondeterministic loops.

Observation 4.12. In a conventional program, if we have a loop "while B do S od," and I is a loop-invariant, then $I \wedge \neg B$ (which implies I) holds on exiting the loop. In contrast, for a conditional or nondeterministic loop in a probabilistic program, a loop-invariant that holds at the beginning and end of the loop body does not necessarily hold on exiting the loop.

Example 4.13. Consider the following program:

while $\neg(x = y)$ **do**
  if prob$(\frac{1}{2})$ **then** $x := x + 1$ **else** $y := y + 1$ **fi**
od

For the loop body, we can derive an expectation invariant $\mathbb{E}[x' - y'] = x - y$; however, for the entire loop this property does not hold: at the end of the loop $x = y$ must hold, and hence $\mathbb{E}[x' - y']$ should be equal to 0.

Because of this issue, I use a pessimistic widening operator for conditional-choice and nondeterministic-choice: the widening operator forgets the expectation invariants and rebuilds them from standard invariants.

$$(P_1, EP_1) \triangledown_c (P_2, EP_2) \stackrel{\text{def}}{=} (P_1 \triangledown P_2, \mathbf{0} \sqcup P_2[\mathbb{E}[x']/x'])$$
$$(P_1, EP_1) \triangledown_n (P_2, EP_2) \stackrel{\text{def}}{=} (P_1 \triangledown P_2, \mathbf{0} \sqcup P_2[\mathbb{E}[x']/x'])$$

I do not have a good method for $(P_1, EP_1) \triangledown_p (P_2, EP_2)$. I found that the following approach loses precision:

$$\textbf{let } P = (P_1 \triangledown P_2) \textbf{ in } (P, (EP_1 \triangledown EP_2) \sqcap (\mathbf{0} \sqcup P[\mathbb{E}[x']/x']))$$

In my experiments, I use $(P_1, EP_1) \triangledown_p (P_2, EP_2) \stackrel{\text{def}}{=} (P_1 \triangledown P_2, EP_2)$, which does no extrapolation in the $\mathcal{EP}$ component.

4.4 Evaluation

In this section, I first describe the implementation of PMAF, and the three instantiations introduced in §4.3. Then, I evaluate the effectiveness and performance of the three analyses.

4.4.1 Implementation

PMAF is implemented in OCaml; the core framework consists of about 400 lines of code. The framework is implemented as a functor parametrized by a module representing a PMA, with some extra functions, such as widening and printing. This organization allows any analysis that can be formulated in PMAF to be implemented as a plugin. Also, the core framework relies on control-flow hyper-graphs, and provides users the flexibility to employ it with any front end. I use OCamlGraph [32] as the implementation of fixed-point computation and Bourdoncle's algorithm.

The plugin for Bayesian inference is about 400 lines of code, including a lexer and a parser for the imperative language that I use in the examples of this paper. I use Lacaml [110] to manipulate matrices. The plugins for the Markov decision problem with rewards and linear expectation-invariant analysis are about 200 lines and 500 lines, respectively. I use APRON [75] for polyhedron operations. Most of the code in the plugins is to implement the PMA structure of the analysis domain.

Because of the numerical reasoning required when analyzing probabilistic programs, I need to be concerned about finite numerical precision in my implementations of the instantiations (although they are sound on a theoretical machine operating on reals). In my implementation, I use the fact that ascending chains of floating numbers always converge in a finite number of steps. The user could use the technique proposed by Darulova and Kuncak [40] to obtain a sound guarantee on numerical precision.

4.4.2 Experiments

Evaluation Platform My experiments were performed on a machine with an Intel Core i5 2.4 GHz processor and 8GB of RAM under Mac OS X 10.13.4.

Bayesian Inference and Markov Decision Problem with Rewards I tested my framework on Bayesian inference and Markov decision problem with rewards on hand-crafted examples. The results of the evaluation of the two analyses are described in Tab. 4.1. The tables contains the number of lines; whether the program is non-recursive, tail-recursive, or recursive; the number of procedure calls; and the time taken by the implementation (measured by running each program 5 times and computing the 20% trimmed mean).

Table 4.1: Top: Bayesian inference. Bottom: Markov decision problem with rewards. (Time is in seconds.)

Program	#loc	rec?	#call	time
compare	17	n	0	2.22
dice	12	n	0	0.02
eg1	10	n	0	0.02
eg1-tail	16	t	2	0.02
eg2	10	n	0	0.02
eg2-tail	16	t	2	0.01
recursive	14	r	1	0.01
binary10	184	n	90	0.03
loop	10	n	0	0.03
quicksort7	109	n	42	0.03
recursive	13	t	1	0.03
student	43	t	8	0.03

My framework computed the same answer (modulo floating-point round-off errors) as PReMo [141], a tool for probabilistic recursive models. I did not compare with probabilistic abstract interpretation [39] because its semantic foundation is substantially different from that of my framework—as I mentioned in the beginning of this chapter, the order for resolving probabilistic behavior and nondeterministic behavior is different.

The analysis time of Bayesian inference grows exponentially with respect to the number of program variables.[3] The time cost comes from the explicit matrix representation of domain elements. One could use Algebraic Decision Diagrams [4] as a compact representation to improve the efficiency.

The analyzer for the Markov decision problem with rewards works quickly and obtains some interesting results. *quicksort7* is a model of a randomized quicksort algorithm on an array of size 7 (obtained from [141]), and my analysis results are consistent with the worst-case expected number of comparisons being $\Theta(n \log n)$.[4] *binary10* is a model of randomized binary search algorithm on an array of size 10, and my analysis results are consistent with the worst-case expected number of comparisons being $\Theta(\log n)$.

[3] One should not assume that exponential growth makes the analysis useless; after all, predicate-abstraction domains [60] also grow exponentially: the universe of assignments to a set of Boolean variables grows exponentially in the number of variables. Finding useful coarser abstractions for Bayesian inference—by analogy with the techniques of Ball et al. [5] for predicate abstraction—might be an interesting direction for future work.

[4] The analysis computes *worst-case expected number* because the underlying semantics resolves nondeterminism first and probabilistic-choice second, and thus the analysis computes $\max_{\text{nondet. resolution}} \mathbb{E}[\#\text{comparisons under resolution}]$.

Table 4.2: Linear expectation-invariant analysis.

Program	Expectation invariants	#loc	rec?	#call	time
2d-walk	$\mathbb{E}[x'] = x$, $\mathbb{E}[y'] = y$, $\mathbb{E}[dist'] = dist$, $\mathbb{E}[count'] \leq count + 1$, $\mathbb{E}[count'] \geq count$	47	n	0	0.24
aggregate-rv	$\mathbb{E}[2x' - i'] = 2x - i$, $\mathbb{E}[x'] \leq x + \frac{1}{2}$, $\mathbb{E}[x'] \geq x$	11	n	0	0.06
biased-coin	$\mathbb{E}[x'] \leq x + \frac{1}{2}$, $\mathbb{E}[x'] \geq x - \frac{1}{2}$	25	n	0	0.06
binom-update (p=$\frac{1}{4}$)	$\mathbb{E}[4x' - n'] = 4x - n$, $\mathbb{E}[x'] \leq x + \frac{1}{4}$, $\mathbb{E}[x'] \geq x$	14	n	0	0.06
coupon5	$\mathbb{E}[count' - i'] = count - i$ (1st), $\mathbb{E}[4count' - 5i'] = 4count - 5i$ (2nd), $\mathbb{E}[3count' - 5i'] = 3count - 5i$ (3rd), $\mathbb{E}[2count' - 5i'] = 2count - 5i$ (4th), $\mathbb{E}[count' - 5i'] = count - 5i$ (5th)	58	n	0	0.07
dist	$\mathbb{E}[x'] = x$, $\mathbb{E}[y'] = y$, $\mathbb{E}[z'] = \frac{1}{2}x + \frac{1}{2}y$	5	n	0	0.05
eg	$\mathbb{E}[x' + y'] = x + y + 3$, $\mathbb{E}[z'] = \frac{1}{4}z + \frac{3}{4}$, $\mathbb{E}[x'] \leq x + 3$, $\mathbb{E}[x'] \geq x$	8	n	0	0.89
eg-tail	$\mathbb{E}[z'] \geq \frac{1}{4}z$, $\mathbb{E}[x'] \geq x$, $\mathbb{E}[y'] \geq y$, $\mathbb{E}[x' + y'] \geq x + y + \frac{3}{4}$	11	t	1	0.13
hare-turtle	$\mathbb{E}[2h' - 5t'] = 2h - 5t$, $\mathbb{E}[h'] \leq h + \frac{5}{2}$, $\mathbb{E}[h'] \geq h$	15	n	0	0.06
hawk-dove	$\mathbb{E}[p1b' - count'] = p1b - count$, $\mathbb{E}[p2b' - count'] = p2b - count$, $\mathbb{E}[p1b'] \leq p1b + 1$, $\mathbb{E}[p1b'] \geq p1b$	29	n	0	0.08
mot-ex	$\mathbb{E}[2x' - y'] = 2x - y$, $\mathbb{E}[4x' - 3count'] = 4x - 3count$, $\mathbb{E}[x'] \leq x + \frac{3}{4}$, $\mathbb{E}[x'] \geq x$	16	n	0	0.06
recursive	$\mathbb{E}[x'] = x + 9$	13	r	2	0.37
uniform-dist	$\mathbb{E}[n'] \leq 2n$, $\mathbb{E}[n'] \geq n$, $\mathbb{E}[g'] \leq 2g + \frac{1}{2}$, $\mathbb{E}[g'] \geq g$	14	n	0	0.06

Linear Expectation-Invariant Analysis I performed a more thorough evaluation of linear expectation-invariant analysis. I collected several examples from the literature on probabilistic invariant generation [26, 81], and handcrafted some new examples to demonstrate particular capabilities of my domain, e.g., analysis of recursive programs. For the examples obtained from the loop-invariant-generation benchmark, I extracted the loop body as my test programs. Also, I performed a positive-negative decomposition to make sure all program variables are nonnegative. That is, I represented each variable x as $x^+ - x^-$ where $x^+, x^- \geq 0$, and replaced every operation on variables with appropriate operations on the decomposed variables.

The results of the evaluation are shown in Tab. 4.2, which lists the expectation invariants obtained, and the time taken by the implementation. In general, the analysis runs quickly—all the examples are processed in less than one second. The analysis time mainly depends on the number of program variables and the size of the control-flow hyper-graph.

As shown in Tab. 4.2, my analysis can derive nontrivial expectation invariants, e.g., relations among different program variables such as $\mathbb{E}[x'+y'] = x+y+3$, $\mathbb{E}[2x'-y'] = 2x-y$. In most cases, my results are at least as precise as those in [26, 81]. Exceptions are *biased-coin* and *uniform-dist*, collected from [81], where their invariant-generation algorithm uses a template-based approach and the form of expectations can be more complicated, e.g., $[P_1] \cdot \mathcal{E}_1 + [P_2] \cdot \mathcal{E}_2$ where P_1, P_2 are linear assertions and $\mathcal{E}_1, \mathcal{E}_2$ are linear expressions. Nevertheless, my analysis is fully automated and applicable to general programs, while [81] requires interactive proofs for nested loops, and [26] works only for single loops.

Chapter 5

Central Moment Analysis of Cost Accumulators in Probabilistic Programs

In this chapter, I propose a novel static analysis for deriving symbolic interval bounds on higher *central moments* for *cost accumulators* in probabilistic programs. Cost accumulators are quantities that can only be incremented or decremented through computation and do not influence the control flow, such as termination time, rewards in Markov decision processes, position information in control systems, and cash flow during bitcoin mining. Central moments $\mathbb{E}[(X - \mathbb{E}[X])^k]$ can be seen as polynomials of raw moments $\mathbb{E}[X], \cdots, \mathbb{E}[X^k]$, e.g., the variance $\mathbb{V}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2]$ can be rewritten as $\mathbb{E}[X^2] - \mathbb{E}^2[X]$, where $\mathbb{E}^k[X]$ denotes $(\mathbb{E}[X])^k$. Central moments can usually provide more information about the distribution of X than raw moments $\mathbb{E}[X^k]$. To derive bounds on central moments, we need both *upper* and *lower* bounds on the raw moments, because of the presence of *subtraction*. For example, to upper-bound $\mathbb{V}[X]$, a static analyzer needs to have an *upper* bound on $\mathbb{E}[X^2]$ and a *lower* bound on $\mathbb{E}^2[X]$.

Recent work has successfully automated inference of upper [112] or lower bounds [137] on the expected cost of probabilistic programs. Kura et al. [94] developed a system to derive upper bounds on higher *raw* moments of program runtimes. However, even in combination, existing approaches can*not* solve tasks such as deriving a lower bound on the second raw moment of runtimes, or deriving an upper bound on the variance of accumulators that count live heap cells. Fig. 5.1(a) summarizes the features of related work on moment inference for probabilistic programs. To the best of my knowledge, my work is the first moment-analysis tool that supports all of the listed programming and analysis features. Fig. 5.1(b) and (c) compare my work with related work in terms of tail-bound analysis on a concrete program (see §5.4). The bounds are derived for the cost accumulator *tick* in a random-walk program that I will present in §5.1. It can be observed that for $d \geq 20$, the

most precise tail bound for *tick* is the one obtained via an upper bound on the variance $\mathbb{V}[tick]$ (*tick*'s second central moment).

My work incorporates ideas known from the literature:

- Using the expected-potential method (or ranking super-martingales) to derive upper bounds on the expected program runtimes or monotone costs [25, 27, 28, 49, 94, 112].
- Using the *Optional Stopping Theorem* from probability theory to ensure the soundness of lower-bound inference for probabilistic programs [6, 62, 123, 137].
- Using *linear programming* (LP) to efficiently automate the (expected) potential method for (expected) cost analysis [68, 70, 135].

The contributions of my work are as follows:

- I develop moment semirings to compose the moments for a cost accumulator from two computations, and to enable interprocedural reasoning about higher moments.
- I instantiate moment semirings with the symbolic interval domain, use that to develop a derivation system for interval bounds on higher central moments for cost accumulators, and automate the derivation via LP solving.
- I prove the soundness of my derivation system for programs that satisfy the criterion of my extension to the Optional Stopping Theorem, and develop an algorithm for checking this criterion automatically.
- I implemented my analysis and evaluated it on a broad suite of benchmarks from the literature. The experimental results show that on a variety of examples, my analyzer is able to use higher central moments to obtain tighter tail bounds on program runtimes than the system of Kura et al. [94], which uses only upper bounds on raw moments.

5.1 Overview

In this section, I demonstrate the expected-potential method for both first-moment analysis (previous work) and higher central-moment analysis (this work) (§5.1.1), and discuss the challenges to supporting interprocedural reasoning and to ensuring the soundness of my approach (§5.1.2).

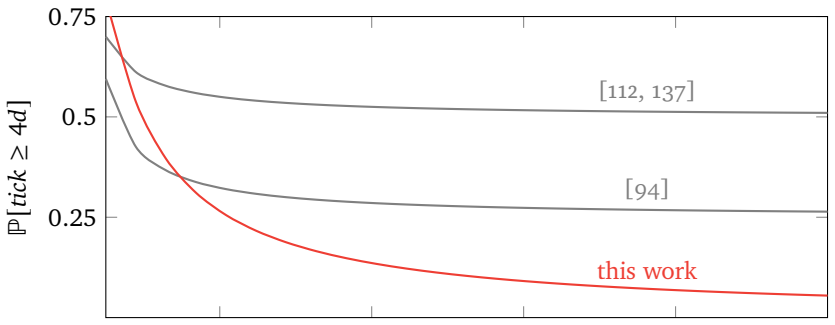
*Example 5.1. The program in Fig. 5.2 implements a bounded, biased random walk. The main function consists of a single statement "*call rdwalk*" that invokes a recursive function. The variables x and d represent the current position and the ending position of the random walk, respectively. We assume that $d > 0$ holds initially. In each step, the program samples the length of the current move from a uniform distribution on the interval $[-1, 2]$. The statement* **tick**(1) *adds one to a cost accumulator that counts the number of steps before the random walk ends. We denote this accumulator by tick in the rest of this section. The program terminates with probability one and its expected accumulated cost is bounded by* $2d + 4$.

feature	[17]	[112]	[94]	[137]	this work
loop		✓	✓	✓	✓
recursion		✓			✓
continuous distributions	✓		✓	✓	✓
non-monotone costs	✓			✓	✓
higher moments	✓		✓		✓
interval bounds	✓			✓	✓

(a)

	[112, 137]	[94]	this work
Derived bound	$\mathbb{E}[tick] \leq 2d + 4$	$\mathbb{E}[tick^2] \leq 4d^2 + 22d + 28$	$\mathbb{V}[tick] \leq 22d + 28$
Moment type	raw	raw	central
Concentration inequality	Markov (degree = 1)	Markov (degree = 2)	Cantelli
Tail bound $\mathbb{P}[tick \geq 4d]$	$\approx \frac{1}{2}$	$\approx \frac{1}{4}$	$\xrightarrow{d\to\infty} 0$

(b)



(c)

Fig. 5.1: (a) Comparison in terms of supporting features. (b) Comparison in terms of moment bounds for the running example. (c) Comparison in terms of derived tail bounds.

```
1  func rdwalk() begin
2    { 2(d − x) + 4 }
3    if x < d then
4      { 2(d − x) + 4 }
5      t ~ Uniform(−1, 2);
6      { 2(d − x − t) + 5 }
7      x := x + t;
8      { 2(d − x) + 5 }
9      call rdwalk;
10     { 1 }
11     tick(1)
12     { 0 }
13   fi
14 end
```

```
1  # XID ≝ {x, d, t}
2  # FID ≝ {rdwalk}
3  # pre-condition: {d > 0}
4  func main() begin
5    x := 0;
6    call rdwalk
7  end
```

Fig. 5.2: A bounded, biased random walk, implemented using recursion. The annotations show the derivation of an *upper* bound on the expected accumulated cost.

5.1.1 The Expected-Potential Method for Higher-Moment Analysis

My approach to higher-moment analysis is inspired by the *expected-potential method* [112], which is also known as *ranking super-martingales* [25, 27, 94, 137], for expected-cost bound analysis of probabilistic programs.

The classic *potential method* of amortized analysis [126] can be automated to derive symbolic cost bounds for non-probabilistic programs [68, 70]. The basic idea is to define a *potential function* $\phi : \Sigma \to \mathbb{R}^+$ that maps program states $\sigma \in \Sigma$ to nonnegative numbers, where we assume each state σ contains a cost-accumulator component $\sigma.\alpha$. If a program executes with initial state σ to final state σ', then it holds that $\phi(\sigma) \geq (\sigma'.\alpha - \sigma.\alpha) + \phi(\sigma')$, where $(\sigma'.\alpha - \sigma.\alpha)$ describes the accumulated cost from σ to σ'. The potential method also enables *compositional* reasoning: if a statement S_1 executes from σ to σ' and a statement S_2 executes from σ' to σ'', then we have $\phi(\sigma) \geq (\sigma'.\alpha - \sigma.\alpha) + \phi(\sigma')$ and $\phi(\sigma') \geq (\sigma''.\alpha - \sigma'.\alpha) + \phi(\sigma'')$; therefore, we derive $\phi(\sigma) \geq (\sigma''.\alpha - \sigma.\alpha) + \phi(\sigma'')$ for the sequential composition $S_1; S_2$. For non-probabilistic programs, the initial potential provides an *upper* bound on the accumulated cost.

This approach has been adapted to reason about expected costs of probabilistic programs [112, 137]. To derive upper bounds on the *expected* accumulated cost of a program S with initial state σ, one needs to take into consideration the *distribution* of all possible executions. More precisely, the potential function should satisfy the following property:

$$\phi(\sigma) \geq \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[C(\sigma, \sigma') + \phi(\sigma')], \tag{5.1}$$

where the notation $\mathbb{E}_{x \sim \mu}[f(x)]$ represents the expected value of $f(x)$, where x is drawn from the distribution μ, $[\![S]\!](\sigma)$ is the distribution over final states of executing S from σ, and $C(\sigma, \sigma') \stackrel{\text{def}}{=} \sigma'.\alpha - \sigma.\alpha$ is the execution cost from σ to σ'.

Example 5.2. Fig. 5.2 annotates the rdwalk *function from Ex. 5.1 with the derivation of an upper bound on the expected accumulated cost. The annotations, taken together, define an expected-potential function $\phi : \Sigma \to \mathbb{R}^+$ where a program state $\sigma \in \Sigma$ consists of a program point and a valuation for program variables. To justify the upper bound $2(d-x)+4$ for the function* rdwalk*, one has to show that the potential right before the* **tick**(1) *statement should be at least* 1*. This property is established by* backward *reasoning on the function body:*

- *For* **call** rdwalk*, we apply the "induction hypothesis" that the expected cost of the function* rdwalk *can be upper-bounded by $2(d-x)+4$. Adding the* 1 *unit of potential needed by the tick statement, we obtain $2(d-x)+5$ as the pre-annotation of the function call.*
- *For $x := x + t$, we substitute x with $x + t$ in the post-annotation of this statement to obtain the pre-annotation.*
- *For $t \sim \textsc{Uniform}(-1,2)$, because its post-annotation is $2(d-x-t)+5$, we compute its pre-annotation as*

$$\begin{aligned}\mathbb{E}_{t\sim\textsc{Uniform}(-1,2)}[2(d-x-t)+5] &= 2(d-x)+5-2\cdot\mathbb{E}_{t\sim\textsc{Uniform}(-1,2)}[t]\\ &= 2(d-x)+5-2\cdot\frac{1}{2}\\ &= 2(d-x)+4,\end{aligned}$$

which is exactly the upper bound we want to justify.

My Approach In this chapter, I focus on derivation of higher central moments. Observing that a central moment $\mathbb{E}[(X-\mathbb{E}[X])^k]$ can be rewritten as a polynomial of raw moments $\mathbb{E}[X], \cdots, \mathbb{E}[X^k]$, I reduce the problem of bounding central moments to reasoning about upper and lower bounds on raw moments. For example, the variance can be written as $\mathbb{V}[X] = \mathbb{E}[X^2] - \mathbb{E}^2[X]$, so it suffices to analyze the *upper* bound of the second moment $\mathbb{E}[X^2]$ and the *lower* bound on the square of the first moment $\mathbb{E}^2[X]$. For higher central moments, this approach requires both upper and lower bounds on higher raw moments. For example, consider the fourth central moment of a *nonnegative* random variable X: $\mathbb{E}[(X-\mathbb{E}[X])^4] = \mathbb{E}[X^4] - 4\mathbb{E}[X^3]\mathbb{E}[X] + 6\mathbb{E}[X^2]\mathbb{E}^2[X] - 3\mathbb{E}^4[X]$. Deriving an upper bound on the fourth central moment requires lower bounds on the first ($\mathbb{E}[X]$) and third ($\mathbb{E}[X^3]$) raw moments.

I now sketch the development of *moment semirings*. I first consider only the upper bounds on higher moments of *nonnegative* costs. To do so, I extend the range of the expected-potential function ϕ to real-valued vectors $(\mathbb{R}^+)^{m+1}$, where $m \in \mathbb{N}$ is the degree of the target moment. I update the potential inequality (5.1) as follows:

$$\phi(\sigma) \geq \mathbb{E}_{\sigma'\sim[\![S]\!](\sigma)}[\overrightarrow{\langle C(\sigma,\sigma')^k\rangle_{0\leq k\leq m}} \otimes \phi(\sigma')], \tag{5.2}$$

where $\overrightarrow{\langle v_k\rangle_{0\leq k\leq m}}$ denotes an $(m+1)$-dimensional vector, the order $\leq$ on vectors is defined pointwise, and $\otimes$ is some *composition* operator. Recall that $[\![S]\!](\sigma)$ denotes the distribution over final states of executing S from σ, and $C(\sigma,\sigma')$ describes the cost for the execution from σ to σ'. Intuitively, for $\phi(\sigma) = \overrightarrow{\langle \phi(\sigma)_k\rangle_{0\leq k\leq m}}$ and each k, the component $\phi(\sigma)_k$

```
1  func rdwalk() begin
2    { ⟨1, 2(d − x) + 4, 4(d − x)² + 22(d − x) + 28⟩ }
3    if x < d then
4      { ⟨1, 2(d − x) + 4, 4(d − x)² + 22(d − x) + 28⟩ }
5      t ~ Uniform(−1, 2);
6      { ⟨1, 2(d − x − t) + 5, 4(d − x − t)² + 26(d − x − t) + 37⟩ }
7      x := x + t;
8      { ⟨1, 2(d − x) + 5, 4(d − x)² + 26(d − x) + 37⟩ }
9      call rdwalk;
10     { ⟨1, 1, 1⟩ }
11     tick(1)
12     { ⟨1, 0, 0⟩ }
13   fi
14 end
```

Fig. 5.3: Derivation of an *upper* bound on the first and second moment of the accumulated cost.

is an upper bound on the k-th moment of the cost for the computation starting from σ. The 0-th moment is the *termination probability* of the computation, and I assume it is always one for now. We *cannot* simply define $\otimes$ as pointwise addition because, for example, $(a+b)^2 \neq a^2+b^2$ in general. If we think of b as the cost for some probabilistic computation, and we prepend a constant cost a to the computation, then by linearity of expectations, we have $\mathbb{E}[(a+b)^2] = \mathbb{E}[a^2+2ab+b^2] = a^2 + 2\cdot a\cdot\mathbb{E}[b] + \mathbb{E}[b^2]$, i.e., reasoning about the second moment requires me to keep track of the first moment. Similarly, we should have

$$\phi(\sigma)_2 \geq \mathbb{E}_{\sigma'\sim[\![S]\!](\sigma)}[C(\sigma,\sigma')^2 + 2\cdot C(\sigma,\sigma')\cdot\phi(\sigma')_1 + \phi(\sigma')_2],$$

for the second-moment component, where $\phi(\sigma')_1$ and $\phi(\sigma')_2$ denote $\mathbb{E}[b]$ and $\mathbb{E}[b^2]$, respectively. Therefore, the composition operator $\otimes$ for second-moment analysis (i.e., $m=2$) should be defined as

$$\langle 1, r_1, s_1\rangle \otimes \langle 1, r_2, s_2\rangle \stackrel{\text{def}}{=} \langle 1, r_1+r_2, s_1+2r_1r_2+s_2\rangle. \tag{5.3}$$

Example 5.3. Fig. 5.3 annotates the rdwalk *function from Ex. 5.1 with the derivation of an upper bound on both the first and second moment of the accumulated cost. To justify the first and second moment of the accumulated cost for the function* rdwalk*, We again perform backward reasoning:*

- *For* **tick**(1)*, it transforms a post-annotation a by $\lambda a.(\langle 1,1,1\rangle \otimes a)$; thus, the pre-annotation is $\langle 1,1,1\rangle \otimes \langle 1,0,0\rangle = \langle 1,1,1\rangle$.*
- *For* **call** rdwalk*, we apply the "induction hypothesis", i.e., the upper bound shown on line 2. We use the $\otimes$ operator to compose the induction hypothesis with the post-annotation of this function call:*

$$\begin{aligned}
&\langle 1, 2(d-x)+4, 4(d-x)^2+22(d-x)+28\rangle \otimes \langle 1,1,1\rangle \\
&= \langle 1, 2(d-x)+5, (4(d-x)^2+22(d-x)+28)+2\cdot(2(d-x)+4)+1\rangle \\
&= \langle 1, 2(d-x)+5, 4(d-x)^2+26(d-x)+37\rangle.
\end{aligned}$$

- *For $x := x+t$, we substitute x with $x+t$ in the post-annotation of this statement to obtain the pre-annotation.*
- *For $t \sim \textsc{Uniform}(-1,2)$, because the post-annotation involves both t and t^2, we compute from the definition of uniform distributions that*

$$\mathbb{E}_{t\sim\textsc{Uniform}(-1,2)}[t] = 1/2, \quad \mathbb{E}_{t\sim\textsc{Uniform}(-1,2)}[t^2] = 1.$$

Then the upper bound on the second moment is derived as follows:

$$\begin{aligned}
&\mathbb{E}_{t\sim\textsc{Uniform}(-1,2)}[4(d-x-t)^2+26(d-x-t)+37] \\
&= (4(d-x)^2+26(d-x)+37) - (8(d-x)+26)\cdot\mathbb{E}_{t\sim\textsc{Uniform}(-1,2)}[t] + 4\cdot\mathbb{E}_{t\sim\textsc{Uniform}(-1,2)}[t^2] \\
&= 4(d-x)^2+22(d-x)+28,
\end{aligned}$$

which is the same as the desired upper bound on the second moment of the accumulated cost for the function rdwalk. *(See Fig. 5.3, line 2.)*

I generalize the composition operator $\otimes$ to moments with arbitrarily high degrees, via a family of algebraic structures, which I name *moment semirings* (see §5.2.2). These semirings are *algebraic* in the sense that they can be instantiated with any partially ordered semiring, not just $\mathbb{R}^+$.

Interval Bounds Moment semirings not only provide a general method to analyze higher moments, but also enable reasoning about upper and lower bounds on moments *simultaneously*. The simultaneous treatment is also essential for analyzing programs with *non-monotone* costs (see §5.2.3).

I instantiate moment semirings with the standard interval semiring $\mathcal{I} = \{[a,b] \mid a \le b\}$. The algebraic approach allows me to systematically incorporate the interval-valued bounds, by *reinterpreting* operations in eq. (5.3) under $\mathcal{I}$:

$$\begin{aligned}
&\langle [1,1], [r_1^{\mathsf{L}}, r_1^{\mathsf{U}}], [s_1^{\mathsf{L}}, s_1^{\mathsf{U}}]\rangle \otimes \langle [1,1], [r_2^{\mathsf{L}}, r_2^{\mathsf{U}}], [s_2^{\mathsf{L}}, s_2^{\mathsf{U}}]\rangle \\
&\overset{\text{def}}{=} \langle [1,1], [r_1^{\mathsf{L}}, r_1^{\mathsf{U}}] +_{\mathcal{I}} [r_2^{\mathsf{L}}, r_2^{\mathsf{U}}], [s_1^{\mathsf{L}}, s_2^{\mathsf{U}}] +_{\mathcal{I}} 2\cdot([r_1^{\mathsf{L}}, r_1^{\mathsf{U}}] \cdot_{\mathcal{I}} [r_2^{\mathsf{L}}, r_2^{\mathsf{U}}]) +_{\mathcal{I}} [s_2^{\mathsf{L}}, s_2^{\mathsf{U}}]\rangle \\
&= \langle [1,1], [r_1^{\mathsf{L}}+r_2^{\mathsf{L}}, r_1^{\mathsf{U}}+r_2^{\mathsf{U}}], [s_1^{\mathsf{L}}+2\cdot\min S + s_2^{\mathsf{L}}, s_1^{\mathsf{U}}+2\cdot\max S + s_2^{\mathsf{U}}]\rangle,
\end{aligned}$$

where $S \overset{\text{def}}{=} \{r_1^{\mathsf{L}} r_2^{\mathsf{L}}, r_1^{\mathsf{L}} r_2^{\mathsf{U}}, r_1^{\mathsf{U}} r_2^{\mathsf{L}}, r_1^{\mathsf{U}} r_2^{\mathsf{U}}\}$. I then update the potential inequality eq. (5.2) as follows:

$$\phi(\sigma) \sqsupseteq \mathbb{E}_{\sigma'\sim[\![S]\!](\sigma)}[\overrightarrow{\langle [C(\sigma,\sigma')^k, C(\sigma,\sigma')^k]\rangle_{0\le k\le m}} \otimes \phi(\sigma')],$$

where the order $\sqsubseteq$ is defined as pointwise interval inclusion.

Example 5.4. Suppose that the interval bound on the first moment of the accumulated cost of the rdwalk *function from Ex. 5.1 is* $[2(d-x), 2(d-x)+4]$. *We can now derive the upper bound on the variance* $\mathbb{V}[tick] \leq 22d + 28$ *shown in Fig. 5.1(b) (where we substitute* x *with* 0 *because the main function initializes* x *to* 0 *on line 5 in Fig. 5.2):*

$$\begin{aligned}\mathbb{V}[tick] &= \mathbb{E}[tick^2] - \mathbb{E}^2[tick]\\ &\leq (\textit{upper bnd. on } \mathbb{E}[tick^2]) - (\textit{lower bnd. on } \mathbb{E}[tick])^2\\ &= (4d^2 + 22d + 28) - (2d)^2\\ &= 22d + 28.\end{aligned}$$

In §5.4, I describe how I use moment bounds to derive the tail bounds shown in Fig. 5.1(c).

5.1.2 Two Major Challenges

Interprocedural Reasoning Recall that in the derivation of Fig. 5.3, I use the $\otimes$ operator to compose the upper bounds on moments for **call** rdwalk and its post-annotation $\langle 1, 1, 1\rangle$. However, this approach does *not* work in general, because the post-annotation might be symbolic (e.g., $\langle 1, x, x^2\rangle$) and the callee might mutate referenced program variables (e.g., x). One workaround is to derive a *pre*-annotation for each possible *post*-annotation of a recursive function, i.e., the moment annotations for a recursive function is *polymorphic*. This workaround would *not* be effective for non-tail-recursive functions: for example, we need to reason about the rdwalk function in Fig. 5.3 with *infinitely* many post-annotations $\langle 1, 0, 0\rangle$, $\langle 1, 1, 1\rangle$, $\langle 1, 2, 4\rangle$, ..., i.e., $\langle 1, i, i^2\rangle$ for all $i \in \mathbb{Z}^+$.

My solution to *moment-polymorphic recursion* is to introduce a *combination* operator $\oplus$ in a way that if ϕ_1 and ϕ_2 are two expected-potential functions, then

$$\phi_1(\sigma) \oplus \phi_2(\sigma) \geq \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[\overrightarrow{\langle C(\sigma,\sigma')^k\rangle_{0\leq k\leq m}} \otimes (\phi_1(\sigma') \oplus \phi_2(\sigma'))].$$

I then use the $\oplus$ operator to derive a *frame* rule:

$$\frac{\{\ Q_1\ \}\ S\ \{\ Q_1'\ \} \qquad \{\ Q_2\ \}\ S\ \{\ Q_2'\ \}}{\{\ Q_1 \oplus Q_2\ \}\ S\ \{\ Q_1' \oplus Q_2'\ \}}$$

I define $\oplus$ as pointwise addition, i.e., for second moments,

$$\langle p_1, r_1, s_1\rangle \oplus \langle p_2, r_2, s_2\rangle \stackrel{\text{def}}{=} \langle p_1 + p_2, r_1 + r_2, s_1 + s_2\rangle, \tag{5.4}$$

and because the 0-th-moment (i.e., termination-probability) component is no longer guaranteed to be one, I redefine $\otimes$ to consider the termination probabilities:

$$\langle p_1, r_1, s_1\rangle \otimes \langle p_2, r_2, s_2\rangle \stackrel{\text{def}}{=} \langle p_1 p_2, p_2 r_1 + p_1 r_2, p_2 s_1 + 2 r_1 r_2 + p_1 s_2\rangle. \tag{5.5}$$

Remark 5.5. As I will show in §5.2.2, the composition operator $\otimes$ *and combination operator* $\oplus$ *form a* moment semiring*; consequently, we can use* algebraic *properties of semirings*

(e.g., distributivity) to aid higher-moment analysis. For example, a vector $\langle 0, r_1, s_1\rangle$ *whose termination-probability component is zero does not seem to make sense, because moments with respect to a zero distribution should also be zero. However, by distributivity, we have*

$$\begin{aligned}&\langle 1, r_3, s_3\rangle \otimes \langle 1, r_1 + r_2, s_1 + s_2\rangle\\ =\ &\langle 1, r_3, s_3\rangle \otimes (\langle 0, r_1, s_1\rangle \oplus \langle 1, r_2, s_2\rangle)\\ =\ &(\langle 1, r_3, s_3\rangle \otimes \langle 0, r_1, s_1\rangle) \oplus (\langle 1, r_3, s_3\rangle \oplus \langle 1, r_2, s_2\rangle).\end{aligned}$$

If we think of $\langle 1, r_1 + r_2, s_1 + s_2\rangle$ *as a post-annotation of a computation whose moments are bounded by* $\langle 1, r_3, s_3\rangle$*, the equation above indicates that we can use* $\oplus$ *to* decompose *the post-annotation into subparts, and then reason about each subpart separately. This fact inspires me to develop a decomposition technique for moment-polymorphic recursion.*

Example 5.6. With the $\oplus$ *operator and the frame rule, we only need to analyze the* rdwalk *function from Ex. 5.1 with* three *post-annotations:* $\langle 1, 0, 0\rangle$*,* $\langle 0, 1, 1\rangle$*, and* $\langle 0, 0, 2\rangle$*, which form a kind of "elimination sequence." We construct this sequence in an* on-demand *manner; the first post-annotation is the identity element* $\langle 1, 0, 0\rangle$ *of the moment semiring.*

For post-annotation $\langle 1, 0, 0\rangle$*, as shown in Fig. 5.3, we need to know the moment bound for* rdwalk *with the post-annotation* $\langle 1, 1, 1\rangle$*. Instead of reanalyzing* rdwalk *with the post-annotation* $\langle 1, 1, 1\rangle$*, We use the* $\oplus$ *operator to compute the "difference" between it and the previous post-annotation* $\langle 1, 0, 0\rangle$*. Observing that* $\langle 1, 1, 1\rangle = \langle 1, 0, 0\rangle \oplus \langle 0, 1, 1\rangle$*, we now analyze* rdwalk *with* $\langle 0, 1, 1\rangle$ *as the post-annotation:*

```
1 call rdwalk; { ⟨0, 1, 3⟩ }  # = ⟨1, 1, 1⟩ ⊗ ⟨0, 1, 1⟩
2 tick(1) { ⟨0, 1, 1⟩ }
```

Again, because $\langle 0, 1, 3\rangle = \langle 0, 1, 1\rangle \oplus \langle 0, 0, 2\rangle$*, we need to further analyze* rdwalk *with* $\langle 0, 0, 2\rangle$ *as the post-annotation:*

```
1 call rdwalk; { ⟨0, 0, 2⟩ }  # = ⟨1, 1, 1⟩ ⊗ ⟨0, 0, 2⟩
2 tick(1) { ⟨0, 0, 2⟩ }
```

With the post-annotation $\langle 0, 0, 2\rangle$*, we can now reason* monomorphically *without analyzing any new post-annotation! We can perform a succession of reasoning steps similar to what we have done in Ex. 5.2 to justify the following bounds ("unwinding" the elimination sequence):*

- $\{\langle 0, 0, 2\rangle\}$ rdwalk $\{\langle 0, 0, 2\rangle\}$*: Directly by backward reasoning with the post-annotation* $\langle 0, 0, 2\rangle$*.*
- $\{\langle 0, 1, 4(d-x)+9\rangle\}$ rdwalk $\{\langle 0, 1, 1\rangle\}$*: To analyze the recursive call with post-annotation* $\langle 0, 1, 3\rangle$*, we use the frame rule with the post-call-site annotation* $\langle 0, 0, 2\rangle$ *to derive* $\langle 0, 1, 4(d-x)+11\rangle$ *as the pre-annotation:*

  ```
  1 { ⟨0, 1, 4(d − x) + 11⟩ }  # = ⟨0, 1, 4(d − x) + 9⟩ ⊕ ⟨0, 0, 2⟩
  2 call rdwalk;
  3 { ⟨0, 1, 3⟩ }  # = ⟨0, 1, 1⟩ ⊕ ⟨0, 0, 2⟩
  ```

- $\{\langle 1, 2(d-x)+4, 4(d-x)^2+22(d-x)+28\rangle\}$ rdwalk $\{\langle 1, 0, 0\rangle\}$*: To analyze the recursive call with post-annotation* $\langle 1, 1, 1\rangle$*, we use the frame rule with the post-call-site annotation* $\langle 0, 1, 1\rangle$ *to derive* $\langle 1, 2(d-x)+5, 4(d-x)^2+26(d-x)+37\rangle$ *as the pre-annotation:*

```
1 func geo() begin { ⟨1, 2^x⟩ }
2   x := x + 1; { ⟨1, 2^{x−1}⟩ }
3   # expected-potential method for lower bounds:
4   # 2^{x−1} < 1/2 · (2^x + 1) + 1/2 · 0
5   if prob(1/2) then { ⟨1, 2^x + 1⟩ }
6     tick(1); { ⟨1, 2^x⟩ }
7     call geo { ⟨1, 0⟩}
8   fi
9 end
```

Fig. 5.4: A purely probabilistic loop with annotations for a *lower* bound on the first moment of the accumulated cost.

```
1 { ⟨1, 2(d − x) + 5, 4(d − x)^2 + 26(d − x) + 37⟩ }
2 # = ⟨1, 2(d − x) + 4, 4(d − x)^2 + 22(d − x) + 28⟩ ⊕ ⟨0, 1, 4(d − x) + 9⟩
3 call rdwalk;
4 { ⟨1, 1, 1⟩ } # = ⟨1, 0, 0⟩ ⊕ ⟨0, 1, 1⟩
```

In §5.2.3, we present an automatic inference system for the expected-potential method that is extended with interval-valued bounds on higher moments, with support for moment-polymorphic recursion.

Soundness of the Analysis Unlike the classic potential method, the expected-potential method is *not* always sound when reasoning about the moments for cost accumulators in probabilistic programs.

Counterexample 5.7. Consider the program in Fig. 5.4 that describes a purely probabilistic loop that exits the loop with probability 1/2 *in each iteration. The expected accumulated cost of the program should be* one *[62]. However, the annotations in Fig. 5.4 justify a potential function* 2^x *as a* lower *bound on the expected accumulated cost, no matter what value x has at the beginning, which is apparently* unsound.

Why does the expected-potential method fail in this case? The short answer is that dualization only works for some problems: upper-bounding the sum of nonnegative ticks is equivalent to lower-bounding the sum of nonpositive ticks; lower-bounding the sum of nonnegative ticks—the issue in Fig. 5.4—is equivalent to upper-bounding the sum of nonpositive ticks; however, the two kinds of problems are *inherently different* [62]. Intuitively, the classic potential method for bounding the costs of non-probabilistic programs is a *partial*-correctness method, i.e., derived upper/lower bounds are sound if the analyzed program terminates [113]. With probabilistic programs, many programs do not terminate *definitely*, but only *almost surely*, i.e., they terminate with probability one, but have some execution traces that are non-terminating. The programs in Figs. 5.2 and 5.4 are both almost-surely terminating. For the expected-potential method, the potential at a program state can be seen as an *average* of potentials needed for all possible computations that

$$
\begin{aligned}
S &::= \textbf{skip} \mid \textbf{tick}(c) \mid x \coloneqq E \mid x \sim D \mid \textbf{call}\ f \mid \textbf{while}\ L\ \textbf{do}\ S\ \textbf{od} \\
&\quad \mid \textbf{if}\ \textbf{prob}(p)\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi} \mid \textbf{if}\ L\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi} \mid S_1; S_2 \\
L &::= \textbf{true} \mid \textbf{not}\ L \mid L_1\ \textbf{and}\ L_2 \mid E_1 \leq E_2 \\
E &::= x \mid c \mid E_1 + E_2 \mid E_1 \times E_2 \\
D &::= \textsc{Uniform}(a, b) \mid \cdots
\end{aligned}
$$

Fig. 5.5: Syntax of the probabilistic programming language, where $p \in [0, 1]$, $a, b, c \in \mathbb{R}$, $a < b$, $x \in$ VID is a variable, and $f \in$ FID is a function identifier.

continue from the state. If the program state can lead to a non-terminating execution trace, the potential associated with that trace might be problematic, and as a consequence, the expected-potential method might fail.

Recent research [6, 62, 123, 137] has employed the *Optional Stopping Theorem* (OST) from probability theory to address this soundness issue. The classic OST provides a collection of *sufficient* conditions for reasoning about expected gain *upon termination* of stochastic processes, where the expected gain at any time is *invariant*. By constructing a stochastic process for executions of probabilistic programs and setting the expected-potential function as the invariant, one can apply the OST to justify the soundness of the expected-potential function. Recently, I have studied and proposed an extension to the classic OST with a *new* sufficient condition that is suitable for reasoning about higher moments [133]. I then prove the soundness of my central-moment inference for programs that satisfy this condition, and develop an algorithm to check this condition automatically (see §5.3).

5.2 Derivation System for Higher Moments

In this section, I describe the inference system used by my analysis. I first present a probabilistic programming language (§5.2.1). I then introduce *moment semirings* to compose higher moments for a cost accumulator from two computations (§5.2.2). I use moment semirings to develop my derivation system, which is presented as a declarative program logic (§5.2.3). Finally, I sketch how I reduce the inference of a derivation to LP solving (§5.2.4).

5.2.1 A Probabilistic Programming Language

In this chapter, I use a syntactic representation of Appl—instead of using CFHGs—to simplify the presentation of the derivation system. Recall that Appl supports general recursion and continuous distributions. I also assume that all the program variables are real-valued for brevity.

Fig. 5.5 presents the syntax as a grammar, where the metavariables S, L, E, and D stand for statements, conditions, expressions, and distributions, respectively. Each distribution D is associated with a probability measure $\mu_D \in \mathbb{D}(\mathbb{R})$. The statement "$x \sim D$" is a *random-sampling* assignment, which draws from the distribution μ_D to obtain a sample value and then assigns it to x. The statement "**if prob**(p) **then** S_1 **else** S_2 **fi**" is a *probabilistic-branching* statement, which executes S_1 with probability p, or S_2 with probability $(1-p)$.

The statement "**call** f" makes a (possibly recursive) call to the function with identifier $f \in \mathsf{FID}$. In this chapter, I assume that the functions only manipulate states that consist of global program variables. The statement **tick**(c), where $c \in \mathbb{R}$ is a constant, is used to define the *cost model*. It adds c to an anonymous global cost accumulator. Note that my implementation supports local variables, function parameters, return statements, as well as accumulation of non-constant costs; the restrictions imposed here are not essential, and are introduced solely to simplify the presentation.

I use a pair $\langle \mathscr{D}, S_{\mathsf{main}} \rangle$ to represent a program, where $\mathscr{D}$ is a finite map from function identifiers to their bodies and S_{main} is the body of the main function. I present an operational semantics for Appl in §5.3.2.

5.2.2 Moment Semirings

As discussed in §5.1.1, I want to design a *composition* operation $\otimes$ and a *combination* operation $\oplus$ to compose and combine higher moments of accumulated costs such that

$$\phi(\sigma) \sqsupseteq \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[\overrightarrow{\langle C(\sigma,\sigma')^k \rangle_{0 \le k \le m}} \otimes \phi(\sigma')],$$
$$\phi_1(\sigma) \oplus \phi_2(\sigma) \sqsupseteq \mathbb{E}_{\sigma' \sim [\![S]\!](\sigma)}[\overrightarrow{\langle C(\sigma,\sigma')^k \rangle_{0 \le k \le m}} \otimes (\phi_1(\sigma') \oplus \phi_2(\sigma'))],$$

where the expected-potential functions ϕ, ϕ_1, ϕ_2 map program states to interval-valued vectors, $C(\sigma,\sigma')$ is the cost for the computation from σ to σ', and m is the degree of the target moment. In eqs. (5.4) and (5.5), I gave a definition of $\otimes$ and $\oplus$ suitable for first and second moments, respectively. In this section, I generalize them to reason about upper and lower bounds of higher moments. My approach is inspired by the work of Li and Eisner [97], which develops a method to "lift" techniques for first moments to those for second moments. Instead of restricting the elements of semirings to be vectors of numbers, I propose *algebraic* moment semirings that can also be instantiated with vectors of intervals, which we need for the interval-bound analysis that was demonstrated in §5.1.1.

Definition 5.8. The m-th order *moment semiring* $\mathcal{M}_{\mathcal{R}}^{(m)} = (|\mathcal{R}|^{m+1}, \oplus, \otimes, \underline{0}, \underline{1})$ is parametrized by a partially ordered semiring $\mathcal{R} = (|\mathcal{R}|, \le, +, \cdot, 0, 1)$, where

$$\overrightarrow{\langle u_k \rangle_{0 \le k \le m}} \oplus \overrightarrow{\langle v_k \rangle_{0 \le k \le m}} \stackrel{\text{def}}{=} \overrightarrow{\langle u_k + v_k \rangle_{0 \le k \le m}}, \tag{5.6}$$
$$\overrightarrow{\langle u_k \rangle_{0 \le k \le m}} \otimes \overrightarrow{\langle v_k \rangle_{0 \le k \le m}} \stackrel{\text{def}}{=} \overrightarrow{\langle \textstyle\sum_{i=0}^{k} \binom{k}{i} \times (u_i \cdot v_{k-i}) \rangle_{0 \le k \le m}}, \tag{5.7}$$

$\binom{k}{i}$ is the binomial coefficient; the scalar product $n \times u$ is an abbreviation for $\sum_{i=1}^{n} u$, for $n \in \mathbb{Z}^+, u \in \mathcal{R}$; $\underline{0} \stackrel{\text{def}}{=} \langle 0, 0, \cdots, 0 \rangle$; and $\underline{1} \stackrel{\text{def}}{=} \langle 1, 0, \cdots, 0 \rangle$. We define the partial order $\sqsubseteq$ as the pointwise extension of the partial order $\leq$ on $\mathcal{R}$.

Intuitively, the definition of $\otimes$ in eq. (5.7) can be seen as the multiplication of two moment-generating functions for distributions with moments $\overrightarrow{\langle u_k \rangle_{0 \leq k \leq m}}$ and $\overrightarrow{\langle v_k \rangle_{0 \leq k \leq m}}$, respectively. I prove a composition property for moment semirings.

LEMMA 5.9. *For all $u, v \in \mathcal{R}$, it holds that*

$$\overrightarrow{\langle (u+v)^k \rangle_{0 \leq k \leq m}} = \overrightarrow{\langle u^k \rangle_{0 \leq k \leq m}} \otimes \overrightarrow{\langle v^k \rangle_{0 \leq k \leq m}},$$

where u^n is an abbreviation for $\prod_{i=1}^{n} u$, for $n \in \mathbb{Z}^+, u \in \mathcal{R}$.

PROOF. Let *RHS* denote the right-hand-side of the target equation. Observe that

$$RHS_k = \sum_{i=0}^{k} \binom{k}{i} \times (u^i \cdot v^{k-i}).$$

We prove by induction on k that $(u+v)^k = RHS_k$.

- $k = 0$: Then $(u+v)^0 = 1$. On the other hand, we have

$$RHS_0 = \binom{0}{0} \times (u^0 \cdot v^0) = 1 \times (1 \cdot 1) = 1.$$

- Suppose that $(u+v)^k = RHS_k$. Then

$$\begin{aligned}
(u+v)^{k+1} &= (u+v) \cdot (u+v)^k \\
&= (u+v) \cdot \sum_{i=0}^{k} \binom{k}{i} \times (u^i \cdot v^{k-i}) \\
&= \sum_{i=0}^{k} \binom{k}{i} \times (u^{i+1} \cdot v^{k-i}) + \sum_{i=0}^{k} \binom{k}{i} \times (u^i \cdot v^{k-i+1}) \\
&= \sum_{i=1}^{k+1} \binom{k}{i-1} \times (u^i \cdot v^{k-i+1}) + \sum_{i=0}^{k} \binom{k}{i} \times (u^i \cdot v^{k-i+1}) \\
&= \sum_{i=0}^{k+1} (\binom{k}{i-1} + \binom{k}{i}) \times (u^i \cdot v^{k-i+1}) \\
&= \sum_{i=0}^{k+1} \binom{k+1}{i} \times (u^k \cdot v^{k-i+1}) \\
&= RHS_{k+1}.
\end{aligned}$$

□

5.2.3 Inference Rules

I present the derivation system as a declarative program logic that uses moment semirings to enable compositional reasoning and moment-polymorphic recursion.

Interval-Valued Moment Semirings My derivation system infers upper and lower bounds simultaneously, rather than separately, which is essential for *non-monotone* costs. Consider a program "$\mathbf{tick}(-1); S$" and suppose that we have $\langle 1, 2, 5\rangle$ and $\langle 1, -2, 5\rangle$ as the upper and lower bound on the first two moments of the cost for S, respectively. If we only use the upper bound, we derive $\langle 1, -1, 1\rangle \otimes \langle 1, 2, 5\rangle = \langle 1, 1, 2\rangle$, which is *not* an upper bound on the moments of the cost for the program; if the *actual* moments of the cost for S are $\langle 1, 0, 5\rangle$, then the *actual* moments of the cost for "$\mathbf{tick}(-1); S$" are $\langle 1, -1, 1\rangle \otimes \langle 1, 0, 5\rangle = \langle 1, -1, 4\rangle \not\leq \langle 1, 1, 2\rangle$. Thus, in the analysis, I instantiate moment semirings with the interval domain $\mathcal{I}$. For the program "$\mathbf{tick}(-1); S$", its interval-valued bound on the first two moments is $\langle [1,1], [-1,-1], [1,1]\rangle \otimes \langle [1,1], [-2,2], [5,5]\rangle = \langle [1,1], [-3,1], [2,10]\rangle$.

Template-Based Expected-Potential Functions The basic approach to automated inference using potential functions is to introduce a *template* for the expected-potential functions. Let me fix $m \in \mathbb{N}$ as the degree of the target moment. Because I use $\mathcal{M}_{\mathcal{I}}^{(m)}$-valued expected-potential functions whose range is vectors of intervals, the templates are vectors of intervals whose ends are represented *symbolically*. In this chapter, I represent the ends of intervals by *polynomials* in $\mathbb{R}[\mathsf{VID}]$ over program variables.

More formally, I lift the interval semiring $\mathcal{I}$ to a *symbolic* interval semiring $\mathcal{PI}$ by representing the ends of the k-th interval by polynomials in $\mathbb{R}_{kd}[\mathsf{VID}]$ up to degree kd for some fixed $d \in \mathbb{N}$. Let $\mathcal{M}_{\mathcal{PI}}^{(m)}$ be the m-th order moment semiring instantiated with the symbolic interval semiring. Then the potential annotation is represented as $Q = \overrightarrow{\langle [L_k, U_k]\rangle_{0\leq k\leq m}} \in \mathcal{M}_{\mathcal{PI}}^{(m)}$, where L_k's and U_k's are polynomials in $\mathbb{R}_{kd}[\mathsf{VID}]$. Q defines an $\mathcal{M}_{\mathcal{I}}^{(m)}$-valued expected-potential function $\phi_Q(\sigma) \stackrel{\text{def}}{=} \overrightarrow{\langle [\sigma(L_k), \sigma(U_k)]\rangle_{0\leq k\leq m}}$, where σ is a program state, and $\sigma(L_k)$ and $\sigma(U_k)$ are L_k and U_k evaluated over σ, respectively.

Inference Rules I formalize my derivation system for moment analysis in a Hoare-logic style. The judgment has the form $\Delta \vdash_h \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\}$, where S is a statement, $\{\Gamma; Q\}$ is a precondition, $\{\Gamma'; Q'\}$ is a postcondition, $\Delta = \langle \Delta_k\rangle_{0\leq k\leq m}$ is a context of function specifications, and $h \in \mathbb{Z}^+$ specifies some restrictions put on Q, Q' that I will explain later. The *logical context* $\Gamma : (\mathsf{VID} \to \mathbb{R}) \to \{\top, \bot\}$ is a predicate that describes reachable states at a program point. The *potential annotation* $Q \in \mathcal{M}_{\mathcal{PI}}^{(m)}$ specifies a map from program states to the moment semiring that is used to define interval-valued expected-potential functions. The semantics of the triple $\{\cdot; Q\}\ S\ \{\cdot; Q'\}$ is that if the rest of the computation after executing S has its moments of the accumulated cost bounded by $\phi_{Q'}$, then the whole computation has its moments of the accumulated cost bounded by ϕ_Q. The parameter

h restricts all i-th-moment components in Q, Q', such that $i < h$, to be $[0, 0]$. I call such potential annotations *h-restricted*; this construction is motivated by an observation from Ex. 5.6, where I illustrated the benefits of carrying out interprocedural analysis using an "elimination sequence" of annotations for recursive function calls, where the successive annotations have a greater number of zeros, filling from the left. *Function specifications* are valid pairs of pre- and post-conditions for all declared functions in a program. For each k, such that $0 \leq k \leq m$, and each function f, a valid specification $(\Gamma; Q, \Gamma'; Q') \in \Delta_k(f)$ is justified by the judgment $\Delta \vdash_k \{\Gamma; Q\}\ \mathscr{D}(f)\ \{\Gamma'; Q'\}$, where $\mathscr{D}(f)$ is the function body of f, and Q, Q' are k-restricted. The validity of a context Δ for function specifications is then established by the validity of all specifications in Δ, denoted by $\vdash \Delta$. To perform context-sensitive analysis, a function can have multiple specifications.

Fig. 5.6 presents the inference rules. The rule (Q-Tick) is the only rule that deals with costs in a program. To accumulate the moments of the cost, I use the $\otimes$ operation in the moment semiring $\mathcal{M}^{(m)}_{\mathcal{P}_I}$. The rule (Q-Sample) accounts for sampling statements. Because "$x \sim D$" randomly assigns a value to x in the support of distribution D, I quantify x out universally from the logical context. To compute $Q = \mathbb{E}_{x\sim\mu_D}[Q']$, where x is drawn from distribution D, I assume the moments for D are well-defined and computable, and substitute x^i, $i \in \mathbb{N}$ with the corresponding moments in Q'. I make this assumption because every component of Q' is a polynomial over program variables. For example, if $D = \textsc{Uniform}(-1, 2)$, we know the following facts

$$\mathbb{E}_{x\sim\mu_D}[x^0] = 1, \mathbb{E}_{x\sim\mu_D}[x^1] = \frac{1}{2}, \mathbb{E}_{x\sim\mu_D}[x^2] = 1, \mathbb{E}_{x\sim\mu_D}[x^3] = \frac{5}{4}.$$

Then for $Q' = \langle [1, 1], [1 + x^2, xy^2 + x^3 y]\rangle$, by the linearity of expectations, we compute $Q = \mathbb{E}_{x\sim\mu_D}[Q']$ as follows:

$$\begin{aligned}\mathbb{E}_{x\sim\mu_D}[Q'] &= \langle [1, 1], [\mathbb{E}_{x\sim\mu_D}[1 + x^2], \mathbb{E}_{x\sim\mu_D}[xy^2 + x^3 y]\rangle \\ &= \langle [1, 1], [1 + \mathbb{E}_{x\sim\mu_D}[x^2], y^2\mathbb{E}_{x\sim\mu_D}[x] + y\mathbb{E}_{x\sim\mu_D}[x^3]]\rangle \\ &= \langle [1, 1], [2, \frac{1}{2}\cdot y^2 + \frac{5}{4}\cdot y]\rangle.\end{aligned}$$

The other probabilistic rule (Q-Prob) deals with probabilistic branching. Intuitively, if the moments of the execution of S_1 and S_2 are q_1 and q_2, respectively, and those of the accumulated cost of the computation after the branch statement is bounded by $\phi_{Q'}$, then the moments for the whole computation should be bounded by a "weighted average" of $(q_1 \otimes \phi_{Q'})$ and $(q_2 \otimes \phi_{Q'})$, with respect to the branching probability p. I implement the weighted average by the combination operator $\oplus$ applied to $\langle [p, p], [0, 0], \cdots, [0, 0]\rangle \otimes q_1 \otimes \phi_{Q'}$ and $\langle [1-p, 1-p], [0, 0], \cdots, [0, 0]\rangle \otimes q_2 \otimes \phi_{Q'}$, because the 0-th moments denote probabilities.

The rules (Q-Call-Poly) and (Q-Call-Mono) handle function calls. Recall that in Ex. 5.6, I use the $\oplus$ operator to combine multiple potential functions for a function to reason about recursive function calls. The restriction parameter h is used to ensure that the derivation system only needs to reason about *finitely* many post-annotations for each call site. In rule (Q-Call-Poly), where h is smaller than the target moment m, I fetch the pre- and

$$\text{(Valid-Ctx)}\quad \frac{\forall (h,f) \in \mathrm{dom}(\Delta) : \forall (\Gamma; Q, \Gamma'; Q') \in \Delta(f) : \Delta \vdash_h \{\Gamma; Q\}\ \mathscr{D}(f)\ \{\Gamma'; Q'\}}{\vdash \Delta}$$

$$\text{(Q-Skip)}\quad \frac{}{\Delta \vdash_h \{\Gamma; Q\}\ \mathbf{skip}\ \{\Gamma; Q\}} \qquad \text{(Q-Tick)}\quad \frac{Q = \overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes Q'}{\Delta \vdash_h \{\Gamma; Q\}\ \mathbf{tick}(c)\ \{\Gamma; Q'\}} \qquad \text{(Q-Assign)}\quad \frac{\Gamma = [E/x]\Gamma' \qquad Q = [E/x]Q'}{\Delta \vdash_h \{\Gamma; Q\}\ x \coloneqq E\ \{\Gamma'; Q'\}}$$

$$\text{(Q-Sample)}\quad \frac{\Gamma = \forall x \in \mathrm{supp}(\mu_D) : \Gamma' \qquad Q = \mathbb{E}_{x \sim \mu_D}[Q']}{\Delta \vdash_h \{\Gamma; Q\}\ x \sim D\ \{\Gamma'; Q'\}} \qquad \text{(Q-Loop)}\quad \frac{\Delta \vdash_h \{\Gamma \wedge L; Q\}\ S\ \{\Gamma; Q\}}{\Delta \vdash_h \{\Gamma; Q\}\ \mathbf{while}\ L\ \mathbf{do}\ S\ \mathbf{od}\ \{\Gamma \wedge \neg L; Q\}}$$

$$\text{(Q-Seq)}\quad \frac{\Delta \vdash_h \{\Gamma; Q\}\ S_1\ \{\Gamma'; Q'\} \qquad \Delta \vdash_h \{\Gamma'; Q'\}\ S_2\ \{\Gamma''; Q''\}}{\Delta \vdash_h \{\Gamma; Q\}\ S_1; S_2\ \{\Gamma''; Q''\}} \qquad \text{(Q-Call-Mono)}\quad \frac{(\Gamma; Q, \Gamma'; Q') \in \Delta_m(f)}{\Delta \vdash_m \{\Gamma; Q\}\ \mathbf{call}\ f\ \{\Gamma'; Q'\}}$$

$$\text{(Q-Call-Poly)}\quad \frac{h < m \qquad \Delta_h(f) = (\Gamma; Q_1, \Gamma'; Q_1') \qquad \Delta \vdash_{h+1} \{\Gamma; Q_2\}\ \mathscr{D}(f)\ \{\Gamma'; Q_2'\}}{\Delta \vdash_h \{\Gamma; Q_1 \oplus Q_2\}\ \mathbf{call}\ f\ \{\Gamma'; Q_1' \oplus Q_2'\}}$$

$$\text{(Q-Prob)}\quad \frac{\begin{array}{c}\Delta \vdash_h \{\Gamma; Q_1\}\ S_1\ \{\Gamma'; Q'\} \qquad \Delta \vdash_h \{\Gamma; Q_2\}\ S_2\ \{\Gamma'; Q'\} \qquad Q = P \oplus R \\ P = \langle [p,p], [0,0], \cdots, [0,0] \rangle \otimes Q_1 \qquad R = \langle [1-p, 1-p], [0,0], \cdots, [0,0] \rangle \otimes Q_2\end{array}}{\Delta \vdash_h \{\Gamma; Q\}\ \mathbf{if}\ \mathbf{prob}(p)\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{\Gamma'; Q'\}}$$

$$\text{(Q-Cond)}\quad \frac{\Delta \vdash_h \{\Gamma \wedge L; Q\}\ S_1\ \{\Gamma'; Q'\} \qquad \Delta \vdash \{\Gamma \wedge \neg L; Q\}\ S_2\ \{\Gamma'; Q'\}}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{if}\ L\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{\Gamma'; Q'\}}$$

$$\text{(Q-Weaken)}\quad \frac{\Delta \vdash_h \{\Gamma_0; Q_0\}\ S\ \{\Gamma_0'; Q_0'\} \qquad \Gamma \models \Gamma_0 \qquad \Gamma_0' \models \Gamma' \qquad \Gamma \models Q \sqsupseteq Q_0 \qquad \Gamma_0' \models Q_0' \sqsupseteq Q'}{\Delta \vdash_h \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\}}$$

Fig. 5.6: Inference rules of the derivation system.

```
1  func rdwalk() begin
2    { x < d + 2; ⟨[1,1], [2(d−x), 2(d−x)+4],
3      [4(d−x)^2 + 6(d−x) − 4, 4(d−x)^2 + 22(d−x) + 28]⟩ }
4    if x < d then
5      { x < d; ⟨[1,1], [2(d−x), 2(d−x)+4],
6        [4(d−x)^2 + 6(d−x) − 4, 4(d−x)^2 + 22(d−x) + 28]⟩ }
7      t ~ Uniform(−1, 2);
8      { x < d ∧ t ≤ 2; ⟨[1,1], [2(d−x−t)+1, 2(d−x−t)+5],
9        [4(d−x−t)^2 + 10(d−x−t) − 3, 4(d−x−t)^2 + 26(d−x−t) + 37)]⟩ }
10     x := x + t;
11     { x < d + 2; ⟨[1,1], [2(d−x)+1, 2(d−x)+5],
12       [4(d−x)^2 + 10(d−x) − 3, 4(d−x)^2 + 26(d−x) + 37]⟩ }
13     call rdwalk;
14     { ⊤; ⟨[1,1], [1,1], [1,1]⟩ }
15     tick(1)
16     { ⊤; ⟨[1,1], [0,0], [0,0]⟩ }
17   fi
18 end
```

Fig. 5.7: The rdwalk function with annotations for the interval-bounds on the first and second moments.

post-condition Q_1, Q_1' for the function f from the specification context Δ_h. I then combine it with a *frame* of $(h+1)$-restricted potential annotations Q_2, Q_2' for the function f. The frame is used to account for the interval bounds on the moments for the computation after the function call for most non-tail-recursive programs. When h reaches the target moment m, I use the rule (Q-Call-Mono) to reason *moment-monomorphically*, because setting h to $m+1$ implies that the frame can only be $\langle [0,0], [0,0], \cdots, [0,0] \rangle$.

The structural rule (Q-Weaken) is used to strengthen the pre-condition and relax the post-condition. The entailment relation $\Gamma \models \Gamma'$ states that the logical implication $\Gamma \implies \Gamma'$ is valid. In terms of the bounds on higher moments for cost accumulators, if the triple $\{\cdot; Q\}\ S\ \{\cdot; Q'\}$ is valid, then I can safely widen the intervals in the pre-condition Q and narrow the intervals in the post-condition Q'.

Example 5.10. Fig. 5.7 presents the logical context and the complete potential annotation for the first and second moments for the cost accumulator tick of the rdwalk *function from Ex. 5.1. Similar to the reasoning in Ex. 5.6, we can justify the derivation using moment-polymorphic recursion and the moment bounds for* rdwalk *with post-annotations* $\langle [0,0], [1,1], [1,1] \rangle$ *and* $\langle [0,0], [0,0], [2,2] \rangle$.

5.2.4 Automatic Linear-Constraint Generation

I adapt existing techniques [22, 112] to automate my inference system by (i) using an abstract interpreter to infer logical contexts, (ii) generating templates and linear constraints by inductively applying the derivation rules to the analyzed program, and (iii) employing

$$\frac{\begin{array}{c} p_1^{tk} = u_1^{tk} \quad q_1^{tk} = u_1^{tk} + v_1^{tk} \quad q_x^{tk} = v_x^{tk} \quad q_N^{tk} = v_N^{tk} \quad q_r^{tk} = v_r^{tk} \\ t_1^{tk} = w_1^{tk} + 2v_1^{tk} + u_1^{tk} \quad t_x^{tk} = w_x^{tk} + 2v_x^{tk} \quad t_{x^2}^{tk} = w_{x^2}^{tk} \quad \cdots \end{array}}{\Delta \vdash \{\top; (P^{tk}, Q^{tk}, T^{tk})\}\ \mathbf{tick}(1)\ \{\top; (U^{tk}, V^{tk}, W^{tk})\}} \text{ (Q-Tick)}$$

$$\frac{\begin{array}{c} p_1^{sa} = u_1^{sa} \quad q_1^{sa} = v_1^{sa} + \frac{1}{2} \cdot v_r^{sa} \quad q_x^{sa} = v_x^{sa} \quad q_N^{sa} = v_N^{sa} \quad q_r^{sa} = 0 \\ t_1^{sa} = w_1^{sa} + \frac{1}{2} \cdot w_r^{sa} + 1 \cdot w_{r^2}^{sa} \quad t_x^{sa} = w_x^{sa} + \frac{1}{2} \cdot w_{r \cdot x}^{sa} \quad t_{x^2}^{sa} = w_{x^2}^{sa} \quad \cdots \end{array}}{\Delta \vdash \{x < N; (P^{sa}, Q^{sa}, T^{sa})\}\ r \sim \textsc{Uniform}(-1, 2)\ \{x < N \wedge r \leq 2; (U^{sa}, V^{sa}, W^{sa})\}} \text{ (Q-Sample)}$$

Fig. 5.8: Generate linear constraints, guided by inference rules.

an off-the-shelf LP solver to discharge the linear constraints. During the generation phase, the coefficients of monomials in the polynomials from the ends of the intervals in every qualitative context $Q \in \mathcal{M}_{\mathcal{PI}}^{(m)}$ are recorded as symbolic names, and the inequalities among those coefficients—derived from the inference rules in Fig. 5.6—are emitted to the LP solver.

Generating Linear Constraints Fig. 5.8 demonstrates the generation process for some of the bounds in Fig. 5.3. Let B_k be a vector of *monomials* over program variables VID of degree up to k. Then a polynomial $\sum_{b \in B_k} q_b \cdot b$, where $q_b \in \mathbb{R}$ for all $b \in B_k$, can be represented as a vector of its coefficients $(q_b)_{b \in B_k}$. I denote coefficient vectors by uppercase letters, while I use lowercase letters as names of the coefficients. I also assume that the degree of the polynomials for the k-th moments is up to k.

For (Q-Tick), I generate constraints that correspond to the composition operation $\otimes$ of the moment semiring. For example, the second-moment component should satisfy

$$\begin{aligned} &\sum_{b \in B_2} t_b^{tk} \cdot b \\ &= \text{the second-moment component of } \big((1,1,1) \otimes (\sum_{b \in B_0} u_b^{tk} \cdot b, \sum_{b \in B_1} v_b^{tk} \cdot b, \sum_{b \in B_2} w_b^{tk} \cdot b)\big) \\ &= \sum_{b \in B_2} w_b^{tk} \cdot b + 2 \cdot \sum_{b \in B_1} v_b^{tk} \cdot b + \sum_{b \in B_0} u_b^{tk} \cdot b. \end{aligned}$$

Then we extract $t_1^{tk} = w_1^{tk} + 2v_1^{tk} + u_1^{tk}$ for $b = 1$, and $t_x^{tk} = w_x^{tk} + 2v_x^{tk}$ for $b = x$, etc. For (Q-Sample), I generate constraints to perform "partial evaluation" on the polynomials by substituting r with the moments of $\textsc{Uniform}(-1, 2)$. As I discussed in §5.2.3, let D denote $\textsc{Uniform}(-1, 2)$, then $\mathbb{E}_{r \sim D}[w_r^{sa} \cdot r] = w_r^{sa} \cdot \mathbb{E}_{r \sim D}[r] = \frac{1}{2} \cdot w_r^{sa}$, $\mathbb{E}_{r \sim D}[w_{r^2}^{sa} \cdot r^2] = w_{r^2}^{sa} \cdot \mathbb{E}_{r \sim D}[r^2] = 1 \cdot w_{r^2}^{sa}$. Then we generate a constraint $t_1^{sa} = w_1^{sa} + \frac{1}{2} \cdot w_r^{sa} + 1 \cdot w_{r^2}^{sa}$ for t_1^{sa}.

The loop rule (Q-Loop) involves constructing loop *invariants* Q, which is in general a non-trivial problem for automated static analysis. Instead of computing the loop invariant Q explicitly, my system represents Q directly as a template with unknown coefficients, then

uses Q as the post-annotation to analyze the loop body and obtain a pre-annotation, and finally generates linear constraints that indicate the pre-annotation equals to Q.

The structural rule (Q-WEAKEN) can be applied at any point during the derivation. In my implementation, I apply it where the control flow has a branch, because different branches might have different costs. To handle the judgment $\Gamma \models Q \sqsupseteq Q'$, i.e., to generate constraints that ensure one interval is always contained in another interval, where the ends of the intervals are polynomials, I adapt the idea of *rewrite functions* [22, 112]. Intuitively, to ensure that $[L_1, U_1] \sqsupseteq [L_2, U_2]$, i.e., $L_1 \leq L_2$ and $U_2 \leq U_1$, under the logical context Γ, I generate constraints indicating that there exist two polynomials T_1, T_2 that are always nonnegative under Γ, such that $L_1 = L_2 + T_1$ and $U_1 = U_2 - T_2$. Here, T_1 and T_2 are like slack variables, except that because all quantities are polynomials, they are too (i.e., slack polynomials). In my implementation, Γ is a set of linear constraints over program variables of the form $\mathcal{E} \geq 0$, then I can represent T_1, T_2 by *conical* combinations (i.e., linear combinations with nonnegative scalars) of expressions $\mathcal{E}$ in Γ.

Solving Linear Constraints The LP solver not only finds assignments to the coefficients that satisfy the constraints, it can also optimize a linear objective function. In my central-moment analysis, I construct an objective function that tries to minimize imprecision. For example, let me consider upper bounds on the variance. I randomly pick a concrete valuation of program variables that satisfies the pre-condition (e.g., $d > 0$ in Fig. 5.2), and then substitute program variables with the concrete valuation in the polynomial for the upper bound on the variance (obtained from bounds on the raw moments). The resulting linear combination of coefficients, which I set as the objective function, stands for the variance under the concrete valuation. Thus, minimizing the objective function produces the most precise upper bound on the variance under the specific concrete valuation. Also, I can extract a *symbolic* upper bound on the variance using the assignments to the coefficients. Because the derivation of the bounds only uses the given pre-condition, the symbolic bounds apply to all valuations that satisfy the pre-condition.

5.3 Soundness of Higher-Moment Analysis

In this section, I study the soundness of my derivation system for higher-moment analysis. I first present a small-step operational cost semantics for the probabilistic programming language (§5.3.1). I then develop a Markov-chain semantics to reason about how *stepwise* costs contribute to the *global* accumulated cost (§5.3.2). With the Markov-chain semantics, I formulate higher-moment analysis with respect to the semantics and prove the soundness of my derivation system for higher-moment analysis based on a recent extension to the *Optional Stopping Theorem* (§5.3.3). Finally, I sketch the algorithmic approach for ensuring the soundness of my analysis (§5.3.4).

5.3.1 A Small-Step Operational Semantics

I start with a small-step operational semantics with continuations, which I will use later to construct the Markov-chain semantics. I follow a distribution-based approach [16, 91] to define an operational cost semantics. A probabilistic semantics steps a program configuration to a probability distribution on configurations. A *program configuration* $\sigma \in \Sigma$ is a quadruple $\langle \gamma, S, K, \alpha \rangle$ where $\gamma : \mathsf{VID} \to \mathbb{R}$ is a program state that maps variables to values, S is the statement being executed, K is a continuation that describes what remains to be done after the execution of S, and $\alpha \in \mathbb{R}$ is the global cost accumulator. To describe these distributions formally, I need to construct a measurable space of program configurations. My approach is to construct a measurable space for each of the four components of configurations, and then use their product measurable space as the semantic domain.

- Valuations $\gamma : \mathsf{VID} \to \mathbb{R}$ are finite real-valued maps, so I define $(V, \mathcal{V}) \stackrel{\text{def}}{=} (\mathbb{R}^{\mathsf{VID}}, \mathcal{B}(\mathbb{R}^{\mathsf{VID}}))$ as the canonical structure on a finite-dimensional space.
- The executing statement S can contain real numbers, so I need to "lift" the Borel σ-algebra on $\mathbb{R}$ to program statements. Intuitively, statements with exactly the same structure can be treated as vectors of parameters that correspond to their real-valued components. Formally, I achieve this by constructing a metric space on statements and then extracting a Borel σ-algebra from the metric space. Fig. 5.9 presents an inductively defined metric d_S on statements, as well as metrics d_E, d_L, and d_D on expressions, conditions, and distributions, respectively, as they are required by d_S. I denote the result measurable space by $(S, \mathcal{S})$.
- A *continuation* K is either an empty continuation **Kstop**, a loop continuation **Kloop** $L\ S\ K$, or a sequence continuation **Kseq** $S\ K$. Similarly, I construct a measurable space $(K, \mathcal{K})$ on continuations by extracting from a metric space. Fig. 5.9 shows the definition of a metric d_K on continuations.
- The cost accumulator $\alpha \in \mathbb{R}$ is a real number, so I define $(W, \mathcal{W}) \stackrel{\text{def}}{=} (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ as the canonical measurable space on $\mathbb{R}$.

Then the semantic domain is defined as the product measurable space of the four components: $(\Sigma, \mathcal{O}) \stackrel{\text{def}}{=} (V, \mathcal{V}) \otimes (S, \mathcal{S}) \otimes (K, \mathcal{K}) \otimes (W, \mathcal{W})$.

An execution of an Appl program $\langle \mathscr{D}, S_{\mathsf{main}} \rangle$ is initialized with $\langle \lambda\_.0, S_{\mathsf{main}}, \mathbf{Kstop}, 0 \rangle$, and the termination configurations have the form $\langle \_, \mathbf{skip}, \mathbf{Kstop}, \_ \rangle$. Different from a standard semantics where each program configuration steps to at most one new configuration, a probabilistic semantics may pick several different new configurations. The evaluation relation has the form $\sigma \mapsto \mu$ where $\mu \in \mathbb{D}(\Sigma)$ is a probability measure over configurations. Fig. 5.10 presents the rules of the evaluation relation. Note that expressions E and conditions L are deterministic, so I define a standard big-step evaluation relation for them, written $\gamma \vdash E \Downarrow r$ and $\gamma \vdash L \Downarrow b$, where γ is a valuation, $r \in \mathbb{R}$, and $b \in \{\top, \bot\}$. Most of the rules in Fig. 5.10, except (E-Sample) and (E-Prob), are also deterministic as they step to a Dirac measure. The rule (E-Prob) constructs a distribution whose support has exactly two elements, which stand for the two branches of the probabilistic choice. The rule (E-Sample)

$$d_E(x, x) \stackrel{\text{def}}{=} 0$$

$$d_E(c_1, c_2) \stackrel{\text{def}}{=} |c_1 - c_2|$$

$$d_E(E_{11} + E_{12}, E_{21} + E_{22}) \stackrel{\text{def}}{=} d_E(E_{11}, E_{21}) + d_E(E_{12}, E_{22})$$

$$d_E(E_{11} \times E_{12}, E_{21} \times E_{22}) \stackrel{\text{def}}{=} d_E(E_{11}, E_{21}) + d_E(E_{12}, E_{22})$$

$$d_E(E_1, E_2) \stackrel{\text{def}}{=} \infty \text{ otherwise}$$

$$d_L(\textbf{true}, \textbf{true}) \stackrel{\text{def}}{=} 0$$

$$d_L(\textbf{not } L_1, \textbf{not } L_2) \stackrel{\text{def}}{=} d_L(L_1, L_2)$$

$$d_L(L_{11} \textbf{ and } L_{12}, L_{21} \textbf{ and } L_{22}) \stackrel{\text{def}}{=} d_L(L_{11}, L_{21}) + d_L(L_{12}, L_{22})$$

$$d_L(E_{11} \leq E_{12}, E_{21} \leq E_{22}) \stackrel{\text{def}}{=} d_E(E_{11}, E_{21}) + d_E(E_{12}, E_{22})$$

$$d_L(L_1, L_2) \stackrel{\text{def}}{=} \infty \text{ otherwise}$$

$$d_D(\textsc{Uniform}(a_1, b_1), \textsc{Uniform}(a_2, b_2)) \stackrel{\text{def}}{=} |a_1 - a_2| + |b_1 - b_2|$$

$$d_D(D_1, D_2) \stackrel{\text{def}}{=} \infty \text{ otherwise}$$

$$d_S(\textbf{skip}, \textbf{skip}) \stackrel{\text{def}}{=} 0$$

$$d_S(\textbf{tick}(c_1), \textbf{tick}(c_2)) \stackrel{\text{def}}{=} |c_1 - c_2|$$

$$d_S(x \coloneqq E_1, x \coloneqq E_2) \stackrel{\text{def}}{=} d_E(E_1, E_2)$$

$$d_S(x \sim D_1, x \sim D_2) \stackrel{\text{def}}{=} d_D(D_1, D_2)$$

$$d_S(\textbf{call } f, \textbf{call } f) \stackrel{\text{def}}{=} 0$$

$$d_S(\textbf{if prob}(p_1) \textbf{ then } S_{11} \textbf{ else } S_{12} \textbf{ fi}, \textbf{if prob}(p_2) \textbf{ then } S_{21} \textbf{ else } S_{22} \textbf{ fi}) \stackrel{\text{def}}{=} |p_1 - p_2| + d_S(S_{11}, S_{21}) + d_S(S_{12}, S_{22})$$

$$d_S(\textbf{if } L_1 \textbf{ then } S_{11} \textbf{ else } S_{12} \textbf{ fi}, \textbf{if } L_2 \textbf{ then } S_{21} \textbf{ else } S_{22} \textbf{ fi}) \stackrel{\text{def}}{=} d_L(L_1, L_2) + d_S(S_{11}, S_{21}) + d_S(S_{12}, S_{22})$$

$$d_S(\textbf{while } L_1 \textbf{ do } S_1 \textbf{ od}, \textbf{while } L_2 \textbf{ do } S_2 \textbf{ od}) \stackrel{\text{def}}{=} d_L(L_1, L_2) + d_S(S_1, S_2)$$

$$d_S(S_{11}; S_{12}, S_{21}; S_{22}) \stackrel{\text{def}}{=} d_S(S_{11}, S_{21}) + d_S(S_{12}, S_{22})$$

$$d_S(S_1, S_2) \stackrel{\text{def}}{=} \infty \text{ otherwise}$$

$$d_K(\textbf{Kstop}, \textbf{Kstop}) \stackrel{\text{def}}{=} 0$$

$$d_K(\textbf{Kloop } L_1\ S_1\ K_1, \textbf{Kloop } L_2\ S_2\ K_2) \stackrel{\text{def}}{=} d_L(L_1, L_2) + d_S(S_1, S_2) + d_K(K_1, K_2)$$

$$d_K(\textbf{Kseq } S_1\ K_1, \textbf{Kseq } S_2\ K_2) \stackrel{\text{def}}{=} d_S(S_1, S_2) + d_K(K_1, K_2)$$

$$d_K(K_1, K_2) \stackrel{\text{def}}{=} \infty \text{ otherwise}$$

Fig. 5.9: Metrics for expressions, conditions, distributions, statements, and continuations.

"pushes" the probability distribution of D to a distribution over post-sampling program configurations.

Example 5.11. Suppose that a random sampling statement is being executed, i.e., the current configuration is

$$\langle \{t \mapsto t_0\}, (t \sim \textsc{Uniform}(-1, 2)), K_0, \alpha_0 \rangle.$$

The probability measure for the uniform distribution is $\lambda O. \int_O \frac{[-1 \le x \le 2]}{3} dx$. *Thus, by the rule (E-Sample), we derive the post-sampling probability measure over configurations:*

$$\lambda A. \int_{\mathbb{R}} [\langle \{t \mapsto r\}, \textbf{skip}, K_0, \alpha_0 \rangle \in A] \cdot \frac{[-1 \le r \le 2]}{3} dr.$$

5.3.2 A Markov-Chain Semantics

In this section, I harness Markov-chain-based reasoning [80, 114] to develop a Markov-chain cost semantics, based on the evaluation relation $\sigma \mapsto \mu$. An advantage of this approach is that it allows me to study how the cost of every single evaluation step contributes to the accumulated cost at the exit of the program.

First, I prove that the evaluation relation $\mapsto$ can be interpreted as a probability kernel.

Lemma 5.12. *Let* $\gamma : \mathsf{VID} \to \mathbb{R}$ *be a valuation.*

- *Let* E *be an expression. Then there exists a unique* $r \in \mathbb{R}$ *such that* $\gamma \vdash E \Downarrow r$.
- *Let* L *be a condition. Then there exists a unique* $b \in \{\top, \bot\}$ *such that* $\gamma \vdash L \Downarrow b$.

Proof. By induction on the structure of E and L. □

Lemma 5.13. *For every configuration* $\sigma \in \Sigma$, *there exists a unique* $\mu \in \mathbb{D}(\Sigma, O)$ *such that* $\sigma \mapsto \mu$.

Proof. Let $\sigma = \langle \gamma, S, K, \alpha \rangle$. Then by case analysis on the structure of S, followed by a case analysis on the structure of K if $S = \textbf{skip}$. The rest of the proof appeals to Lem. 5.12. □

Theorem 5.14. *The evaluation relation* $\mapsto$ *defines a probability kernel on program configurations.*

Proof. Lem. 5.13 tells me that $\mapsto$ can be seen as a function $\hat{\mapsto}$ defined as follows:

$$\hat{\mapsto}(\sigma, A) \stackrel{\text{def}}{=} \mu(A) \quad \text{where} \quad \sigma \mapsto \mu.$$

It is clear that $\lambda A. \hat{\mapsto}(\sigma, A)$ is a probability measure. On the other hand, to show that $\lambda \sigma. \hat{\mapsto}(\sigma, A)$ is measurable for any $A \in \mathcal{O}$, I need to prove that for $B \in \mathcal{B}(\mathbb{R})$, it holds that $\mathcal{O}(A, B) \stackrel{\text{def}}{=} (\lambda \sigma. \hat{\mapsto}(\sigma, A))^{-1}(B) \in \mathcal{O}$.

$\boxed{\gamma \vdash E \Downarrow r}$ "the expression E evaluates to a real value r under the valuation γ"

$$\text{(E-Var)}\ \frac{\gamma(x) = r}{\gamma \vdash x \Downarrow r} \qquad \text{(E-Const)}\ \frac{}{\gamma \vdash c \Downarrow c} \qquad \text{(E-Add)}\ \frac{\gamma \vdash E_1 \Downarrow r_1 \qquad \gamma \vdash E_2 \Downarrow r_2 \qquad r = r_1 + r_2}{\gamma \vdash E_1 + E_2 \Downarrow r}$$

$$\text{(E-Mul)}\ \frac{\gamma \vdash E_1 \Downarrow r_1 \qquad \gamma \vdash E_2 \Downarrow r_2 \qquad r = r_1 \cdot r_2}{\gamma \vdash E_1 \times E_2 \Downarrow r}$$

$\boxed{\gamma \vdash L \Downarrow b}$ "the condition L evaluates to a Boolean value b under the valuation γ"

$$\text{(E-Top)}\ \frac{}{\gamma \vdash \textbf{true} \Downarrow \top} \qquad \text{(E-Neg)}\ \frac{\gamma \vdash L \Downarrow b}{\gamma \vdash \textbf{not}\ L \Downarrow \neg b} \qquad \text{(E-Conj)}\ \frac{\gamma \vdash L_1 \Downarrow b_1 \qquad \gamma \vdash L_2 \Downarrow b_2}{\gamma \vdash L_1\ \textbf{and}\ L_2 \Downarrow b_1 \wedge b_2} \qquad \text{(E-Le)}\ \frac{\gamma \vdash E_1 \Downarrow r_1 \qquad \gamma \vdash E_2 \Downarrow r_2}{\gamma \vdash E_1 \leq E_2 \Downarrow [r_1 \leq r_2]}$$

$\boxed{\langle \gamma, S, K, \alpha \rangle \mapsto \mu}$ "the configuration $\langle \gamma, S, K, \alpha \rangle$ steps to a probability distribution μ on $\langle \gamma', S', K', \alpha' \rangle$'s"

$$\text{(E-Skip-Stop)}\ \frac{}{\langle \gamma, \textbf{skip}, \textbf{Kstop}, \alpha \rangle \mapsto \delta(\langle \gamma, \textbf{skip}, \textbf{Kstop}, \alpha \rangle)}$$

$$\text{(E-Skip-Loop)}\ \frac{\gamma \vdash L \Downarrow b}{\langle \gamma, \textbf{skip}, \textbf{Kloop}\ S\ L\ K, \alpha \rangle \mapsto [b] \cdot \delta(\langle \gamma, S, \textbf{Kloop}\ S\ L\ K, \alpha \rangle) + [\neg b] \cdot \delta(\langle \gamma, \textbf{skip}, K, \alpha \rangle)}$$

$$\text{(E-Skip-Seq)}\ \frac{}{\langle \gamma, \textbf{skip}, \textbf{Kseq}\ S\ K, \alpha \rangle \mapsto \delta(\langle \gamma, S, K, \alpha \rangle)} \qquad \text{(E-Tick)}\ \frac{}{\langle \gamma, \textbf{tick}(c), K, \alpha \rangle \mapsto \delta(\langle \gamma, \textbf{skip}, K, \alpha + c \rangle)}$$

$$\text{(E-Assign)}\ \frac{\gamma \vdash E \Downarrow r}{\langle \gamma, x := E, K, \alpha \rangle \mapsto \delta(\langle \gamma[x \mapsto r], \textbf{skip}, K, \alpha \rangle)} \qquad \text{(E-Sample)}\ \frac{}{\langle \gamma, x \sim D, K, \alpha \rangle \mapsto \mu_D \gg\!= \lambda r.\delta(\langle \gamma[x \mapsto r], \textbf{skip}, K, \alpha \rangle)}$$

$$\text{(E-Call)}\ \frac{}{\langle \gamma, \textbf{call}\ f, K, \alpha \rangle \mapsto \delta(\langle \gamma, \mathscr{D}(f), K, \alpha \rangle)}$$

$$\text{(E-Prob)}\ \frac{}{\langle \gamma, \textbf{if}\ \textbf{prob}(p)\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}, K, \alpha \rangle \mapsto p \cdot \delta(\langle \gamma, S_1, K, \alpha \rangle) + (1-p) \cdot \delta(\langle \gamma, S_2, K, \alpha \rangle)}$$

$$\text{(E-Cond)}\ \frac{\gamma \vdash L \Downarrow b}{\langle \gamma, \textbf{if}\ L\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}, K, \alpha \rangle \mapsto [b] \cdot \delta(\langle \gamma, S_1, K, \alpha \rangle) + [\neg b] \cdot \delta(\langle \gamma, S_2, K, \alpha \rangle)}$$

$$\text{(E-Loop)}\ \frac{}{\langle \gamma, \textbf{while}\ L\ \textbf{do}\ S\ \textbf{od}, K, \alpha \rangle \mapsto \delta(\langle \gamma, \textbf{skip}, \textbf{Kloop}\ L\ S\ K, \alpha \rangle)} \qquad \text{(E-Seq)}\ \frac{}{\langle \gamma, S_1; S_2, K, \alpha \rangle \mapsto \delta(\langle \gamma, S_1, \textbf{Kseq}\ S_2\ K, \alpha \rangle)}$$

Fig. 5.10: Rules of the operational semantics of Appl.

I introduce *skeletons* of programs to separate real numbers and discrete structures.

$$\begin{aligned}
\hat{S} &::= \textbf{skip} \mid \textbf{tick}(\square_\ell) \mid x \coloneqq \hat{E} \mid x \sim \hat{D} \mid \textbf{call}\ f \mid \textbf{while}\ \hat{L}\ \textbf{do}\ \hat{S}\ \textbf{od} \\
&\quad \mid \textbf{if}\ \textbf{prob}(\square_\ell)\ \textbf{then}\ \hat{S}_1\ \textbf{else}\ \hat{S}_2\ \textbf{fi} \mid \textbf{if}\ \hat{L}\ \textbf{then}\ \hat{S}_1\ \textbf{else}\ \hat{S}_2\ \textbf{fi} \mid \hat{S}_1; \hat{S}_2 \\
\hat{L} &::= \textbf{true} \mid \textbf{not}\ \hat{L} \mid \hat{L}_1\ \textbf{and}\ \hat{L}_2 \mid \hat{E}_1 \le \hat{E}_2 \\
\hat{E} &::= x \mid \square_\ell \mid \hat{E}_1 + \hat{E}_2 \mid \hat{E}_1 \times \hat{E}_2 \\
\hat{D} &::= \textbf{uniform}(\square_{\ell_a}, \square_{\ell_b}) \mid \cdots \\
\hat{K} &::= \textbf{Kstop} \mid \textbf{Kloop}\ \hat{L}\ \hat{S}\ \hat{K} \mid \textbf{Kseq}\ \hat{S}\ \hat{K}
\end{aligned}$$

The *holes* $\square_\ell$ are placeholders for real numbers parameterized by *locations* $\ell \in \mathsf{LOC}$. I assume that the holes in a program structure are always pairwise distinct. Let $\eta : \mathsf{LOC} \to \mathbb{R}$ be a map from holes to real numbers and $\eta(\hat{S})$ (resp., $\eta(\hat{L})$, $\eta(\hat{E})$, $\eta(\hat{D})$, $\eta(\hat{K})$) be the instantiation of a statement (resp., condition, expression, distribution, continuation) skeleton by substituting $\eta(\ell)$ for $\square_\ell$. One important property of skeletons is that the "distance" between any concretizations of two different skeletons is always infinity with respect to the metrics in Fig. 5.9.

Observe that

$$\mathcal{O}(A,B) = \bigcup_{\hat{S},\hat{K}} \mathcal{O}(A,B) \cap \{\langle \gamma, \eta(\hat{S}), \eta(\hat{K}), \alpha\rangle \mid \text{any } \gamma, \alpha, \eta\}$$

and that $\hat{S}, \hat{K}$ are countable families of statement and continuation skeletons. Thus it suffices to prove that every set in the union, which I denote by $\mathcal{O}(A,B) \cap \mathscr{C}(\hat{S},\hat{K})$ later in the proof, is measurable. Note that $\mathscr{C}(\hat{S},\hat{K})$ itself is indeed measurable. Further, the skeletons $\hat{S}$ and $\hat{K}$ are able to determine the evaluation rule for all concretized configurations. Thus I can proceed by a case analysis on the evaluation rules.

To aid the case analysis, I define a deterministic evaluation relation $\overset{\text{det}}{\longmapsto}$ by getting rid of the $\delta(\cdot)$ notations in the rules in Fig. 5.10 except probabilistic ones (E-Sample) and (E-Prob). Obviously, $\overset{\text{det}}{\longmapsto}$ can be interpreted as a measurable function on configurations.

- If the evaluation rule is deterministic, then we have

$$\begin{aligned}
&\mathcal{O}(A,B) \cap \mathscr{C}(\hat{S},\hat{K}) \\
&= \{\sigma \mid \sigma \mapsto \mu, \mu(A) \in B\} \cap \mathscr{C}(\hat{S},\hat{K}) \\
&= \{\sigma \mid \sigma \overset{\text{det}}{\longmapsto} \sigma', [\sigma' \in A] \in B\} \cap \mathscr{C}(\hat{S},\hat{K}) \\
&= \begin{cases}
\mathscr{C}(\hat{S},\hat{K}) & \text{if } \{0,1\} \subseteq B \\
\overset{\text{det}}{\longmapsto}^{-1}(A) \cap \mathscr{C}(\hat{S},\hat{K}) & \text{if } 1 \in B \text{ and } 0 \notin B \\
\overset{\text{det}}{\longmapsto}^{-1}(A^c) \cap \mathscr{C}(\hat{S},\hat{K}) & \text{if } 0 \in B \text{ and } 1 \notin B \\
\emptyset & \text{if } \{0,1\} \cap B = \emptyset.
\end{cases}
\end{aligned}$$

The sets in all the cases are measurable, so is the set $\mathcal{O}(A,B) \cap \mathscr{C}(\hat{S},\hat{K})$.

- (E-Prob): Consider B with the form $(-\infty, t]$ with $t \in \mathbb{R}$. If $t \geq 1$, then $\mathcal{O}(A, B) = \Sigma$. Otherwise, let me assume $t < 1$. Let $\hat{S} = \textbf{if } \textbf{prob}(\square) \textbf{ then } \hat{S}_1 \textbf{ else } \hat{S}_2 \textbf{ fi}$. Then we have

$$\begin{aligned}
&\mathcal{O}(A, B) \cap \mathcal{C}(\hat{S}, \hat{K}) \\
&= \{\sigma \mid \sigma \mapsto \mu, \mu(A) \in B\} \cap \mathcal{C}(\hat{S}, \hat{K}) \\
&= \{\sigma \mid \sigma \mapsto p \cdot \delta(\sigma_1) + (1-p) \cdot \delta(\sigma_2), \\
&\qquad p \cdot [\sigma_1 \in A] + (1-p) \cdot [\sigma_2 \in A] \in B\} \cap \mathcal{C}(\hat{S}, \hat{K}) \\
&= \mathcal{C}(\hat{S}, \hat{K}) \cap \{\langle \gamma, \textbf{if } \textbf{prob}(p) \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, K, \alpha\rangle \mid \\
&\qquad p \cdot [\langle \gamma, S_1, K, \alpha\rangle \in A] + (1-p) \cdot [\langle \gamma, S_2, K, \alpha\rangle \in A] \leq t\} \\
&= \mathcal{C}(\hat{S}, \hat{K}) \cap \\
&\quad \{\langle \gamma, \textbf{if } \textbf{prob}(p) \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, K, \alpha\rangle \mid \\
&\qquad (\langle \gamma, S_1, K, \alpha\rangle \in A, \langle \gamma, S_2, K, \alpha\rangle \notin A, p \leq t) \vee \\
&\qquad (\langle \gamma, S_2, K, \alpha\rangle \in A, \langle \gamma, S_1, K, \alpha\rangle \notin A, 1 - p \leq t)\}.
\end{aligned}$$

  The set above is measurable because A and A^c are measurable, as well as $\{p \leq t\}$ and $\{p \geq 1 - t\}$ are measurable in $\mathbb{R}$.
- (E-Sample): Consider B with the form $(-\infty, t]$ with $t \in \mathbb{R}$. Similar to the previous case, we assume that $t < 1$. Let $\hat{S} = x \sim \textbf{uniform}(\square_{\ell_a}, \square_{\ell_b})$, without loss of generality. Then we have

$$\begin{aligned}
&\mathcal{O}(A, B) \cap \mathcal{C}(\hat{S}, \hat{K}) \\
&= \{\sigma \mid \sigma \mapsto \mu, \mu(A) \in B\} \cap \mathcal{C}(\hat{S}, \hat{K}) \\
&= \{\sigma \mid \sigma \mapsto \mu_D \gg\!= \kappa_\sigma, \int \kappa_\sigma(r)(A)\mu_D(dr) \leq t\} \cap \mathcal{C}(\hat{S}, \hat{K}) \\
&= \mathcal{C}(\hat{S}, \hat{K}) \cap \{\sigma \mid \sigma \mapsto \mu_D \gg\!= \kappa_\sigma, \\
&\qquad \mu_D(\{r \mid \langle \gamma[x \mapsto r], \textbf{skip}, K, \alpha\rangle \in A\}) \leq t\} \\
&= \mathcal{C}(\hat{S}, \hat{K}) \cap \{\langle \gamma, x \sim \textbf{uniform}(a, b), K, \alpha\rangle \mid a < b, \\
&\qquad \mu_{\textbf{uniform}(a,b)}(\{r \mid \langle \gamma[x \mapsto r], \textbf{skip}, K, \alpha\rangle \in A\}) \leq t\},
\end{aligned}$$

  where $\kappa_{\langle \gamma, S, K, \alpha\rangle} \stackrel{\text{def}}{=} \lambda r.\delta(\langle \gamma[x \mapsto r], \textbf{skip}, K, \alpha\rangle)$. For fixed γ, K, α, the set $\{r \mid \langle \gamma[x \mapsto r], \textbf{skip}, K, \alpha\rangle \in A\}$ is measurable in $\mathbb{R}$. For the distributions considered in this paper, there is a sub-probability kernel $\kappa_D : \mathbb{R}^{\text{ar}(D)} \rightsquigarrow \mathbb{R}$. For example, $\kappa_{\textbf{uniform}}(a, b)$ is defined to be $\mu_{\textbf{uniform}(a,b)}$ if $a < b$, or $\mathbf{0}$ otherwise. Therefore, $\lambda(a, b).\kappa_{\textbf{uniform}}(a, b)(\{r \mid \langle \gamma[x \mapsto r], \textbf{skip}, K, \alpha\rangle \in A\})$ is measurable, and its inversion on $(-\infty, t]$ is a measurable set on distribution parameters (a, b). Hence the set above is measurable.

□

I now review a standard mechanism in measure theory that constructs a probability measure by composing probability kernels. If μ is a probability measure on $(S, \mathcal{S})$ and

$\kappa : (S, \mathcal{S}) \rightsquigarrow (T, \mathcal{T})$ is a probability kernel, then I can construct the a probability measure on $(S, \mathcal{S}) \otimes (T, \mathcal{T})$ that captures all transitions from μ through κ: $\mu \otimes \kappa \stackrel{\text{def}}{=} \lambda(A, B). \int_A \kappa(x, B)\mu(dx)$. If μ is a probability measure on $(S_0, \mathcal{S}_0)$ and $\kappa_i : (S_{i-1}, \mathcal{S}_{i-1}) \rightsquigarrow (S_i, \mathcal{S}_i)$ is a probability kernel for $i = 1, \cdots, n$,where $n \in \mathbb{Z}^+$, then I can construct a probability measure on $\bigotimes_{i=0}^{n}(S_i, \mathcal{S}_i)$, i.e., the space of sequences of n transitions by iteratively applying the kernels to μ:

$$\mu \otimes \bigotimes_{i=1}^{0} \kappa_i \stackrel{\text{def}}{=} \mu,$$
$$\mu \otimes \bigotimes_{i=1}^{k+1} \kappa_i \stackrel{\text{def}}{=} (\mu \otimes \bigotimes_{i=1}^{k} \kappa_i) \otimes \kappa_{k+1}, \qquad 0 \leq k < n.$$

Let $(S_i, \mathcal{S}_i)$, $i \in \mathcal{I}$ be a family of measurable spaces. Their product, denoted by $\bigotimes_{i\in\mathcal{I}}(S_i, \mathcal{S}_i) = (\prod_{i\in\mathcal{I}} S_i, \bigotimes_{i\in\mathcal{I}} \mathcal{S}_i)$, is the product space with the smallest σ-algebra such that for each $i \in \mathcal{I}$, the coordinate map π_i is measurable. The theorem below is widely used to construct a probability measures on an infinite product via probability kernels.

Proposition 5.15 (Ionescu-Tulcea). *Let $(S_i, \mathcal{S}_i)$, $i \in \mathbb{Z}^+$ be a sequence of measurable spaces. Let μ_0 be a probability measure on $(S_0, \mathcal{S}_0)$. For each $i \in \mathbb{N}$, let $\kappa_i : \bigotimes_{k=0}^{i-1}(S_k, \mathcal{S}_k) \rightsquigarrow (S_i, \mathcal{S}_i)$ be a probability kernel. Then there exists a sequence of probability measures $\mu_i \stackrel{\text{def}}{=} \mu_0 \otimes \bigotimes_{k=1}^{i} \kappa_k$, $i \in \mathbb{Z}^+$, and there exists a uniquely defined probability measure μ on $\bigotimes_{k=0}^{\infty}(S_k, \mathcal{S}_k)$ such that $\mu_i(A) = \mu(A \times \prod_{k=i+1}^{\infty} S_k)$ for all $i \in \mathbb{Z}^+$ and $A \in \bigotimes_{k=0}^{i} \mathcal{S}_i$.*

Let $(\Omega, \mathcal{F}) \stackrel{\text{def}}{=} \bigotimes_{n=0}^{\infty}(\Sigma, O)$ be a measurable space of infinite traces on program configurations. Let $\{\mathcal{F}_n\}_{n\in\mathbb{Z}^+}$ be a filtration, i.e., an increasing sequence $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}$ of sub-σ-algebras in $\mathcal{F}$, generated by coordinate maps $X_n(\omega) \stackrel{\text{def}}{=} \omega_n$ for $n \in \mathbb{Z}^+$. Let $\langle \mathscr{D}, S_{\mathsf{main}} \rangle$ be an Appl program. Let $\mu_0 \stackrel{\text{def}}{=} \delta(\langle \lambda\_.0, S_{\mathsf{main}}, \mathbf{Kstop}, 0 \rangle)$ be the initial distribution. Let $\mathbb{P}$ be the probability measure on infinite traces induced by Prop. 5.15 and Thm. 5.14. Then $(\Omega, \mathcal{F}, \mathbb{P})$ forms a probability space on *infinite* traces of the program. Intuitively, $\mathbb{P}$ specifies the probability distribution over all possible executions of a probabilistic program. The probability of an assertion θ with respect to $\mathbb{P}$, written $\mathbb{P}[\theta]$, is defined as $\mathbb{P}(\{\omega \mid \theta(\omega) \text{ is true}\})$.

To formulate the accumulated cost at the exit of the program, I define the *stopping time* $T : \Omega \to \mathbb{Z}^+ \cup \{\infty\}$ of a probabilistic program as a random variable on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ of program traces:

$$T(\omega) \stackrel{\text{def}}{=} \inf\{n \in \mathbb{Z}^+ \mid \omega_n = \langle \_, \mathbf{skip}, \mathbf{Kstop}, \_ \rangle\},$$

i.e., $T(\omega)$ is the number of evaluation steps before the trace ω reaches some termination configuration $\langle \_, \mathbf{skip}, \mathbf{Kstop}, \_ \rangle$. I define the accumulated cost $A_T : \Omega \to \mathbb{R}$ with respect to the stopping time T as

$$A_T(\omega) \stackrel{\text{def}}{=} A_{T(\omega)}(\omega),$$

where $A_n : \Omega \to \mathbb{R}$ captures the accumulated cost at the n-th evaluation step for $n \in \mathbb{Z}^+$, which is defined as

$$A_n(\omega) \stackrel{\text{def}}{=} \alpha_n \text{ where } \omega_n = \langle \_, \_, \_, \alpha_n \rangle.$$

The m-th moment of the accumulated cost is given by the expectation $\mathbb{E}[A_T^m]$ with respect to $\mathbb{P}$.

5.3.3 Soundness of the Derivation System

The Expected-Potential Method for Moment Analysis I fix a degree $m \in \mathbb{N}$ and let $\mathcal{M}_{\mathcal{I}}^{(m)}$ be the m-th order moment semiring instantiated with the interval semiring $\mathcal{I}$. I now define $\mathcal{M}_{\mathcal{I}}^{(m)}$-valued expected-potential functions.

Definition 5.16. A measurable map $\phi : \Sigma \to \mathcal{M}_{\mathcal{I}}^{(m)}$ is said to be an *expected-potential function* if

(i) $\phi(\sigma) = \underline{1}$ if $\sigma = \langle \_, \textbf{skip}, \textbf{Kstop}, \_ \rangle$, and

(ii) $\phi(\sigma) \sqsupseteq \mathbb{E}_{\sigma' \sim \mapsto (\sigma)}[\overrightarrow{([(\alpha' - \alpha)^k, (\alpha' - \alpha)^k])_{0 \le k \le m}} \otimes \phi(\sigma')]$ where $\sigma = \langle \_, \_, \_, \alpha \rangle$, $\sigma' = \langle \_, \_, \_, \alpha' \rangle$ for all $\sigma \in \Sigma$.

Intuitively, $\phi(\sigma)$ is an interval bound on the moments for the accumulated cost of the computation that *continues from* the configuration σ. I define $\mathbf{\Phi}_n$ and $\mathbf{Y}_n$, where $n \in \mathbb{Z}^+$, to be $\mathcal{M}_{\mathcal{I}}^{(m)}$-valued random variables on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ of the Markov-chain semantics as

$$\mathbf{\Phi}_n(\omega) \stackrel{\text{def}}{=} \phi(\omega_n), \mathbf{Y}_n(\omega) \stackrel{\text{def}}{=} \overrightarrow{\langle [A_n(\omega)^k, A_n(\omega)^k] \rangle_{0 \le k \le m}} \otimes \mathbf{\Phi}_n(\omega).$$

In the definition of $\mathbf{Y}_n$, I use $\otimes$ to compose the powers of the accumulated cost at step n and the expected potential function that stands for the moments of the accumulated cost for the rest of the computation.

Lemma 5.17. *By the properties of potential functions, we can prove that* $\mathbb{E}[\mathbf{Y}_{n+1} \mid \mathbf{Y}_n] \sqsubseteq \mathbf{Y}_n$ *almost surely, for all* $n \in \mathbb{Z}^+$.

To prove Lem. 5.17, I first investigate several properties of the interval-valued moment semiring. I show that $\otimes$ and $\oplus$ are monotone if the operations of the underlying semiring are monotone.

Lemma 5.18. *Let* $\mathcal{R} = (|\mathcal{R}|, \le, +, \cdot, 0, 1)$ *be a partially ordered semiring. If* $+$ *and* $\cdot$ *are monotone with respect to* $\le$*, then* $\otimes$ *and* $\oplus$ *in the moment semiring* $\mathcal{M}_{\mathcal{R}}^{(m)}$ *are also monotone with respect to* $\sqsubseteq$.

Proof. It is straightforward to show $\oplus$ is monotone. For the rest of the proof, without loss of generality, we show that $\overrightarrow{\langle u_k \rangle_{0 \le k \le m}} \otimes \overrightarrow{\langle v_k \rangle_{0 \le k \le m}} \sqsubseteq \overrightarrow{\langle u_k \rangle_{0 \le k \le m}} \otimes \overrightarrow{\langle w_k \rangle_{0 \le k \le m}}$ if $\overrightarrow{\langle v_k \rangle_{0 \le k \le m}} \sqsubseteq$

$\overrightarrow{\langle w_k\rangle_{0\le k\le m}}$. By the definition of $\sqsubseteq$, we know that $v_k \le w_k$ for all $k = 0, 1, \cdots, m$. Then for each k, we have

$$\begin{aligned}
&(\overrightarrow{\langle u_k\rangle_{0\le k\le m}} \otimes \overrightarrow{\langle v_k\rangle_{0\le k\le m}})_k \\
=& \sum_{i=0}^{k} \binom{k}{i} \times (u_i \cdot v_{k-i}) \\
\le& \sum_{i=0}^{k} \binom{k}{i} \times (u_i \cdot w_{k-i}) \\
=& (\overrightarrow{\langle u_k\rangle_{0\le k\le m}} \otimes \overrightarrow{\langle w_k\rangle_{0\le k\le m}})_k.
\end{aligned}$$

Then we conclude by the definition of $\sqsubseteq$. □

As I allow potential functions to be *interval*-valued, I show that the interval semiring $\mathcal{I}$ satisfies the monotonicity required in Lem. 5.18.

LEMMA 5.19. *The operations $+_{\mathcal{I}}$ and $\cdot_{\mathcal{I}}$ are monotone with respect to $\le_{\mathcal{I}}$.*

PROOF. It is straightforward to show $+_{\mathcal{I}}$ is monotone. For the rest of the proof, it suffices to show that $[a, b] \cdot_{\mathcal{I}} [c, d] \le_{\mathcal{I}} [a', b'] \cdot_{\mathcal{I}} [c, d]$ if $[a, b] \le_{\mathcal{I}} [a', b']$, i.e., $[a, b] \subseteq [a', b']$ or $a \ge a', b \le b'$.

We claim that $\min S_{a,b,c,d} \ge \min S_{a',b',c,d}$, i.e., $\min\{ac, ad, bc, bd\} \ge \min\{a'c, a'd, b'c, b'd\}$.

- If $0 \le c \le d$: Then $ac \le bc$, $ad \le bd$, $a'c \le b'c$, $a'd \le b'd$. It then suffices to show that $\min\{ac, ad\} \ge \min\{a'c, a'd\}$. Because $d \ge c \ge 0$ and $a \ge a'$, we conclude that $ac \ge a'c$ and $ad \ge a'd$.
- If $c < 0 \le d$: Then $ac \ge bc$, $ad \le bd$, $a'c \ge b'c$, $a'd \ge b'd$. It then suffices to show that $\min\{bc, ad\} \ge \min\{b'c, a'd\}$. Because $d \ge 0 > c$ and $a \ge a', b \le b'$, we conclude that $bc \ge b'c$ and $ad \le a'd$.
- If $c \le d < 0$: Then $ac \ge bc$, $ad \ge bd$, $a'c \ge b'c$, $a'd \ge b'd$. It then suffices to show that $\min\{bc, bd\} \ge \min\{b'c, b'd\}$. Because $0 > d \ge c$ and $b \le b'$, we conclude that $bc \ge b'c$ and $bd \ge b'd$.

In a similar way, we can also prove that $\max S_{a,b,c,d} \le \max S_{a',b',c,d}$. Therefore, we show that $\cdot_{\mathcal{I}}$ is monotone. □

LEMMA 5.20. *If $\{[a_n, b_n]\}_{n\in\mathbb{Z}^+}$ is a montone sequence in $\mathcal{I}$, i.e., $[a_0, b_0] \le_{\mathcal{I}} [a_1, b_1] \le_{\mathcal{I}} \cdots \le_{\mathcal{I}} [a_n, b_n] \le_{\mathcal{I}} \cdots$, and $[a_n, b_n] \le_{\mathcal{I}} [c, d]$ for all $n \in \mathbb{Z}^+$. Let $[a, b] = \lim_{n\to\infty}[a_n, b_n]$ (the limit is well-defined by the monotone convergence theorem for series). Then $[a, b] \le_{\mathcal{I}} [c, d]$.*

PROOF. By the definition of $\le_{\mathcal{I}}$, we know that $\{a_n\}_{n\in\mathbb{Z}^+}$ is non-increasing and $\{b_n\}_{n\in\mathbb{Z}^+}$ is non-decreasing. Because $a_n \ge c$ for all $n \in \mathbb{Z}^+$, we conclude that $\lim_{n\to\infty} a_n \ge c$.

Because $b_n \leq d$ for all $n \in \mathbb{Z}^+$, we conclude that $\lim_{n\to\infty} b_n \leq d$. Thus we conclude that $[a,b] \leq_{\mathcal{I}} [c,d]$. □

LEMMA 5.21. *Let $X, Y : \Omega \to \mathcal{M}_{\mathcal{I}}^{(m)}$ be integrable. Then $\mathbb{E}[X \oplus Y] = \mathbb{E}[X] \oplus \mathbb{E}[Y]$.*

PROOF. Appeal to linearity of expectations and the fact that $\oplus$ is the pointwise extension of $+_{\mathcal{I}}$, as well as $+_{\mathcal{I}}$ is the pointwise extension of numeric addition. □

LEMMA 5.22. *If $X : \Omega \to \mathcal{M}_{\mathcal{I}}^{(m)}$ is $\mathcal{G}$-measurable and bounded, a.s., as well as $X(\omega) = \overrightarrow{\langle [a_k(\omega), a_k(\omega)] \rangle_{0\leq k\leq m}}$ for all $\omega \in \Omega$, then $\mathbb{E}[X \otimes Y \mid \mathcal{G}] = X \otimes \mathbb{E}[Y \mid \mathcal{G}]$ almost surely.*

PROOF. Fix $\omega \in \Omega$. Let $Y(\omega) = \overrightarrow{\langle [b_k(\omega), c_k(\omega)] \rangle_{0\leq k\leq m}}$. Then we have

$$\begin{aligned}
\mathbb{E}[X \otimes Y \mid \mathcal{G}](\omega) &= \mathbb{E}[\overrightarrow{\langle [a_k, a_k] \rangle_{0\leq k\leq m}} \otimes \overrightarrow{\langle [b_k, c_k] \rangle_{0\leq k\leq m}} \mid \mathcal{G}](\omega) \\
&= \mathbb{E}[\overrightarrow{\langle \textstyle\sum_{\mathcal{I}}{}_{i=0}^{k} \binom{k}{i} \times_{\mathcal{I}} ([a_i, a_i] \cdot_{\mathcal{I}} [b_{k-i}, c_{k-i}]) \rangle_{0\leq k\leq m}} \mid \mathcal{G}](\omega) \\
&= \overrightarrow{\langle \mathbb{E}[\textstyle\sum_{\mathcal{I}}{}_{i=0}^{k} \binom{k}{i} \times_{\mathcal{I}} ([a_i, a_i] \cdot_{\mathcal{I}} [b_{k-i}, c_{k-i}]) \mid \mathcal{G}](\omega) \rangle_{0\leq k\leq m}} \\
&= \overrightarrow{\langle \textstyle\sum_{\mathcal{I}}{}_{i=0}^{k} \binom{k}{i} \times_{\mathcal{I}} \mathbb{E}[[a_i, a_i] \cdot_{\mathcal{I}} [b_{k-i}, c_{k-i}] \mid \mathcal{G}](\omega) \rangle_{0\leq k\leq m}}.
\end{aligned}$$

On the other hand, we have

$$\begin{aligned}
X(\omega) \otimes \mathbb{E}[Y \mid \mathcal{G}](\omega) &= \langle X_0(\omega), \cdots, X_m(\omega) \rangle \otimes \mathbb{E}[\langle Y_0, \cdots, Y_m \rangle \mid \mathcal{G}](\omega) \\
&= \langle X_0(\omega), \cdots, X_m(\omega) \rangle \otimes \langle \mathbb{E}[Y_0 \mid \mathcal{G}](\omega), \cdots, \mathbb{E}[Y_m \mid \mathcal{G}](\omega) \rangle \\
&= \overrightarrow{\langle \textstyle\sum_{\mathcal{I}}{}_{i=0}^{k} \binom{k}{i} \times_{\mathcal{I}} (X_i(\omega) \cdot_{\mathcal{I}} \mathbb{E}[Y_{k-i} \mid \mathcal{G}](\omega)) \rangle_{0\leq k\leq m}} \\
&= \overrightarrow{\langle \textstyle\sum_{\mathcal{I}}{}_{i=0}^{k} \binom{k}{i} \times_{\mathcal{I}} ([a_i(\omega), a_i(\omega)] \cdot_{\mathcal{I}} \mathbb{E}[[b_{k-i}, c_{k-i}] \mid \mathcal{G}](\omega)) \rangle_{0\leq k\leq m}}.
\end{aligned}$$

Thus, it suffices to show that for each i, $\mathbb{E}[[a_i, a_i] \cdot_{\mathcal{I}} [b_{k-i}, c_{k-i}] \mid \mathcal{G}] = [a_i, a_i] \cdot_{\mathcal{I}} \mathbb{E}[[b_{k-i}, c_{k-i}] \mid \mathcal{G}]$ almost surely.

For ω such that $a_i(\omega) \geq 0$:

$$\begin{aligned}
\mathbb{E}[[a_i, a_i] \cdot_{\mathcal{I}} [b_{k-i}, c_{k-i}] \mid \mathcal{G}](\omega) &= \mathbb{E}[[a_i b_{k-i}, a_i c_{k-i}] \mid \mathcal{G}](\omega) \\
&= [\mathbb{E}[a_i b_{k-i} \mid \mathcal{G}](\omega), \mathbb{E}[a_i c_{k-i} \mid \mathcal{G}](\omega)] \\
&= [a_i(\omega) \cdot \mathbb{E}[b_{k-i} \mid \mathcal{G}](\omega), a_i(\omega) \cdot \mathbb{E}[c_{k-i} \mid \mathcal{G}](\omega)], a.s., \\
&= [a_i(\omega), a_i(\omega)] \cdot_{\mathcal{I}} \mathbb{E}[[b_{k-i}, c_{k-i}] \mid \mathcal{G}](\omega).
\end{aligned}$$

For ω such that $a_i(\omega) < 0$:

$$\begin{aligned}
\mathbb{E}[[a_i, a_i] \cdot_{\mathcal{I}} [b_{k-i}, c_{k-i}] \mid \mathcal{G}](\omega) &= \mathbb{E}[[a_i c_{k-i}, a_i b_{k-i}] \mid \mathcal{G}](\omega) \\
&= [\mathbb{E}[a_i c_{k-i} \mid \mathcal{G}](\omega), \mathbb{E}[a_i b_{k-i} \mid \mathcal{G}](\omega)] \\
&= [a_i(\omega) \cdot \mathbb{E}[c_{k-i} \mid \mathcal{G}](\omega), a_i(\omega) \cdot \mathbb{E}[b_{k-i} \mid \mathcal{G}](\omega)], a.s., \\
&= [a_i(\omega), a_i(\omega)] \cdot_{\mathcal{I}} \mathbb{E}[[b_{k-i}, c_{k-i}] \mid \mathcal{G}](\omega).
\end{aligned}$$

□

PROOF OF LEM. 5.17. A sequence of random variables $\{X_n\}_{n\in\mathbb{Z}^+}$ is said to be *adapted* to a filtration $\{\mathcal{F}_n\}_{n\in\mathbb{Z}^+}$ if for each $n \in \mathbb{Z}^+$, X_n is $\mathcal{F}_n$-measurable. Then $\{\Phi_n\}_{n\in\mathbb{Z}^+}$ and $\{A_n\}_{n\in\mathbb{Z}^+}$ are adapted to the coordinate-generated filtration $\{\mathcal{F}_n\}_{n\in\mathbb{Z}^+}$ as $\Phi_n(\omega)$ and $A_n(\omega)$ depend on ω_n. By the property of the operational semantics, we know that $\alpha_n(\omega) \le C \cdot n$ almost surely for some $C \ge 0$. Then using Lem. 5.22, we have

$$\begin{aligned}
\mathbb{E}[Y_{n+1} \mid \mathcal{F}_n](\omega) &= \mathbb{E}[A_{n+1} \otimes \Phi_{n+1} \mid \mathcal{F}_n](\omega) \\
&= \mathbb{E}[A_n \otimes \overrightarrow{\langle [(\alpha_{n+1} - \alpha_n)^k, (\alpha_{n+1} - \alpha_n)^k] \rangle_{0\le k\le m}} \otimes \Phi_{n+1} \mid \mathcal{F}_n](\omega) \\
&= A_n(\omega) \otimes \mathbb{E}[\overrightarrow{\langle [(\alpha_{n+1} - \alpha_n)^k, (\alpha_{n+1} - \alpha_n)^k] \rangle_{0\le k\le m}} \otimes \Phi_{n+1} \mid \mathcal{F}_n](\omega), a.s., \\
&= A_n(\omega) \otimes \mathbb{E}[\overrightarrow{\langle [(\alpha_{n+1} - \alpha_n)^k, (\alpha_{n+1} - \alpha_n)^k] \rangle_{0\le k\le m}} \otimes \phi(\omega_{n+1}) \mid \mathcal{F}_n].
\end{aligned}$$

Recall the property of the expected-potential function ϕ in Defn. 5.16. Then by Lem. 5.18 with Lem. 5.19, we have

$$\begin{aligned}
\mathbb{E}[Y_{n+1} \mid \mathcal{F}_n](\omega) &\sqsubseteq A_n(\omega) \otimes \phi(\omega_n), a.s., \\
&= A_n(\omega) \otimes \Phi_n(\omega) \\
&= Y_n.
\end{aligned}$$

As a corollary, we have $\mathbb{E}[Y_n] \sqsubseteq \mathbb{E}[Y_0]$ for all $n \in \mathbb{Z}^+$. □

I call $\{\mathbf{Y}_n\}_{n\in\mathbb{Z}^+}$ a *moment invariant*. My goal is to establish that $\mathbb{E}[\mathbf{Y}_T] \sqsubseteq \mathbb{E}[\mathbf{Y}_0]$, i.e., the initial interval-valued potential $\mathbb{E}[\mathbf{Y}_0] = \mathbb{E}[\underline{1} \otimes \mathbf{\Phi}_0] = \mathbb{E}[\mathbf{\Phi}_0]$ brackets the higher moments of the accumulated cost $\mathbb{E}[\mathbf{Y}_T] = \mathbb{E}[\overrightarrow{\langle [A_T^k, A_T^k] \rangle_{0\le k\le m}} \otimes \underline{1}] = \overrightarrow{\langle [\mathbb{E}[A_T^k], \mathbb{E}[A_T^k]] \rangle_{0\le k\le m}}$.

Soundness The soundness of the derivation system is proved with respect to the Markov-chain semantics. Let $\|\overrightarrow{\langle [a_k, b_k] \rangle_{0\le k\le m}}\|_\infty \stackrel{\text{def}}{=} \max_{0\le k\le m}\{\max\{|a_k|, |b_k|\}\}$.

THEOREM 5.23. *Let $\langle \mathscr{D}, S_{\mathsf{main}} \rangle$ be a probabilistic program. Suppose $\Delta \vdash \{\Gamma; Q\}\ S_{\mathsf{main}}\ \{\Gamma'; \underline{1}\}$, where $Q \in \mathcal{M}_{\mathcal{P}_I}^{(m)}$ and the ends of the k-th interval in Q are polynomials in $\mathbb{R}_{kd}[\mathsf{VID}]$. Let $\{\mathbf{Y}_n\}_{n\in\mathbb{Z}^+}$ be the moment invariant extracted from the Markov-chain semantics with respect to the derivation of $\Delta \vdash \{\Gamma; Q\}\ S_{\mathsf{main}}\ \{\Gamma'; \underline{1}\}$. If the following conditions hold:*

(i) $\mathbb{E}[T^{md}] < \infty$, *and*

(ii) there exists $C \ge 0$ such that for all $n \in \mathbb{Z}^+$, $\|\mathbf{Y}_n\|_\infty \le C \cdot (n+1)^{md}$ almost surely,

Then $\overrightarrow{\langle [\mathbb{E}[A_T^k], \mathbb{E}[A_T^k]] \rangle_{0\le k\le m}} \sqsubseteq \phi_Q(\lambda\_.0)$.

The intuitive meaning of $\overrightarrow{\langle [\mathbb{E}[A_T^k], \mathbb{E}[A_T^k]] \rangle_{0\le k\le m}} \sqsubseteq \phi_Q(\lambda\_.0)$ is that the moment $\mathbb{E}[A_T^k]$ of the accumulated cost upon program termination is bounded by the interval in the k^{th}-moment component of $\phi_Q(\lambda\_.0)$, where Q is the quantitative context and $\lambda\_.0$ is the initial state.

As I discussed in §5.1.2 and Ex. 5.7, the expected-potential method is *not* always sound for deriving bounds on higher moments for cost accumulators in probabilistic programs. The extra conditions Thm. 5.23(i) and (ii) impose constraints on the analyzed program and the expected-potential function, which allow me to reduce the soundness to the *optional stopping problem* from probability theory.

Optional Stopping Let me represent the moment invariant $\{\mathbf{Y}_n\}_{n\in\mathbb{Z}^+}$ as

$$\{\langle [L_n^{(0)}, U_n^{(0)}], [L_n^{(1)}, U_n^{(1)}], \cdots, [L_n^{(m)}, U_n^{(m)}]\rangle\}_{n\in\mathbb{Z}^+},$$

where $L_n^{(k)}, U_n^{(k)} : \Omega \to \mathbb{R}$ are real-valued random variables on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ of the Markov-chain semantics, for $n \in \mathbb{Z}^+$, $0 \le k \le m$. I then have the observations below as direct corollaries of Lem. 5.17:

- For any k, the sequence $\{U_n^{(k)}\}_{n\in\mathbb{Z}^+}$ satisfies $\mathbb{E}[U_{n+1}^{(k)} \mid U_n^{(k)}] \le U_n^{(k)}$ almost surely, for all $n \in \mathbb{Z}^+$, and we want to find sufficient conditions for $\mathbb{E}[U_T^{(k)}] \le \mathbb{E}[U_0^{(k)}]$.
- For any k, the sequence $\{L_n^{(k)}\}_{n\in\mathbb{Z}^+}$ satisfies $\mathbb{E}[L_{n+1}^{(k)} \mid L_n^{(k)}] \ge L_n^{(k)}$ almost surely, for all $n \in \mathbb{Z}^+$, and we want to find sufficient conditions for $\mathbb{E}[L_T^{(k)}] \ge \mathbb{E}[L_0^{(k)}]$.

These kinds of questions can be reduced to *optional stopping problem* from probability theory. Recent research [6, 62, 123, 137] has used and extended the *Optional Stopping Theorem* (OST) from probability theory to establish *sufficient* conditions for the soundness for analysis of probabilistic programs. However, the classic OST turns out to be *not* suitable for higher-moment analysis. I extend OST with a *new* sufficient condition that allows me to prove Thm. 5.23. I first review an important convergence theorem for series of random variables.

Proposition 5.24 (Dominated convergence theorem). *If $\{f_n\}_{n\in\mathbb{Z}^+}$ is a sequence of measurable functions on a measure space $(S, \mathcal{S}, \mu)$, $\{f_n\}_{n\in\mathbb{Z}^+}$ converges to f pointwise, and $\{f_n\}_{n\in\mathbb{Z}^+}$ is dominated by a nonnegative integrable function g (i.e., $|f_n(x)| \le g(x)$ for all $n \in \mathbb{Z}^+$, $x \in S$), then f is integrable and $\lim_{n\to\infty} \mu(f_n) = \mu(f)$.*

Further, the theorem still holds if f is chosen as a measurable function and "$\{f_n\}_{n\in\mathbb{Z}^+}$ converges to f pointwise and is dominated by g" holds almost everywhere.

Now I prove the following extension of OST to deal with interval-valued potential functions.

Theorem 5.25. *If $\mathbb{E}[\|Y_n\|_\infty] < \infty$ for all $n \in \mathbb{Z}^+$, then $\mathbb{E}[Y_T]$ exists and $\mathbb{E}[Y_T] \sqsubseteq \mathbb{E}[Y_0]$ in the following situation:*

There exist $\ell \in \mathbb{N}$ and $C \ge 0$ such that $\mathbb{E}[T^\ell] < \infty$ and for all $n \in \mathbb{Z}^+$, $\|Y_n\|_\infty \le C \cdot (n+1)^\ell$ almost surely.

Proof. By $\mathbb{E}[T^\ell] < \infty$ where $\ell \ge 1$, we know that $\mathbb{P}[T < \infty] = 1$. By the property of the operational semantics, for $\omega \in \Omega$ such that $T(\omega) < \infty$, we have $Y_n(\omega) = Y_T(\omega)$ for all

$n \geq T(\omega)$. Then we have

$$\begin{aligned}
\mathbb{P}[\lim_{n\to\infty} Y_n = Y_T] &= \mathbb{P}(\{\omega \mid \lim_{n\to\infty} Y_n(\omega) = Y_T(\omega)\}) \\
&\geq \mathbb{P}(\{\omega \mid \lim_{n\to\infty} Y_n(\omega) = Y_T(\omega) \wedge T(\omega) < \infty\}) \\
&= \mathbb{P}(\{\omega \mid Y_{T(\omega)}(\omega) = Y_T(\omega) \wedge T(\omega) < \infty\}) \\
&= \mathbb{P}(\{\omega \mid T(\omega) < \infty\}) \\
&= 1.
\end{aligned}$$

On the other hand, $Y_n(\omega)$ can be treated as a vector of real numbers. Let $a_n : \Omega \to \mathbb{R}$ be a real-valued component in Y_n. Because $\mathbb{E}[\|Y_n\|_\infty] < \infty$ and $\|Y_n\|_\infty \leq C \cdot (n+1)^\ell$ almost surely, we know that $\mathbb{E}[|a_n|] \leq \mathbb{E}[\|Y_n\|_\infty] < \infty$ and $|a_n| \leq \|Y_n\|_\infty \leq C \cdot (n+1)^\ell$ almost surely. Therefore,

$$|a_n| = |a_{\min(T,n)}| \leq C \cdot (\min(T,n)+1)^\ell \leq C \cdot (T+1)^\ell, \text{a.s.}$$

Recall that $\mathbb{E}[T^\ell] < \infty$. Then $\mathbb{E}[(T+1)^\ell] = \mathbb{E}[T^\ell + O(T^{\ell-1})] < \infty$. By Prop. 5.24, with the function g set to $\lambda\omega.C \cdot (T(\omega)+1)^\ell$, we know that $\lim_{n\to\infty} \mathbb{E}[a_n] = \mathbb{E}[a_T]$. Because a_n is an arbitrary real-valued component in Y_n, we know that $\lim_{n\to\infty} \mathbb{E}[Y_n] = \mathbb{E}[Y_T]$. By Lem. 5.17, we know that $\mathbb{E}[Y_n] \sqsubseteq \mathbb{E}[Y_0]$ for all $n \in \mathbb{Z}^+$. By Lem. 5.20, we conclude that $\lim_{n\to\infty} \mathbb{E}[Y_n] \sqsubseteq \mathbb{E}[Y_0]$, i.e., $\mathbb{E}[Y_T] \sqsubseteq \mathbb{E}[Y_0]$. □

Proof of Thm. 5.23 To reduce the soundness proof to the extended OST for interval-valued bounds, I construct an *annotated transition kernel* from validity judgements $\vdash \Delta$ and $\Delta \vdash \{\Gamma; Q\}\ S_{\mathsf{main}}\ \{\Gamma'; Q'\}$. Before proceeding to the proof, I extend the derivation system with rules for program configurations and include them in Fig. 5.11. Moreover, I use a more declarative rule shown below for function calls that merges the two rules for function calls in Fig. 5.6 into one:

$$\frac{\forall i\colon (\Gamma; Q_i, \Gamma'; Q_i') \in \Delta(f)}{\Delta \vdash \{\Gamma; \bigoplus_i Q_i\}\ \textbf{call}\ f\ \{\Gamma'; \bigoplus_i Q_i'\}}\ \text{(Q-Call)}$$

Lemma 5.26. *Suppose* $\vdash \Delta$ *and* $\Delta \vdash \{\Gamma; Q\}\ S_{\mathsf{main}}\ \{\Gamma'; Q'\}$. *An* annotated program configuration *has the form* $\langle \Gamma, Q, \gamma, S, K, \alpha\rangle$ *such that* $\Delta \vdash \{\Gamma; Q\}\ \langle \gamma, S, K, \alpha\rangle$. *Then there exists a probability kernel* κ *over annotated program configurations such that:*

For all $\sigma = \langle \Gamma, Q, \gamma, S, K, \alpha\rangle \in \mathrm{dom}(\kappa)$, *it holds that*

(i) κ *is the same as the evaluation relation* $\mapsto$ *if the annotations are omitted, i.e.,*

$$\kappa(\sigma) \gg\!= \lambda\langle \_, \_, \gamma', S', K', \alpha'\rangle.\delta(\langle \gamma', S', K', \alpha'\rangle) = \mapsto(\langle \gamma, S, K, \alpha\rangle),$$

and

$$\frac{\gamma \models \Gamma \qquad \Delta \vdash \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\} \qquad \Delta \vdash \{\Gamma'; Q'\}\ K}{\Delta \vdash \{\Gamma; Q\}\ \langle \gamma, S, K, \alpha \rangle}\ (\textsc{Valid-Cfg}) \qquad \frac{}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{Kstop}}\ (\textsc{QK-Stop})$$

$$\frac{\Delta \vdash \{\Gamma \wedge L; Q\}\ S\ \{\Gamma; Q\} \qquad \Delta \vdash \{\Gamma \wedge \neg L; Q\}\ K}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{Kloop}\ L\ S\ K}\ (\textsc{QK-Loop}) \qquad \frac{\Delta \vdash \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\} \qquad \Delta \vdash \{\Gamma'; Q'\}\ K}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{Kseq}\ S\ K}\ (\textsc{QK-Seq})$$

$$\frac{\Delta \vdash \{\Gamma'; Q'\}\ K \qquad \Gamma \models \Gamma' \qquad \Gamma \models Q \sqsupseteq Q'}{\Delta \vdash \{\Gamma; Q\}\ K}\ (\textsc{QK-Weaken})$$

Fig. 5.11: Extra inference rules of the derivation system.

(ii) $\phi_Q(\gamma) \quad \sqsupseteq \quad \mathbb{E}_{\sigma' \sim \kappa(\sigma)}[\overrightarrow{\langle [(\alpha' - \alpha)^k, (\alpha' - \alpha)^k] \rangle_{0 \le k \le m}} \otimes \phi_{Q'}(\gamma')]$ *where* $\sigma' = \langle \_, Q', \gamma', \_, \_, \alpha' \rangle$.

Before proving the soundness, I show that the derivation system for bound inference admits a *relaxation* rule.

Lemma 5.27. *Suppose that* $\vdash \Delta$. *If* $\Delta \vdash \{\Gamma; Q_1\}\ S\ \{\Gamma'; Q_1'\}$ *and* $\Delta \vdash \{\Gamma; Q_2\}\ S\ \{\Gamma'; Q_2'\}$, *then the judgment* $\Delta \vdash \{\Gamma; Q_1 \oplus Q_2\}\ S\ \{\Gamma'; Q_1' \oplus Q_2'\}$ *is derivable.*

Proof. By nested induction on the derivation of the judgment $\Delta \vdash \{\Gamma; Q_1\}\ S\ \{\Gamma'; Q_1'\}$, followed by inversion on $\Delta \vdash \{\Gamma; Q_2\}\ S\ \{\Gamma'; Q_2'\}$. We assume the derivations have the same shape and the same logical contexts; in practice, we can ensure this by explicitly inserting weakening positions, e.g., all the branching points, and by doing a first pass to obtain logical contexts.

- $\dfrac{}{\Delta \vdash \{\Gamma; Q_1\}\ \mathbf{skip}\ \{\Gamma; Q_1\}}$ (Q-Skip)
  By inversion, we have $Q_2 = Q_2'$. By (Q-Skip), we immediately have $\Delta \vdash \{\Gamma; Q_1 \oplus Q_2\}\ \mathbf{skip}\ \{\Gamma; Q_1 \oplus Q_2\}$.
- $\dfrac{Q_1 = \overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes Q_1'}{\Delta \vdash \{\Gamma; Q_1\}\ \mathbf{tick}(c)\ \{\Gamma; Q_1'\}}$ (Q-Tick)
  By inversion, we have $Q_2 = \overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes Q_2'$. By distributivity, we have $\overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes (Q_1' \oplus Q_2') = (\overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes Q_1') \oplus (\overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \oplus Q_2') = Q_1 \oplus Q_2$. Then we conclude by (Q-Tick).
- $\dfrac{\Gamma = [E/x]\Gamma' \qquad Q_1 = [E/x]Q_1'}{\Delta \vdash \{\Gamma; Q_1\}\ x \coloneqq E\ \{\Gamma'; Q_1'\}}$ (Q-Assign)

By inversion, we have $Q_2 = [E/x]Q_2'$. Then we know that $[E/x](Q_1' \oplus Q_2') = [E/x]Q_1' \oplus [E/x]Q_2' = Q_1 \oplus Q_2$. Then we conclude by (Q-Assign).

- (Q-Sample)
$$\frac{\Gamma = \forall x \in \mathrm{supp}(\mu_D) : \Gamma' \qquad Q_1 = \mathbb{E}_{x\sim\mu_D}[Q_1']}{\Delta \vdash \{\Gamma; Q_1\}\ x \sim D\ \{\Gamma'; Q_1'\}}$$
By inversion, we have $Q_2 = \mathbb{E}_{x\sim\mu_D}[Q_2']$. By Lem. 5.21, we know that $\mathbb{E}_{x\sim\mu_D}[Q_1' \oplus Q_2'] = \mathbb{E}_{x\sim\mu_D}[Q_1'] \oplus \mathbb{E}_{x\sim\mu_D}[Q_2'] = Q_1 \oplus Q_2$. Then we conclude by (Q-Sample).

- (Q-Call)
$$\frac{\forall i : (\Gamma; Q_{1i}, \Gamma'; Q_{1i}') \in \Delta(f)}{\Delta \vdash \{\Gamma; \bigoplus_i Q_{1i}\}\ \textbf{call}\ f\ \{\Gamma'; \bigoplus_i Q_{1i}'\}}$$
By inversion, $Q_2 = \bigoplus_j Q_{2j}$ and $Q_2' = \bigoplus_j Q_{2j}'$ where $(\Gamma; Q_{2j}, \Gamma'; Q_{2j}') \in \Delta(f)$ for each j. Then by (Q-Call), I have $\Delta \vdash \{\Gamma; \bigoplus_i Q_{1i} \oplus \bigoplus_j Q_{2j}\}\ \textbf{call}\ f\ \{\Gamma'; \bigoplus_i Q_{1i}' \oplus \bigoplus_j Q_{2j}'\}$.

- (Q-Prob)
$$\frac{\Delta \vdash \{\Gamma; Q_{11}\}\ S_1\ \{\Gamma'; Q_1'\} \qquad \Delta \vdash \{\Gamma; Q_{12}\}\ S_2\ \{\Gamma'; Q_1'\} \qquad Q_1 = P_1 \oplus R_1 \qquad P_1 = \langle [p,p], [0,0], \cdots, [0,0]\rangle \otimes Q_{11} \qquad R_1 = \langle [1-p, 1-p], [0,0], \cdots, [0,0]\rangle \otimes Q_{12}}{\Delta \vdash \{\Gamma; Q_1\}\ \textbf{if}\ \textbf{prob}(p)\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{\Gamma'; Q_1'\}}$$
By inversion, we know that $Q_2 = P_2 \oplus R_2$ for some Q_{21}, Q_{22} such that $\Delta \vdash \{\Gamma; Q_{21}\}\ S_1\ \{\Gamma'; Q_2'\}$, $\Delta \vdash \{\Gamma; Q_{22}\}\ S_2\ \{\Gamma'; Q_2'\}$, $P_2 = \langle [p,p], [0,0], \cdots, [0,0]\rangle \otimes Q_{21}$, and $R_2 = \langle [1-p, 1-p], [0,0], \cdots, [0,0]\rangle \otimes Q_{22}$. By induction hypothesis, we have $\Delta \vdash \{\Gamma; Q_{11} \oplus Q_{21}\}\ S_1\ \{\Gamma'; Q_1' \oplus Q_2'\}$ and $\Delta \vdash \{\Gamma; Q_{12} \oplus Q_{22}\}\ S_2\ \{\Gamma'; Q_1' \oplus Q_2'\}$. Then

$$\begin{aligned}
&\langle [p,p], \cdots, [0,0]\rangle \otimes (Q_{11} \oplus Q_{21}) \\
&= (\langle [p,p], \cdots, [0,0]\rangle \otimes Q_{11}) \oplus (\langle [p,p], \cdots, [0,0]\rangle \oplus Q_{21}) \\
&= P_1 \oplus P_2, \\
&\langle [1-p, 1-p], \cdots, [0,0]\rangle \otimes (Q_{12} \oplus Q_{22}) \\
&= (\langle [1-p, 1-p], \cdots, [0,0]\rangle \otimes Q_{12}) \oplus (\langle [1-p, 1-p], \cdots, [0,0]\rangle \otimes Q_{22}) \\
&= R_1 \oplus R_2, \\
&(P_1 \oplus P_2) \oplus (R_1 \oplus R_2) \\
&= (P_1 \oplus R_1) \oplus (P_2 \oplus R_2) \\
&= Q_1 \oplus Q_2.
\end{aligned}$$

Thus we conclude by (Q-Prob).

- (Q-Cond)
$$\frac{\Delta \vdash \{\Gamma \wedge L; Q_1\}\ S_1\ \{\Gamma'; Q_1'\} \qquad \Delta \vdash \{\Gamma \wedge \neg L; Q_1\}\ S_2\ \{\Gamma'; Q_1'\}}{\Delta \vdash \{\Gamma; Q_1\}\ \textbf{if}\ L\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{\Gamma'; Q_1'\}}$$
By inversion, we have $\Delta \vdash \{\Gamma \wedge L; Q_2\}\ S_1\ \{\Gamma'; Q_2'\}$, and $\Delta \vdash \{\Gamma \wedge \neg L; Q_2\}\ S_2\ \{\Gamma'; Q_2'\}$. By induction hypothesis, we have $\Delta \vdash \{\Gamma \wedge L; Q_1 \oplus Q_2\}\ S_1\ \{\Gamma'; Q_1' \oplus Q_2'\}$ and $\Delta \vdash \{\Gamma \wedge \neg L; Q_1 \oplus Q_2\}\ S_2\ \{\Gamma'; Q_1' \oplus Q_2'\}$. Then we conclude by (Q-Cond).

- (Q-Loop)
$$\frac{\Delta \vdash \{\Gamma \wedge L; Q_1\}\ S\ \{\Gamma; Q_1\}}{\Delta \vdash \{\Gamma; Q_1\}\ \textbf{while}\ L\ \textbf{do}\ S\ \textbf{od}\ \{\Gamma \wedge \neg L; Q_1\}}$$

By inversion, we have $\Delta \vdash \{\Gamma \wedge L; Q_2\}\ S\ \{\Gamma; Q_2\}$. By induction hypothesis, we have $\Delta \vdash \{\Gamma \wedge L; Q_1 \oplus Q_2\}\ S\ \{\Gamma; Q_1 \oplus Q_2\}$. Then we conclude by (Q-Loop).

- (Q-Seq)
$$\frac{\Delta \vdash \{\Gamma; Q_1\}\ S_1\ \{\Gamma''; Q_1''\} \qquad \Delta \vdash \{\Gamma''; Q_1''\}\ S_2\ \{\Gamma'; Q_1'\}}{\Delta \vdash \{\Gamma; Q_1\}\ S_1; S_2\ \{\Gamma'; Q_1'\}}$$
By inversion, there exists Q_2'' such that $\Delta \vdash \{\Gamma; Q_2\}\ S_1\ \{\Gamma''; Q_2''\}$, and $\Delta \vdash \{\Gamma''; Q_2''\}\ S_2\ \{\Gamma'; Q_2'\}$. By induction hypothesis, we have $\Delta \vdash \{\Gamma; Q_1 \oplus Q_2\}\ S_1\ \{\Gamma''; Q_1'' \oplus Q_2''\}$ and $\Delta \vdash \{\Gamma''; Q_1'' \oplus Q_2''\}\ S_2\ \{\Gamma'; Q_1' \oplus Q_2'\}$. Then we conclude by (Q-Seq).

- (Q-Weaken)
$$\frac{\Delta \vdash \{\Gamma_0; Q_0\}\ S\ \{\Gamma_0'; Q_0'\} \qquad \Gamma \models \Gamma_0 \qquad \Gamma_0' \models \Gamma' \qquad \Gamma \models Q_1 \sqsupseteq Q_0 \qquad \Gamma_0' \models Q_0' \sqsupseteq Q_1'}{\Delta \vdash \{\Gamma; Q_1\}\ S\ \{\Gamma'; Q_1'\}}$$
By inversion, there exist Q_3, Q_3' such that $\Gamma \models Q_2 \sqsupseteq Q_3$, $\Gamma_0' \models Q_3' \sqsupseteq Q_2'$, and $\Delta \vdash \{\Gamma_0; Q_3\}\ S\ \{\Gamma_0'; Q_3'\}$. By induction hypothesis, we have $\Delta \vdash \{\Gamma_0; Q_0 \oplus Q_3\}\ S\ \{\Gamma_0'; Q_0' \oplus Q_3'\}$. To apply (Q-Weaken), we need to show that $\Gamma \models Q_1 \oplus Q_2 \sqsupseteq Q_0 \oplus Q_3$ and $\Gamma_0' \models Q_0' \oplus Q_3' \sqsupseteq Q_1' \oplus Q_2'$. Then appeal to Lemmas 5.18 and 5.19.

□

Now I can construct the annotated transition kernel to reduce the soundness proof to OST.

Proof of Lem. 5.26. Let $\nu \overset{\text{def}}{=} \mapsto(\langle \gamma, S, K, \alpha\rangle)$. By inversion on $\Delta \vdash \{\Gamma; Q\}\ \langle \gamma, S, K, \alpha\rangle$, we know that $\gamma \models \Gamma$, $\Delta \vdash \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\}$, and $\Delta \vdash \{\Gamma'; Q'\}\ K$ for some Γ', Q'. We construct a probability measure μ as $\kappa(\langle \Gamma, Q, \gamma, S, K, \alpha\rangle)$ by induction on the derivation of $\Delta \vdash \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\}$.

- (Q-Skip)
$$\frac{}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{skip}\ \{\Gamma; Q\}}$$
By induction on the derivation of $\Delta \vdash \{\Gamma; Q\}\ K$.
  - (QK-Stop)
$$\frac{}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{Kstop}}$$
We have $\nu = \delta(\langle \gamma, \mathbf{skip}, \mathbf{Kstop}, \alpha\rangle)$. Then we set $\mu = \delta(\langle \Gamma, Q, \gamma, \mathbf{skip}, \mathbf{Kstop}, \alpha\rangle)$. It is clear that $\phi_Q(\gamma) = \underline{1} \otimes \phi_Q(\gamma)$.
  - (QK-Loop)
$$\frac{\Delta \vdash \{\Gamma \wedge L; Q\}\ S\ \{\Gamma; Q\} \qquad \Delta \vdash \{\Gamma \wedge \neg L; Q\}\ K}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{Kloop}\ L\ S\ K}$$
Let $b \in \{\bot, \top\}$ be such that $\gamma \vdash L \Downarrow b$.
If $b = \top$, then $\nu = \delta(\langle \gamma, S, \mathbf{Kloop}\ L\ S\ K, \alpha\rangle)$. We set $\mu = \delta(\langle \Gamma \wedge L, Q, \gamma, S, \mathbf{Kloop}\ L\ S\ K, \alpha\rangle)$. In this case, we know that $\gamma \models \Gamma \wedge L$. By the premise, we know that $\Delta \vdash \{\Gamma \wedge L; Q\}\ S\ \{\Gamma; Q\}$. It then remains to show that $\Delta \vdash \{\Gamma; Q\}\ \mathbf{Kloop}\ L\ S\ K$. By (QK-Loop), it suffices to show that $\Delta \vdash \{\Gamma \wedge L; Q\}\ S\ \{\Gamma; Q\}$ and $\Delta \vdash \{\Gamma \wedge \neg L; Q\}\ K$. Then appeal to the premise.
If $b = \bot$, then $\mu = \delta(\langle \gamma, \mathbf{skip}, K, \alpha\rangle)$. We set $\mu = \delta(\langle \Gamma \wedge \neg L, Q, \gamma, \mathbf{skip}, K, \alpha\rangle)$. In this case, we know that $\gamma \models \Gamma \wedge \neg L$. By (Q-Skip), we have $\Delta \vdash \{\Gamma \wedge \neg L; Q\}\ \mathbf{skip}\ \{\Gamma \wedge$

$\neg L; Q\}$. It then remains to show that $\Delta \vdash \{\Gamma \wedge \neg L; Q\}\ K$. Then appeal to the premise.

In both cases, γ and Q do not change, thus we conclude that $\phi_Q(\gamma) = \underline{1} \otimes \phi_Q(\gamma)$.

(QK-Seq)

$$\frac{\Delta \vdash \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\} \qquad \Delta \vdash \{\Gamma'; Q'\}\ K}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{Kseq}\ S\ K}$$

– We have $\nu = \delta(\langle \gamma, S, K, \alpha \rangle)$. Then we set $\mu = \delta(\langle \Gamma, Q, \gamma, S, K, \alpha \rangle)$. By the premise, we know that $\Delta \vdash \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\}$ and $\Delta \vdash \{\Gamma'; Q'\}\ K$. Because γ and Q do not change, we conclude that $\phi_Q(\gamma) = \underline{1} \otimes \phi_Q(\gamma)$.

(QK-Weaken)

$$\frac{\Delta \vdash \{\Gamma'; Q'\}\ K \qquad \Gamma \models \Gamma' \qquad \Gamma \models Q \sqsupseteq Q'}{\Delta \vdash \{\Gamma; Q\}\ K}$$

– Because $\gamma \models \Gamma$ and $\Gamma \models \Gamma'$, we know that $\gamma \models \Gamma'$. Let μ' be obtained from the induction hypothesis on $\Delta \vdash \{\Gamma'; Q'\}\ K$. Then $\phi_{Q'}(\gamma) \sqsupseteq \overrightarrow{\mathbb{E}_{\sigma' \sim \mu'}[\langle [(\alpha' - \alpha)^k, (\alpha' - \alpha)^k] \rangle_{0 \le k \le m} \otimes \phi_{Q''}(\gamma')]}$, where $\sigma' = \langle \_, Q'', \gamma', \_, \_, \alpha' \rangle$. We set $\mu = \mu'$. Because $\Gamma \models Q \sqsupseteq Q'$ and $\gamma \models \Gamma$, we conclude that $\phi_Q(\gamma) \sqsupseteq \phi_{Q'}(\gamma)$.

(Q-Tick)

$$\frac{Q = \overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes Q'}{\Delta \vdash \{\Gamma; Q\}\ \mathbf{tick}(c)\ \{\Gamma; Q'\}}$$

• We have $\nu = \delta(\langle \gamma, \mathbf{skip}, K, \alpha + c \rangle)$. Then we set $\mu = \delta(\langle \Gamma, Q', \gamma, \mathbf{skip}, K, \alpha + c \rangle)$. By (Q-Skip), we have $\Delta \vdash \{\Gamma; Q'\}\ \mathbf{skip}\ \{\Gamma; Q'\}$. Then by the assumption, we have $\Delta \vdash \{\Gamma; Q'\}\ K$. It remains to show that $\phi_Q(\gamma) \sqsupseteq \overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes \phi_{Q'}(\gamma)$. Indeed, we have $\phi_Q(\gamma) = \overrightarrow{\langle [c^k, c^k] \rangle_{0 \le k \le m}} \otimes \phi_{Q'}(\gamma)$ by the premise.

(Q-Assign)

$$\frac{\Gamma = [E/x]\Gamma' \qquad Q = [E/x]Q'}{\Delta \vdash \{\Gamma; Q\}\ x := E\ \{\Gamma'; Q'\}}$$

• Let $r \in \mathbb{R}$ be such that $\gamma \vdash E \Downarrow r$. We have $\nu = \delta(\langle \gamma[x \mapsto r], \mathbf{skip}, K, \alpha \rangle)$. Then we set $\mu = \delta(\langle \Gamma', Q', \gamma[x \mapsto r], \mathbf{skip}, K, \alpha \rangle)$. Because $\gamma \vdash \Gamma$, i.e., $\gamma \vdash [E/x]\Gamma'$, we know that $\gamma[x \mapsto r] \vdash \Gamma'$. By (Q-Skip), we have $\Delta \vdash \{\Gamma'; Q'\}\ \mathbf{skip}\ \{\Gamma'; Q'\}$. Then by the assumption, we have $\Delta \vdash \{\Gamma'; Q'\}\ K$. It remains to show that $\phi_Q(\gamma) = \underline{1} \otimes \phi_{Q'}(\gamma[x \mapsto r])$. By the premise, we have $Q = [E/x]Q'$, thus $\phi_Q(\gamma) = \phi_{[E/x]Q'}(\gamma) = \phi_{Q'}(\gamma[x \mapsto r])$.

(Q-Sample)

$$\frac{\Gamma = \forall x \in \mathrm{supp}(\mu_D) : \Gamma' \qquad Q = \mathbb{E}_{x \sim \mu_D}[Q']}{\Delta \vdash \{\Gamma; Q\}\ x \sim D\ \{\Gamma'; Q'\}}$$

• We have $\nu = \mu_D \gg\!= \lambda r.\delta(\langle \gamma[x \mapsto r], \mathbf{skip}, K, \alpha \rangle)$. Then we set $\mu = \mu_D \gg\!= \lambda r.\delta(\langle \Gamma', Q', \gamma[x \mapsto r], \mathbf{skip}, K, \alpha \rangle)$. For all $r \in \mathrm{supp}(\mu_D)$, because $\gamma \models \forall x \in \mathrm{supp}(\mu_D) : \Gamma'$, we know that $\gamma[x \mapsto r] \models \Gamma'$. By (Q-Skip), we have $\Delta \vdash \{\Gamma'; Q'\}\ \mathbf{skip}\ \{\Gamma'; Q'\}$. Then by the assumption, we have $\Delta \vdash \{\Gamma'; Q'\}\ K$. It remains to show that $\phi_Q(\gamma) \sqsupseteq \mathbb{E}_{r \sim \mu_D}[\underline{1} \otimes \phi_{Q'}(\gamma[x \mapsto r])]$. Indeed, because $Q = \mathbb{E}_{x \sim \mu_D}[Q']$, we know that $\phi_Q(\gamma) = \phi_{\mathbb{E}_{x \sim \mu_D}[Q']}(\gamma) = \mathbb{E}_{r \sim \mu_D}[\phi_{Q'}(\gamma[x \mapsto r])]$.

(Q-Call)
$$\frac{\forall i\colon (\Gamma; Q_i, \Gamma'; Q_i') \in \Delta(f)}{\Delta \vdash \{\Gamma; \bigoplus_i Q_i\}\ \textbf{call}\ f\ \{\Gamma'; \bigoplus_i Q_i'\}}$$

- We have $\nu = \delta(\langle \gamma, \mathscr{D}(f), K, \alpha\rangle)$. Then we set $\mu = \delta(\langle \Gamma, \bigoplus_i Q_i, \gamma, \mathscr{D}(f), K, \alpha\rangle)$. By the premise, we know that $\Delta \vdash \{\Gamma; Q_i\}\ \mathscr{D}(f)\ \{\Gamma'; Q_i'\}$ for each i. By Lem. 5.27 and simple induction, we know that $\Delta \vdash \{\Gamma; \bigoplus_i Q_i\}\ \mathscr{D}(f)\ \{\Gamma'; \bigoplus_i Q_i'\}$. Because γ and $\bigoplus_i Q_i$ do not change, we conclude that $\phi_{\bigoplus_i Q_i}(\gamma) = \underline{1} \otimes \phi_{\bigoplus_i Q_i}(\gamma)$.

(Q-Prob)
$$\frac{\Delta \vdash \{\Gamma; Q_1\}\ S_1\ \{\Gamma'; Q'\} \qquad \Delta \vdash \{\Gamma; Q_2\}\ S_2\ \{\Gamma'; Q'\} \qquad Q = P \oplus R \qquad P = \langle [p,p], [0,0], \cdots, [0,0]\rangle \otimes Q_1 \qquad R = \langle [1-p, 1-p], [0,0], \cdots, [0,0]\rangle \otimes Q_2}{\Delta \vdash \{\Gamma; Q\}\ \textbf{if}\ \textbf{prob}(p)\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{\Gamma'; Q'\}}$$

- We have $\nu = p \cdot \delta(\langle \gamma, S_1, K, \alpha\rangle) + (1-p) \cdot \delta(\langle \gamma, S_2, K, \alpha\rangle)$. Then we set $\mu = p \cdot \delta(\langle \Gamma, Q_1, \gamma, S_1, K, \alpha\rangle + (1-p) \cdot \delta(\langle \Gamma, Q_2, \gamma, S_2, K, \alpha\rangle)$. From the assumption and the premise, we know that $\gamma \models \Gamma$, $\Delta \vdash \{\Gamma'; Q'\}\ K$, and $\Delta \vdash \{\Gamma; Q_1\}\ S_1\ \{\Gamma'; Q'\}$, $\Delta \vdash \{\Gamma; Q_2\}\ S_2\ \{\Gamma'; Q'\}$. It remains to show that $\phi_Q(\gamma)_k \geq_{\mathcal{I}} (p \cdot \phi_{Q_1}(\gamma)_k) +_{\mathcal{I}} ((1-p) \cdot \phi_{Q_2}(\gamma)_k)$, where the scalar product $p \cdot [a,b] \stackrel{\text{def}}{=} [pa, pb]$ for $p \geq 0$. On the other hand, from the premise, we have $Q_k = P_k +_{\mathcal{PI}} R_k$ and $P_k = (([p,p], \cdots, [0,0]) \otimes Q_1)_k = \binom{k}{0} \times_{\mathcal{PI}} ([p,p] \cdot_{\mathcal{PI}} (Q_1)_k) = p \cdot (Q_1)_k$, as well as $R_k = (1-p) \cdot (Q_2)_k$. Therefore, we have $\phi_Q(\gamma)_k = \phi_{\xrightarrow{\langle P_k +_{\mathcal{PI}} R_k\rangle_{0 \leq k \leq m}}}(\gamma)_k = p \cdot \phi_{Q_1}(\gamma)_k +_{\mathcal{I}} (1-p) \cdot \phi_{Q_2}(\gamma)_k$.

(Q-Cond)
$$\frac{\Delta \vdash \{\Gamma \wedge L; Q\}\ S_1\ \{\Gamma'; Q'\} \qquad \Delta \vdash \{\Gamma \wedge \neg L; Q\}\ S_2\ \{\Gamma'; Q'\}}{\Delta \vdash \{\Gamma; Q\}\ \textbf{if}\ L\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{\Gamma'; Q'\}}$$

- Let $b \in \{\top, \bot\}$ be such that $\gamma \vdash L \Downarrow b$.
  If $b = \top$, then $\nu = \delta(\langle \gamma, S_1, K, \alpha\rangle)$. We set $\mu = \delta(\langle \Gamma \wedge L, Q, \gamma, S_1, K, \alpha\rangle)$. In this case, we know that $\gamma \models \Gamma \wedge L$. By the premise and the assumption, we know that $\Delta \vdash \{\Gamma \wedge L; Q\}\ S_1\ \{\Gamma'; Q'\}$ and $\Delta \vdash \{\Gamma'; Q'\}\ K$.
  If $b = \bot$, then $\nu = \delta(\langle \gamma, S_2, K, \alpha\rangle)$. We set $\mu = \delta(\langle \Gamma \wedge \neg L, Q, \gamma, S_2, K, \alpha\rangle)$. In this case, we know that $\gamma \models \Gamma \wedge \neg L$. By the premise and the assumption, we know that $\Delta \vdash \{\Gamma \wedge \neg L; Q\}\ S_2\ \{\Gamma'; Q'\}$ and $\Delta \vdash \{\Gamma'; Q'\}\ K$.
  In both cases, γ and Q do not change, thus we conclude that $\phi_Q(\gamma) = \underline{1} \otimes \phi_Q(\gamma)$.

(Q-Loop)
$$\frac{\Delta \vdash \{\Gamma \wedge L; Q\}\ S\ \{\Gamma; Q\}}{\Delta \vdash \{\Gamma; Q\}\ \textbf{while}\ L\ \textbf{do}\ S_1\ \textbf{od}\ \{\Gamma \wedge \neg L; Q\}}$$

- We have $\nu = \delta(\langle \gamma, \textbf{skip}, \textbf{Kloop}\ L\ S\ K, \alpha\rangle)$. Then we set $\mu = \delta(\langle \Gamma, Q, \gamma, \textbf{skip}, \textbf{Kloop}\ L\ S\ K, \alpha\rangle)$. By (Q-Skip), we have $\Delta \vdash \{\Gamma; Q\}\ \textbf{skip}\ \{\Gamma; Q\}$. Then by the assumption $\Delta \vdash \{\Gamma \wedge \neg L; Q\}\ K$ and the premise, we know that $\Delta \vdash \{\Gamma; Q\}\ \textbf{Kloop}\ L\ S\ K$ by (QK-Loop). Because γ and Q do not change, we conclude that $\phi_Q(\gamma) = \underline{1} \otimes \phi_Q(\gamma)$.

(Q-Seq)
$$\frac{\Delta \vdash \{\Gamma; Q\}\ S_1\ \{\Gamma'; Q'\} \qquad \Delta \vdash \{\Gamma'; Q'\}\ S_2\ \{\Gamma''; Q''\}}{\Delta \vdash \{\Gamma; Q\}\ S_1; S_2\ \{\Gamma''; Q''\}}$$

- We have $\nu = \delta(\langle \gamma, S_1, \textbf{Kseq}\ S_2\ K, \alpha\rangle)$. Then we set $\mu = \delta(\langle \Gamma, Q, \gamma, S_1, \textbf{Kseq}\ S_2\ K, \alpha\rangle)$.

By the first premise, we have $\Delta \vdash \{\Gamma; Q\}\ S_1\ \{\Gamma'; Q'\}$. By the assumption $\Delta \vdash \{\Gamma''; Q''\}\ K$ and the second premise, we know that $\Delta \vdash \{\Gamma'; Q'\}$ **Kseq** $S_2\ K$ by (QK-Seq). Because γ and Q do not change, we conclude that $\phi_Q(\gamma) = \underline{1} \otimes \phi_Q(\gamma)$.

- (Q-Weaken)
$$\frac{\Delta \vdash \{\Gamma_0; Q_0\}\ S\ \{\Gamma_0'; Q_0'\} \qquad \Gamma \models \Gamma_0 \qquad \Gamma_0' \models \Gamma' \qquad \Gamma \models Q \sqsupseteq Q_0 \qquad \Gamma_0' \models Q_0' \sqsupseteq Q'}{\Delta \vdash \{\Gamma; Q\}\ S\ \{\Gamma'; Q'\}}$$
By $\gamma \models \Gamma$ and $\Gamma \models \Gamma_0$, we know that $\gamma \models \Gamma_0$. By the assumption $\Delta \vdash \{\Gamma'; Q'\}\ K$ and the premise $\Gamma_0' \models \Gamma'$, $\Gamma_0' \models Q_0' \sqsupseteq Q'$, we derive $\Delta \vdash \{\Gamma_0'; Q_0'\}\ K$ by (QK-Weaken). Thus let μ_0 be obtained by the induction hypothesis on $\Delta \vdash \{\Gamma_0; Q_0\}\ S\ \{\Gamma_0'; Q_0'\}$. Then $\phi_{Q_0}(\gamma) \sqsupseteq \mathbb{E}_{\sigma' \sim \mu_0}[\overrightarrow{\langle [(\alpha' - \alpha)^k, (\alpha' - \alpha)^k] \rangle_{0 \le k \le m}} \otimes \phi_{Q''}(\gamma')]$, where $\sigma' = \langle \_, Q'', \gamma', \_, \_, \alpha' \rangle$. We set $\mu = \mu_0$. By the premise $\Gamma \models Q \sqsupseteq Q_0$ and $\gamma \models \Gamma$, we conclude that $\phi_Q(\gamma) \sqsupseteq \phi_{Q_0}(\gamma)$. □

Therefore, I can use the annotated kernel κ above to re-construct the trace-based moment semantics in §5.3.2. Then I can define the potential function on annotated program configurations as $\phi(\sigma) \stackrel{\text{def}}{=} \phi_Q(\gamma)$ where $\sigma = \langle \_, Q, \gamma, \_, \_, \_ \rangle$.

The next step is to apply the extended OST for interval bounds (Thm. 5.25). Recall that the theorem requires that for some $\ell \in \mathbb{N}$ and $C \ge 0$, $\|Y_n\|_\infty \le C \cdot (n+1)^\ell$ almost surely for all $n \in \mathbb{Z}^+$. One sufficient condition for the requirement is to assume the *bounded-update* property, i.e., every (deterministic or probabilistic) assignment to a program variable updates the variable with a bounded change. As observed by Wang et al. [137], bounded updates are common in practice. I formulate the idea as follows.

Lemma 5.28. *If there exists $C_0 \ge 0$ such that for all $n \in \mathbb{Z}^+$ and $x \in \mathsf{VID}$, it holds that $\mathbb{P}[|\gamma_{n+1}(x) - \gamma_n(x)| \le C_0] = 1$ where ω is an infinite trace, $\omega_n = \langle \gamma_n, \_, \_, \_ \rangle$, and $\omega_{n+1} = \langle \gamma_{n+1}, \_, \_, \_ \rangle$, then there exists $C \ge 0$ such that for all $n \in \mathbb{Z}^+$, $\|Y_n\|_\infty \le C \cdot (n+1)^{md}$ almost surely.*

Proof. Let $C_1 \ge 0$ be such that for all **tick**(c) statements in the program, $|c| \le C_1$. Then for all ω, if $\omega_n = \langle \_, \_, \_, \alpha_n \rangle$, then $|\alpha_n| \le n \cdot C_1$. On the other hand, we know that $\mathbb{P}[|\gamma_n(x) - \gamma_0(x)| \le C_0 \cdot n] = 1$ for any variable x. As we assume all the program variables are initialized to zero, we know that $\mathbb{P}[|\gamma_n(x)| \le C_0 \cdot n] = 1$. From the construction in the proof of Lem. 5.26, we know that all the templates used to define the interval-valued potential function should have almost surely bounded coefficients. Let $C_2 \ge 0$ be such a bound. Also, the k-th component in a template is a polynomial in $\mathbb{R}_{kd}[\mathsf{VID}]$. Therefore, $\Phi_n(\omega) = \phi(\omega_n) = \phi_{Q_n}(\gamma_n)$, and

$$|\phi_{Q_n}(\gamma_n)_k| \le \sum_{i=0}^{kd} C_2 \cdot |\mathsf{VID}|^i \cdot |C_0 \cdot n|^i \le C_3 \cdot (n+1)^{kd}, \text{ a.s.},$$

for some sufficiently large constant C_3. Thus

$$\begin{aligned}|(Y_n)_k| = |(A_n \otimes \Phi_n)_k| &= |\sum_{i=0}^{k}{}_{\mathcal{I}} \binom{k}{i} \times_{\mathcal{I}} ((A_n)_i \cdot_{\mathcal{I}} (\Phi_n)_{k-i})| \\ &\leq \sum_{i=0}^{k} \binom{k}{i} \cdot (n \cdot C_1)^i \cdot (C_3 \cdot (n+1))^{(k-i)d} \\ &\leq C_4 \cdot (n+1)^{kd}, \text{a.s.},\end{aligned}$$

for some sufficiently large constant C_4. Therefore $\|Y_n\|_\infty \leq C_5 \cdot (n+1)^{md}$, a.s., for some sufficiently large constant C_5. □

Now I prove the soundness of bound inference.

Proof of Thm. 5.23. By Lem. 5.28, there exists $C \geq 0$ such that $\|Y_n\|_\infty \leq C \cdot (n+1)^{md}$ almost surely for all $n \in \mathbb{Z}^+$. By the assumption, we also know that $\mathbb{E}[T^{md}] < \infty$. Thus by Thm. 5.25, we conclude that $\mathbb{E}[Y_T] \sqsubseteq \mathbb{E}[Y_0]$, i.e., $\mathbb{E}[A_T] \sqsubseteq \mathbb{E}[\Phi_0] = \phi_Q(\lambda\_.0)$. □

5.3.4 An Algorithm for Checking Soundness Criteria

Termination Analysis I reuse my system for automatically deriving higher moments, which I developed in §5.2.3 and §5.2.4, for checking if $\mathbb{E}[T^{md}] < \infty$ (Thm. 5.23(i)). To reason about termination time, I assume that every program statement increments the cost accumulator by one. For example, the inference rule (Q-Sample) becomes

$$\frac{\Gamma = \forall x \in \mathrm{supp}(\mu_D) : \Gamma' \qquad Q = \langle 1, 1, \cdots, 1\rangle \otimes \mathbb{E}_{x\sim\mu_D}[Q']}{\Delta \vdash \{\Gamma; Q\}\ x \sim D\ \{\Gamma'; Q'\}}$$

However, I cannot apply Thm. 5.23 for the soundness of the termination-time analysis, because that would introduce a circular dependence. Instead, I use a different proof technique to reason about $\mathbb{E}[T^{md}]$, taking into account the *monotonicity* of the runtimes. My upper-bound analysis of higher moments of runtimes is similar to the approach of Kura et al. [94]. In this section, I assume $\mathcal{R} = ([0,\infty], \leq, +, \cdot, 0, 1)$ to be a partially ordered semiring on extended nonnegative real numbers.

Definition 5.29. A map $\psi : \Sigma \to \mathcal{M}_{\mathcal{R}}^{(m)}$ is said to be an *expected-potential function* for upper bounds on stopping time if

(i) $\psi(\sigma)_0 = 1$ for all $\sigma \in \Sigma$,
(ii) $\psi(\sigma) = \underline{1}$ if $\sigma = \langle \_, \textbf{skip}, \textbf{Kstop}, \_\rangle$, and
(iii) $\psi(\sigma) \sqsupseteq \mathbb{E}_{\sigma' \sim \mapsto(\sigma)}[\langle 1, 1, \cdots, 1\rangle \otimes \psi(\sigma')]$ for all non-terminating configuration $\sigma \in \Sigma$.

Intuitively, $\psi(\sigma)$ is an upper bound on the moments of the evaluation steps upon termination for the computation that *continues from* the configuration σ. I define A_n and Ψ_n where $n \in \mathbb{Z}^+$ to be random variables on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ of the trace

semantics as $A_n(\omega) \stackrel{\text{def}}{=} \overrightarrow{\langle n^k \rangle_{0 \le k \le m}}$ and $\Psi_n(\omega) \stackrel{\text{def}}{=} \psi(\omega_n)$. Then I define $A_T(\omega) \stackrel{\text{def}}{=} A_{T(\omega)}(\omega)$. Note that $A_T = \overrightarrow{\langle T^k \rangle_{0 \le k \le m}}$.

I now show that a valid potential function for stopping time *always* gives a sound upper bound by applying the Monotone Convergence Theorem.

Proposition 5.30 (Monotone convergence theorem). *If $\{f_n\}_{n \in \mathbb{Z}^+}$ is a non-decreasing sequence of nonnegative measurable functions on a measure space $(S, \mathcal{S}, \mu)$, and $\{f_n\}_{n \in \mathbb{Z}^+}$ converges to f pointwise, then f is measurable and $\lim_{n \to \infty} \mu(f_n) = \mu(f) \le \infty$.*

Further, the theorem still holds if f is chosen as a measurable function and "$\{f_n\}_{n \in \mathbb{Z}^+}$ converges to f pointwise" holds almost everywhere*, rather than* everywhere*.*

Theorem 5.31. $\mathbb{E}[A_T] \sqsubseteq \mathbb{E}[\Psi_0]$.

Proof. Let $C_n(\omega) \stackrel{\text{def}}{=} \langle 1, 1, \cdots, 1 \rangle$ if $n < T(\omega)$, otherwise $C_n(\omega) \stackrel{\text{def}}{=} \langle 1, 0, \cdots, 0 \rangle$. Then $A_T = \bigotimes_{i=0}^{\infty} C_i$. By Prop. 5.30, we know that $\mathbb{E}[A_T] = \lim_{n \to \infty} \mathbb{E}[\bigotimes_{i=0}^{n} C_i]$. Thus it suffices to show that for all $n \in \mathbb{Z}^+$, $\mathbb{E}[\bigotimes_{i=0}^{n} C_i] \sqsubseteq \mathbb{E}[\Psi_0]$.

Observe that $\{C_n\}_{n \in \mathbb{Z}^+}$ is adapted to $\{\mathcal{F}_n\}_{n \in \mathbb{Z}^+}$, because the event $\{T \le n\}$ is $\mathcal{F}_n$-measurable. Then we have

$$\begin{aligned} \mathbb{E}[\textstyle\bigotimes_{i=0}^{n} C_i \otimes \Psi_{n+1} \mid \mathcal{F}_n] &= \textstyle\bigotimes_{i=0}^{n-1} C_i \otimes \mathbb{E}[C_n \otimes \Psi_{n+1} \mid \mathcal{F}_n], \text{a.s.}, \\ &\sqsubseteq \textstyle\bigotimes_{i=0}^{n-1} C_i \otimes \Psi_n. \end{aligned}$$

Therefore, $\mathbb{E}[\bigotimes_{i=0}^{n} C_i \otimes \Psi_{n+1}] \sqsubseteq \mathbb{E}[\Psi_0]$ for all $n \in \mathbb{Z}^+$ by a simple induction. Because $\Psi_{n+1} \sqsupseteq \underline{1}$, we conclude that

$$\mathbb{E}[\textstyle\bigotimes_{i=0}^{n} C_i] \sqsubseteq \mathbb{E}[\bigotimes_{i=0}^{n} C_i \otimes \Psi_{n+1}] \sqsubseteq \mathbb{E}[\Psi_0].$$

□

Boundedness of $\|Y_n\|_\infty$, $n \in \mathbb{Z}^+$ To ensure that the condition in Thm. 5.23(ii) holds, I check if the analyzed program satisfies the *bounded-update* property: every (deterministic or probabilistic) assignment to a program variable updates the variable with a change bounded by a constant C almost surely. Then the absolute value of every program variable at evaluation step n can be bounded by $C \cdot n = O(n)$. Thus, a polynomial up to degree $\ell \in \mathbb{N}$ over program variables can be bounded by $O(n^\ell)$ at evaluation step n. As observed by Wang et al. [137], bounded updates are common in practice.

5.4 Tail-Bound Analysis

One application of the central-moment analysis is to bound the probability that the accumulated cost deviates from some given quantity. In this section, I sketch how I produce the tail bounds shown in Fig. 5.1(c).

There are a lot of *concentration-of-measure* inequalities in probability theory [43]. Among those, one of the most important is *Markov's inequality*:

PROPOSITION 5.32. *If X is a nonnegative random variable and $a > 0$, then $\mathbb{P}[X \geq a] \leq \frac{\mathbb{E}[X^k]}{a^k}$ for any $k \in \mathbb{N}$.*

Recall that Fig. 5.1(b) presents upper bounds on the raw moments $\mathbb{E}[tick] \leq 2d + 4$ and $\mathbb{E}[tick^2] \leq 4d^2 + 22d + 28$ for the cost accumulator *tick*. With Markov's inequality, we derive the following tail bounds:

$$\mathbb{P}[tick \geq 4d] \leq \frac{\mathbb{E}[tick]}{4d} \leq \frac{2d+4}{4d} \xrightarrow{d\to\infty} \frac{1}{2}, \tag{5.8}$$

$$\mathbb{P}[tick \geq 4d] \leq \frac{\mathbb{E}[tick^2]}{(4d)^2} \leq \frac{4d^2+22d+28}{16d^2} \xrightarrow{d\to\infty} \frac{1}{4}. \tag{5.9}$$

Note that (5.9) provides an asymptotically more precise bound on $\mathbb{P}[tick \geq 4d]$ than (5.8) does, when d approaches infinity.

Central-moment analysis can obtain an even more precise tail bound. As presented in Fig. 5.1(b), my analysis derives $\mathbb{V}[tick] \leq 22d + 28$ for the variance of *tick*. We can now employ concentration inequalities that involve variances of random variables. Recall *Cantelli's inequality*:

PROPOSITION 5.33. *If X is a random variable and $a > 0$, then $\mathbb{P}[X - \mathbb{E}[X] \geq a] \leq \frac{\mathbb{V}[X]}{\mathbb{V}[X]+a^2}$ and $\mathbb{P}[X - \mathbb{E}[X] \leq -a] \leq \frac{\mathbb{V}[X]}{\mathbb{V}[X]+a^2}$.*

With Cantelli's inequality, we obtain the following tail bound, where we assume $d \geq 2$:

$$\begin{aligned} &\mathbb{P}[tick \geq 4d] = \mathbb{P}[tick - (2d+4) \geq 2d-4] \\ &\leq \mathbb{P}[tick - \mathbb{E}[tick] \geq 2d-4] \leq \frac{\mathbb{V}[tick]}{\mathbb{V}[tick] + (2d-4)^2} \\ &= 1 - \frac{(2d-4)^2}{\mathbb{V}[tick] + (2d-4)^2} \leq \frac{22d+28}{4d^2+6d+44} \xrightarrow{d\to\infty} 0. \end{aligned} \tag{5.10}$$

For all $d \geq 15$, (5.10) gives a more precise bound than both (5.8) and (5.9). It is also clear from Fig. 5.1(c), where I plot the three tail bounds (5.8), (5.9), and (5.10), that the asymptotically most precise bound is the one obtained via variances.

In general, for higher central moments, I employ *Chebyshev's inequality* to derive tail bounds:

PROPOSITION 5.34. *If X is a random variable and $a > 0$, then $\mathbb{P}[|X - \mathbb{E}[X]| \geq a] \leq \frac{\mathbb{E}[(X-\mathbb{E}[X])^{2k}]}{a^{2k}}$ for any $k \in \mathbb{N}$.*

In my experiments, I use Chebyshev's inequality to derive tail bounds from the fourth central moments. I will show in Fig. 5.12 that these tail bounds can be more precise than those obtained from both raw moments and variances.

5.5 Implementation and Experiments

Implementation My tool is implemented in OCaml, and consists of about 5,300 LOC. The tool works on imperative arithmetic probabilistic programs using a CFG-based IR [130]. The language supports recursive functions, continuous distributions, unstructured control-flow, and local variables. To infer the bounds on the central moments for a cost accumulator in a program, the user needs to specify the order of the analyzed moment, and a maximal degree for the polynomials to be used in potential-function templates. Using APRON [75], I implemented an interprocedural numeric analysis to infer the logical contexts used in the derivation. I use the off-the-shelf solver Gurobi [98] for LP solving.

Evaluation Setup I evaluated my tool to answer the following three research questions:

1. How does the raw-moment inference part of my tool compare to existing techniques for expected-cost bound analysis [112, 137]?
2. How does my tool compare to the state of the art in tail-probability analysis (which is based only on higher *raw* moments [94])?
3. How scalable is my tool? Can it analyze programs with many recursive functions?

For the first question, I collected a broad suite of challenging examples from related work [94, 112, 137] with different loop and recursion patterns, as well as probabilistic branching, discrete sampling, and continuous sampling. My tool achieved comparable precision and efficiency with the prior work on expected-cost bound analysis [112, 137]. The details are included in Tabs. 5.3 and 5.4.

For the second question, I evaluated my tool on the complete benchmarks from Kura et al. [94]. I also conducted a case study of a timing-attack analysis for a program provided by DARPA during engagements of the STAC program [139], where central moments are more useful than raw moments to bound the success probability of an attacker. I include the case study in §5.6.

For the third question, I conducted case studies on two sets of synthetic benchmark programs:

- coupon-collector programs with N coupons ($N \in [1, 10^3]$), where each program is implemented as a set of tail-recursive functions, each of which represents a state of coupon collection, i.e., the number of coupons collected so far; and
- random-walk programs with N consecutive one-dimensional random walks ($N \in [1, 10^3]$), each of which starts at a position that equals the number of steps taken by the previous random walk to reach the ending position (the origin). Each program is implemented as a set of non-tail-recursive functions, each of which represents a random walk. The random walks in the same program can have different transition probabilities.

The largest synthetic program has nearly 16,000 LOC. I then ran my tool to derive an upper bound on the fourth (resp., second) central moment of the runtime for each coupon-collector

(resp., random-walk) program.

The experiments were performed on a machine with an Intel Core i7 3.6GHz processor and 16GB of RAM under macOS Catalina 10.15.7.

Results Some of the evaluation results to answer the second research question are presented in Tab. 5.1. The programs (1-1) and (1-2) are coupon-collector problems with a total of two and four coupons, respectively. The other five are variants of random walks. The first three are 1-dimensional random walks: (2-1) is integer-valued, (2-2) is real-valued with continuous sampling, and (2-3) exhibits adversarial nondeterminism. The programs (2-4) and (2-5) are two-dimensional random walks. The table contains the inferred upper bounds on the moments for runtimes of these programs, and the running times of the analyses. I compared my results with Kura et al. [94]'s inference tool for raw moments. My tool is as precise as, and sometimes more precise than the prior work on all the benchmark programs. Meanwhile, my tool is able to infer an upper bound on the raw moments of degree up to four on all the benchmarks, while the prior work reports failure on some higher moments for the random-walk programs. In terms of efficiency, my tool completed each example in less than 10 seconds, while the prior work took more than a few minutes on some programs. One reason why my tool is more efficient is that I always reduce higher-moment inference with non-linear polynomial templates to efficient LP solving, but the prior work requires semidefinite programming (SDP) for polynomial templates.

Besides raw moments, my tool is also capable of inferring upper bounds on the central moments of runtimes for the benchmarks. To evaluate the quality of the inferred central moments, Fig. 5.12 plots the upper bounds of tail probabilities on runtimes T obtained by Kura et al. [94], and those by my central-moment analysis. Specifically, the prior work uses Markov's inequality (Prop. 5.32), while I am also able to apply Cantelli's and Chebyshev's inequality (Props. 5.33 and 5.34) with central moments. My tool outperforms the prior work on programs (1-1), (1-2), (2-3), and (2-5), and derives better tail bounds when d is large on programs (2-2) and (2-4), while it obtains similar curves on program (2-1).

Scalability In Fig. 5.13, I demonstrate the running times of my tool on the two sets of synthetic benchmark programs; Fig. 5.13a plots the analysis times for coupon-collector programs as a function of the independent variable N (the total number of coupons), and Fig. 5.13b plots the analysis times for random-walk programs as a function of N (the total number of random walks). The evaluation statistics show that my tool achieves good scalability in both case studies: the runtime is almost a linear function of the program size, which is the number of recursive functions for both case studies. Two reasons why my tool is scalable on the two sets of programs are (i) my analysis is compositional and uses function summaries to analyze function calls, and (ii) for a fixed set of templates and a fixed diameter of the call graph, the number of linear constraints generated by my tool grows linearly with the size of the program, and the LP solvers available nowadays can handle large problem instances efficiently.

Table 5.1: Upper bounds on the raw/central moments of runtimes, with comparison to Kura et al. [94]. "T/O" stands for timeout after 30 minutes. "N/A" means that the tool is not applicable. "-" indicates that the tool fails to infer a bound. Entries with more precise results or less analysis time are marked in bold.

program	moment	this work		Kura et al. [94]	
		upper bnd.	time (sec)	upper bnd.	time (sec)
(1-1)	2^{nd} raw	**201**	0.019	**201**	**0.015**
	3^{rd} raw	**3829**	**0.019**	**3829**	0.020
	4^{th} raw	**90705**	**0.023**	**90705**	0.027
	2^{nd} central	**32**	**0.029**	N/A	N/A
	4^{th} central	**9728**	**0.058**	N/A	N/A
(1-2)	2^{nd} raw	**2357**	1.068	3124	**0.037**
	3^{rd} raw	**148847**	1.512	171932	**0.062**
	4^{th} raw	**11285725**	1.914	12049876	**0.096**
	2^{nd} central	**362**	**3.346**	N/A	N/A
	4^{th} central	**955973**	**9.801**	N/A	N/A
(2-1)	2^{nd} raw	**2320**	**0.016**	**2320**	11.380
	3^{rd} raw	**691520**	**0.018**	-	16.056
	4^{th} raw	**340107520**	**0.021**	-	23.414
	2^{nd} central	**1920**	**0.026**	N/A	N/A
	4^{th} central	**289873920**	**0.049**	N/A	N/A
(2-2)	2^{nd} raw	**8375**	**0.022**	**8375**	38.463
	3^{rd} raw	**1362813**	**0.028**	-	73.408
	4^{th} raw	**306105209**	**0.035**	-	141.072
	2^{nd} central	**5875**	**0.029**	N/A	N/A
	4^{th} central	**447053126**	**0.086**	N/A	N/A
(2-3)	2^{nd} raw	**3675**	**0.039**	6710	48.662
	3^{rd} raw	**618584**	0.049	19567045	**0.039**
	4^{th} raw	**164423336**	**0.055**	-	T/O
	2^{nd} central	**3048**	**0.053**	N/A	N/A
	4^{th} central	**196748763**	**0.123**	N/A	N/A
(2-4)	2^{nd} raw	**6625**	**0.035**	10944	216.352
	3^{rd} raw	**742825**	**0.048**	-	453.435
	4^{th} raw	**101441320**	**0.072**	-	964.579
	2^{nd} central	**6624**	**0.051**	N/A	N/A
	4^{th} central	**313269063**	**0.215**	N/A	N/A
(2-5)	2^{nd} raw	**21060**	**0.045**	-	216.605
	3^{rd} raw	**9860940**	**0.063**	-	467.577
	4^{th} raw	**7298339760**	**0.101**	-	1133.947
	2^{nd} central	**20160**	**0.068**	N/A	N/A
	4^{th} central	**8044220161**	**0.271**	N/A	N/A

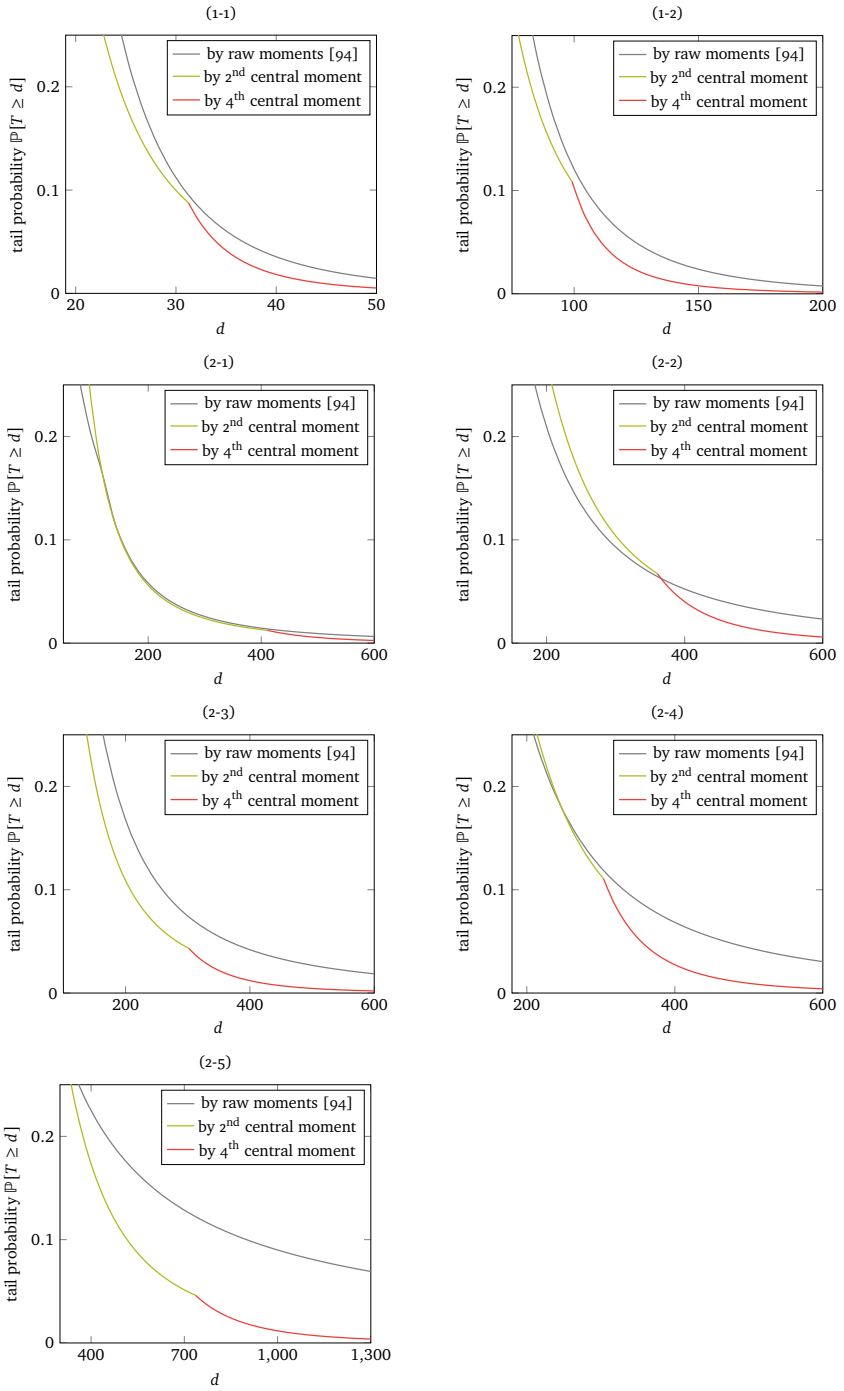
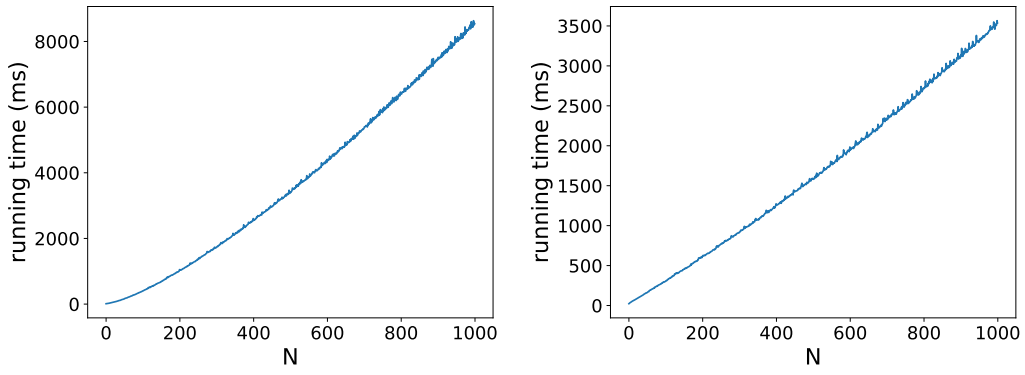


Fig. 5.12: Upper bound of the tail probability $\mathbb{P}[T \geq d]$ as a function of d, with comparison to Kura et al. [94]. Each gray line is the minimum of tail bounds obtained from the raw moments of degree up to four inferred by Kura et al. [94]. Green lines and red lines are the tail bounds given by 2nd and 4th central moments inferred by my tool, respectively.



(a) Coupon Collector (b) Random Walk

Fig. 5.13: Running times of my tool on two sets of synthetic benchmark programs. Each figure plots the runtimes as a function of the size of the analyzed program.

Discussion Higher central moments can also provide more information about the *shape* of a distribution, e.g., the *skewness* (i.e., $\frac{\mathbb{E}[(T-\mathbb{E}[T])^3]}{(\mathbb{V}[T])^{3/2}}$) indicates how lopsided the distribution of T is, and the *kurtosis* (i.e., $\frac{\mathbb{E}[(T-\mathbb{E}[T])^4]}{(\mathbb{V}[T])^2}$) measures the heaviness of the tails of the distribution of T. I used my tool to analyze two variants of the random-walk program (2-1). The two random walks have different transition probabilities and step lengths, but they have the same expected runtime $\mathbb{E}[T]$. Tab. 5.2 presents the skewness and kurtosis derived from the moment bounds inferred by my tool. A positive skew indicates that the mass of the distribution is concentrated on the *left*, and larger skew means the concentration is more *left*. A larger kurtosis, on the other hand, indicates that the distribution has *fatter* tails. Therefore, as the derived skewness and kurtosis indicate, the distribution of the runtime T for rdwalk-2 should be more left-leaning and have fatter tails than the distribution for rdwalk-1. Density estimates for the runtime T, obtained by simulation, are shown in Fig. 5.14.

Table 5.2: Skewness & kurtosis.

program	skewness	kurtosis
rdwalk-1	2.1362	10.5633
rdwalk-2	2.9635	17.5823

My tool can also derive *symbolic* bounds on higher moments. The table below presents the inferred upper bounds on the variances for the random-walk benchmarks, where I replace the concrete inputs with symbolic pre-conditions.

program	pre-condition	upper bnd. on the variance
(2-1)	$x \geq 0$	$1920x$
(2-2)	$x \geq 0$	$2166.6667x + 1541.6667$
(2-3)	$x \geq 0$	$284.2060x^2 + 955.25x$
(2-4)	$x \geq 0 \wedge y > 0$	$144(x+y)^2 + 816(x+y) + 1056$
(2-5)	$x \geq y$	$7920(x-y) + 12240$

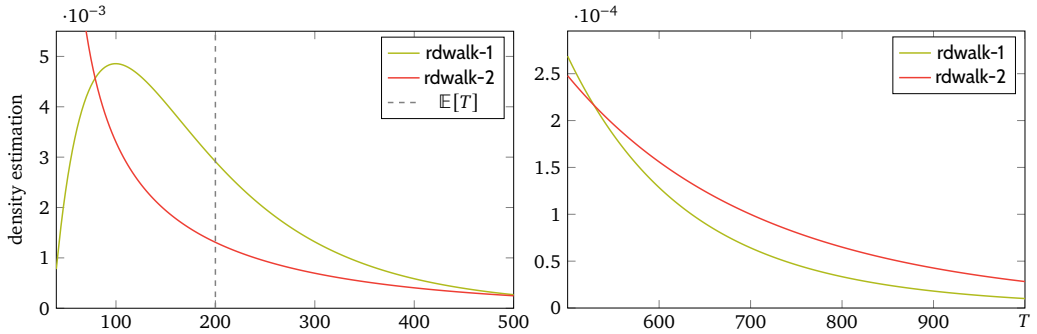


Fig. 5.14: Density estimation for the runtime T of two variants rdwalk-1 and rdwalk-2 of (2-1).

5.6 Case Study: Timing Attack

I motivate my work on central-moment analysis using a probabilistic program with a *timing-leak* vulnerability, and demonstrate how the results from an analysis can be used to bound the success rate of an attack program that attempts to exploit the vulnerability. The program is extracted and modified from a web application provided by DARPA during engagements as part of the STAC program [139]. In essence, the program models a password checker that compares an input *guess* with an internally stored password *secret*, represented as two N-bit vectors. The program in Fig. 5.15(a) is the interface of the checker, and Fig. 5.15(b) is the comparison function compare, which carries out most of the computation. The statements of the form "**tick**$(\cdot)$" represent a cost model for the running time of compare, which is assumed to be observable by the attacker. compare iterates over the bits from high-index to low-index, and the running time expended during the processing of bit i depends on the current comparison result (stored in *cmp*), as well as on the values of the i-th bits of *guess* and *secret*. Because the running time of compare might *leak* information about the relationship between *guess* and *secret*, compare introduces some *random delays* to add noise to its running time. However, we will see shortly that such a countermeasure does *not* protect the program from a timing attack.

I now show how the moments of the running time of compare—the kind of information provided by my central-moment analysis (§5.2)—are useful for analyzing the success probability of the attack program given in Fig. 5.15(c). Let T be the random variable for the running time of compare. A standard timing attack for such programs is to guess the bits of *secret* successively. The idea is the following: Suppose that we have successfully obtained the bits $secret[i+1]$ through $secret[N]$; we now guess that the next bit, $secret[i]$, is 1 and set $guess[i] := 1$. Theoretically, if the following two conditional expectations

$$\mathbb{E}[T_1] \stackrel{\text{def}}{=} \mathbb{E}[T \mid \textstyle\bigwedge_{j=i+1}^{N}(secret[j] = guess[j]) \wedge (secret[i] = 1 \wedge guess[i] = 1)] \tag{5.11}$$

$$\mathbb{E}[T_0] \stackrel{\text{def}}{=} \mathbb{E}[T \mid \textstyle\bigwedge_{j=i+1}^{N}(secret[j] = guess[j]) \wedge (secret[i] = 0 \wedge guess[i] = 1)] \tag{5.12}$$

have a significant difference, then there is an opportunity to check my guess by running

```
1  func compare(guess, secret) begin
2    i := N; cmp := 0;
3    while i > 0 do
4      tick(2);
5      while i > 0 do
6        if prob(0.5) then break fi
7        tick(5);
8        if cmp > 0 ∨ (cmp = 0 ∧ guess[i] > secret[i])
9        then cmp := 1
10       else
11         tick(5);
12         if cmp < 0 ∨ (cmp = 0 ∧ guess[i] < secret[i])
13         then cmp := −1
14         fi
15       fi
16       tick(1);
17       i := i − 1
18     od
19   od;
20   return cmp
21 end
```

(b)

```
1  func check(guess) begin
2    cmp := compare(guess, secret);
3    if cmp = 0 then
4      login()
5    fi
6  end
```

(a)

```
1  guess := 0⃗;
2  i := N;
3  while i > 0 do
4    next := guess;
5    next[i] := 1;
6    est := estimateTime(K, check(next));
7    if est ≤ 13N − 1.5i then
8      guess[i] := 0
9    else
10     guess[i] := 1
11   fi;
12   i := i − 1
13 od
```

(c)

Fig. 5.15: (a) The interface of the password checker. (b) A function that compares two bit vectors, adding some random noise. (c) An attack program that attempts to exploit the timing properties of compare to find the value of the password stored in *secret*.

the program multiple times, using the *average* running time as an estimate of $\mathbb{E}[T]$, and choosing the value of $guess[i]$ according to whichever of (5.11) and (5.12) is closest to my estimate. However, if the difference between $\mathbb{E}[T_1]$ and $\mathbb{E}[T_0]$ is not significant enough, or the program produces a large amount of noise in its running time, the attack might not be realizable in practice. To determine whether the timing difference represents an exploitable vulnerability, we need to reason about the attack program's success rate.

Toward this end, we can analyze the failure probability for setting $guess[i]$ incorrectly, which happens when, due to an unfortunate fluctuation, the running-time estimate est is closer to one of $\mathbb{E}[T_1]$ and $\mathbb{E}[T_0]$, but the truth is actually the other. For instance, suppose that $\mathbb{E}[T_0] < \mathbb{E}[T_1]$ and $est < \frac{\mathbb{E}[T_0]+\mathbb{E}[T_1]}{2}$; the attack program would pick T_0 as the truth, and set $guess[i]$ to 0. If such a choice is *incorrect*, then the actual distribution of est on the i-th round of the attack program satisfies $\mathbb{E}[est] = \mathbb{E}[T_1]$, and the probability of this failure event is

$$\begin{aligned}\mathbb{P}\left[est < \frac{\mathbb{E}[T_0]+\mathbb{E}[T_1]}{2}\right] &= \mathbb{P}\left[est - \mathbb{E}[T_1] < \frac{\mathbb{E}[T_0]-\mathbb{E}[T_1]}{2}\right] \\ &= \mathbb{P}\left[est - \mathbb{E}[est] < \frac{\mathbb{E}[T_0]-\mathbb{E}[T_1]}{2}\right]\end{aligned}$$

under the condition given by the conjunction in (5.11). This formula has exactly the same shape as a *tail probability*, which makes it possible to utilize moments and *concentration-of-measure* inequalities [43] to bound the probability.

The attack program is parameterized by $K > 0$, which represents the number of trials it performs for each bit position to obtain an estimate of the running time. Assume that I have applied my central-moment-analysis technique (§5.2), and obtained the following inequalities on the mean (i.e., the first moment), the second moment, and the variance (i.e., the second central moment) of the quantities (5.11) and (5.12).

$$\begin{aligned}&\mathbb{E}[T_1] \geq 13N, && \mathbb{E}[T_1] \leq 15N, \\ &\mathbb{V}[T_1] \leq 26N^2 + 42N, && && (5.13)\\ &\mathbb{E}[T_0] \geq 13N - 5i, && \mathbb{E}[T_0] \leq 13N - 3i, \\ &\mathbb{V}[T_0] \leq 8N - 36i^2 + 52Ni + 24i. && && (5.14)\end{aligned}$$

To bound the probability that the attack program makes an incorrect guess for the i-th bit, I do case analysis:

- Suppose that $secret[i] = 1$, but the attack program assigns $guess[i] := 0$. The truth—with respect to the actual distribution of the running time T of compare for the i-th bit—is that $\mathbb{E}[est] = \mathbb{E}[T_1]$, but the attack program in Fig. 5.15(c) executes the then-branch of the conditional statement. Thus, my task reduces to that of bounding $\mathbb{P}[est < 13N - 1.5i]$. The estimate est is the average of K i.i.d. random variables drawn from a distribution with mean $\mathbb{E}[T_1]$ and variance $\mathbb{V}[T_1]$. We derive the following,

using the inequalities from (5.13):

$$\begin{aligned}\mathbb{E}[est] &= \mathbb{E}[T_1] \geq 13N,\\ \mathbb{V}[est] &= \frac{\mathbb{V}[T_1]}{K} \leq \frac{26N^2 + 42N}{K}.\end{aligned} \tag{5.15}$$

We are now able to derive an upper bound on $\mathbb{P}[est < 13N - 1.5i]$ using Cantelli's inequality:

$$\begin{aligned}&\mathbb{P}[est \leq 13N - 1.5i]\\ &= \mathbb{P}[est - 13N \leq -1.5i]\\ &\leq \mathbb{P}[est - \mathbb{E}[est] \leq -1.5i] && \dagger\ \mathbb{E}[est] \geq 13N \text{ by (5.15)}\ \dagger\\ &\leq \frac{\mathbb{V}[est]}{\mathbb{V}[est] + (1.5i)^2} && \dagger\ \text{Prop. 5.33}\ \dagger\\ &= \frac{26N^2 + 42N}{26N^2 + 42N + 2.25Ki^2}.\end{aligned}$$

- The other case, in which $secret[i] = 0$ but the attack program chooses to set $guess[i] := 1$, can be analyzed in a similar way to the previous case, and the bound obtained is the following:

$$\begin{aligned}&\mathbb{P}[est > 13N - 1.5i]\\ &\leq \mathbb{P}[est \geq 13N - 1.5i]\\ &\leq \frac{8N - 36i^2 + 52Ni + 24i}{8N - 36i^2 + 52Ni + 24i + 2.25Ki^2}.\end{aligned}$$

Let F_1^i and F_0^i, respectively, denote the two upper bounds on the failure probabilities for the i-th bit.

For the attack program to succeed, it has to succeed for all bits. If the number of bits is $N = 32$, and in each iteration the number of trials that the attack program uses to estimate the running time is $K = 10^4$, we derive a *lower* bound on the success rate of the attack program from the upper bounds on the failure probabilities derived above:

$$\mathbb{P}[\textsc{Success}] \geq \prod_{i=1}^{32}(1 - \max(F_1^i, F_0^i)) \geq 0.219413,$$

which is low, but not insignificant. However, the somewhat low probability is caused by a property of compare: if *guess* and *secret* share a very long prefix, then the running-time behavior on different values of *guess* becomes indistinguishable. However, if instead we bound the success rate for all but the last six bits, we obtain:

$$\mathbb{P}[\textsc{Success for all but the last six bits}] \geq 0.830561,$$

Table 5.3: Upper bounds of the expectations of monotone costs, with comparison to Ngo et al. [112]. ABSYNTH uses a finer-grained set of base functions, and it supports bounds of the form $|[x, y]|$, which is defined as $\max(0, y - x)$.

program	pre-condition	upper bound by my tool	upper bound by ABSYNTH [112]
2drdwalk	$d < n$	$2(n - d + 1)$ (deg 1, 0.269s)	$2\|[d, n + 1]\|$ (deg 1, 0.170s)
C4B_t09	$x > 0$	$17x$ (deg 1, 0.061s)	$17\|[0, x]\|$ (deg 1, 0.014s)
C4B_t13	$x > 0 \wedge y > 0$	$1.25x + y$ (deg 1, 0.060s)	$1.25\|[0, x]\| + \|[0, y]\|$ (deg 1, 0.008s)
C4B_t15	$x > y \wedge y > 0$	$1.1667x$ (deg 1, 0.072s)	$1.3333\|[0, x]\|$ (deg 1, 0.014s)
C4B_t19	$i > 100 \wedge k > 0$	$k + 2i - 49$ (deg 1, 0.059s)	$\|[0, 51 + i + k]\| + 2\|[0, i]\|$ (deg 1, 0.010s)
C4B_t30	$x > 0 \wedge y > 0$	$0.5x + 0.5y + 2$ (deg 1, 0.044s)	$0.5\|[0, x + 2]\| + 0.5\|[0, y + 2]\|$ (deg 1, 0.007s)
C4B_t61	$l \geq 8$	$1.4286l$ (deg 1, 0.046s)	$\|[0, l]\| + 0.5\|[1, l]\|$ (deg 1, 0.007s)
bayesian_network	$n > 0$	$4n$ (deg 1, 0.264s)	$4\|[0, n]\|$ (deg 1, 0.057s)
ber	$x < n$	$2(n - x)$ (deg 1, 0.035s)	$2\|[x, n]\|$ (deg 1, 0.004s)
bin	$n > 0$	$0.2(n + 9)$ (deg 1, 0.036s)	$0.2\|[0, n + 9]\|$ (deg 1, 0.030s)
condand	$n > 0 \wedge m > 0$	$2m$ (deg 1, 0.042s)	$2\|[0, m]\|$ (deg 1, 0.004s)
cooling	$mt > st \wedge t > 0$	$mt - st + 0.42t + 2.1$ (deg 1, 0.071s)	$0.42\|[0, t + 5]\| + \|[st, mt]\|$ (deg 1, 0.017s)
coupon	$\top$	11.6667 (deg 4, 0.066s)	15 (deg 1, 0.016s)
cowboy_duel	$\top$	1.2 (deg 1, 0.030s)	1.2 (deg 1, 0.004s)
cowboy_duel_3way	$\top$	2.0833 (deg 1, 0.110s)	2.0833 (deg 1, 0.142s)
fcall	$x < n$	$2(n - x)$ (deg 1, 0.053s)	$2\|[x, n]\|$ (deg 1, 0.004s)
filling_vol	$volTF > 0$	$0.6667volTF + 7$ (deg 1, 0.082s)	$0.3333\|[0, volTF + 10]\| + 0.3333\|[0, volTF + 11]\|$ (deg 1, 0.079s)
geo	$\top$	5 (deg 1, 0.061s)	5 (deg 1, 0.003s)
hyper	$x < n$	$5(n - x)$ (deg 1, 0.035s)	$5\|[x, n]\|$ (deg 1, 0.005s)
linear01	$x > 2$	$0.6x$ (deg 1, 0.034s)	$0.6\|[0, x]\|$ (deg 1, 0.008s)
prdwalk	$x < n$	$1.1429(n - x + 4)$ (deg 1, 0.037s)	$1.1429\|[x, n + 4]\|$ (deg 1, 0.011s)
prnes	$y > 0 \wedge n < 0$	$-68.4795n + 0.0526(y - 1)$ (deg 1, 0.071s)	$68.4795\|[n, 0]\| + 0.0526\|[0, y]\|$ (deg 1, 0.016s)
prseq	$y > 9 \wedge x - y > 2$	$1.65x - 1.5y$ (deg 1, 0.061s)	$1.65\|[y, x]\| + 0.15\|[0, y]\|$ (deg 1, 0.011s)
prspeed	$x + 1 < n \wedge y < m$	$2(m - y) + 0.6667(n - x)$ (deg 1, 0.065s)	$2\|[y, m]\| + 0.6667\|[x, n]\|$ (deg 1, 0.010s)
race	$h < t$	$0.6667(t - h + 9)$ (deg 1, 0.039s)	$0.6667\|[h, t + 9]\|$ (deg 1, 0.027s)
rdseql	$x > 0 \wedge y > 0$	$2.25x + y$ (deg 1, 0.059s)	$2.25\|[0, x]\| + \|[0, y]\|$ (deg 1, 0.008s)
rdspeed	$x + 1 < n \wedge y < m$	$2(m - y) + 0.6667(n - x)$ (deg 1, 0.062s)	$2\|[y, m]\| + 0.6667\|[x, n]\|$ (deg 1, 0.010s)
rdwalk	$x < n$	$2(n - x + 1)$ (deg 1, 0.035s)	$2\|[x, n + 1]\|$ (deg 1, 0.004s)
rejection_sampling	$n > 0$	$2n$ (deg 2, 0.071s)	$2\|[0, n]\|$ (deg 1, 0.350s)
rfind_lv	$\top$	2 (deg 1, 0.033s)	2 (deg 1, 0.004s)
rfind_mc	$k > 0$	2 (deg 1, 0.047s)	$\|[0, k]\|$ (deg 1, 0.007s)
robot	$n > 0$	$0.2778(n + 7)$ (deg 1, 0.047s)	$0.3846\|[0, n + 6]\|$ (deg 1, 0.017s)
roulette	$n < 10$	$-4.9333n + 98.6667$ (deg 1, 0.057s)	$4.9333\|[n, 20]\|$ (deg 1, 0.073s)
sprdwalk	$x < n$	$2(n - x)$ (deg 1, 0.036s)	$2\|[x, n]\|$ (deg 1, 0.004s)
trapped_miner	$n > 0$	$7.5n$ (deg 1, 0.081s)	$7.5\|[0, n]\|$ (deg 1, 0.015s)
complex	$y, w, n, m > 0$	$4mn + 2n + w + 0.6667(y + 1)$ (deg 2, 0.142s)	$(4\|[0, m]\| + 2)\|[0, n]\| + (\|[0, w]\| + 0.3333\|[0, y]\|)\|[0, y + 1]\| + 0.6667$ (deg 2, 0.451s)
multirace	$n > 0 \wedge m > 0$	$2mn + 4n$ (deg 2, 0.080s)	$2\|[0, m]\|\|[0, n]\| + 4\|[0, n]\|$ (deg 2, 0.692s)
pol04	$x > 0$	$4.5x^2 + 10.5x$ (deg 2, 0.052s)	$1.5\|[0, x]\|^2 + 3\|[1, x]\|\|[0, x]\| + 10.5\|[0, x]\|$ (deg 2, 0.101s)
pol05	$x > 0$	$0.6667(x^2 + 3x)$ (deg 2, 0.059s)	$2\|[0, x + 1]\|\|[0, x]\|$ (deg 2, 0.133s)
pol07	$n > 1$	$1.5n^2 - 4.5n + 3$ (deg 2, 0.057s)	$1.5\|[1, n]\|\|[2, n]\|$ (deg 2, 0.203s)
rdbub	$n > 0$	$3n^2$ (deg 2, 0.055s)	$3\|[0, n]\|^2$ (deg 2, 0.030s)
recursive	$arg_l < arg_h$	$0.25(arg_h - arg_l)^2 + 1.75(arg_h - arg_l)$ (deg 2, 0.313s)	$0.25\|[arg_l, arg_h]\|^2 + 1.75\|[arg_l, arg_h]\|$ (deg 2, 0.398s)
trader	$mP > 0 \wedge sP > mP$	$1.5(sP^2 - mP^2 + sP - mP)$ (deg 2, 0.073s)	$1.5\|[mP, sP]\|^2 + 3\|[mP, sP]\|\|[0, mP]\| + 1.5\|[mP, sP]\|$ (deg 2, 0.192s)

which is a much higher probability! The attack program can enumerate all the possibilities to resolve the last six bits. (Moreover, by brute-forcing the last six bits, a total of $10{,}000 \times 26 + 64 = 260{,}064$ calls to check would be performed, rather than 320,000.)

Overall, my analysis concludes that the check and compare procedures in Fig. 5.15 are vulnerable to a timing attack.

Table 5.4: Upper and lower bounds of the expectation of (possibly) non-monotone costs, with comparison to Wang et al. [137]. To ensure soundness, my tool has to perform an extra termination check required by Thm. 5.23.

program	pre-cond.	termination		bounds by my tool	bounds by Wang et al. [137]
Bitcoin Mining	$x \geq 1$	$\mathbb{E}[T] < \infty$ (deg 1, 0.080s)	ub.	$-1.475x$ (deg 1, 0.078s)	$-1.475x + 1.475$ (deg 2, 3.705s)
			lb.	$-1.5x$ (deg 1, 0.088s)	$-1.5x$ (deg 2, 3.485s)
Bitcoin Mining Pool	$y \geq 0$	$\mathbb{E}[T^2] < \infty$ (deg 4, 0.168s)	ub.	$-7.375y^2 - 66.375y$ (deg 2, 0.127s)	$-7.375y^2 - 41.625y + 49$ (deg 2, 5.936s)
			lb.	$-7.5y^2 - 67.5y$ (deg 2, 0.134s)	$-7.5y^2 - 67.5y$ (deg 2, 6.157s)
Queueing Network	$n > 0$	$\mathbb{E}[T^2] < \infty$ (deg 2, 0.208s)	ub.	$0.0531n$ (deg 2, 0.134s)	$0.0492n$ (deg 3, 69.669s)
			lb.	$0.028n$ (deg 2, 0.139s)	$0.0384n$ (deg 3, 68.849s)
Running Example	$x \geq 0$	$\mathbb{E}[T^2] < \infty$ (deg 2, 0.064s)	ub.	$0.3333(x^2 + x)$ (deg 2, 0.063s)	$0.3333x^2 + 0.3333x$ (deg 2, 3.766s)
			lb.	$0.3333(x^2 + x)$ (deg 2, 0.059s)	$0.3333x^2 + 0.3333x - 0.6667$ (deg 2, 3.555s)
Nested Loop	$i \geq 0$	$\mathbb{E}[T^2] < \infty$ (deg 4, 0.127s)	ub.	$0.3333i^2 + i$ (deg 2, 0.117s)	$0.3333i^2 + i$ (deg 2, 28.398s)
			lb.	$0.3333i^2 + i$ (deg 2, 0.115s)	$0.3333i^2 - i$ (deg 2, 7.299s)
Random Walk	$x \leq n$	$\mathbb{E}[T] < \infty$ (deg 1, 0.063s)	ub.	$2.5x - 2.5n - 2.5$ (deg 1, 0.064s)	$2.5x - 2.5n$ (deg 2, 4.536s)
			lb.	$2.5x - 2.5n - 2.5$ (deg 1, 0.068s)	$2.5x - 2.5n - 2.5$ (deg 2, 4.512s)
2D Robot	$x \geq y$	$\mathbb{E}[T^2] < \infty$ (deg 2, 0.145s)	ub.	$1.7280(x-y)^2 + 31.4539(x-y) + 126.5167$ (deg 2, 0.132s)	$1.7280(x-y)^2 + 31.4539(x-y) + 126.5167$ (deg 2, 7.133s)
			lb.	$1.7280(x-y)^2 + 31.4539(x-y) + 29.7259$ (deg 2, 0.121s)	$1.7280(x-y)^2 + 31.4539(x-y)$ (deg 2, 7.040s)
Good Discount	$d \leq 30 \wedge$ $n \geq 1$	$\mathbb{E}[T^2] < \infty$ (deg 2, 0.093s)	ub.	$-0.5n - 3.6667d + 117.3333$ (deg 2, 0.093s)	$0.0067dn - 0.7n - 3.8035d + 0.0022d^2 + 119.4351$ (deg 2, 5.272s)
			lb.	$-0.005n^2 - 0.5n$ (deg 2, 0.092s)	$0.0067dn - 0.7133n - 3.8123d + 0.0022d^2 + 112.3704$ (deg 2, 5.323s)
Pollutant Disposal	$n \geq 0$	$\mathbb{E}[T^2] < \infty$ (deg 2, 0.091s)	ub.	$-0.2n^2 + 50.2n$ (deg 2, 0.092s)	$-0.2n^2 + 50.2n$ (deg 2, 5.851s)
			lb.	$-0.2n^2 + 50.2n - 435.6$ (deg 2, 0.094s)	$-0.2n^2 + 50.2n - 482$ (deg 2, 5.215s)
Species Fight	$a \geq 5 \wedge$ $b \geq 5$	$\mathbb{E}[T^2] < \infty$ (deg 2, 0.042s)	ub.	$40ab - 180a - 180b + 810$ (deg 2, 0.045s)	$40ab - 180a - 180b + 810$ (deg 3, 5.545s)
			lb.	N/A	N/A

Chapter 6

Proposed Work

My main goal in this thesis proposal is (i) enhancing the *usability* and *efficiency* of the central-moment analysis tool that I developed in Chapter 5, and (ii) reformulating it as a new *abstract domain* for the PMAF framework that I presented in Chapter 4. For the second proposal, I will also explore the relation between *iteration*-based and *constraint*-based static analysis. Beyond the two proposals, if time permits, I would like to explore how to use static-analysis techniques to aid the development of probabilistic programs used for *probabilistic inference*.

6.1 Enhance the Central Moment Analysis Implementation

In §5.5, I have described a prototype implementation of the central-moment analysis framework proposed in Chapter 5. Although the implementation has already supported recursive functions, continuous distributions, unstructured control-flow, and local variables, its *usability* is still not satisfactory because it can only handle arithmetic probabilistic programs with real-valued program variables. In fact, in the timing-attack case study presented in §5.6, I had to manually reimplement the analyzed program (shown in Fig. 5.15(b)) to remove all the array-related operations, whereas there was no guarantee for the correctness of the reimplementation process. Beside array operations (or more generally, heap manipulation), a usable programming language usually supports multiple data types, such as real numbers, integers, Booleans, etc. It is already nontrivial to support those features in a static analyzer for non-probabilistic programs, and probabilities may pose new challenges; for example, as discussed in §5.3.3, probabilistic termination is very different from and much trickier than non-probabilistic termination. Another concern is the *efficiency* of the central-moment analysis tool. Despite the fact that my prototype implementation achieved reasonable efficiency and scalability in the experimental evaluation (see §5.5), the worst-case time complexity can become exponential in terms of the number of program variables and the depth of the call graph. Therefore, enhancing the usability and efficiency

```
1  logical state invariant: 1 ≤ k ≤ N ∧ ∧_{j=k+1}^{N} (secret[j] = guess[j]) ∧ (secret[k] < guess[k])
2  func compare(guess, secret) begin
3    i := N; cmp := 0;
4    while i > 0 do
5      assert(i > k ⟹ secret[i] = guess[i]);
6      assert(i = k ⟹ secret[i] < guess[i]);
7      tick(2);
8      while i > 0 do
9        assert(i > k ⟹ secret[i] = guess[i]);
10       assert(i = k ⟹ secret[i] < guess[i]);
11       if prob(0.5) then break fi
12       tick(5);
13       if cmp > 0 ∨ (cmp = 0 ∧ guess[i] > secret[i])
14       then cmp := 1
15       else
16         tick(5);
17         if cmp < 0 ∨ (cmp = 0 ∧ guess[i] < secret[i])
18         then cmp := −1
19         fi
20       fi
21       tick(1);
22       i := i − 1
23     od
24   od;
25   return cmp
26 end
```

Fig. 6.1: Assisted moment-bound derivation using logical state. The derived upper bound on the variance of the accumulated cost is $8N - 36k^2 + 52Nk + 24k$.

is the main topic of this section.

Array Manipulation Soundly and precisely handling the program heap in static analysis is a hard problem and researchers have proposed many techniques to analyze heap manipulation (e.g., [59, 74]). I am considering using a relatively lightweight semi-automatic approach that is first used by Carbonneaux et al. [23]; the idea is to introduce a *logical state* using *auxiliary variables* that are used in bound derivation but do *not* influence program behavior. The auxiliary variables provide a mechanism for users to specify logical or quantitative invariants, which can reflect properties of the heap contents and thus aid the moment-bound derivation.

Fig. 6.1 illustrates the idea on the compare function from Fig. 5.15. The parts of the code in blue are user-provided annotations, whereas other code forms the original program. The logical variable k is introduced to specify a complicated array-related pre-condition that corresponds to the case where the attacker has successfully obtained the bits $secret[k+1]$

through $secret[N]$ but guessed the bit $secret[k]$ incorrectly. Because the program never mutates *secret* or *guess*, the assertions on lines 5, 6, 9, and 10 can be directly justified using the logical state invariant. With the extra information, the central-moment analysis tool should be able to derive a fine-grained upper bound (which involves k) on the variance of the accumulated cost.

More Basic Data Types It is often more desirable to support multiple data types in a programming language than just allowing real-valued program variables. Once one has multiple types in the system, the moment-bound analysis tool must find a proper way to reference program variables of different types in symbolic bounds. For example, consider the function f below with a Boolean-valued variable b:

```
1 func f() begin
2   if b then
3     if prob(0.4) then tick(1)
4     else tick(2) fi
5   else
6     if prob(0.6) then tick(1)
7     else tick(2) fi
8   fi
9 end
```

Instead of using 1.6 as an upper bound on the expected accumulated cost of the function, I would like to derive a more precise bound that uses b, i.e., $[b] \cdot 1.6 + [\neg b] \cdot 1.4$. To support such reasoning, I plan to adapt a systematic approach proposed by Hoffmann et al. [66] to express resource bounds in terms of values of different types for functional programs. The basic idea is to introduce for each type τ a set of *base monomials* that map values of type τ to real numbers, and then use those base monomials to construct polynomials that express moment bounds. To illustrate this approach, let us consider a simple type system as follows:

$$\tau ::= \mathsf{real} \mid \mathsf{bool} \mid \tau_1 \times \tau_2,$$

where $\tau_1 \times \tau_2$ is the binary product type, whose inhabitants are pairs of values of type τ_1 and τ_2. I now define base monomials $\mathcal{B}(\tau)$ for each type τ inductively:

$$\begin{aligned}
\mathcal{B}(\mathsf{real}) &\stackrel{\text{def}}{=} \{\lambda v.v\},\\
\mathcal{B}(\mathsf{bool}) &\stackrel{\text{def}}{=} \{\lambda v.v, \lambda v.\neg v\},\\
\mathcal{B}(\tau_1 \times \tau_2) &\stackrel{\text{def}}{=} \{\lambda(\langle v_1, v_2\rangle).b_1(v_1) \cdot b_2(v_2) \mid b_1 \in \mathcal{B}(\tau_1), b_2 \in \mathcal{B}(\tau_2)\}.
\end{aligned}$$

Then, for a context $\Gamma = x_1 : \tau_1, \cdots, x_n : \tau_n$ of variable-type bindings, I can define the set of monomials over program variables $x_1, \cdots, x_n$ as follows:

$$\mathcal{M}(\Gamma) \stackrel{\text{def}}{=} \{\lambda V. \prod_{i=1}^{n} b_i(V(x_i))^{d_i} \mid b_i \in \mathcal{B}(\tau_i), d_i \in \mathbb{Z}^+\},$$

and thus the polynomial moment bounds over Γ can be expressed as linear combinations of finitely many monomials from $\mathcal{M}(\Gamma)$.

Probability-Encapsulation Type Another interesting direction for extending the type system is to add a type that *encapsulates* probabilities [135]; that is, the programming language has a type for *non-constant* probabilities and programs can manipulate those probabilities and flip a coin with respect to them in probabilistic branching. The motivation for introducing a probability-encapsulation type is that although most of the static analyzers for probabilistic programs assume that probabilistic branches are of constant probabilities, *non-constant* probabilities can become useful in verification tasks. For example, let us consider the following function that simulates a fair coin using a coin that shows heads with probability p and we want to verify the function is correct for any $p \in (0, 1)$:

```
1 func fair() begin
2   if prob(p) then
3     if prob(p) then call fair()
4     else tick(1) fi # In this case, the outcome should be heads.
5   else
6     if prob(p) then skip # In this case, the outcome should be tails.
7     else call fair() fi
8   fi
9 end
```

Because there is exactly one unit of cost when the simulation function decides to return heads as the coin-flip outcome, the expected accumulated cost is exactly the probability that the function returns heads. Thus, it is desirable for the moment-bound analysis tool to derive $\frac{1}{2}$ as both the lower and the upper bounds on the expected accumulated cost for any possible coin-flip probability p.

I am considering adapting the approach by Wang et al. [135] that adds a probability-encapsulation type to a functional programming language. I can extend the simpler type system mentioned earlier in this section with a primitive probability type as follows:

$$\tau ::= \mathsf{real} \mid \mathsf{bool} \mid \mathsf{prob} \mid \tau_1 \times \tau_2,$$

where the introduction form for values of type prob simply takes a number $p \in (0, 1)$, and the elimination form is a probabilistic branch. Also, I can define the set of base monomials for prob by

$$\mathcal{B}(\mathsf{prob}) \stackrel{\text{def}}{=} \{\lambda p.p, \lambda p.(1-p)\}.$$

Furthermore, it might be useful to incorporate the ability to multiply and complement encapsulated probabilities in the programming language. The major challenge is then to add inference rules for the new type and new program constructs, as well as effectively automate newly added rules in the moment analysis tool.

Neededness Analysis Besides the analysis capability, the analysis efficiency is also an important aspect to evaluate a static analyzer. Recall that my central-moment analysis

framework uses polynomials over program variables as the templates for moment bounds; therefore, even with a fixed maximal degree on the polynomials, the number of possible monomials grows *exponentially* in terms of the number of program variables. It is obviously redundant to use all program variables to express the moment bound at every program point. For example, consider the program below:

```
1 while x < d do
2   t ~ Uniform(−1, 2);
3   x := x + t;
4   tick(1)
5 od
```

There are three program variables x, d, and t, but not all variables are *needed* to encode the moment bound at every program location; for example, the variable t is just used to save a temporary value drawn from a distribution, so it is only needed between the time it is created and the time it is used, i.e., between line 2 and line 3 of the code. I am considering implementing an intra-procedural data-flow analysis, which should be similar to liveness analysis, to evaluate the *neededness* of every program variable at every program point for expressing moment bounds.

Tunable Context Sensitivity Another source of the exponential time complexity of the moment analysis is my treatment of *context sensitivity*, i.e., function calls to the same function at different call sites can use different function specifications in the bound derivation. More specifically, in my prototype implementation, I collapse the cycles in the call graph and analyze each function at least once for every path in the resulting graph. This implementation strategy can make the moment analysis extremely slow when the call graph has a complex structure.

I am considering adapting different treatments for context sensitivity of static analysis, and then the research question is to figure out what kind of context sensitivity is suitable for my central moment analysis. One possibility is to use *k-limiting* context sensitivity [124], i.e., the static analysis does not distinguish two function calls if both have the same k immediate call sites. Another possibility is to devise function summaries that can be reused at all call sites. The major challenge here is that my central-moment analysis framework is constraint-based, which is different from standard abstract interpretation. I would like to adapt Jost et al. [78]'s approach to handling polymorphic resource bounds for functional programs; that is, instead of re-analyzing a function at different call sites, I can analyze every function once and record the linear constraints generated by the derivation system, and then instantiate those constraints with fresh variables to analyze different call sites.

6.2 Instantiate PMAF for Central Moment Reasoning

In Chapter 4, I have proposed a general framework PMAF for static analysis of probabilistic programs, but later in Chapter 5 I did *not* use PMAF in the development of the central-

moment analysis framework. One reason for this is that the analysis algorithm of PMAF (see §4.2.3) is *iteration* based, whereas the central-moment analysis algorithm (see §5.2.4) is *constraint* based. The main topic in this section is to establish a connection between PMAF and the central moment analysis, e.g., by designing a new abstract domain for PMAF to track central moments of cost accumulators. Then the research question is that if iteration-based and constraint-based methods can be integrated together to enhance the performance of the central moment analysis.

There are two possibilities I would like to explore. The first approach I am considering is to extend the abstract domain for linear expectation-invariant analysis, or LEIA (presented in §4.3.3), in a similar way that I extend the expected potential method to reason about higher moments of cost accumulators in Chapter 5. The challenge is that as discussed in §4.3.3, the probabilistic termination complicates the analysis of expectations and I have to make many conservative design choices for LEIA. On the other hand, one observation from Chapter 5 is that the Optional Stopping Theorem (OST) can be used to reason about probabilistic programs with almost-sure termination, and it has been applied in static analysis of probabilistic loops (e.g., [62]). Thus, I propose to use OST to develop a new summarization technique for loops in probabilistic programs. For example, consider the following program:

```
1 # pre-condition: x < d
2 while x < d do
3   if prob(3/4) then
4     x := x + 1
5   else
6     x := x − 1
7   fi
8   tick(1)
9 od
```

Let *tick* denote the accumulated cost and let us consider the problem of deriving the expected accumulated cost. Using an abstract domain that is similar to LEIA, one should be able to derive for the loop body that $\mathbb{E}[x'] = x + \frac{1}{2}$, $tick' = tick + 1$, $x' \leq d$, and $|x' - x| \leq 1$, where non-primed versions of the program variables stand for the initial values, and the primed versions stand for the final values after the loop body is executed. To construct a loop summary, we notice that for the loop body, $\mathbb{E}[2x' - tick'] = (2x + 1) - (tick + 1) = 2x - tick$ is an expectation invariant, and updates to both x and *tick* are bounded, thus by applying a proper sufficient condition of OST, we can derive for the whole loop that $x' = d$ and $\mathbb{E}[2x' - tick'] = 2x - tick$, and therefore $\mathbb{E}[tick'] = tick + 2(d - x)$. However, it remains a challenge to incorporate loop summarization in PMAF. A possible approach to applying loop summarization on unstructured programs is using Tarjan's path-expression algorithm [125] on the control-flow graph to obtain a regular expression that encodes all possible program executions, and then reinterpreting the regular expression in a way that the Kleene-star operation is interpreted as loop summarization. This approach has been successfully applied to analyze non-probabilistic programs (e.g., [48, 84, 119]). The gap here is that PMAF uses *hyper-graphs* to encode control-flow graphs, so I cannot use Tarjan's algorithm for

path-expression construction. One possible way to bridge the gap is to construct *regular-tree expressions* [31] to encode all hyper-paths on a hyper-graph. I am also considering a restricted case where the control-flow hyper-graphs can be equivalently broken down to normal graphs; for example, I would like to remove nondeterminism from the programming language, and then adapt a slightly different semantic algebra from probabilistic NetKAT [51], which takes the form of a Kleene algebra.

Another approach I am considering is to develop recurrence relations for central moment analysis and then use recurrence solvers (e.g., [85, 86]) to derive moment bounds. Because recurrence relations are also used to analyze loops, this approach shares the same challenge about path-expression construction on hyper-graphs as the previous approach. One benefit of recurrence relations is that one does not need to specify a set of solution templates, therefore one may derive complex non-polynomial bounds, such as exponential bounds. I observe the need for such complex bounds in applications that track position information in control systems. For example, consider the program below, which models a simple lane-keeping controller:

```
1 θ := 0;
2 i := 0;
3 while i < n do
4   tick(0.1 · θ);
5   θ := 0.8 · θ + UnknownDist([−0.01, 0.01], 0, 0.001);
6   # an unknown distribution with support [−0.01, 0.01], 1st moment 0, and 2nd moment 0.001
7   i := i + 1
8 od
```

In the program, I use the cost accumulator to keep track of the distance between the car and the middle of the lane. In each time unit, the car changes its position with respect to its angle θ, and then updates the angle *randomly*. In this example, the number of loop iterations is totally deterministic, thus I can extract recurrence relations about moments of the variable θ and the accumulated cost *tick* as follows, where I use a subscript k to denote the value at the end of the k-th iteration:

$$\begin{aligned} \theta_0 &= 0, \\ \mathbb{E}[\theta_{k+1} \mid \theta_k] &= 0.8 \cdot \theta_k, \qquad & \mathbb{E}[\theta_{k+1}^2 \mid \theta_k] &= 0.64 \cdot \theta_k^2 + 0.001, \\ tick_0 &= 0, \\ \mathbb{E}[tick_{k+1} \mid tick_k, \theta_k] &= tick_k + 0.1 \cdot \theta_k, & \mathbb{E}[tick_{k+1}^2 \mid tick_k, \theta_k] &= tick_k^2 + 0.2 \cdot tick_k \cdot \theta_k + 0.01 \cdot \theta_k^2. \end{aligned}$$

Because of the *tower property* in probability theory, i.e., $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]]$, where X and Y are two random variables, and $\mathbb{E}[X \mid Y]$ is a conditional expectation, it is sound to "ignore" the expectation operator in the recurrence relations, thus I obtain the results below using an existing recurrence solver:

$$\begin{aligned} \mathbb{E}[\theta'] &= 0, & \mathbb{E}[\theta'^2] &= \frac{1}{360} - \frac{1}{360} \cdot (\frac{16}{25})^n, \\ \mathbb{E}[tick'] &= 0, & \mathbb{E}[tick'^2] &= -\frac{1}{12960} + \frac{1}{12960} \cdot (\frac{16}{25})^n + \frac{1}{36000} \cdot n. \end{aligned}$$

```
1 k ~ Gaussian(0, 2);
2 b ~ Gaussian(0, 2);
3 σ ~ Gamma(1, 1);
4 i := 0;
5 while i < n do
6    y' ~ Gaussian(k · xs[i] + b, σ);
7    observe(ys[i] = y');
8    i := i + 1
9 od
```

Fig. 6.2: A probabilistic program that describes a linear-regression model.

Recall that *tick* keeps track of the position of the car. Therefore, combining with concentration inequalities (see §5.4), these moment bounds can be used to reason about tail bounds such as $\mathbb{P}[tick' \geq 0.1]$.

6.3 Further Proposed Work

Nowadays, probabilistic programs are also used to express and perform *probabilistic inference*, a method of inferring a statistical model from observed data, which is good at capturing uncertainty in model parameters and integrating expert domain knowledge (e.g., *a priori* distributions), but requires considerable expertise from the practitioner [56]. For example, the code in Fig. 6.2 describes the following linear-regression model between y and x,

$$y \sim \mathcal{N}(k \cdot x + b, \sigma),$$

where the prior on the parameters k,b, and σ is $k \sim \mathcal{N}(0, 2)$, $b \sim \mathcal{N}(0, 2)$ and $\sigma \sim \Gamma(1, 1)$. The primitive **observe**(φ) conditions on the predicate φ that is usually used to fit the model to observed data. Probabilistic programming systems take in the model program and the data xs, ys, then perform Bayesian inference to obtain a posterior distribution of the parameters k, b and σ, *conditioned* on the given data xs, ys. There have been a lot of probabilistic programming languages (PPL) in active development [3, 15, 24, 58, 100, 128, 134], but balancing between *language expressibility* and *inference efficiency* remains a challenge for the design and implementation of PPLs.

If time permits, I would like to improve the inference efficiency *without* restricting language expressibility or compromising on high-level language clarity, through developing static-analysis techniques. One promising direction is to use static analysis to detect initialization bugs for *Markov-chain Monte Carlo* (MCMC) algorithms. MCMC is a commonly-used family of sampling methods for probabilistic inference. Given a probabilistic program, an MCMC algorithm usually starts with randomly picking an execution path from the program that satisfies all the conditioning predicates in the program. For general-purpose PPLs, it is

sometimes hard to come up with a satisfactory path. For example, the program in Fig. 6.2 is *not* suitable for MCMC to perform inference because the probability that the sampling statement on line 6 returns exactly $ys[i]$ is zero. A standard workaround is to transform the *hard* constraint **observe**$(ys[i] = y')$ to a *soft* one, i.e., weighting the trace with the likelihood that the data point $ys[i]$ is sampled from the Gaussian distribution. However, as the size and complexity of the program increase, it is not always obvious *where* and *how* to transform the program. A static analyzer may take in a probabilistic program and a path condition (e.g., the probability of the execution path is greater than some threshold), then either generates a path with the condition satisfied, or indicates program fragments that may invalidate the condition. Further, in the latter case, the tool may be able to synthesize a patch, which can come from formal deduction or real-world code corpus, to transform the program to an equivalent one with some execution paths satisfying the condition.

Chapter 7

Timeline

PhD Timeline for 2021–22

Sep – Oct 2021	Enhance the implementation for central-moment analysis
Nov – Dec 2021	Instantiate PMAF for central-moment reasoning
Sep – Dec 2021	Apply for faculty positions
Jan – Apr 2022	Write thesis

Bibliography

[1] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2005. Control-Flow Integrity. In *Comp. and Comm. Sec. (CCS'05)*. https://doi.org/10.1145/1102120.1102165

[2] Samson Abramsky and Achim Jung. 1995. Domain Theory. In *Handbook of Logic in Computer Science (Vol. 3): Semantic Structures*. Oxford University Press, Inc. https://dl.acm.org/doi/10.5555/218742.218744

[3] Jessica Ai, Nimar S. Arora, Ning Dong, Beliz Gokkaya, Thomas Jiang, Anitha Kubendran, Arun Kumar, Michael Tingley, and Narjes Torabi. 2019. HackPPL: A Universal Probabilistic Programming Language. In *Int. Workshop on Machine Learning and Prog. Lang. (MAPL'19)*. https://doi.org/10.1145/3315508.3329974

[4] 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, 2 (April 1997). https://doi.org/10.1023/A:1008699807402

[5] Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. 2001. Boolean and Cartesian Abstraction for Model Checking C Programs. In *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'01)*. https://doi.org/10.1007/3-540-45319-9_19

[6] 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

[7] Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. An Assertion-Based Program Logic for Probabilistic Programs. In *European Symp. on Programming (ESOP'18)*. https://doi.org/10.1007/978-3-319-89884-1_5

[8] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal Certification of Code-Based Cryptographic Proofs. In *Princ. of Prog. Lang. (POPL'09)*. https://doi.org/10.1145/1480881.1480894

[9] Ezio Bartocci, Laura Kovács, and Miroslav Stankovič. 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

[10] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2018. How long, O Bayesian network, will I sample thee?. In *European Symp. on Programming (ESOP'18)*. https://doi.org/10.1007/978-3-319-89884-1_7

[11] Mihir Bellare and Phillip Rogaway. 1994. Optimal Asymmetric Encryption. In *Int. Conf. on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'94)*. https://doi.org/10.1007/BFb0053428

[12] Richard Bellman. 1957. A Markovian Decision Process. *J. Mathematics and Mechanics* 6, 5 (1957). https://www.jstor.org/stable/24900506

[13] Benjamin Bichsel, Timon Gehr, and Martin Vechev. 2018. Fine-Grained Semantics for Probabilistic Programs. In *European Symp. on Programming (ESOP'18)*. https://doi.org/10.1007/978-3-319-89884-1_6

[14] Patrick Billingsley. 2012. *Probability and Measure*. John Wiley & Sons, Inc.

[15] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rishabh Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. 2018. Pyro: Deep Universal Probabilistic Programming. *J. Machine Learning Research* 20, 1 (January 2018). https://dl.acm.org/doi/10.5555/3322706.3322734

[16] Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A Lambda-Calculus Foundation for Universal Probabilistic Programming. In *Int. Conf. on Functional Programming (ICFP'16)*. https://doi.org/10.1145/2951913.2951942

[17] 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

[18] François Bourdoncle. 1993. Efficient Chaotic Iteration Strategies With Widenings. In *Formal Methods in Prog. and Their Applications*. https://doi.org/10.1007/BFb0039704

[19] Tomá Brázdil, Stefan Kiefer, and Antonín Kučera. 2014. Efficient Analysis of Probabilistic Programs with an Unbounded Counter. *J. ACM* 61, 6 (November 2014). https://doi.org/10.1145/2629599

[20] Tomá Brázdil, Stefan Kiefer, Antonín Kučera, and Ivana Hutařová Vařeková. 2015. Runtime Analysis of Probabilistic Programs with Unbounded Recursion. *J. Comput. Syst. Sci.* 81, 1 (February 2015). https://doi.org/10.1016/j.jcss.2014.06.005

[21] Jason Breck, John Cyphert, Zachary Kincaid, and Thomas Reps. 2020. Templates and Recurrences: Better Together. In *Prog. Lang. Design and Impl. (PLDI'20)*. https://doi.org/10.1145/3385412.3386035

[22] Quentin Carbonneaux, 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

[23] Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. 2015. Compositional Certified Resource Bounds. In *Prog. Lang. Design and Impl. (PLDI'15)*. https://doi.org/10.1145/2737924.2737955

[24] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *J. Statistical Softw.* 76, 1 (2017). https://doi.org/10.18637/jss.v076.i01

[25] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verif. (CAV'13)*. https://doi.org/10.1007/978-3-642-39799-8_34

[26] Aleksandar Chakarov and Sriram Sankaranarayanan. 2014. Expectation Invariants for Probabilistic Program Loops as Fixed Points. In *Static Analysis Symp. (SAS'14)*. https://doi.org/10.1007/978-3-319-10936-7_6

[27] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination Analysis of Probabilistic Programs Through Positivstellensatz's. In *Computer Aided Verif. (CAV'16)*. https://doi.org/10.1007/978-3-319-41528-4_1

[28] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. In *Princ. of Prog. Lang. (POPL'16)*. https://doi.org/10.1145/2837614.2837639

[29] Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. 2017. Stochastic Invariants for Probabilistic Termination. In *Princ. of Prog. Lang. (POPL'17)*. https://doi.org/10.1145/3093333.3009873

[30] 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)*. https://doi.org/10.1145/2491411.2491423

[31] 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.

[32] Sylvain Conchon, Jean-Christophe Filliâtre, and Julien Signoles. 2007. Designing a Generic Graph Library Using ML Functors. In *Trends in Functional Programming (TFP'07)*.

[33] John Horton Conway. 1971. *Regular Algebra and Finite Machines*. London: Chapman and Hall.

[34] Patrick Cousot. 1981. Semantic Foundations of Program Analysis. In *Program Flow Analysis: Theory and Applications*. Prentice-Hall, Inc.

[35] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Latice 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

[36] Patrick Cousot and Radhia Cousot. 1978. Static Determination of Dynamic Properties of Recursive Procedures. In *Formal Descriptions of Programming Concepts*.

[37] Patrick Cousot and Radhia Cousot. 1979. Systematic Design of Program Analysis Frameworks. In *Princ. of Prog. Lang. (POPL'79)*. https://doi.org/10.1145/567752.567778

[38] Patrick Cousot and Nicolas Halbwachs. 1978. Automatic Discovery of Linear Constraints Among Variables of a Program. In *Princ. of Prog. Lang. (POPL'78)*. https://doi.org/10.1145/512760.512770

[39] Patrick Cousot and Michael Monerau. 2012. Probabilistic Abstract Interpretation. In *European Symp. on Programming (ESOP'12)*. https://doi.org/10.1007/978-3-642-28869-2_9

[40] Eva Darulova and Viktor Kuncak. 2014. Sound Compilation of Reals. In *Princ. of Prog. Lang. (POPL'14)*. https://doi.org/10.1145/2578855.2535874

[41] J. I. den Hartog and Erik P. de Vink. 1999. Mixing Up Nondeterminism and Probability: a preliminary report. *Electr. Notes Theor. Comp. Sci.* 22 (1999). https://doi.org/10.1016/S1571-0661(05)82521-6 PROBMIV'98, First International Workshop on Probabilistic Methods in Verification.

[42] Edsger W. Dijkstra. 1997. *A Discipline of Programming*. Prentice Hall PTR. https://dl.acm.org/doi/book/10.5555/550359

[43] 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

[44] Thomas Ehrhard, Michele Pagani, and Christine Tasson. 2017. Measurable Cones and Stable, Measurable Functions. *Proc. ACM Program. Lang.* 2, POPL (December 2017). https://doi.org/10.1145/3158147

[45] 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

[46] 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

[47] Kousha Etessami and Mihalis Yannakakis. 2015. Recursive Markov Decision Processes and Recursive Stochastic Games. *J. ACM* 62, 2 (May 2015). https://doi.org/10.1145/2699431

[48] 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

[49] Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Princ. of Prog. Lang. (POPL'15)*. https://doi.org/10.1145/2676726.2677001

[50] Robert W. Floyd. 1967. Assigning Meanings to Programs. *Symposium on Applied Mathematics* 19 (1967).

[51] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. 2016. Probabilistic NetKAT. In *European Symp. on Programming (ESOP'16)*. https://doi.org/10.1007/978-3-662-49498-1_12

[52] Björn Franke, Michael O'Boyle, John Thomson, and Grigori Fursin. 2005. Probabilistic Source-Level Optimisation of Embedded Programs. In *Lang., Comp., and Tools for Embeded Syst. (LCTES'05)*. https://doi.org/10.1145/1070891.1065922

[53] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. 1993. Directed Hypergraphs and Applications. *Disc. Appl. Math.* 42, 2 (April 1993). https://doi.org/10.1016/0166-218X(93)90045-P

[54] Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verif. (CAV'16)*. https://doi.org/10.1007/978-3-319-41528-4_4

[55] Zoubin Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature* 521 (May 2015). https://doi.org/10.1038/nature14541

[56] Noah D. Goodman. 2013. The Principles and Practice of Probabilistic Programming. In *Princ. of Prog. Lang. (POPL'13)*. https://doi.org/10.1145/2429069.2429117

[57] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith A. Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A language for generative models. In *Uncertainty in Artificial Intelligence (UAI'08)*. https://dl.acm.org/doi/10.5555/3023476.3023503

[58] Noah D. Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. Available on http://dippl.org.

[59] Denis Gopan, Frank DiMaio, Nurit Dor, Thomas Reps, and Shmuel Sagiv. 2004. Numeric Domains with Summarized Dimensions. In *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'04)*.

[60] Susanne Graf and Hassen Saidi. 1997. Construction of Abstract State Graphs with PVS. In *Computer Aided Verif. (CAV'97)*. https://doi.org/10.1007/3-540-63166-6_10

[61] Nicolas Halbwachs. 1979. *Détermination automatique de relations linéaires vérifiées par les variables d'un programme*. Ph.D. Dissertation. Université Joseph-Fourier.

[62] Marcel Hark, Benjamin Lucien Kaminski, Jürgen Giesl, and Joost-Pieter Katoen. 2019. Aiming Low Is Harder: Induction for Lower Bounds in Probabilistic Program Verification. *Proc. ACM Program. Lang.* 4, POPL (December 2019). https://doi.org/10.1145/3371105

[63] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A Convenient Category for Higher-Order Probability Theory. In *Logic in Computer Science (LICS'17)*. https://doi.org/10.1109/LICS.2017.8005137

[64] C. A. R. Hoare. 1962. Quicksort. *Comput. J.* 5, 1 (January 1962). https://doi.org/10.1093/comjnl/5.1.10

[65] C. A. R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (October 1969). https://doi.org/10.1145/363235.363259

[66] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. 2011. Multivariate Amortized Resource Analysis. In *Princ. of Prog. Lang. (POPL'11)*. https://doi.org/10.1145/1926385.1926427

[67] Jan Hoffmann, Ankush Das, and Shu-Chun Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *Princ. of Prog. Lang. (POPL'17)*. https://doi.org/10.1145/3009837.3009842

[68] Jan Hoffmann and Martin Hofmann. 2010. Amortized Resource Analysis with Polymorphic Recursion and Partial Big-Step Operational Semantics. In *Asian Symp. on Prog. Lang. and Systems (APLAS'10)*. https://doi.org/10.1007/978-3-642-17164-2_13

[69] Karl H. Hofmann and Michael W. Mislove. 1981. Local Compactness and Continuous Lattices. In *Continuous Lattices*. https://doi.org/10.1007/BFb0089908

[70] Martin Hofmann and Steffen Jost. 2003. Static Prediction of Heap Space Usage for First-Order Functional Programs. In *Princ. of Prog. Lang. (POPL'03)*. https://doi.org/10.1145/604131.604148

[71] Susan Horwitz, Thomas Reps, and Mooly Sagiv. 1995. Demand Interprocedural Dataflow Analysis. In *Found. of Softw. Eng. (FSE'95)*. https://doi.org/10.1145/222132.222146

[72] Frederick James and Lorenzo Moneta. 2020. Review of High-Quality Random Number Generators. *Computing and Software for Big Science* 4, 2 (January 2020). https://doi.org/10.1007/s41781-019-0034-3

[73] Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Federico Olmedo, Friedrich Gretz, and Annabelle K. McIver. 2015. Conditioning in Probabilistic Programming. *Electr. Notes Theor. Comp. Sci.* 319 (December 2015). https://doi.org/10.1016/j.entcs.2015.12.013 The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

[74] Bertrand Jeannet, Denis Gopan, and Thomas Reps. 2005. A Relational Abstraction for Functions. In *Static Analysis Symp. (SAS'05)*. https://doi.org/10.1007/11547662_14

[75] Bertrand Jeannet and Antoine Miné. 2009. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *Computer Aided Verif. (CAV'09)*. https://doi.org/10.1007/978-3-642-02658-4_52

[76] Claire Jones. 1989. *Probabilistic Nondeterminism*. Ph.D. Dissertation. University of Edinburgh.

[77] Claire Jones and Gordon D. Plotkin. 1989. A Probabilistic Powerdomain of Evaluations. In *Logic in Computer Science (LICS'89)*. https://doi.org/10.1109/LICS.1989.39173

[78] Steffen Jost, Kevin Hammond, Hans-Wolfgang Loidl, and Martin Hofmann. 2010. Static Determination of Quantitative Resource Usage for Higher-Order Programs. In *Princ. of Prog. Lang. (POPL'10)*. https://doi.org/10.1145/1706299.1706327

[79] Achim Jung and Regina Tix. 1998. The Troublesome Probabilistic Powerdomain. *Electr. Notes Theor. Comp. Sci.* 13 (1998). https://doi.org/10.1016/S1571-0661(05)80216-6 Comprox III, Third Workshop on Computation and Approximation.

[80] 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)*. https://doi.org/10.1007/978-3-662-49498-1_15

[81] 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)*. https://doi.org/10.1007/978-3-642-15769-1_24

[82] Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. 2009. Abstraction Refinement for Probabilistic Software. In *Verif., Model Checking, and Abs. Interp. (VMCAI'09)*. https://doi.org/10.1007/978-3-540-93900-9_17

[83] Gary A. Kildall. 1973. A Unified Approach to Global Program Optimization. In *Princ. of Prog. Lang. (POPL'73)*. https://doi.org/10.1145/512927.512945

[84] Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas Reps. 2017. Compositional Recurrence Analysis Revisited. In *Prog. Lang. Design and Impl. (PLDI'17)*. https://doi.org/10.1145/3062341.3062373

[85] Zachary Kincaid, Jason Breck, John Cyphert, and Thomas Reps. 2019. Closed Forms for Numerical Loops. *Proc. ACM Program. Lang.* 3, POPL (January 2019). https://doi.org/10.1145/3290368

[86] Zachary Kincaid, John Cyphert, Jason Breck, and Thomas Reps. 2017. Non-Linear Reasoning for Invariant Synthesis. *Proc. ACM Program. Lang.* 2, POPL (December 2017). https://doi.org/10.1145/3158142

[87] S. C. Kleene. 1951. Representation of Events in Nerve Nets and Finite Automata. Available on https://www.rand.org/pubs/research_memoranda/RM704.html.

[88] Jens Knoop and Bernhard Steffen. 1992. The Interprocedural Coincidence Theorem. In *Comp. Construct. (CC'92)*. https://doi.org/10.1007/3-540-55984-1_13

[89] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press. https://dl.acm.org/doi/book/10.5555/1795555

[90] Dexter Kozen. 1981. On Induction vs. *-Continuity. In *Logics of Programs*. https://doi.org/10.1007/BFb0025782

[91] Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. Syst. Sci.* 22, 3 (June 1981). https://doi.org/10.1016/0022-0000(81)90036-2

[92] Dexter Kozen. 1985. A Probabilistic PDL. *J. Comput. Syst. Sci.* 30, 2 (April 1985). https://doi.org/10.1016/0022-0000(85)90012-1

[93] Dexter Kozen. 1991. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. In *Logic in Computer Science (LICS'91)*. https://doi.org/10.1109/LICS.1991.151646

[94] 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

[95] Akash Lal, Thomas Reps, and Gogul Balakrishnan. 2005. Extended Weighted Pushdown Systems. In *Computer Aided Verif. (CAV'05)*. https://doi.org/10.1007/11513988_44

[96] Akash Lal, Tayssir Touili, Nicholas Kidd, and Thomas Reps. 2008. Interprocedural Analysis of Concurrent Programs Under a Context Bound. In *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'08)*. https://doi.org/10.1007/978-3-540-78800-3_20

[97] 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

[98] Gurobi Optimization LLC. 2020. Gurobi Optimizer Reference Manual. Available on https://www.gurobi.com.

[99] Guido Manfredi and Yannick Jestin. 2016. An Introduction to ACAS Xu and the Challenges Ahead. In *Digital Avionics Systems Conference (DASC'16)*. https://doi.org/10.1109/DASC.2016.7778055

[100] Vikash K. Mansinghka, Ulrich Schaechtle, Shivam Handa, Alexey Radul, Yutian Chen, and Martin C. Rinard. 2018. Probabilistic Programming with Programmable Inference. In *Prog. Lang. Design and Impl. (PLDI'18)*. https://doi.org/10.1145/3296979.3192409

[101] Annabelle K. McIver and Carroll C. Morgan. 2001. Partial correctness for probabilistic demonic programs. *Theor. Comp. Sci.* 266, 1 (September 2001). https://doi.org/10.1016/S0304-3975(00)00208-5

[102] 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

[103] Antoine Miné. 2006. Symbolic Methods to Enhance the Precision of Numerical Abstract Domains. In *Verif., Model Checking, and Abs. Interp. (VMCAI'06)*. https://doi.org/10.1007/11609773_23

[104] Michael W. Mislove. 1998. Topology, domain theory and theoretical computer science. *Topology and its Applications* 89, 1 (November 1998). https://doi.org/10.1016/S0166-8641(97)00222-8

[105] Michael W. Mislove. 2000. Nondeterminism and Probabilistic Choice: Obeying the Laws. In *Int. Conf. on Concurrency Theory (CONCUR'00)*. https://doi.org/10.1007/3-540-44618-4_26

[106] Michael W. Mislove, Joël Ouaknine, and James Worrell. 2004. Axioms for Probability and Nondeterminism. *Electr. Notes Theor. Comp. Sci.* 96 (June 2004). https://doi.org/10.1016/j.entcs.2004.04.019 Proceedings of the 10th International Workshop on Expressiveness in Concurrency.

[107] David Monniaux. 2000. Abstract Interpretation of Probabilistic Semantics. In *Static Analysis Symp. (SAS'00)*. https://doi.org/10.1007/978-3-540-45099-3_17

[108] David Monniaux. 2001. Backwards Abstract Interpretation of Probabilistic Programs. In *European Symp. on Programming (ESOP'01)*. https://doi.org/10.1007/3-540-45309-1_24

[109] David Monniaux. 2003. Abstract Interpretation of Programs as Markov Decision Processes. In *Static Analysis Symp. (SAS'03)*. https://doi.org/10.1007/3-540-44898-5_13

[110] Markus Mottl. 2017. Lacaml - Linear Algebra for OCaml. Available on https://github.com/mmottl/lacaml.

[111] Markus Müller-Olm and Helmut Seidl. 2004. Precise Interprocedural Analysis through Linear Algebra. In *Princ. of Prog. Lang. (POPL'04)*. https://doi.org/10.1145/964001.964029

[112] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded Expectations: Resource Analysis for Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'18)*. https://doi.org/10.1145/3192366.3192394

[113] Van Chan Ngo, Mario Dehesa-Azuara, Matthew Fredrikson, and Jan Hoffmann. 2017. Verifying and Synthesizing Constant-Resource Implementations with Types. In *Symp. on Sec. and Privacy (SP'17)*. https://doi.org/10.1109/SP.2017.53

[114] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about Recursive Probabilistic Programs. In *Logic in Computer Science (LICS'16)*. https://doi.org/10.1145/2933575.2935317

[115] Brooks Paige and Frank Wood. 2014. A Compilation Target for Probabilistic Programming Languages. In *Int. Conf. on Machine Learning (ICML'14)*. https://dl.acm.org/doi/10.5555/3044805.3045108

[116] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. https://dl.acm.org/doi/book/10.5555/528623

[117] Ganesan Ramalingam. 1996. *Bounded Incremental Computation*. Springer-Verlag. https://doi.org/10.1007/BFb0028290

[118] Thomas Reps, Susan Horwitz, and Mooly Sagiv. 1995. Precise Interprocedural Dataflow Analysis via Graph Reachability. In *Princ. of Prog. Lang. (POPL'95)*. https://doi.org/10.1145/199448.199462

[119] Thomas Reps, Emma Turetsky, and Prathmesh Prabhu. 2016. Newtonian Program Analysis via Tensor Product. In *Princ. of Prog. Lang. (POPL'16)*. https://doi.org/10.1145/2837614.2837659

[120] Reuven Y. Rubinstein and Dirk P. Kroese. 2016. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc. https://doi.org/10.1002/9781118631980

[121] Mooly Sagiv, Thomas Reps, and Susan Horwitz. 1996. Precise Interprocedural Dataflow Analysis with Applications to Constant Propagation. *Theor. Comp. Sci.* 167, 1 (1996). https://doi.org/10.1016/0304-3975(96)00072-2

[122] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. 2013. Static Analysis for Probabilistic Programs: Inferring Whole Program Properties from Finitely Many Paths. In *Prog. Lang. Design and Impl. (PLDI'13)*. https://doi.org/10.1145/2491956.2462179

[123] Anne Schreuder and Luke Ong. 2019. Polynomial Probabilistic Invariants and the Optional Stopping Theorem. https://arxiv.org/abs/1910.12634

[124] Micha Sharir and Amir Pnueli. 1981. Two Approaches to Interprocedural Data Flow Analysis. In *Program Flow Analysis: Theory and Applications*. Prentice-Hall, Inc.

[125] Robert Endre Tarjan. 1981. A Unified Approach to Path Problems. *J. ACM* 28, 3 (July 1981). https://doi.org/10.1145/322261.322272

[126] Robert Endre Tarjan. 1985. Amortized Computational Complexity. *SIAM J. Algebraic Discrete Methods* 6, 2 (August 1985). https://doi.org/10.1137/0606031

[127] Regina Tix, Klaus Keimel, and Gordon D. Plotkin. 2009. Semantic Domains for Combining Probability and Non-Determinism. *Electr. Notes Theor. Comp. Sci.* 222 (February 2009). https://doi.org/10.1016/j.entcs.2009.01.002

[128] Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. 2017. Deep Probabilistic Programming. In *Int. Conf. on Learning Representations (ICLR'17)*.

[129] Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A Domain Theory for Statistical Probabilistic Programming. *Proc. ACM Program. Lang.* 3, POPL (January 2019). https://doi.org/10.1145/3290349

[130] 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

[131] Di Wang, Jan Hoffmann, and Thomas Reps. 2019. 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.

[132] 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

[133] Di Wang, Jan Hoffmann, and Thomas Reps. 2021. Expected-Cost Analysis for Probabilistic Programs and Semantics-Level Adaption of Optional Stopping Theorems. https://arxiv.org/abs/2103.16105

[134] Di Wang, Jan Hoffmann, and Thomas Reps. 2021. Sound Probabilistic Inference via Guide Types. In *Prog. Lang. Design and Impl. (PLDI'21)*. https://doi.org/10.1145/3453483.3454077

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

[136] Peng Wang, Santiago Cuellar, and Adam Chlipala. 2014. Compiler Verification Meets CrossLanguage Linking via Data Abstraction. In *Object-Oriented Prog., Syst., Lang., and Applications (OOPSLA'14)*. https://doi.org/10.1145/2660193.2660201

[137] Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost Analysis of Nondeterministic Probabilistic Programs. In *Prog. Lang. Design and Impl. (PLDI'19)*. https://doi.org/10.1145/3314221.3314581

[138] Website. 2014. Infer.NET - Microsoft Research. Available on https://www.microsoft.com/en-us/research/project/infernet.

[139] Website. 2015. Space/Time Analysis for Cybersecurity (STAC). Available on https://www.darpa.mil/program/space-time-analysis-for-cybersecurity.

[140] David Williams. 1991. *Probability with Martingales*. Cambridge University Press. https://doi.org/10.1017/CBO9780511813658

[141] Dominik Wojtczak and Kousha Etessami. 2017. PReMo – Probabilistic Recursive Models analyzer. Available on https://groups.inf.ed.ac.uk/premo.