Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 347 (2019) 303–324

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

# A Denotational Semantics for Low-Level Probabilistic Programs with Nondeterminism

Di Wang,[a] Jan Hoffmann[a] and Thomas Reps[b,c]

[a] *Carnegie Mellon University*

[b] *University of Wisconsin*

[c] *GrammaTech, Inc.*

**Abstract**

Probabilistic programming is an increasingly popular formalism for modeling randomness and uncertainty. Designing semantic models for probabilistic programs has been extensively studied, but is technically challenging. Particular complications arise when trying to account for (i) unstructured control-flow, a natural feature in low-level imperative programs; (ii) general recursion, an extensively used programming paradigm; and (iii) nondeterminism, which is often used to represent adversarial actions in probabilistic models, and to support refinement-based development. This paper presents a denotational-semantics framework that supports the three features mentioned above, while allowing nondeterminism to be handled in different ways. To support both probabilistic choice and nondeterministic choice, the semantics is given over control-flow *hyper*-graphs. The semantics follows an *algebraic* approach: it can be instantiated in different ways as long as certain algebraic properties hold. In particular, the semantics can be instantiated to support nondeterminism among either *program states* or *state transformers*. We develop a new formalization of nondeterminism based on *powerdomains* over *sub-probability kernels*. Semantic objects in the powerdomain enjoy a notion we call *generalized convexity*, which is a generalization of convexity. As an application, the paper sketches an algebraic framework for static analysis of probabilistic programs, which has been proposed in a companion paper.

*Keywords:* Probabilistic programming, denotational semantics, control-flow hyper-graphs, nondeterminism, powerdomains

## 1 Introduction

Probabilistic programming provides a powerful framework for implementing randomized algorithms [2], cryptographic protocols [3], cognitive models [31], and machine-learning algorithms [29]. One important focus of recent studies on probabilistic programming is to reason *rigorously* about probabilistic programs and systems. The first step in such works 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 [8], higher-order functions [32,20], and recursive

https://doi.org/10.1016/j.entcs.2019.09.016

types [63], we observe that *low*-level features could arise naturally. For example, when developing a compiler for a probabilistic programming language [26,56], we need a semantics for the imperative target language to prove compiler correctness. There have been studies on *denotational* semantics for *well-structured* imperative programs [43,44,47,48,62,34,38,55,7], as well as *operational* semantics for *control-flow graphs* (CFGs) based on Markov chains (MCs) and Markov decision processes (MDPs) ([25,14,15]). On the one hand, we 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 rigorous 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, in this paper, we devise a denotational semantics for low-level probabilistic programs. Our work makes three main contributions:

- We use *hyper-graphs* as the representation for low-level probabilistic programs with unstructured control-flow, general recursion, and nondeterminism.
- We 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*.
- We devise an *algebraic* framework for denotational semantics. The advantage of having a framework is that it can be instantiated with different models of nondeterminism. We show how to instantiate the framework using two different approaches to formalizing nondeterminism in Ex. 5.2. We also show that for programs without procedure calls and nondeterminism, the resulting denotational semantics is equivalent to a distribution-based operational semantics (§5.2).

We define the denotational semantics *directly* as an interpretation of the *control-flow hyper-graphs* (CFHGs) of low-level probabilistic programs, introduced in §2. 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, our 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."

Some high-level decision choices about *nondeterminism* arise when we are developing the low-level semantics. Nondeterminism itself is an important feature from two perspectives: (i) it arises naturally from probabilistic models, such as the agent for an MDP [6], or the unknown input distribution for modeling *fault tolerance* [40], and (ii) it is required by the common paradigm of *abstraction* and *refinement* [1] on programs [19,48]. 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 [18,50,47,51,48,62]. 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. We call the common model *nondeterminism-last* and the other *nondeterminism-first*. In §4, we present a domain-theoretic study of nondeterminism-first. Technically, we 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 our ultimate goal of developing a denotational semantics, instead of restricting ourselves to one specific model for nondeterminism, we propose a general *algebraic* denotational semantics in §5, 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. In §6, we sketch a static-analysis framework proposed in a companion paper [64], as an application of the denotational semantics.

The *algebraic* approach we take in this paper 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* [41,16,42,45], 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's path-expression algorithm [61]; and

[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.

$$\textbf{if } \star \textbf{ then } \textbf{if } \textbf{prob}(1/2) \textbf{ then } t \coloneqq 0 \textbf{ else } t \coloneqq 1 \textbf{ fi}$$
$$\textbf{else } \textbf{if } \textbf{prob}(1/3) \textbf{ then } t \coloneqq 0 \textbf{ else } t \coloneqq 1 \textbf{ fi } \textbf{ fi}$$

Fig. 1. A nondeterministic, probabilistic program

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 with a *nondeterministic* choice $\star$ in Fig. 1. 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.

Although the denotational semantics proposed in this paper supports interesting features including unstructured control-flow, general recursion, and nondeterminism, there are some other important features that the semantics does not support *yet*, such as continuous distributions and higher-order functions. We discuss those missing features in §7, and leave them for future work.

## 2 An Operational Semantics for Low-Level Probabilistic Programs

In this section, we sketch an operational semantics for an imperative, single-procedure, deterministic,[2] probabilistic programming language, following the approach of Borgström et al.'s distribution-based semantics [8]. We use the operational semantics to (i) illustrate how to model executions of probabilistic programs operationally, and (ii) justify the development of a denotational semantics in later sections.

### *2.1 A Hyper-Graph Program Model*

We define the operational semantics on CFHGs of programs. We 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., [24,54,46]). Instead of standard directed graphs, we make use of *hyper-graphs* [27].

**Definition 2.1** 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*

[2] The term "deterministic" is used in the sense "not nondeterministic."

*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 $v^{\mathsf{entry}}$ does not have incoming edges, and $v^{\mathsf{exit}}$ has no outgoing edges.

**Definition 2.2** 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 $v_i^{\mathsf{exit}}$ has *at least* one outgoing hyper-edge, and $v_i^{\mathsf{exit}}$ 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* $\mathsf{Act}$ and *deterministic conditions* $\mathsf{Cond}$ 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:

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

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 2.3** Fig. 2(b) shows the CFHG of the program in Fig. 2(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.

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 $\mathsf{Act}$ and deterministic conditions $\mathsf{Cond}$ that could be used for an arithmetic program, where $p \in [0, 1]$, $c \in \mathbb{Q}$, $a, b \in \mathbb{Z}$, and $n \in \mathbb{N}$.

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

$\mathsf{Dist}$ stands for a collection of discrete probability distributions. For example, $\mathsf{Binomial}(n, p)$ 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 $p$; $\mathsf{Uniform}(a, b)$ represents a discrete uniform distribution on $[a, b] \cap \mathbb{Z}$.

### *2.2 A Distribution-Based Small-Step Operational Semantics*

The next step is to define a semantics based on CFHGs. We adopt Borgström et al.'s distribution-based small-step operational semantics for lambda calculus [8] to our hyper-graph setting, while we suppress the features of multiple procedures and nondeterminism for now.

Three components are used to define the semantics:

- A *program state space* $\Omega$, e.g., for arithmetic programs, we can define $\Omega \stackrel{\text{def}}{=} \mathsf{Var} \rightharpoonup_{\text{fin}} \mathbb{Q}$, i.e., a set of finite partial maps from program variables to their values.

$n \coloneqq 0;$

**while** $\mathbf{prob}(0.5) \wedge \mathbf{prob}(0.5)$ **do**

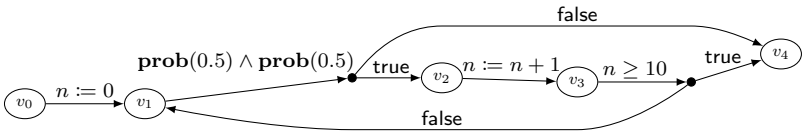
  $n \coloneqq n + 1;$

  **if** $n \geq 10$ **then break**

  **else continue**

**od**

(a)




(b)

Fig. 2. (a) An example of probabilistic programs; (b) The corresponding CFHG

- A function $[\![\mathsf{act}]\!]$ from program states to *(sub-probability) distributions* over program states for each data action $\mathsf{act}$. A distribution is a function $\Delta : \Omega \to [0,1]$ such that $\sum_{\omega \in \Omega} \Delta(\omega) \leq 1$. Intuitively, $[\![\mathsf{act}]\!](\omega)(\omega')$ is the probability that the action $\mathsf{act}$, starting in state $\omega \in \Omega$, halts in a state $\omega' \in \Omega$ [44].
- A $[0,1]$-valued function $[\![\varphi]\!]$ from program states for each deterministic condition $\varphi$. Intuitively, $[\![\varphi]\!](\omega)$ is the probability that the condition $\varphi$ holds in state $\omega \in \Omega$.

The *point distribution* $\delta(\omega)$ is defined as $\lambda\omega'.[\omega = \omega']$ where $[\psi]$ is an *Iverson bracket* that evaluates to 1 if $\psi$ is true and 0 otherwise. If $\Delta$ is a distribution and $r \in [0,1]$, we write $r \cdot \Delta$ for the distribution $\lambda\omega.r \cdot \Delta(\omega)$. If $\Delta_1, \Delta_2$ are distributions and $r_1, r_2 \in [0,1]$ satisfy $r_1 + r_2 \leq 1$, we write $r_1 \cdot \Delta_1 + r_2 \cdot \Delta_2$ for the distribution $\lambda\omega.r_1 \cdot \Delta_1(\omega) + r_2 \cdot \Delta(\omega)$.

Fig. 3 shows interpretation of the data actions and deterministic conditions given in §2.1, where $\omega(e)$ evaluates expression $e$ in state $\omega$, $[x \mapsto v]\omega$ updates $x$ in $\omega$ with $v$, and $\Delta_D : \mathbb{Q} \to [0,1]$ is the *probability mass function* of the distribution $D$. If $\varphi$ does not contain any probabilistic choices $\mathbf{prob}(p)$, then $[\![\varphi]\!](\omega)$ is either 0 or 1. Intuitively, $[\![\varphi]\!](\omega)$ is the probability that $\varphi$ is true in the state $\omega$, w.r.t. a probability space specified by all the $\mathbf{prob}(p)$'s in $\varphi$. Then the probability of $\varphi_1 \wedge \varphi_2$ is defined as the product of the individual probabilities of $\varphi_1$ and $\varphi_2$, because $\varphi_1$ and $\varphi_2$ are interpreted w.r.t. probabilistic choices in $\varphi_1$ and $\varphi_2$, 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 $v^{\mathsf{exit}}$ 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

$$\begin{array}{|rl|rl|}\hline [\![x \coloneqq e]\!] \stackrel{\text{def}}{=} \lambda\omega.\delta([x \mapsto \omega(e)]\omega) & [\![\mathbf{skip}]\!] \stackrel{\text{def}}{=} \lambda\omega.\delta(\omega) & [\![\top]\!] \stackrel{\text{def}}{=} \lambda\omega.1 & [\![\neg\varphi]\!] \stackrel{\text{def}}{=} \lambda\omega.1 - [\![\varphi]\!](\omega) \\ [\![x \sim D]\!] \stackrel{\text{def}}{=} \lambda\omega.\sum_{v\in\text{supp}(\Delta_D)} \Delta_D(v)\cdot\delta([x\mapsto v]\omega) & & [\![\mathbf{prob}(p)]\!] \stackrel{\text{def}}{=} \lambda\omega.p & [\![e_1 \le e_2]\!] \stackrel{\text{def}}{=} \lambda\omega.[\omega(e_1) \le \omega(e_2)] \\ [\![\mathbf{observe}(\varphi)]\!] \stackrel{\text{def}}{=} \lambda\omega.[\![\varphi]\!](\omega)\cdot\delta(\omega) & & & [\![\varphi_1 \wedge \varphi_2]\!] \stackrel{\text{def}}{=} \lambda\omega.[\![\varphi_1]\!](\omega)\cdot[\![\varphi_2]\!](\omega) \\ \hline\end{array}$$

Fig. 3. Interpretation of data actions and deterministic conditions

node in the CFHG, and $\omega \in \Omega$ is a program state.

We define *one-step evaluation* as a relation $\langle v, \omega\rangle \longrightarrow \Delta$ between configurations $\langle v, \omega\rangle$ and distributions $\Delta$ on configurations, as shown in Fig. 4.

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

Fig. 4. One-step evaluation relation

**Example 2.4** For the program in Fig. 2, 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 $\langle v_3, n \mapsto 9\rangle \longrightarrow \delta(\langle v_1, \{n \mapsto 9\}\rangle)$.

We now define *step-indexed evaluation* as the family of $n$-indexed relations $\langle v, \omega\rangle \longrightarrow_n \Delta$ between configurations $\langle v, \omega\rangle$ and distributions $\Delta$ on program states inductively, as shown in Fig. 5.

$$\begin{array}{ll} \langle v,\omega\rangle \longrightarrow_0 \lambda\omega'.0 & \\ \langle v^{\mathsf{exit}},\omega\rangle \longrightarrow_n \delta(\omega) & \text{if } n>0 \\ \langle v,\omega\rangle \longrightarrow_{n+1} \sum_{\tau\in\text{supp}(\Delta)} \Delta(\tau)\cdot\Delta'_\tau & \text{where} \langle v,\omega\rangle \longrightarrow \Delta \text{ and } \tau \longrightarrow_n \Delta'_\tau \text{ for any } \tau\in\text{supp}(\Delta) \end{array}$$

Fig. 5. Step-indexed evaluation relation

**Example 2.5** For the program in Fig. 2, 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 $\langle v_1, \{n \mapsto 0\}\rangle \longrightarrow_5 0.75\cdot\delta(\{n \mapsto 0\}) + 0.1875\cdot\delta(\{n \mapsto 1\})$.

For the program $P = \langle V, E, v^{\mathsf{entry}}, v^{\mathsf{exit}}\rangle$, we define its semantics $[\![P]\!]_{\mathsf{os}}(\omega) \stackrel{\text{def}}{=} \sup_{n\in\mathbb{N}}\{\Delta \mid \langle v^{\mathsf{entry}},\omega\rangle \longrightarrow_n \Delta\}$.

**Example 2.6** For the program $P$ in Fig. 2, $[\![P]\!]_{\mathsf{os}}(\omega)$ for any initial state $\omega$ with $n \in \text{dom}(\omega)$ is given by $\sum_{k=0}^{9}(0.75\times 0.25^k)\cdot\delta([n\mapsto k]\omega) + 0.00000095367431640625\cdot\delta([n\mapsto 10]\omega)$.

### *2.3 Why is a Denotational Semantics Desirable?*

We have already shown how probabilistic programs execute *operationally*. As mentioned in §1, we are instead interested in developing a *denotational* semantics, which concentrates on the *effects* of programs and abstracts from how the program executes. This characterization of denotational semantics is indeed 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 could develop *local*—and thus *scalable*—reasoning techniques based on a denotational semantics. In contrast, the operational semantics in §2.2 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. In the rest of this section, we 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 §2.2, one needs to introduce a notion of *stacks* to keep track of procedure calls, as in [22,23,55]. 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. 4 and 5 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 $C$. If $C_1$ is indeed 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., [25,14,15]). When the scheduler is fixed, a program can be executed deterministically (as shown in §2.2). 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 should 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.

# 3 A Summary of Existing Domain-Theoretic Developments

Our development of models for nondeterminism makes great use of existing domain-theoretic studies of powerdomains, thus in this section, we present a brief summary of them. We review some standard notions from domain theory [33,1,49], as well as some results on probabilistic powerdomains [36,35] and nondeterministic powerdo-

mains [18,50,47,51,48,62].

### 3.1 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 $\downarrow A \stackrel{\text{def}}{=} \{x \in P \mid \exists a \in A\colon x \sqsubseteq a\}$. The *upper closure* of a subset $A$ is defined as $\uparrow A \stackrel{\text{def}}{=} \{x \in P \mid \exists a \in A\colon a \sqsubseteq x\}$. 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 *$\omega$-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 $\mathop{\downarrow} A \stackrel{\text{def}}{=} \{x \in D \mid \exists a \in A\colon x \ll a\}$ and $\mathop{\uparrow} A \stackrel{\text{def}}{=} \{x \in D \mid \exists a \in A\colon a \ll x\}$. 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 $\mathcal{O}(X)$. A *cover* $\mathcal{C}$ of a subset $A$ of $X$ is a collection of subsets whose union contains $A$ as a subset. A *sub-cover* of $\mathcal{C}$ is a subset of $\mathcal{C}$ that still covers $A$. The cover $\mathcal{C}$ 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.

We are going to use the following theorems in our technical development.

**Proposition 3.1 (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 $\omega$-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 3.2 (Cor. of [33, 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.*

### *3.2 Probabilistic Powerdomains*

Jones et al.'s pioneer work on probabilistic powerdomains [36,35] extends the complete partially ordered sets, which are pervasively used in computer science, to model probabilistic computations. Let $X$ be a nonempty countable set. The set of all distributions on $X$ is denoted by $\underline{\mathcal{D}}(X)$, i.e., a *probabilistic powerdomain* over $X$. Recall that a distribution on $X$ is a function $\Delta : X \to [0,1]$ such that $\sum_{x\in X} \Delta(x) \leq 1$, and the point distribution $\delta(x)$ for some $x \in X$ is defined as $\lambda x'.[x = x']$. Distributions are ordered pointwise, i.e., $\Delta_1 \sqsubseteq_D \Delta_2 \stackrel{\text{def}}{=} \forall x \in X : \Delta_1(x) \leq \Delta_2(x)$. We define the *probabilistic-choice* of distributions $\Delta_1, \Delta_2$ 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 following theorems provide a characterization of the probabilistic powerdomains.

**Proposition 3.3 ([36,35,47,62])** *The poset $\langle \underline{\mathcal{D}}(X), \sqsubseteq_D \rangle$ forms a coherent dcpo with a countable basis $\{\sum_{i=1}^n r_i \cdot \delta(x_i) \mid n \in \mathbb{N} \wedge r_i \in \mathbb{Q}_0^+ \wedge \sum_{i=1}^n r_i \leq 1 \wedge x_i \in X\}$. 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 3.4 ([35,62])** *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 $\widehat{f} : \underline{\mathcal{D}}(X) \to \underline{\mathcal{D}}(X)$.*

### *3.3 Nondeterministic Powerdomains*

When nondeterminism comes into the picture, as we discussed in §1, existing studies usually resolve program inputs *prior to* nondeterminism [37,18,50,47,51,48,62]. In §1, we 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 3.5 ([47,62])** *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* $conv(A \cup B)$. *Intuitively, the formal union operation stands for nondeterministic choices.*

**Proposition 3.6 ([62])** *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 3.7** 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 semantic object 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 $t \in \mathbb{Q}$.

# 4 Nondeterminism-First

In this section, we 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 §1), 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.

We first introduce *kernels*, then propose a new notion of *generalized convexity* (*g-convexity*, for short), and finally develop a powerdomain for nondeterminism-first.

## *4.1 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 4.1** *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.*

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 $\kappa$ is a kernel and $\phi \in \mathbb{W}(X)$, we write $\phi \cdot \kappa$ for the kernel $\lambda x.\phi(x) \cdot \kappa(x)$. If $\kappa_1, \kappa_2$ 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 $\bigsqcup_{n \in \mathbb{N}}^{\uparrow} \sum_{i=1}^{n} \phi_i \cdot \kappa_i$. Then we define *conditional-choice* of kernels $\kappa_1, \kappa_2$ 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 $\kappa_1, \kappa_2$ 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 4.2** (i) *The conditional-choice operation ${}_{\phi}\!\diamond$ is Scott-continuous for all $\phi \in \mathbb{W}(X)$.*

(ii) *The composition operation $\otimes$ is Scott-continuous.*

### *4.2 Generalized Convexity*

As shown in §3.3, 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 $\phi \in \mathbb{W}(X)$. 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. We formalize the idea by defining a notion of *g-convexity*.

**Definition 4.3** 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 $A$.

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

**Lemma 4.4** *Let $A$ be a g-convex subset of $\underline{\mathcal{K}}(X)$. Then*

(i) *The saturation $\uparrow A$ and the lower closure $\downarrow A$ are g-convex.*

(ii) *The closure $\overline{A}$ is g-convex.*

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.

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

**Lemma 4.5** *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 $\phi \in \mathbb{W}(X)$.*

**Corollary 4.6** *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. 4.5. □

For a finite subset $F$ of $\underline{\mathcal{K}}(X)$, as an immediate corollary of Cor. 4.6, 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 4.7** *For an arbitrary $A \subseteq \mathcal{K}(X)$, we have*

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

**Lemma 4.8** (i) *For an arbitrary $A \subseteq \underline{\mathcal{K}}(X)$, we have $\overline{gconv(A)} = \overline{gconv\left(\overline{A}\right)}$.*

(ii) *If $\{A_i\}_{i\in\mathcal{I}}$ is a directed collection of Scott-closed subsets of $\underline{\mathcal{K}}(X)$ ordered by set inclusion, then $\overline{gconv\left(\bigcup A_i\right)} = \bigcup \overline{gconv(A_i)}$.*

**Lemma 4.9** *Let $A$ and $B$ be Scott-compact g-convex subsets of $\underline{\mathcal{K}}(X)$. Then $gconv(A \cup B)$ is also Scott-compact.*

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

**Lemma 4.10** (i) *If $A \subseteq \underline{\mathcal{K}}(X)$ is g-convex, then for all $x$, $\{\kappa(x) \mid \kappa \in A\}$ is convex.*

(ii) *If $A \subseteq \underline{\mathcal{K}}(X)$ is Scott-compact, then for all $x$, $\{\kappa(x) \mid \kappa \in A\}$ is Scott-compact.*

(iii) *If $A \subseteq \underline{\mathcal{K}}(X)$ is Scott-closed, then for all $x$, $\{\kappa(x) \mid \kappa \in A\}$ is Scott-closed.*

**Lemma 4.11** *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 $A$.*

**Lemma 4.12** *If $K \subseteq \underline{\mathcal{K}}(X)$ is nonempty and Scott-compact, then $gconv(K)$ is Scott-compact.*

### *4.3 A g-convex Powerdomain for Nondeterminism-First*

From the literature, a *Plotkin powertheory* [1] 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, we hope to develop a powerdomain on

$\underline{\mathcal{K}}(X)$, i.e., kernels on $X$. To achieve this goal, we 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 [1,47,62], we 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 $A \sqsubseteq_G B \stackrel{\text{def}}{=} A \subseteq {\downarrow}B \wedge {\uparrow}A \supseteq B$. We 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 4.13** *$\langle \mathcal{G}\underline{\mathcal{K}}(X), \sqsubseteq_G\rangle$ forms a dcpo, with a least element $\bot_G \stackrel{\text{def}}{=} \{\bot_K\}$.*

We now lift conditional-choice ${}_{\phi}\Diamond$ (where $\phi \in \mathbb{W}(X)$) and composition $\otimes$ for kernels to the powerdomain $\mathcal{G}\underline{\mathcal{K}}(X)$ 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 4.4 and 4.12. As conditional-choice and composition operations are Scott-continuous on kernels, the lifted operations are also Scott-continuous in the powerdomain.

**Lemma 4.14** *The operations ${}_{\phi}\Diamond_G$ and $\otimes_G$ are Scott-continuous for all $\phi \in \mathbb{W}(X)$.*

Finally, we define a *formal union* operation $\uplus_G$ as in Prop. 3.5 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 4.15** *The formal union $\uplus_G$ is a Scott-continuous semilattice operation on $\mathcal{G}\underline{\mathcal{K}}(X)$.*

**Example 4.16** Recall the probabilistic program $P$ in Ex. 3.7:

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

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 $\textbf{prob}(r)$), 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 $\Delta_1$ starting with $t = t_1$ and $\Delta_2$ with $t = t_2$.

With the g-convex powerdomain $\mathcal{G}\underline{\mathcal{K}}(X)$ for 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. 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 we started with the convex powerdomain $\mathcal{P}\underline{\mathcal{D}}(X)$ reviewed in §3.3 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. 3.7. Now the refinements of $P$ include some $\kappa$ 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 $\mathbb{E}[t_1'-t_2']=t_1-t_2$.

# 5 An Algebraic Denotational Semantics

The operational semantics described in §2.2 presents a reasonable model for evaluating single-procedure probabilistic programs without nondeterminism. In this section, we develop a general denotational semantics for CFHGs (introduced in §2.1) 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 §3.3, as well as nondeterminism-first developed in §4.3. We will show the denotational semantics is equivalent to the operational semantics in §2.2 if we suppress procedure calls and nondeterminism in the programming model.

## *5.1 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 $\mathsf{act}\in\mathsf{Act}$. The semantic algebras that we use are *Markov algebras* introduced in [64]:

**Definition 5.1** A *Markov algebra* (MA) over a set $\mathsf{Cond}$ 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 $\varphi\in\mathsf{Cond}$ we have $a\ {}_{\varphi}\diamond_M\ b\le_M a\uplus_M b$ where $\le_M$ is the semilattice ordering induced by $\uplus_M$ (i.e., $a\le_M b$ if $a\uplus_M b=b$); and $\otimes_M,{}_{\varphi}\diamond_M,\uplus_M$ are Scott-continuous.

**Example 5.2** Let $\Omega$ be a nonempty countable set of program states and $\mathsf{Cond}$ be a set of deterministic conditions, the definition and meaning of which are given in §2.1 and §2.2.

(i) The convex powerdomain $\mathcal{P}\underline{\mathcal{D}}(\Omega)$ admits an MA $\langle\Omega\to\mathcal{P}\underline{\mathcal{D}}(\Omega),\dot{\sqsubseteq}_P,\otimes_P,{}_{\varphi}\diamond_P,\dot{\uplus}_P,\dot{\bot}_P,1_P\rangle$, where $\dot{\sqsubseteq}_P,\dot{\uplus}_P,\dot{\bot}_P$ are pointwise extensions of $\sqsubseteq_P,\uplus_P,\bot_P$, defined in §3.3, and $g\otimes_P h\stackrel{\text{def}}{=}\widehat{\widehat{h}}\circ g$ where $\widehat{\widehat{h}}$ is given by Prop. 3.6, $g\ {}_{\varphi}\diamond_P\ h\stackrel{\text{def}}{=}\lambda\omega.g(\omega)\ {}_{[\![\varphi]\!](\omega)}\oplus_P\ h(\omega)$, as well as $1_P\stackrel{\text{def}}{=}\lambda\omega.\{\delta(\omega)\}$.

(ii) The g-convex powerdomain $\mathcal{G}\underline{\mathcal{K}}(\Omega)$ admits an MA $\langle\mathcal{G}\underline{\mathcal{K}}(\Omega),\sqsubseteq_G$

$, \otimes_G, {}_\varphi\Diamond_G, \uplus_G, \bot_G, 1_G\rangle$, where $\sqsubseteq_G, \otimes_G, {}_\varphi\Diamond_G, \uplus_G, \bot_G$ come from §4.3,[3] and $1_G \stackrel{\text{def}}{=} \{\lambda\omega.\delta(\omega)\}$.

**Definition 5.3** An *interpretation* is a pair $\mathscr{I} = \langle \mathcal{M}, [\![\cdot]\!]^{\mathscr{I}} \rangle$, where $\mathcal{M}$ is an MA and $[\![\cdot]\!]^{\mathscr{I}} : \mathsf{Act} \to \mathcal{M}$. We call $\mathcal{M}$ the *semantic algebra* of the interpretation and $[\![\cdot]\!]^{\mathscr{I}}$ the *semantic function*.

**Example 5.4** We can lift the interpretation of data actions defined in Fig. 3 to semantic functions with respect to convex or g-convex powerdomains—$\mathscr{P} = \langle \mathcal{P}\underline{\mathcal{D}}(\Omega), [\![\cdot]\!]^{\mathscr{P}} \rangle$ with $[\![\mathsf{act}]\!]^{\mathscr{P}} \stackrel{\text{def}}{=} \lambda\omega.\{[\![\mathsf{act}]\!](\omega)\}$ and $\mathscr{G} = \langle \mathcal{G}\underline{\mathcal{K}}(\Omega), [\![\cdot]\!]^{\mathscr{G}} \rangle$ with $[\![\mathsf{act}]\!]^{\mathscr{G}} \stackrel{\text{def}}{=} \{[\![\mathsf{act}]\!]\}$.

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

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

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

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

The least fixpoint of $F_P$ exists by Prop. 3.1 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 5.5** *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 $\otimes_M$, ${}_\varphi\Diamond_M$, and $\uplus_M$. □

### *5.2 An Equivalence Result*

To justify the denotational semantics proposed in §5.1, we go back to the restricted programming language used to define the operational semantics in §2.2. 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 5.6** *Let $P = \langle V, E, v^{\mathsf{entry}}, v^{\mathsf{exit}} \rangle$ be a deterministic single-procedure probabilistic program.*

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

(ii) *If we interpret $P$ using $\mathscr{G} = \langle \mathcal{G}\underline{\mathcal{K}}(\Omega), [\![\cdot]\!]^{\mathscr{G}} \rangle$, we will have $[\![P]\!]_{\mathsf{ds}} = \{[\![P]\!]_{\mathsf{os}}\}$.*

**Proof.** Recall the definition $[\![P]\!] \stackrel{\text{def}}{=} \lambda\omega.\sup_{n \in \mathbb{N}} \{\Delta \mid \langle v^{\mathsf{entry}}, \omega \rangle \longrightarrow_n \Delta\}$. On the other hand, the fixpoint $(\mathrm{lfp}^{\dot{\sqsubseteq}_M}_{\dot{\bot}_M} F_P)(v_i^{\mathsf{entry}})$ is actually obtained by

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

$\bigsqcup^{\uparrow}_{n\in\mathbb{N}} F^n_P(\dot{\bot}_M)(v_i^{\mathsf{entry}})$ by Prop. 3.1. The proof proceeds by induction on $n$. □

# 6 Application: Static Analysis for Probabilistic Programs with Nondeterminism

A lot of recent studies on probabilistic programming focus on rigorous reasoning about probabilistic programs (e.g., [52,53,39,4,17,11,57,12,10,34,28,13,38,55,9,5]). In this section, we discuss an application of the new denotational semantics as the concrete semantics of a static-analysis framework for probabilistic programs. More details about the static analysis and its soundness proof can be found in a companion paper [64].

**Definition 6.1** A *pre-Markov algebra* (PMA) over a set $\mathsf{Cond}$ of deterministic conditions is a 7-tuple $\mathcal{M}^\sharp = \langle M, \sqsubseteq_M, \otimes_M, {}_\varphi\diamondsuit_M, \uplus_M, \bot_M, 1_M\rangle$, which is essentially an MA, except that $\langle M, \sqsubseteq_M\rangle$ forms a complete lattice, and $\otimes_M$, ${}_\varphi\diamondsuit_M$, and $\uplus_M$ are only required to be monotone.

Intuitively, PMAs specify *abstract* semantics used in static analyses. We can define interpretations with respect to PMAs in the same way, except that we obtain the least fixpoint $\mathscr{I}^\sharp[P]$ of the function $F_P$ by the Knaster-Tarski theorem, given a probabilistic program $P$ and an interpretation $\mathscr{I} = \langle \mathcal{M}^\sharp, [\![\cdot]\!]^{\mathscr{I}}\rangle$.

**Definition 6.2** A *probabilistic over-abstraction* (resp., *under-abstraction*) from an MA $\mathcal{C}$ (i.e., a concrete semantics such as $\mathcal{PD}(\Omega)$ and $\mathcal{G}\underline{\mathcal{K}}(\Omega)$) to a PMA $\mathcal{Y}$ is a concretization mapping, $\gamma : Y \to C$, such that

- $\bot_C \sqsubseteq_C \gamma(\bot_Y)$ (resp., $\gamma(\bot_Y) \sqsubseteq_C \bot_C$),
- $1_C \sqsubseteq_C \gamma(1_Y)$ (resp., $\gamma(1_Y) \sqsubseteq_C 1_C$),
- for all $Q_1, Q_2 \in Y$, $\gamma(Q_1) \otimes_C \gamma(Q_2) \sqsubseteq_C \gamma(Q_1 \otimes_Y Q_2)$ (resp., $\gamma(Q_1 \otimes_Y Q_2) \sqsubseteq_C \gamma(Q_1) \otimes_C \gamma(Q_2)$),
- for all $Q_1, Q_2 \in Y$, $\gamma(Q_1)\ {}_\varphi\diamondsuit_C\ \gamma(Q_2) \sqsubseteq_C \gamma(Q_1\ {}_\varphi\diamondsuit_Y\ Q_2)$ (resp., $\gamma(Q_1\ {}_\varphi\diamondsuit_Y\ Q_2) \sqsubseteq_C \gamma(Q_1)\ {}_\varphi\diamondsuit_C\ \gamma(Q_2)$), and
- for all $Q_1, Q_2 \in Y$, $\gamma(Q_1) \uplus_C \gamma(Q_2) \sqsubseteq_C \gamma(Q_1 \uplus_Y Q_2)$, (resp., $\gamma(Q_1 \uplus_Y Q_2) \sqsubseteq_C \gamma(Q_1) \uplus_C \gamma(Q_2)$).

A probabilistic abstraction leads to a sound analysis:

**Theorem 6.3** *Let $\mathscr{C}$ and $\mathscr{Y}$ be interpretations over an MA $\mathcal{C}$ and a PMA $\mathcal{Y}$; let $\gamma$ be a probabilistic over-abstraction (resp., under-abstraction) from $\mathcal{C}$ to $\mathcal{Y}$; and let $P$ be an arbitrary program. If for all data actions* $\mathsf{act}$, $[\![\mathsf{act}]\!]^{\mathscr{C}} \sqsubseteq_C \gamma([\![\mathsf{act}]\!]^{\mathscr{Y}})$ *(resp., $\gamma([\![\mathsf{act}]\!]^{\mathscr{Y}}) \sqsubseteq_C [\![\mathsf{act}]\!]^{\mathscr{C}}$), then we have $\mathscr{C}[P] \dot{\sqsubseteq}_C \dot{\gamma}(\mathscr{Y}^\sharp[P])$ (resp., $\dot{\gamma}(\mathscr{Y}^\sharp[P]) \dot{\sqsubseteq}_C \mathscr{C}[P]$).*

# 7 Discussion

## *7.1 Continuous Distributions*

One of the most important features of probabilistic programming is *continuous* probability distributions over real numbers, such as Gaussian distributions. Notions from measure theory, such as *measures* and *kernels*, are extensively used to model continuous distributions in probabilistic programming. Kozen studied the relation between deterministic probabilistic programs and continuous distributions via a metric on measures [43]. Many approaches use probability kernels [44,58], sub-probability kernels [8], and s-finite kernels [59,7]. A different approach uses measurable functions $A \to \mathcal{D}(\mathbb{R}_{\geq 0} \times B)$ where $\mathcal{D}(S)$ stands for the set of all probability measures on $S$ [60]. For higher-order languages, Jones and Plotkin [35,36] have developed a probabilistic powerdomain that consists of continuous *evaluations*, which are a reformulation of distributions in domain theory, on a state space. They show that the powerdomain can be used to solve recursive domain equations. Smolka et al. [58] study the semantics of probabilistic networks. Ehrhard et al. [20] provide a Cartesian-closed category on stable and measurable maps between cones, and use it to give a semantics for probabilistic PCF.

However, those measure-theoretic developments do not work properly when nondeterminism comes into the picture. To overcome this challenge, people have been adapting domain-theoretic results. McIver and Morgan build a Plotkin-style powerdomain over probability distributions on a discrete state space [47,48]. Mislove et al. [50,51] study powerdomain constructions for probabilistic CSP. Tix et al. [62] generalize McIver and Morgan's results to continuous state spaces, and construct three powerdomains for the extended probabilistic powerdomains. Although there has been a lot of work on this direction, one has to keep in mind that the domain-theoretic notion of "continuous" distributions is different from the notion in measure theory—instead, the domain-theoretic studies are focused on *computable* distributions. In other words, real numbers are realized by some computable models, such as *partial reals* [21]. These models would become unsatisfactory when one wants to *observe* a random value drawn from a continuous distribution, e.g., the meaning of $x \coloneqq \mathsf{Normal}(0,1); \mathbf{if}\ x = 0\ \mathbf{then}\ \cdots\ \mathbf{fi}$ is not expressible. We leave the semantic development of combining nondeterminism and continuous distributions (from a measure-theoretic perspective) for future work.

## *7.2 Higher-Order Functions*

In functional programming, higher-order functions are functions that can take functions as arguments, as well as return a function as a result. Some probabilistic programming languages, such as Church [30], are indeed functional programming languages and can express higher-order functions. While operational models for probabilistic functional programming have been proposed [8], developing a denotational semantics for higher-order probabilistic programming has been an open problem for years.

The major challenge is to propose a Cartesian-closed category for semantic objects

of probabilistic programming. Intuitively, the Cartesian-closure property ensures that if type $A$ and type $B$ are two objects in the category, then the function space $B^A$ (i.e., an object for the arrow type $A \to B$) is also contained in the category. The category of measures is clearly *not* Cartesian-closed; a lot of probabilistic powerdomains also do *not* admit a Cartesian-closed category [37]. Recently, Heunen et al. [32] propose quasi-Borel measures for higher-order functions in probabilistic programming. The measure-theoretic approach is further extended by Vákár et al. [63] to support recursive types. However, it is unclear how to model nondeterminism in the framework of quasi-Borel measures. We leave the combination of nondeterminism and higher-order functions for future work.

## 8 Conclusion

We have developed a framework for denotational semantics of low-level probabilistic programs with unstructured control-flow, general recursion, and nondeterminism, represented by control-flow hyper-graphs. The semantics is algebraic and it can be instantiated with different models of nondeterminism. We have demonstrated two instantiations with nondeterminism-first and nondeterminism-last, respectively. We have proposed a powerdomain for nondeterminism-first that consists of collections of kernels and enjoys generalized convexity. As an application, we have reviewed a static-analysis framework for probabilistic programs, which has been proposed in a companion paper.

In the future, we plan to combine continuous distributions and higher-order functions with nondeterminism in our semantics framework. We will also work on models of nondeterminism, especially nondeterminism-first, and investigate its connection with relational reasoning. Another research direction is to develop more formal reasoning techniques based on the denotational semantics.

## Acknowledgement

This work was supported, in part, by a gift from Rajiv and Ritu Batra; by AFRL under DARPA MUSE award FA8750-14-2-0270, DARPA STAC award FA8750-15-C-0082 and DARPA AA award FA8750-18-C-0092; by ONR under grant N00014-17-1-2889; by NSF under SaTC award 1801369, SHF grant 1812876, and CAREER award 1845514; and by the UW-Madison OVRGE with funding from WARF.

## References

[1] Abramsky, S. and A. Jung, *Domain Theory*, in: *Handbook of Logic in Computer Science*, Oxford University Press Oxford, UK, 1994 .

[2] Barthe, G., T. Espitau, M. Gaboardi, B. Grégoire, J. Hsu and P.-Y. Strub, *A Program Logic for Probabilistic Programs*, Available on: https://justinh.su/files/papers/ellora.pdf (2016).

[3] Barthe, G., B. Grégoire and S. Zanella Béguelin, *Formal Certification of Code-based Cryptographic Proofs*, in: *Princ. of Prog. Lang. (POPL'09)*, 2009.

[4] Barthe, G., B. Köpf, F. Olmedo and S. Zanella Béguelin, *Probabilistic Relational Reasoning for Differential Privacy*, in: *Princ. of Prog. Lang. (POPL'12)*, 2012.

[5] Batz, K., B. L. Kaminski, J.-P. Katoen and C. Matheja, *How long, O Bayesian network, will I sample thee?*, in: *European Symp. on Programming (ESOP'18)*, 2018.

[6] Bellman, R., *A Markovian Decision Process*, Indiana Univ. Math. J. **6** (1957).

[7] Bichsel, B., T. Gehr and M. Vechev, *Fine-grained Semantics for Probabilistic Programs*, in: *European Symp. on Programming (ESOP'18)*, 2018.

[8] Borgström, J., U. D. Lago, A. D. Gordon and M. Szymczak, *A Lambda-Calculus Foundation for Universal Probabilistic Programming*, in: *Int. Conf. on Functional Programming (ICFP'16)*, 2016.

[9] Bouissou, O., E. Goubault, S. Putot, A. Chakarov and S. Sankaranarayanan, *Uncertainty Propagation Using Probabilistic Affine Forms and Concentration of Measure Inequalities*, in: *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'16)*, 2016.

[10] Brázdil, T., S. Kiefer, A. Kučera and I. H. Vařeková, *Runtime Analysis of Probabilistic Programs with Unbounded Recursion*, J. Comput. Syst. Sci. **81** (2015).

[11] Chakarov, A. and S. Sankaranarayanan, *Probabilistic Program Analysis with Martingales*, in: *Computer Aided Verif. (CAV'13)*, 2013.

[12] Chakarov, A. and S. Sankaranarayanan, *Expectation Invariants for Probabilistic Program Loops as Fixed Points*, in: *Static Analysis Symp. (SAS'14)*, 2014.

[13] Chatterjee, K., H. Fu and A. K. Goharshady, *Termination Analysis of Probabilistic Programs Through Positivstellensatz's*, in: *Computer Aided Verif. (CAV'16)*, 2016.

[14] Chatterjee, K., H. Fu, P. Novotný and R. Hasheminezhad, *Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs*, in: *Princ. of Prog. Lang. (POPL'16)*, 2016.

[15] Chatterjee, K., P. Novotný and Đ. Žikelić, *Stochastic Invariants for Probabilistic Termination*, in: *Princ. of Prog. Lang. (POPL'17)*, 2017.

[16] Conway, J. H., "Regular algebra and finite machines," London: Chapman and Hall, 1971.

[17] Cousot, P. and M. Monerau, *Probabilistic Abstract Interpretation*, in: *European Symp. on Programming (ESOP'12)*, 2012.

[18] den Hartog, J. I. and E. P. de Vink, *Mixing Up Nondeterminism and Probability: a preliminary report*, Electr. Notes Theor. Comp. Sci. **22** (1999).

[19] Dijkstra, E. W., "A Discipline of Programming," Prentice-Hall, 1997.

[20] Ehrhard, T., M. Pagani and C. Tasson, *Measurable Cones and Stable, Measurable Functions*, in: *Princ. of Prog. Lang. (POPL'18)*, 2018.

[21] Escardó, M. H., *PCF extended with real numbers*, Theor. Comp. Sci. **162** (1996).

[22] Etessami, K. and M. Yannakakis, *Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations*, in: *Symp. on Theor. Aspects of Comp. Sci. (STACS'05)*, 2005.

[23] Etessami, K. and M. Yannakakis, *Recursive Markov Decision Processes and Recursive Stochastic Games*, J. ACM **62** (2015).

[24] Farzan, A. and Z. Kincaid, *Compositional Recurrence Analysis*, in: *Formal Methods in Compiter-Aided Design (FMCAD'15)*, 2015.

[25] Ferrer Fioriti, L. M. and H. Hermanns, *Probabilistic Termination: Soundness, Completeness, and Compositionality*, in: *Princ. of Prog. Lang. (POPL'15)*, 2015.

[26] Franke, B., M. O'Boyle, J. Thomson and G. Fursin, *Probabilistic Source-Level Optimisation of Embedded Programs*, in: *Lang., Comp., and Tools for Embeded Syst. (LCTES'05)*, 2005.

[27] Gallo, G., G. Longo, S. Pallottino and S. Nguyen, *Directed Hypergraphs and Applications*, Disc. Appl. Math. **42** (1993).

[28] Gehr, T., S. Misailovic and M. Vechev, *PSI: Exact Symbolic Inference for Probabilistic Programs*, in: *Computer Aided Verif. (CAV'16)*, 2016.

[29] Ghahramani, Z., *Probabilistic machine learning and artificial intelligence*, Nature (2015).

[30] Goodman, N. D., V. K. Mansinghka, D. M. Roy and J. B. Tenenbaum, *Church: a language for generative models*, in: *Uncertainty in Artif. Intelligence*, 2008.

[31] Gordon, A. D., T. A. Henzinger, A. V. Nori and S. K. Rajamani, *Probabilistic Programming*, in: *Future of Softw. Eng. (FOSE'14)*, 2014.

[32] Heunen, C., O. Kammar, S. Staton and H. Yang, *A Convenient Category for Higher-Order Probability Theory*, in: *Logic in Computer Science (LICS'17)*, 2017.

[33] Hofmann, K. H. and M. Mislove, *Local compactness and continuous lattices*, in: *Continuous Lattices*, 1981.

[34] Jansen, N., B. L. Kaminski, J.-P. Katoen, F. Olmedo, F. Gretz and A. K. McIver, *Conditioning in Probabilistic Programming*, Electr. Notes Theor. Comp. Sci. **319** (2015).

[35] Jones, C., "Probabilistic Non-determinism," Ph.D. thesis, University of Edinburgh (1989).

[36] Jones, C. and G. Plotkin, *A Probabilistic Powerdomain of Evaluations*, in: *Logic in Computer Science (LICS'89)*, 1989.

[37] Jung, A. and R. Tix, *The Troublesome Probabilistic Powerdomain*, Electr. Notes Theor. Comp. Sci. **13** (1998).

[38] Kaminski, B. L., J.-P. Katoen, C. Matheja and F. Olmedo, *Weakest Precondition Reasoning for Expected Run—Times of Probabilistic Programs*, in: *European Symp. on Programming (ESOP'16)*, 2016.

[39] Katoen, J.-P., A. K. McIver, L. A. Meinicke and C. C. Morgan, *Linear-Invariant Generation for Probabilistic Programs: Automated Support for Proof-Based Methods*, in: *Static Analysis Symp. (SAS'10)*, 2010.

[40] Kattenbelt, M., M. Kwiatkowska, G. Norman and D. Parker, *Abstraction Refinement for Probabilistic Software*, in: *Verif., Model Checking, and Abs. Interp. (VMCAI'09)*, 2009.

[41] Kleene, S. C., *Representation of Events in Nerve Nets and Finite Automata*, Available on https://www.rand.org/pubs/research_memoranda/RM704.html (1951).

[42] Kozen, D., *On induction vs. *-continuity*, in: *Workshop on Logic of Programs*, 1981.

[43] Kozen, D., *Semantics of Probabilistic Programs*, J. Comput. Syst. Sci. **22** (1981).

[44] Kozen, D., *A Probabilistic PDL*, J. Comput. Syst. Sci. **30** (1985).

[45] Kozen, D., *A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events*, J. Information and Computation **110** (1991).

[46] Lal, A., T. Touili, N. Kidd and T. Reps, *Interprocedural Analysis of Concurrent Programs Under a Context Bound*, in: *Tools and Algs. for the Construct. and Anal. of Syst. (TACAS'08)*, 2008.

[47] McIver, A. K. and C. C. Morgan, *Partial correctness for probabilistic demonic programs*, Theor. Comp. Sci. **266** (2001).

[48] McIver, A. K. and C. C. Morgan, "Abstraction, Refinement and Proof for Probabilistic Systems," Springer Science+Business Media, Inc., 2005.

[49] Mislove, M., *Topology, domain theory and theoretical computer science*, Topology and its Applications **89** (1998).

[50] Mislove, M., *Nondeterminism and Probabilistic Choice: Obeying the Laws*, in: *Concurrency Theory*, 2000.

[51] Mislove, M., J. Ouaknine and J. Worrell, *Axioms for Probability and Nondeterminism*, Electr. Notes Theor. Comp. Sci. **96** (2004).

[52] Monniaux, D., *Abstract Interpretation of Probabilistic Semantics*, in: *Static Analysis Symp. (SAS'00)*, 2000.

[53] Monniaux, D., *Abstract Interpretation of Programs as Markov Decision Processes*, in: *Static Analysis Symp. (SAS'03)*, 2003.

[54] Müller-Olm, M. and H. Seidl, *Precise Interprocedural Analysis through Linear Algebra*, in: *Princ. of Prog. Lang. (POPL'04)*, 2004.

[55] Olmedo, F., B. L. Kaminski, J.-P. Katoen and C. Matheja, *Reasoning about Recursive Probabilistic Programs*, in: *Logic in Computer Science (LICS'16)*, 2016.

[56] Paige, B. and F. Wood, *A Compilation Target for Probabilistic Programming Languages*, in: *Int. Conf. on Machine Learning (ICML'14)*, 2014.

[57] Sankaranarayanan, S., A. Chakarov and S. Gulwani, *Static Analysis for Probabilistic Programs: Inferring Whole Program Properties from Finitely Many Paths*, in: *Prog. Lang. Design and Impl. (PLDI'13)*, 2013.

[58] Smolka, S., P. Kumar, N. Foster, D. Kozen and A. Silva, *Cantor meets Scott: Semantic Foundations for Probabilistic Networks*, in: *Princ. of Prog. Lang. (POPL'17)*, 2017.

[59] Staton, S., *Commutative Semantics for Probabilistic Programming*, in: *European Symp. on Programming (ESOP'17)*, 2017.

[60] Staton, S., H. Yang, C. Heunen and O. Kammar, *Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints*, in: *Logic in Computer Science (LICS'16)*, 2016.

[61] Tarjan, R. E., *A Unified Approach to Path Problems*, J. ACM **28** (1981).

[62] Tix, R., K. Keimel and G. Plotkin, *Semantic Domains for Combining Probability and Non-Determinism*, Electr. Notes Theor. Comp. Sci. **222** (2009).

[63] Vákár, M., O. Kammar and S. Staton, *A Domain Theory for Statistical Probabilistic Programming*, in: *Princ. of Prog. Lang. (POPL'19)*, 2019.

[64] Wang, D., J. Hoffmann and T. Reps, *PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs*, in: *Prog. Lang. Design and Impl. (PLDI'18)*, 2018.