A Demonic Outcome Logic for Randomized Nondeterminism

NOAM ZILBERSTEIN, Cornell University, USA
DEXTER KOZEN, Cornell University, USA
ALEXANDRA SILVA, Cornell University, USA
JOSEPH TASSAROTTI, New York University, USA

Programs increasingly rely on randomization in applications such as cryptography and machine learning. Analyzing randomized programs has been a fruitful research direction, but there is a gap when programs also exploit nondeterminism (for concurrency, efficiency, or algorithmic design). In this paper, we introduce *Demonic Outcome Logic* for reasoning about programs that exploit *both* randomization and nondeterminism. The logic includes several novel features, such as reasoning about multiple executions in tandem and manipulating pre- and postconditions using familiar equational laws—including the distributive law of probabilistic choices over nondeterministic ones. We also give rules for loops that both establish termination and quantify the distribution of final outcomes from a single premise. We illustrate the reasoning capabilities of Demonic Outcome Logic through several case studies, including the Monty Hall problem, an adversarial protocol for simulating fair coins, and a heuristic based probabilistic SAT solver.

CCS Concepts: • **Theory of computation** → **Programming logic**; **Hoare logic**; **Probabilistic computation**; **Denotational semantics**.

Additional Key Words and Phrases: Program Logics, Probabilistic Programming, Demonic Nondeterminism

ACM Reference Format:
Noam Zilberstein, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. 2025. A Demonic Outcome Logic for Randomized Nondeterminism. *Proc. ACM Program. Lang.* 9, POPL, Article 19 (January 2025), 30 pages. https://doi.org/10.1145/3704855

1 Introduction

Randomization is critical in sensitive software domains such as cryptography and machine learning. While it is difficult to establish correctness of these systems alone, the difficulty is increased as they become distributed, since nondeterminism is introduced by scheduling the concurrent processes. Verification techniques exist for reasoning about programs that are both randomized and nondeterministic using expectations [Morgan et al. 1996a] and refinement [Tassarotti and Harper 2019], but there are currently no logics that allow for specifying and reasoning about the multiple probabilistic executions that arise from this combination of effects.

In program logics such as Hoare Logic [Hoare 1969], preconditions and postconditions are propositions about the start and end states of the program. When moving to a probabilistic setting, it is not enough for these propositions to merely describe states, they must also quantify how likely the program is to end up in each of those states, as correctness is a property of the *distribution* of outcomes. Several logics exist for reasoning about purely probabilistic programs in this way, including Probabilistic Hoare Logic [Corin and den Hartog 2006; den Hartog 1999, 2002], Ellora

Authors' Contact Information: Noam Zilberstein, Cornell University, Ithaca, USA, noamz@cs.cornell.edu; Dexter Kozen, Cornell University, Ithaca, USA, kozen@cs.cornell.edu; Alexandra Silva, Cornell University, Ithaca, USA, alexandra.silva@cornell.edu; Joseph Tassarotti, New York University, New York, USA, jt4767@nyu.edu.

ACM 2475-1421/2025/1-ART19
https://doi.org/10.1145/3704855

[Barthe et al. 2018], Outcome Logic [Zilberstein 2024; Zilberstein et al. 2023, 2024b], and Quantitative Weakest Hyper Pre [Zhang et al. 2024]. The benefits of reasoning about multiple executions are:

Outcomes. As opposed to expectation reasoning, program logics can describe multiple outcomes in a single specification, giving a more comprehensive account of the *distribution* of behaviors. This is displayed in Section 6.2, where we prove that a program simulates a fair coin by enumerating the outcomes and showing that they are uniformly distributed.

Compositionality. Inference rules allow us to reason about programs in a compositional, but also concise way. This is evident in our termination rules (Section 5), which have fewer premises compared to similar rules in other reasoning systems.

This paper introduces *Demonic Outcome Logic*, a program logic for reasoning about *randomized nondeterministic* programs—programs that have both probabilistic and nondeterministic choice operators. This work builds both on Outcome Logic—which can be used to reason about randomization *or* nondeterminism, but not both together—and a large body of work on the semantics of randomized nondeterministic programs [He et al. 1997; Jacobs 2008; Morgan et al. 1996a,b; Tix et al. 2009; Varacca 2002]. Our contributions can be grouped in four categories, as follows:

(1) *From Equations to Propositions.* Semantic objects to capture both randomization and nondeterminism are often described in terms of *equations*, stating properties of relevant operators such as idempotence and distributivity. In logic, implications are used to manipulate assertions and facilitate reasoning. In our proposed program logic, we want to bring these worlds together and have logical implications mirror the equational laws. The challenge is that—as we will see in Sections 2.1 and 4.1—the equations do not immediately hold as implications, so a carefully designed assertion language is needed in which the laws indeed hold.

(2) *Demonic Outcome Logic.* This paper presents the first program logic for reasoning about *distributions of outcomes* with both randomization and nondeterminism. Making the logic *demonic*—meaning that the postcondition applies to every nondeterministic possibility—allowed us to create simple and convenient inference rules.

While our logic has similarities to Weakest Pre-Expectation calculi, Demonic Outcome Logic involves some key differences. Demonic Outcome Logic can reason about many executions together, which allows us to specify the distribution of outcomes rather than just quantitative properties of that distribution. This is demonstrated in Section 6.2, where a program is specified in terms of multiple distinct outcomes and Section 6.3, where case analysis is done over multiple nondeterministic executions. It was necessary to develop new sound rules for this more expressive form of reasoning and idempotence of the logical connectives proved crucial in ways that do not appear in prior work.

(3) *Loops and Termination.* Our rules for reasoning about loops in Section 5 allow us to prove termination, while simultaneously specifying the precise distribution of outcomes upon termination. This goes beyond prior work on expectation based reasoning [McIver and Morgan 2005; McIver et al. 2018], where termination is established with a propositional invariant describing only a singe outcome. Our rules also have fewer premises, making them simpler to apply in our experience.

(4) *Case Studies.* We investigate three case studies in Section 6 to demonstrate the applicability of our logic. For example, we present a protocol to simulate a fair coin flip given a coin whose bias is continually altered by an adversary, and show that this program terminates with the outcomes being uniformly distributed. We also prove that a probabilistic SAT solver terminates, even if some of the heuristics involved are adversarially chosen.

We begin in Section 2 by outlining the challenges of reasoning about randomized nondeterminism, and how this informed the design of Demonic Outcome Logic. Next, in Section 3, we describe the denotational model that we use for semantics of programs. In Section 4, we introduce Demonic Outcome Logic and the inference rules for reasoning about sequential programs. We discuss reasoning about loops in Section 5. We examine three case studies in Section 6 to demonstrate the utility of the logic. Finally, we discuss related work and conclude in Sections 7 and 8.

2 An Overview of Demonic, Outcome-Based Reasoning

The issue of combining randomization and nondeterminism is one of the most difficult and subtle challenges in program semantics. In this section, we outline the desired properties of a logic for that purpose and how the design of that logic intersects with prior work on semantics.

Most applications of randomized nondeterminism take a *demonic* view of nondeterminism, wherein the nondeterminism is controlled by an *adversary*, and the program is correct only if the distribution of outcomes satisfies a certain post-condition, *regardless* of how the adversary might have resolved the nondeterminism. One such domain is verification of distributed cryptographic protocols, where the probability that an adversary can guess a secret message must be negligible regardless of how the scheduler interleaves the concurrent processes.

To demonstrate the complex interaction between demonic non-determinism and probabilistic choice, we consider an example in which an adversary tries to guess the outcome of a fair coin flip. The coin flip is represented by the program $x \coloneqq \mathsf{flip}\left(\frac{1}{2}\right)$, whose denotation is a singleton set containing a distribution of outcomes in which $x = \mathsf{true}$ and $x = \mathsf{false}$ both occur with probability $\frac{1}{2}$. The adversarial choice is performed by the program $y \leftarrow \mathbb{B}$, where $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$. Operationally, we presume that the adversary can make this choice in any way it pleases, including by flipping a biased coin. That means that the adversary can force y to be true, it can force y to be false, or it can make both outcomes possible with any probability. Denotationally, the semantics of these programs is a map $[\![C]\!] : \Sigma \to 2^{\mathcal{D}(\Sigma)}$ from states Σ to sets of distributions of states, shown below.

$$[\![x \coloneqq \mathsf{flip}\left(\tfrac{1}{2}\right)]\!](\sigma) = \left\{ \begin{smallmatrix} \sigma[x \coloneqq \mathsf{true}\,] \mapsto \frac{1}{2} \\ \sigma[x \coloneqq \mathsf{false}] \mapsto \frac{1}{2} \end{smallmatrix} \right\} \qquad [\![y \leftarrow \mathbb{B}]\!](\sigma) = \left\{ \begin{smallmatrix} \sigma[y \coloneqq \mathsf{true}\,] \mapsto p \\ \sigma[y \coloneqq \mathsf{false}] \mapsto 1-p \end{smallmatrix} \;\middle|\; p \in [0,1] \right\}$$

Now, we consider two variants of composing these programs, shown below. On the left is a variant in which the adversary picks last and on the right is a variant in which the adversary picks first.

$$x \coloneqq \mathsf{flip}\left(\tfrac{1}{2}\right) \fatsemi y \leftarrow \mathbb{B} \qquad\qquad y \leftarrow \mathbb{B} \fatsemi x \coloneqq \mathsf{flip}\left(\tfrac{1}{2}\right)$$

We wish to know the probability that $x = y$. In the program on the left, the value of x is fixed before the adversary makes its choice, meaning that it can choose a distribution in which $x = y$ with any probability $p \in [0, 1]$. However, in the program on the right, the adversary chooses first, and so the later coin flip will ensure that $x = y$ with probability exactly $\frac{1}{2}$.

We will examine how to prove this fact using program logics. First, in Section 2.1 we will lay out the semantic properties of random and nondeterministic choices using equations, and we will show how those equations inform propositional reasoning about outcomes. Next, in Section 2.2, we will see how to make such propositional inferences about program behavior using our new logic. We will then overview how to reason about more complicated looping programs in Section 2.3.

2.1 From Equational Laws to Propositional Reasoning

Equational theories are a useful tool for defining the behavior of programatic operators in terms of laws that must be upheld. This has been studied extensively in the context of semantics of probabilistic nondeterminism [Bonchi et al. 2019, 2021b, 2022; Mio and Vignudelli 2020; Mislove 2000; Tix 2000], where equations are used to describe properties of nondeterminism, random choice,

and the interaction between the two. We will now explore the link between equational theories and propositional reasoning about program outcomes. As we explain in this section, care has to be taken to craft a model in which the desired properties of program operations can be used not only to establish equality of semantic objects, but also as logical implications.

In the following, the variables X, Y, and Z denote the outcomes of a program. The nondeterministic choice operator—denoted $X \,\&\, Y$—is an adversarial choice between the outcomes X and Y. It should be idempotent, commutative, and associative, as captured by the following equations:

$$X \,\&\, X = X \qquad X \,\&\, Y = Y \,\&\, X \qquad (X \,\&\, Y) \,\&\, Z = X \,\&\, (Y \,\&\, Z)$$

That is, choosing between X and X is equivalent to making no choice at all, and the ordering of choices makes no difference. The probabilistic choice operator $X \oplus_p Y$, where $p \in [0, 1]$, represents a random choice where X and Y occur with probability p and $1 - p$, respectively. This operator obeys similar laws, with probabilities adjusted appropriately.

$$X \oplus_p X = X \qquad X \oplus_p Y = Y \oplus_{1-p} X \qquad (X \oplus_p Y) \oplus_q Z = X \oplus_{pq} (Y \oplus_{\frac{(1-p)q}{1-pq}} Z)$$

In addition, the following *distributive law* requires that random choices distribute over nondeterministic ones, much like multiplication distributes over addition in standard arithmetic.

$$X \oplus_p (Y \,\&\, Z) = (X \oplus_p Y) \,\&\, (X \oplus_p Z)$$

This law corresponds to our interpretation of demonic nondeterminism. On the left-hand side of the equation, we first randomly choose to execute either X or $Y \,\&\, Z$, and then—if the second option is taken—the nondeterministic choice is resolved. Applying this axiom as a rewrite rule from left to right would push the nondeterministic choice to the top above the probabilistic choice.

Traditionally, equational theories have been used to decide equality between programs [Kozen 1997]. Here, we repurpose the equations for propositional reasoning about program outcomes. That is, if φ, ψ, and ϑ are assertions about outcomes, then $\varphi \,\&\, \psi$ asserts that φ and ψ are two possible nondeterministic outcomes, and $\varphi \oplus_p \psi$ asserts that φ occurs with probability p and ψ occurs with probability $1 - p$. This is inspired by Outcome Logic [Zilberstein et al. 2023], but there are now two types of outcomes (probabilistic and non-deterministic). We want to rewrite the desired equations above as logical equivalences, e.g. the distributive law would be transformed to:

$$\varphi \oplus_p (\psi \,\&\, \vartheta) \;\Leftrightarrow\; (\varphi \oplus_p \psi) \,\&\, (\varphi \oplus_p \vartheta)$$

However, one has to be careful. For example, as we illustrate below, the idempotence property $\varphi \,\&\, \varphi \Leftrightarrow \varphi$ only holds as an implication in a carefully crafted model.

One benefit of propositional reasoning vs equational reasoning is the ability to *weaken* assertions. For instance, returning to the coin flip example, the following proposition precisely captures the result of the program in which the adversary chooses first, where $\lceil P \rceil$ means that P occurs with probability 1 (almost surely).

$$(\lceil y = \mathsf{true} \rceil \wedge (\lceil x = \mathsf{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \mathsf{false} \rceil)) \,\&\, (\lceil y = \mathsf{false} \rceil \wedge (\lceil x = \mathsf{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \mathsf{false} \rceil)) \tag{1}$$

But the precise values of x and y are cumbersome to remember, and obfuscate the property that we want to convey. Instead, we can weaken the assertion to record only whether $x = y$ or $x \neq y$.

$$(\lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil) \,\&\, (\lceil x \neq y \rceil \oplus_{\frac{1}{2}} \lceil x = y \rceil)$$

It is now tempting to use commutativity of $\oplus_{\frac{1}{2}}$ and idempotence of $\&$ to perform the following simplification, concisely asserting the probability that the adversary can determine the value of x.

$$(\lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil) \,\&\, (\lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil) \quad \Rightarrow \quad \lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil \tag{2}$$

However, care had be taken to craft a model in which implication (2) is valid. Unlike the idempotence equation $X \,\&\, X = X$—which applies when the exact same set of distributions appears on each side—the implication version (2) operates on *approximations* of those sets of distributions. Recall from (1) that $x = y$ is satisfied by $x = y = \mathsf{true}$ on the left hand side of the &, whereas it is satisfied by $x = y = \mathsf{false}$ on the right, so even though both sets of distributions satisfy $x = y \oplus_{\frac{1}{2}} x \neq y$, they are not equal. The full details of this example are shown in Zilberstein et al. [2024a, §A.1].

In Section 4.1, we give a full account of how our demonic logic supports idempotence and all of the other properties that were stated equationally above. These properties do not hold by default, but rather required some intentional choices in the design of the logic. In particular, as we will detail in Section 4.1, the assertion language will not include disjunctions or existential quantification. The result is a deductive system that is able to express more intuitive and concise specifications.

2.2 Program Logics and Compositionality

Inspired by Hoare Logic, our goal is to develop a logic where programs are specified in terms of pre- and postconditions using *triples* of the form $\langle \varphi \rangle\ C\ \langle \psi \rangle$. Here, φ and ψ are outcome assertions from Section 2.1, whose models are distributions of states. Since the program C is nondeterministic—and is interpreted as a map into sets of distributions—the postcondition ψ must be satisfied by every distribution in that resulting set. We call this logic Demonic Outcome Logic, as it supports probabilistic reasoning in the style of Outcome Logic [Zilberstein 2024; Zilberstein et al. 2023, 2024b] with the crucial addition of demonic nondeterminism.

Compositional reasoning—the ability to analyze a complex program in terms of its subprograms—is the hallmark of program logics. This is exemplified by the inference rule for sequential composition; we infer the behavior of a composite program from the behavior of its constituent parts.

$$\frac{\langle \varphi \rangle\ C_1\ \langle \vartheta \rangle \qquad \langle \vartheta \rangle\ C_2\ \langle \psi \rangle}{\langle \varphi \rangle\ C_1 \fatsemi C_2\ \langle \psi \rangle}\ \textsc{Seq}$$

The soundness of this rule is not a given in the randomized nondeterminism setting. It relies on being able to define the semantics $[\![C_1 \fatsemi C_2]\!]$ in terms of $[\![C_1]\!]$ and $[\![C_2]\!]$. As we have already seen at the beginning of Section 2, the semantics is a map from states to sets of distributions of states: $[\![C]\!] : \Sigma \to 2^{\mathcal{D}(\Sigma)}$. We compose the semantics of program fragments using a lifted version, known as the *Kleisli extension* $[\![C]\!]^\dagger : 2^{\mathcal{D}(\Sigma)} \to 2^{\mathcal{D}(\Sigma)}$. If $2^{\mathcal{D}(-)}$ were a *monad*, then we would have the compositionality property to guarantee the soundness of the Seq rule: $[\![C_1 \fatsemi C_2]\!]^\dagger = [\![C_2]\!]^\dagger \circ [\![C_1]\!]^\dagger$. Unfortunately, as originally shown by Varacca and Winskel [2006] (see also Parlant [2020]; Zwart and Marsden [2019]), no such composition operator exists.

But compositionality can be retained by requiring the sets of distributions to be *convex* [Jacobs 2008; Mislove 2000; Morgan et al. 1996b; Tix 1999]. That is, whenever two distributions μ and ν are in the set of outcomes, then all convex combinations $(p \cdot \mu + (1-p) \cdot \nu$ for $p \in [0,1])$ are also in the set of possible results. Convexity corresponds to our operational interpretation of nondeterminism—the adversary may flip biased coins to resolve choices [Varacca 2002, Theorem 6.12]. Returning to the coin flip example, we can derive the following specifications for the primitive operations.

$$\langle \lceil \mathsf{true} \rceil \rangle\ y \leftarrow \mathbb{B}\ \langle \lceil y = \mathsf{true} \rceil \,\&\, \lceil y = \mathsf{false} \rceil \rangle$$

$$\langle \lceil y = \ \mathsf{true} \rceil \rangle\ x := \mathsf{flip}\left(\tfrac{1}{2}\right)\ \langle \lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil \rangle$$
$$\langle \lceil y = \mathsf{false} \rceil \rangle\ x := \mathsf{flip}\left(\tfrac{1}{2}\right)\ \langle \lceil x \neq y \rceil \oplus_{\frac{1}{2}} \lceil x = y \rceil \rangle$$

That is, the adversarial choice results in two nondeterministic outcomes, separated by &. Executing the probabilistic choice in either of those states yields two further probabilistic outcomes.

Since the first command splits the execution into two outcomes, we need one more type of composition in order to stitch these together into a specification for the composite program. That

is, we need the ability to decompose the precondition and analyze the program with each resulting sub-assertion individually. We may also wish to do the same for probabilistic choices.

$$\frac{\langle \varphi_1 \rangle\ C\ \langle \psi_1 \rangle \qquad \langle \varphi_2 \rangle\ C\ \langle \psi_2 \rangle}{\langle \varphi_1 \,\&\, \varphi_2 \rangle\ C\ \langle \psi_1 \,\&\, \psi_2 \rangle}\text{ND Split} \qquad \frac{\langle \varphi_1 \rangle\ C\ \langle \psi_1 \rangle \qquad \langle \varphi_2 \rangle\ C\ \langle \psi_2 \rangle}{\langle \varphi_1 \oplus_p \varphi_2 \rangle\ C\ \langle \psi_1 \oplus_p \psi_2 \rangle}\text{Prob Split}$$

Using Seq, ND Split, and the idempotence rule (2), we derive the triple below on the left (shown fully in Zilberstein et al. [2024a, §A.2]). A similar derivation for the reversed version yields the triple on the right.

$$\begin{array}{l}\langle \lceil \mathsf{true} \rceil \rangle \\ \quad y \leftarrow \mathbb{B} \fatsemi \\ \quad x \coloneqq \mathsf{flip}\left(\frac{1}{2}\right) \\ \langle \lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil \rangle\end{array} \qquad \begin{array}{l}\langle \lceil \mathsf{true} \rceil \rangle \\ \quad x \coloneqq \mathsf{flip}\left(\frac{1}{2}\right) \fatsemi \\ \quad y \leftarrow \mathbb{B} \\ \langle \lceil x = y \rceil \,\&\, \lceil x \neq y \rceil \rangle\end{array}$$

If the adversary picks first, then it can only guess the value of x with probability $\frac{1}{2}$. But if the coin flip is first, we only know that $x = y$ occurs with some probability. In fact, $x = y \,\&\, x \neq y$ is equivalent to true, so it certainly does not give us a robust security guarantee, leaving open the possibility that the adversary can guess x.

2.3 Reasoning about Loops

Reasoning about loops is challenging in any program logic, and Demonic Outcome Logic is no exception. When reasoning about probabilistic loops, one often wants to prove not only that some property holds upon termination, but also that the program *almost surely terminates*—the probability of nontermination is 0. An example of an almost surely terminating program is shown below. It is an adversarial random walk, where the agent steps towards 0 with probability $\frac{1}{2}$, otherwise the adversary moves the agent to an arbitrary position between 1 and 5.

$$\begin{array}{l}\textbf{while}\ x > 0\ \textbf{do} \\ \quad (x \coloneqq x - 1) \oplus_{\frac{1}{2}} (x \leftarrow \{1, \ldots, 5\})\end{array}$$

It may seem surprising that this program almost surely terminates; after all, the adversary can always choose the worst possible option of resetting the position to 5. However, as the number of iterations goes to infinity, the probability of decrementing x five times in a row goes to 1.

Demonic Outcome Logic has a simple inference rule for proving almost sure termination, inspired by a rule of McIver and Morgan [2005], which uses *ranking functions* to quantify how close the loop is to termination. The rule states that the program almost surely terminates if the rank strictly decreases on each iteration with probability bounded away from 0, while also preserving some invariant P.

$$\frac{\langle \lceil P \wedge e \wedge e_{\mathsf{rank}} = n \rceil \rangle\ C\ \langle \lceil P \wedge e_{\mathsf{rank}} < n \rceil \oplus_p \lceil P \rceil \rangle, \quad p > 0}{\langle \lceil P \rceil \rangle\ \textbf{while}\ e\ \textbf{do}\ C\ \langle \lceil P \wedge \neg e \rceil \rangle}\text{Bounded Rank}$$

We prove the soundness of this rule in Section 5. To instantiate it for the program above, we use the invariant $P \triangleq 0 \leq x \leq 5$, the ranking function $e_{\mathsf{rank}} \triangleq x$, and the probability $p = \frac{1}{2}$. This means that x is always between 0 and 5 and the value of x strictly decreases with probability $\frac{1}{2}$ in each iteration of the loop. Applying the inference, we get the following specification for the program.

$$\langle \lceil 0 \leq x \leq 5 \rceil \rangle\ \textbf{while}\ x > 0\ \textbf{do}\ (x \coloneqq x - 1) \oplus_{\frac{1}{2}} (x \leftarrow \{1, \ldots, 5\})\ \langle \lceil x = 0 \rceil \rangle$$

This says that the program terminates in a state satisfying $x = 0$ with probability 1 (*i.e.,* almost surely). Compared to the rule of McIver and Morgan [2005]—which is based on *weakest pre-expectations*—our approach has two key advantages. First, in the pre-expectation approach, the

preservation of the invariant and the decrease in rank are verified separately, whereas our rule combines the two in a single premise. Second, our rules allow the invariant to have multiple outcomes, allowing them to express a *distribution* of end states, rather than a single assertion. A concrete example of this appears in Section 6.2.

We have now seen the key ideas behind Demonic Outcome Logic, including how equational laws translate to propositional reasoning over pre- and postconditions, the challenges in making the logic compositional, and strategies for analyzing loops to establish almost sure termination. We now proceed by making these ideas formal, starting in Section 3 where we define the program semantics, and continuing with Sections 4 and 5 where we define the logic and rules for analyzing loops. In Section 6 we examine case studies to show how the logic is used.

3 Denotational Semantics for Probabilistic Nondeterminism

In this section, we present the semantics of a simple imperative language with both probabilistic and nondeterministic choice operators, originally due to He et al. [1997] and Morgan et al. [1996a]. The syntax of the language, below, includes familiar constructs such as no-ops, variable assignment, sequential composition, if-statements, and while-loops, plus two kinds of branching choice.

$$
\begin{array}{rll}
\mathsf{Cmd} \ni C ::= & \mathbf{skip} & \text{(No-op)} \\
\mid & x := e & \text{(Variable Assignment)} \\
\mid & C_1 \fatsemi C_2 & \text{(Sequential Composition)} \\
\mid & C_1 \,\&\, C_2 & \text{(Nondeterministic Choice)} \\
\mid & C_1 \oplus_e C_2 & \text{(Probabilistic Choice)} \\
\mid & \mathbf{if}\ e\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2 & \text{(Conditional)} \\
\mid & \mathbf{while}\ e\ \mathbf{do}\ C & \text{(While Loop)}
\end{array}
$$

Expressions $e \in \mathsf{Exp}$ range over typical arithmetic and Boolean operations, and we evaluate these expressions in the usual way. Nondeterministic choice $C_1 \,\&\, C_2$ represents a program that arbitrarily chooses to execute either C_1 or C_2, whereas probabilistic choice $C_1 \oplus_e C_2$, in which e evaluates to a rational probability p, represents a program in which C_1 is executed with probability p and C_2 with probability $1 - p$. In the remainder of this section, we precisely describe the semantics of the language, building on the informal account given in Section 2.

3.1 States, Probability Distributions, and Convex Sets

Before we present the semantics, we review some preliminary definitions. We begin by describing the *program states* $\sigma \in \Sigma \triangleq \mathsf{Var} \to \mathsf{Val}$, which are mappings from a finite set of variables $x \in \mathsf{Var}$ to values $v \in \mathsf{Val}$. Values consist of Booleans and rational numbers, making the set of states countable. To define the semantics, we will work with discrete probability distributions over states.

Definition 3.1 (Discrete Probability Distribution). Let $\mathcal{D}(X) \triangleq X \to [0,1]$ be the set of discrete probability distributions on X. The support of a distribution μ is the set of elements to which μ assigns nonzero probability $\mathrm{supp}(\mu) \triangleq \{x \in X \mid \mu(x) > 0\}$. We only consider proper distributions such that the total probability mass $|\mu| = \sum_{x \in \mathrm{supp}(\mu)} \mu(x)$ is 1.

We denote the the Dirac distribution centered at a point $x \in X$ by δ_x, with $\delta_x(x) = 1$ and $\delta_x(y) = 0$ if $x \neq y$. Addition and scalar multiplication are lifted to distributions pointwise:

$$(\mu_1 + \mu_2)(x) = \mu_1(x) + \mu_2(x) \qquad (p \cdot \mu)(x) = p \cdot \mu(x)$$

The semantics of our language will be based on nonempty subsets of distributions $\mathcal{D}(\Sigma_\perp)$, where $\perp$ is a special element symbolizing divergence and $\Sigma_\perp = \Sigma \cup \{\perp\}$. We will need three *closure properties*

to make the semantics well-defined: convexity, topological closure, and up-closure. We begin by describing convexity, which makes the semantics of sequential composition associative, and the overall program semantics compositional (Section 2.2).

Definition 3.2 (Convex Sets). A set $S \subseteq \mathcal{D}(X)$ of distributions is *convex* if $\mu \in S$ and $\nu \in S$ implies that $p \cdot \mu + (1-p) \cdot \nu \in S$ for every $p \in [0,1]$.

In order to formally define the semantics of while loops, we will have to compute certain fixpoints (explained fully in Section 3.3), which requires us to restrict our semantic domain to up-closed sets.

Definition 3.3 (Up-closed Sets). A set $S \subseteq \mathcal{D}(\Sigma_\perp)$ is *up-closed* $\mu \in S$ and $\mu \sqsubseteq_{\mathcal{D}} \nu$ implies $\nu \in S$. The order $\sqsubseteq_{\mathcal{D}} \subseteq \mathcal{D}(\Sigma_\perp) \times \mathcal{D}(\Sigma_\perp)$ is defined as $\mu \sqsubseteq_{\mathcal{D}} \nu$ iff $\forall \sigma \in \Sigma.\ \mu(\sigma) \leq \nu(\sigma)$. The *up-closure* of a set S is the set $\uparrow S \triangleq \{\nu \mid \mu \in S,\ \mu \sqsubseteq_{\mathcal{D}} \nu\}$. Thus S is up-closed iff $S = \uparrow S$.

Note that $\mu \sqsubseteq_{\mathcal{D}} \nu$ implies that $\nu(\perp) \leq \mu(\perp)$, and that $\delta_\perp$ is the bottom of this order, since $\delta_\perp(\sigma) = 0$ for all $\sigma \in \Sigma$, therefore $\delta_\perp(\sigma) \leq \mu(\sigma)$ for any $\mu \in \mathcal{D}(\Sigma_\perp)$. If $\mu(\perp) = 0$, then μ is already maximal and so $\uparrow\{\mu\} = \{\mu\}$, but if $\mu(\perp) > 0$, then μ can be made larger by reassigning probability mass from $\perp$ to proper states, *e.g.*, $\delta_\perp \sqsubseteq_{\mathcal{D}} \delta_\sigma$. As a consequence, $\uparrow\{\delta_\perp\} = \mathcal{D}(\Sigma_\perp)$, the set of all distributions. Up-closure means that we cannot be sure whether a program truly diverges, or instead exhibits erratic nondeterministic behavior, which is a common limitation of the Smyth powerdomain [Søndergaard and Sestoft 1992]. However, the program logic that we develop in this paper is concerned with proving almost sure termination of programs, so the loss of precision in the semantics when nontermination might occur does not affect the accuracy of our logic.

Finally, we require sets to be closed in the usual topological sense. A subset of $\mathcal{D}(X)$ is *closed* if it is closed in the product topology $[0,1]^X$, where $[0,1]$ has the Euclidean topology. So closure means that a set S contains all of its limit points. This will later help us to ensure that the semantics is Scott continuous by precluding unbounded nondeterminism. More precisely, it will not be possible to define a primitive command $x := \bigstar$, which surely terminates and nondeterministically selects a value for x from an infinite set (such as $\mathbb{N}$). While this is certainly a limitation of the semantics, it is a typical one; an impossibility result due to Apt and Plotkin [1986] showed that it is not possible to define a semantics that both determines whether a program terminates and also allows unbounded nondeterminism. This corresponds to Dijkstra's [1976] operational observation that a machine cannot choose between infinitely many branches in a finite amount of time, so any computation with infinitely many nondeterministic outcomes may not terminate. We now have all the ingredients to define our semantic domain:

$$C(X) \triangleq \left\{S \subseteq \mathcal{D}(X_\perp) \mid S \text{ is nonempty, convex, (topologically) closed, and up-closed}\right\}$$

C is a functor and, interestingly from a semantics point of view, a monad [Jacobs 2008]. For any $f\colon X \to C(Y)$, the Kleisli extension $f^\dagger\colon C(X) \to C(Y)$ and unit operation $\eta\colon X \to C(X)$ are defined as follows:

$$\eta(x) \triangleq \uparrow\{\delta_x\} \qquad f^\dagger(S) \triangleq \left\{ \sum_{x \in \mathsf{supp}(\mu)} \mu(x) \cdot \nu_x \;\middle|\; \mu \in S, \forall x \in \mathsf{supp}(\mu).\ \nu_x \in f_\perp(x) \right\}$$

Where for any function $f\colon X \to C(Y)$, we define $f_\perp\colon X_\perp \to C(Y)$ such that $f_\perp(x) = f(x)$ for $x \in X$ and $f_\perp(\perp) = \uparrow\{\delta_\perp\}$. The C monad presented here has subtle differences to that of Jacobs [2008]—it is composed with an error monad to handle $\perp$, and the unit performs an up-closure—but it still upholds the monad laws, shown below, which we prove in Zilberstein et al. [2024a, §B.2].

$$\eta^\dagger = \mathsf{id} \qquad f^\dagger \circ \eta = f \qquad (f^\dagger \circ g)^\dagger = f^\dagger \circ g^\dagger$$

It was also shown by He et al. [1997] that $f^\dagger$ preserves up-closedness and convexity.

$$
\begin{aligned}
[\![\mathbf{skip}]\!]\,(\sigma) &\triangleq \eta(\sigma)\\
[\![x \coloneqq e]\!]\,(\sigma) &\triangleq \eta(\sigma[x \coloneqq [\![e]\!]\,(\sigma)])\\
[\![C_1 \fatsemi C_2]\!]\,(\sigma) &\triangleq [\![C_2]\!]^\dagger([\![C_1]\!]\,(\sigma))\\
[\![C_1 \,\&\, C_2]\!]\,(\sigma) &\triangleq [\![C_1]\!]\,(\sigma) \,\&\, [\![C_2]\!]\,(\sigma)\\
[\![C_1 \oplus_e C_2]\!]\,(\sigma) &\triangleq [\![C_1]\!]\,(\sigma) \oplus_{[\![e]\!](\sigma)} [\![C_2]\!]\,(\sigma)\\
[\![\mathbf{if}\ e\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2]\!]\,(\sigma) &\triangleq \begin{cases} [\![C_1]\!]\,(\sigma) & \text{if } [\![e]\!]\,(\sigma) = \text{true}\\ [\![C_2]\!]\,(\sigma) & \text{if } [\![e]\!]\,(\sigma) = \text{false}\end{cases}\\
[\![\mathbf{while}\ e\ \mathbf{do}\ C]\!]\,(\sigma) &\triangleq \mathsf{lfp}\left(\Phi_{\langle C,e\rangle}\right)(\sigma)
\end{aligned}
$$

$$
\text{where } \Phi_{\langle C,e\rangle}(f)(\sigma) \triangleq \begin{cases} f^\dagger_\bot([\![C]\!]\,(\sigma)) & \text{if } [\![e]\!]\,(\sigma) = \text{true}\\ \eta(\sigma) & \text{if } [\![e]\!]\,(\sigma) = \text{false}\end{cases}
$$

Fig. 1. Denotational Semantics for programs $[\![C]\!] : \Sigma \to \mathcal{C}(\Sigma)$, where $[\![e]\!] : \Sigma \to \mathsf{Val}$ is the interpretation of expressions, defined in the obvious way.

The last ingredient we need (for the semantics of loops) is an order on $\mathcal{C}(\Sigma)$:

$$S \sqsubseteq_{\mathcal{C}} T \quad \text{iff} \quad \forall \nu \in T.\ \exists \mu \in S.\ \mu \sqsubseteq_{\mathcal{D}} \nu$$

This order, due to Smyth [1978], is not generally antisymmetric, but in this case it is antisymmetric because the sets in $\mathcal{C}$ are up-closed. In fact, due to up-closure, the Smyth order is equivalent to reverse subset inclusion $S \sqsubseteq_{\mathcal{C}} T$ iff $S \supseteq T$. The bottom element of $\mathcal{C}(\Sigma)$ in this order is $\eta(\bot) = \uparrow\{\delta_\bot\} = \mathcal{D}(\Sigma_\bot)$, the set of all distributions. Operationally, this means that nontermination is identified with total uncertainty about the program outcome. As we unroll loops to obtain tighter and tighter approximations of their semantics, we rule out more and more possible behaviors.

In addition, we note that $\langle \mathcal{C}(\Sigma), \sqsubseteq_{\mathcal{C}}\rangle$ is a directed complete partial order (dcpo), meaning that all increasing chains of elements $S_1 \sqsubseteq_{\mathcal{C}} S_2 \sqsubseteq_{\mathcal{C}} \cdots$ have a supremum. Since $S \sqsubseteq_{\mathcal{C}} T$ is equivalent to $S \supseteq T$, then suprema are given by standard set intersection. So, to show that $\langle \mathcal{C}(\Sigma), \sqsubseteq_{\mathcal{C}}\rangle$ is a dcpo we need to show that any chain $S_1 \supseteq S_2 \supseteq \cdots$ has a supremum (*i.e.,* intersection) in $\mathcal{C}(\Sigma)$. McIver and Morgan [2005, Lemma B.4.4], showed that $\mathcal{D}(\Sigma)$ is compact using Tychonoff's Theorem, and therefore it is well known that such a chain has a nonempty intersection. The remaining properties (convexity, closure, up-closure) are well known to be preserved by intersections too.

3.2 Semantics of Sequential Commands

We are now ready to define the semantics, shown in Figure 1. We interpret commands denotationally as maps from states to convex sets of distributions, *i.e.,* $[\![C]\!] : \Sigma \to \mathcal{C}(\Sigma)$. No-ops and variable assignment are defined as point-mass distributions. Sequential composition is a Klesili composition. The probabilistic and nondeterministic choice operations are defined in terms of new operators:

$$S \oplus_p T \triangleq \{p \cdot \mu + (1-p)\cdot \nu \mid \mu \in S, \nu \in T\} \qquad S \,\&\, T \triangleq \bigcup_{p\in[0,1]} S \oplus_p T$$

As expected, probabilistic branching chooses an element of $[\![C_1]\!]\,(\sigma)$ with probability $p = [\![e]\!]\,(\sigma)$, and chooses an element of $[\![C_2]\!]\,(\sigma)$ with probability $1-p$. Nondeterministic choices are equivalent to a union of all the possible probabilistic choices between C_1 and C_2. If we think of nondeterminism being resolved by a scheduler, this operationally corresponds to the scheduler picking a bias p (which could be 0 or 1, corresponding to certainty), then flipping a coin with bias p to decide which command to execute [Segala and Lynch 1994; Varacca 2002].

He et al. [1997] showed that all of these operations preserve up-closedness and convexity and Morgan et al. [1996a] showed that they preserve topological closure (Morgan et al. refer to this as Cauchy Closure) and non-emptiness.

Conditional statements are defined in the standard way. A branch is taken deterministically depending on whether the guard e evaluates to true or false. As syntactic sugar, we define special syntax for biased coin flips and nondeterministic choice from a nonempty finite set $S = \{v_1, \ldots, v_n\}$:

$$x := \mathsf{flip}(e) \triangleq (x := \mathsf{true}) \oplus_e (x := \mathsf{false}) \qquad x \leftarrow S \triangleq (x := v_1) \,\&\, \cdots \,\&\, (x := v_n)$$

3.3 Semantics of Loops

Loops are interpreted as the least fixed point of $\Phi_{\langle C,e\rangle}$ (see Figure 1), which essentially means that:

$$[\![\textbf{while } e \textbf{ do } C]\!] = [\![\textbf{if } e \textbf{ then } (C \fatsemi \textbf{while } e \textbf{ do } C)]\!]$$

We will use the Kleene fixed point theorem to prove that a least fixed point exists. To do so, we first define an ordering on functions $\sqsubseteq^\bullet_C \subseteq (\Sigma \to C(\Sigma)) \times (\Sigma \to C(\Sigma))$, which is the pointwise extension of the order $\sqsubseteq_C$ from Section 3.1 and is defined as follows:

$$f \sqsubseteq^\bullet_C g \qquad \text{iff} \qquad \forall \sigma \in \Sigma.\, f(\sigma) \sqsubseteq_C g(\sigma) \qquad \text{iff} \qquad \forall \sigma \in \Sigma.\, f(\sigma) \sqsupseteq g(\sigma)$$

Clearly, the function $\bot^\bullet_C(\sigma) \triangleq \eta(\bot)$ is the bottom of this order, since $\eta(\bot)$ is the bottom of $\sqsubseteq_C$. The characteristic function $\Phi_{\langle C,e\rangle}$ is also Scott continuous in this order, meaning that it preserves suprema of directed sets [Zilberstein et al. 2024a, Lemma B.12]:

$$\sup_{f \in D} \Phi_{\langle C,e\rangle}(f) = \Phi_{\langle C,e\rangle}(\sup D)$$

So, by the Kleene fixed point theorem, we conclude that the least fixed point exists, and is characterized as the supremum of the iterates of $\Phi_{\langle C,e\rangle}$ over all the natural numbers.

$$\mathsf{lfp}\left(\Phi_{\langle C,e\rangle}\right)(\sigma) = \left(\sup_{n \in \mathbb{N}} \Phi^n_{\langle C,e\rangle}\left(\bot^\bullet_C\right)\right)(\sigma) = \bigcap_{n \in \mathbb{N}} \Phi^n_{\langle C,e\rangle}\left(\bot^\bullet_C\right)(\sigma)$$

These iterates are defined as $f^0 \triangleq \mathsf{id}$ and $f^{n+1} \triangleq f \circ f^n$, where $\circ$ is function composition.

4 Demonic Outcome Logic

We now present Demonic Outcome Logic, a new logic for reasoning about programs that are both randomized and nondeterministic. This logic has constructs for reasoning about probabilistic branching, inspired by Outcome Logic (OL) [Zilberstein et al. 2023] and probabilistic Hoare logics [Barthe et al. 2018; den Hartog 2002]. In addition, nondeterminism is treated *demonically*: the postcondition must hold regardless of how the nondeterminism is resolved.

4.1 Outcome Assertions

Outcome assertions $\varphi, \psi \in \mathsf{Prop}$ are used in pre- and postconditions of triples in Demonic Outcome Logic. The syntax is shown below, where $p \in [0,1]$ is a probability, and $P, Q \in \mathsf{Atom}$ are atoms.

$$\mathsf{Prop} \ni \varphi ::= \top \mid \bot \mid \varphi \wedge \psi \mid \varphi \oplus_p \psi \mid \varphi \,\&\, \psi \mid \lceil P \rceil \qquad p \in [0,1]$$
$$\mathsf{Atom} \ni P ::= \mathsf{true} \mid \mathsf{false} \mid P \wedge Q \mid P \vee Q \mid \neg P \mid e_1 = e_2 \mid e_1 \leq e_2 \mid \cdots$$

Atomic assertions $P, Q \in \mathsf{Atom}$ describe states, and are interpreted using $(\!|-|\!) : \mathsf{Atom} \to 2^\Sigma$, giving the set of states satisfying P, defined as usual. The satisfaction relation $\vDash \subseteq \mathcal{D}(\Sigma_\bot) \times \mathsf{Prop}$ defined in Figure 2 relates each assertion to probability distributions $\mu \in \mathcal{D}(\Sigma_\bot)$ (not *sets* distributions). As explained in Section 4.2, this corresponds to the logic's demonic treatment of nondeterminism.

$$
\begin{array}{lll}
\mu \vDash \top & & \text{always} \\
\mu \vDash \bot & & \text{never} \\
\mu \vDash \varphi \wedge \psi & \text{iff} & \mu \vDash \varphi \quad \text{and} \quad \mu \vDash \psi \\
\mu \vDash \varphi \oplus_p \psi & \text{iff} & \exists \mu_1, \mu_2.\ \ \mu = p \cdot \mu_1 + (1-p) \cdot \mu_2 \quad \text{and} \quad \mu_1 \vDash \varphi \quad \text{and} \quad \mu_2 \vDash \psi \\
\mu \vDash \varphi \,\&\, \psi & \text{iff} & \mu \vDash \varphi \oplus_p \psi \quad \text{for some } p \in [0,1] \\
\mu \vDash \lceil P \rceil & \text{iff} & \mathrm{supp}(\mu) \subseteq (\!|P|\!)
\end{array}
$$

Fig. 2. Definition of the satisfaction relation $\vDash \subseteq \mathcal{D}(\Sigma_\perp) \times \mathrm{Prop}$ for Outcome Assertions.

As expected, $\top$ is satisfied by any distribution, whereas $\bot$ is satisfied by nothing. The logical conjunction $\varphi \wedge \psi$ is true iff both conjuncts are true. If a distribution μ satisfies the probabilistic outcome conjunction $\varphi \oplus_p \psi$, then μ must be a convex combination with parameter p of a distribution satisfying φ and one satisfying ψ. Similarly, $\mu \vDash \varphi \,\&\, \psi$ means that μ is some convex combination (where the parameter is existentially quantified) of distributions satisfying φ and ψ. Finally, a distribution satisfies $\lceil P \rceil$ if its support is contained in $(\!|P|\!)$. Note that P can only describe states (and not $\bot$), so $\mu \vDash \lceil P \rceil$ implies that $\mu(\bot) = 0$, *i.e.,* that the program that generated μ almost surely terminated. This is a crucial difference between true and $\top$; whereas true guarantees almost sure termination, $\top$ is satisfied by any distribution.

As an example, $\lceil x = 1 \rceil \oplus_{\frac{1}{3}} \lceil x = 2 \rceil$ means that the event $x = 1$ occurs with probability $\frac{1}{3}$ and the event $x = 2$ occurs with probability $\frac{2}{3}$. On the other hand, $\lceil x = 1 \rceil \,\&\, \lceil x = 2 \rceil$ means that $x = 1$ occurs with some probability p and $x = 2$ occurs with probability $1 - p$. Given that we represent nondeterminism as convex union, $\varphi \,\&\, \psi$ characterizes nondeterministic choice. In addition, we can *forget* about the probabilities of outcomes by weakening $\varphi \oplus_p \psi \Rightarrow \varphi \,\&\, \psi$. As a shorthand, we will often write $\&_{k=1}^{n} \varphi_k$ instead of $\varphi_1 \,\&\, \cdots \,\&\, \varphi_n$ for finite & conjunctions of assertions. Unlike in standard Outcome Logic [Zilberstein et al. 2023], $\varphi \,\&\, \psi$ does not imply that both φ and ψ are realizable via an actual trace; for example if $\mu \vDash \lceil x = 1 \rceil \,\&\, \lceil x = 2 \rceil$, it is possible that the event $x = 1$ occurs with probability 0 according to μ. This is an intentional choice, as it allows us to retain desirable propositional properties such as idempotence of &, as explained in Section 2.1.

Echoing the equational laws from Section 2.1, outcome assertions can be manipulated using the following implications, where $\varphi \Rightarrow \psi$ means that if $\mu \vDash \varphi$ then $\mu \vDash \psi$ for all $\mu \in \mathcal{D}(\Sigma_\perp)$. These implications are not included in the syntax of Prop, since they are not allowed to be used in the pre- and postconditions of triples.

$$
\begin{array}{cc}
\varphi \,\&\, \varphi \Leftrightarrow \varphi & \varphi \oplus_p \varphi \Leftrightarrow \varphi \qquad \text{(Idempotence)} \\
\varphi \,\&\, \psi \Leftrightarrow \psi \,\&\, \varphi & \varphi \oplus_p \psi \Leftrightarrow \psi \oplus_{1-p} \varphi \qquad \text{(Commutativity)} \\
\varphi \,\&\, (\psi \,\&\, \vartheta) \Leftrightarrow (\varphi \,\&\, \psi) \,\&\, \vartheta & (\varphi \oplus_p \psi) \oplus_q \vartheta \Leftrightarrow \varphi \oplus_{pq} (\psi \oplus_{\frac{(1-p)q}{1-pq}} \vartheta) \qquad \text{(Associativity)} \\
\multicolumn{2}{c}{\varphi \oplus_p (\psi \,\&\, \vartheta) \Leftrightarrow (\varphi \oplus_p \psi) \,\&\, (\varphi \oplus_p \vartheta) \qquad \text{(Distributivity)}}
\end{array}
$$

We remark that these laws depend on what is—and, crucially, what is *not*—included in the assertion language. As we saw in Section 2.1, idempotence is delicate due to the fact that assertions are only *approximations* of the distributions that they model. Despite this, idempotence turns out to be crucial to the usability of the logic, as the soundness of several inference rules depends on it (*e.g.,* Nondet and Bounded Variant). The idempotence laws would be invalidated if the syntax included disjunctions or existential quantification. For example, in the following implication, x having value 1 or 2 each with probability $\frac{1}{2}$ does not imply that x is always 1 or always 2.

$$
\lceil x = 1 \rceil \oplus_{\frac{1}{2}} \lceil x = 2 \rceil \quad \Longrightarrow \quad (\lceil x = 1 \rceil \vee \lceil x = 2 \rceil) \oplus_{\frac{1}{2}} (\lceil x = 1 \rceil \vee \lceil x = 2 \rceil) \quad \not\Longrightarrow \quad \lceil x = 1 \rceil \vee \lceil x = 2 \rceil
$$

Note that the first implication is valid since $\lceil x = 1 \rceil \Rightarrow \lceil x = 1 \rceil \vee \lceil x = 2 \rceil$ and weakening can be applied inside of $\oplus_p$ as follows: $\lceil P \rceil \Rightarrow \lceil P' \rceil \vdash \lceil P \rceil \oplus_p \lceil Q \rceil \Rightarrow \lceil P' \rceil \oplus_p \lceil Q \rceil$.

We do not believe that the exclusion of disjunctions and existential quantification poses a severe restriction in practice. Existentials are often used to quantify over the values of certain program variables; in Demonic Outcome Logic, we quantify over values in a different way. In a typical logic, pre- and postconditions are predicates over individual states, so $\exists v : T.x = v$ asserts that the value of x takes on some value from the set T. In our case, we use predicates over distributions, so it is more appropriate to say $\&_{v \in T} \lceil x = v \rceil$, which asserts that the value of x is in T for every state in the support of the distribution. We use this technique in Section 6.2.

4.2 Semantics of Triples

Similar to Hoare Logic and Outcome Logic [Zilberstein et al. 2023], specifications in Demonic Outcome Logic are triples of the form $\langle \varphi \rangle\ C\ \langle \psi \rangle$. Intuitively, the semantics of these triples is that if states are initially distributed according to a distribution $\mu \in \mathcal{D}(\Sigma_\perp)$ that satisfies φ, then after the program C is run, the resulting states will be distributed according to some distribution $\nu \in \mathcal{D}(\Sigma_\perp)$ that satisfies ψ, regardless of how nondeterministic choices in C are resolved. We formalize the semantics of triples, denoted by $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$, as follows.

Definition 4.1 (Semantics of Demonic Outcome Triples).

$$\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \text{iff} \qquad \forall \mu \in \mathcal{D}(\Sigma_\perp).\quad \mu \vDash \varphi \quad \Longrightarrow \quad \forall \nu \in [\![C]\!]^\dagger(\uparrow\{\mu\}).\quad \nu \vDash \psi$$

We note that when limited to basic assertions $P, Q \in \mathrm{Atom}$, $\vDash \langle \lceil P \rceil \rangle\ C\ \langle \lceil Q \rceil \rangle$ is semantically equivalent to a total correctness Hoare triple [Manna and Pnueli 1974] (albeit, in a language with randomization). That is, for any start state $\sigma \in (\!|P|\!)$, the program will terminate in a state $\tau \in (\!|Q|\!)$.

4.3 Inference Rules

The inference rules for reasoning about non-looping commands are shown in Figure 3 (we will revisit loops in Section 5). We write $\vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$ to mean that a triple is derivable using these rules. This relates to the semantics of triples via the following soundness theorem, which is proved by induction on the derivation [Zilberstein et al. 2024a, §C.2].

Theorem 4.2 (Soundness).

$$\vdash \langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \Longrightarrow \qquad \vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$$

Now, we will describe the rules in more depth.

Sequential and Probabilistic Commands. Many of the rules for analyzing commands are as expected. The Skip rule simply preserves the precondition, as no-ops do not affect the distribution of outcomes. The Assign rule uses standard backward substitution, where $\varphi[e/x]$ is the assertion obtained by syntactically substituting e for all occurrences of x. The Seq rule allows us to analyze sequences of commands compositionally, and relies on the fact that $[\![C]\!]^\dagger(S) = \bigcup_{\mu \in S} [\![C]\!]^\dagger(\uparrow\{\mu\})$ for soundness.

In order to analyze a probabilistic choice $C_1 \oplus_e C_2$ using Prob, the precondition φ must contain enough information to ascertain that the expression e evaluates to a precise probability $p \in [0, 1]$. If e is a literal p, then this restriction is trivial since $\varphi \Rightarrow \lceil p = p \rceil$ for any φ. The postcondition then joins the outcomes of the two branches using $\oplus_p$. Similarly, the rules for analyzing if statements require that the precondition selects one of the two branches deterministically. If1 applies when the precondition forces the true branch to be taken and If2 applies when it forces the false branch. We will soon see derived rules that allow both branches to be analyzed in a single rule.

Commands

$$\frac{}{\langle\varphi\rangle\ \mathbf{skip}\ \langle\varphi\rangle}\textsc{Skip} \qquad \frac{}{\langle\varphi[e/x]\rangle\ x \coloneqq e\ \langle\varphi\rangle}\textsc{Assign} \qquad \frac{\langle\varphi\rangle\ C_1\ \langle\vartheta\rangle \qquad \langle\vartheta\rangle\ C_2\ \langle\psi\rangle}{\langle\varphi\rangle\ C_1 \fatsemi C_2\ \langle\psi\rangle}\textsc{Seq}$$

$$\frac{\varphi \Rightarrow \lceil e = p\rceil \quad \langle\varphi\rangle\ C_1\ \langle\psi_1\rangle \quad \langle\varphi\rangle\ C_2\ \langle\psi_2\rangle}{\langle\varphi\rangle\ C_1 \oplus_e C_2\ \langle\psi_1 \oplus_p \psi_2\rangle}\textsc{Prob} \qquad \frac{\langle\lceil P\rceil\rangle\ C_1\ \langle\psi_1\rangle \quad \langle\lceil P\rceil\rangle\ C_2\ \langle\psi_2\rangle}{\langle\lceil P\rceil\rangle\ C_1 \,\&\, C_2\ \langle\psi_1 \,\&\, \psi_2\rangle}\textsc{Nondet}$$

$$\frac{\varphi \Rightarrow \lceil e\rceil \qquad \langle\varphi\rangle\ C_1\ \langle\psi\rangle}{\langle\varphi\rangle\ \mathbf{if}\ e\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle\psi\rangle}\textsc{If1} \qquad \frac{\varphi \Rightarrow \lceil \neg e\rceil \qquad \langle\varphi\rangle\ C_2\ \langle\psi\rangle}{\langle\varphi\rangle\ \mathbf{if}\ e\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle\psi\rangle}\textsc{If2}$$

Structural Rules

$$\frac{\langle\varphi_1\rangle\ C\ \langle\psi_1\rangle \qquad \langle\varphi_2\rangle\ C\ \langle\psi_2\rangle}{\langle\varphi_1 \oplus_p \varphi_2\rangle\ C\ \langle\psi_1 \oplus_p \psi_2\rangle}\textsc{Prob Split} \qquad \frac{\langle\varphi_1\rangle\ C\ \langle\psi_1\rangle \qquad \langle\varphi_2\rangle\ C\ \langle\psi_2\rangle}{\langle\varphi_1 \,\&\, \varphi_2\rangle\ C\ \langle\psi_1 \,\&\, \psi_2\rangle}\textsc{ND Split}$$

$$\frac{\varphi' \Rightarrow \varphi \quad \langle\varphi\rangle\ C\ \langle\psi\rangle \quad \psi \Rightarrow \psi'}{\langle\varphi'\rangle\ C\ \langle\psi'\rangle}\textsc{Consequence} \qquad \frac{\langle\varphi\rangle\ C\ \langle\psi\rangle \quad \mathsf{mod}(C) \cap \mathsf{fv}(P) = \varnothing}{\langle\varphi \wedge \lceil P\rceil\rangle\ C\ \langle\psi \wedge \lceil P\rceil\rangle}\textsc{Constancy}$$

Fig. 3. Inference rules for non-looping commands in Demonic Outcome Logic.

Structural Rules. The bottom of Figure 3 also contains structural rules, which do not depend on the program command. The Prob Split and ND Split rules allow us to deconstruct pre- and postconditions in order to build derivations compositionally. As we will see shortly, these rules are necessary to analyze nondeterministic choices, since the Nondet rule requires the precondition to be a basic assertion $\lceil P\rceil$. They are also useful for analyzing if statements, since the If1 and If2 rules require the precondition to imply the truth and falsity of the guard, respectively. The soundness of these rules relies on the following equality [Zilberstein et al. 2024a, Lemma C.2].

$$[\![C]\!]^\dagger(S_1 \oplus_p S_2) \quad = \quad [\![C]\!]^\dagger(S_1) \oplus_p [\![C]\!]^\dagger(S_2)$$

Next, we have the usual rule of Consequence, which allows the precondition to be strengthened and the postcondition to be weakened. These implications are semantic ones; we do not provide proof rules to dispatch them beyond the laws at the end of Section 4.1.

Finally, the rule of Constancy allows us to conjoin additional information P about the program state, so long as it does not involve any of the modified program variables. We let $\mathsf{mod}(C)$ denote the set of variables modified by the program C, defined inductively on its structure. One subtlety is that possibly nonterminating programs must be considered to modify all the program variables, meaning that Constancy only applies to terminating programs. However, this restriction does not matter much in practice, since all the loop rules we present in Section 5 guarantee almost sure termination. In addition $\mathsf{fv}(P)$ is the set of variables occurring free in P. Just like the frame rule from Outcome Separation Logic, P is a basic assertion rather than an outcome assertion, since this extra information concerns only the local state and not the branching behavior of the program [Zilberstein et al. 2024b]. The soundness of Constancy is considerably simpler than that of the frame rule, since it does not deal with dynamically allocated pointers and aliasing.

Nondeterministic Branching. The Nondet rule can only be applied if the precondition is a basic assertion P. If the precondition contained information about probabilistic outcomes, then this rule would be unsound. To demonstrate why this is the case, let us revisit the coin flip game:

$$x \coloneqq \mathsf{flip}\left(\tfrac{1}{2}\right) \fatsemi y \leftarrow \mathbb{B}$$

If we imagine that nondeterminism is controlled by an adversary, then it is always possible for the adversary to *guess* the coin flip, that is, to force x and y to be equal. However, if we allowed the precondition in the Nondet rule to contain probabilistic outcomes, then we could derive the triple below, stating that $x = y$ always occurs with probability $\frac{1}{2}$, which is untrue.

$$\dfrac{\dfrac{\vdots}{\langle \lceil x = \text{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \text{false} \rceil \rangle\ y \coloneqq \text{false}\ \langle \lceil x \neq y \rceil \oplus_{\frac{1}{2}} \lceil x = y \rceil \rangle} \qquad \dfrac{\vdots}{\langle \lceil x = \text{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \text{false} \rceil \rangle\ y \coloneqq \text{true}\ \langle \lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil \rangle}}{\langle \lceil x = \text{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \text{false} \rceil \rangle\ y \leftarrow \mathbb{B}\ \langle (\lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil)\ \&\ (\lceil x \neq y \rceil \oplus_{\frac{1}{2}} \lceil x = y \rceil) \rangle}\ \textsc{Nondet}\ \text{(incorrect usage)}$$
$$\dfrac{\langle \lceil x = \text{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \text{false} \rceil \rangle\ y \leftarrow \mathbb{B}\ \langle (\lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil)\ \&\ (\lceil x \neq y \rceil \oplus_{\frac{1}{2}} \lceil x = y \rceil) \rangle}{\langle \lceil x = \text{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \text{false} \rceil \rangle\ y \leftarrow \mathbb{B}\ \langle \lceil x = y \rceil \oplus_{\frac{1}{2}} \lceil x \neq y \rceil \rangle}\ \textsc{Consequence}$$

Instead—as shown below—we must de-structure the precondition using Prob Split, and then apply Nondet twice, using each of the basic assertions ($\lceil x = \text{true} \rceil$ and $\lceil x = \text{false} \rceil$) as preconditions. This has the effect of expanding each basic outcome inside of the $\oplus_{\frac{1}{2}}$. After applying idempotence in the postcondition, we see that $x = y$ occurs with unknown probability, which does not preclude that the adversary could force it to occur. In fact, the postcondition is equivalent to true.

$$\dfrac{\dfrac{\dfrac{\vdots}{\langle \lceil x = \text{true} \rceil \rangle\ y \leftarrow \mathbb{B}\ \langle \lceil x = y \rceil\ \&\ \lceil x \neq y \rceil \rangle}\ \textsc{Nondet} \qquad \dfrac{\vdots}{\langle \lceil x = \text{false} \rceil \rangle\ y \leftarrow \mathbb{B}\ \langle \lceil x \neq y \rceil\ \&\ \lceil x = y \rceil \rangle}\ \textsc{Nondet}}{\langle \lceil x = \text{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \text{false} \rceil \rangle\ y \leftarrow \mathbb{B}\ \langle (\lceil x = y \rceil\ \&\ \lceil x \neq y \rceil) \oplus_{\frac{1}{2}} (\lceil x \neq y \rceil\ \&\ \lceil x = y \rceil) \rangle}\ \textsc{Prob Split}}{\langle \lceil x = \text{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \text{false} \rceil \rangle\ y \leftarrow \mathbb{B}\ \langle \lceil x = y \rceil\ \&\ \lceil x \neq y \rceil \rangle}\ \textsc{Consequence}$$

Requiring basic assertions as the precondition may seem restrictive, but the Nondet rule can still be applied in all scenarios by deconstructing the precondition, as we saw in Section 2.2 [Zilberstein et al. 2024a, §A.2]. The soundness of Nondet fundamentally depends on idempotence. In the proof, we show that if $\mu \vDash P$, then $\nu \vDash \psi_1\ \&\ \psi_2$ for each $\nu \in [\![C_1\ \&\ C_2]\!]\,(\sigma)$ and $\sigma \in \mathrm{supp}(\mu)$. Any distribution in $[\![C_1\ \&\ C_2]\!]^\dagger(\uparrow\{\mu\})$ is therefore a convex combination of distributions, all of which satisfy $\psi_1\ \&\ \psi_2$. We collapse that convex combination using, *e.g.*, $(\psi_1\ \&\ \psi_2) \oplus_p (\psi_1\ \&\ \psi_2) \Rrightarrow \psi_1\ \&\ \psi_2$ (see Lemma C.5 for the more general property). Soundness must be established in this way, since $[\![C_1\ \&\ C_2]\!]^\dagger(S) \neq [\![C_1]\!]^\dagger(S)\ \&\ [\![C_2]\!]^\dagger(S)$ and so:

$$[\![C_1]\!]^\dagger(S) \vDash \psi_1 \quad \text{and} \quad [\![C_2]\!]^\dagger(S) \vDash \psi_2 \qquad \not\Rightarrow \qquad [\![C_1\ \&\ C_2]\!]^\dagger(S) \vDash \psi_1\ \&\ \psi_2$$

On the other hand, $[\![C_1\ \&\ C_2]\!]\,(\sigma) = [\![C_1]\!]\,(\sigma)\ \&\ [\![C_2]\!]\,(\sigma)$, so we can analyze nondeterministic branching *compositionally* only when starting from a single state.

Derived Rules. In addition to the rules in Figure 3, we also provide some derived rules for convenience in common scenarios. All the derivations are shown in Zilberstein et al. [2024a, §C.3]. The first rules pertain to conditional statements. These rules use Prob Split and ND Split to deconstruct the pre- and postconditions in order to analyze both branches of the conditional statement. These are similar to the rules found in Ellora [Barthe et al. 2018] and Outcome Logic [Zilberstein 2024].

$$\dfrac{\substack{\varphi_1 \Rightarrow \lceil e \rceil \\ \varphi_2 \Rightarrow \lceil \neg e \rceil} \quad \langle \varphi_1 \rangle\ C_1\ \langle \psi_1 \rangle \quad \langle \varphi_2 \rangle\ C_2\ \langle \psi_2 \rangle}{\langle \varphi_1 \oplus_p \varphi_2 \rangle\ \texttt{if}\ e\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \psi_1 \oplus_p \psi_2 \rangle} \qquad \dfrac{\substack{\varphi_1 \Rightarrow \lceil e \rceil \\ \varphi_2 \Rightarrow \lceil \neg e \rceil} \quad \langle \varphi_1 \rangle\ C_1\ \langle \psi_1 \rangle \quad \langle \varphi_2 \rangle\ C_2\ \langle \psi_2 \rangle}{\langle \varphi_1\ \&\ \varphi_2 \rangle\ \texttt{if}\ e\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \psi_1\ \&\ \psi_2 \rangle}$$

In addition, if the precondition of the conditional statement is a basic assertion P, then we can use the typical conditional rule from Hoare logic. This relies on the Hahn decomposition theorem: $\lceil P \rceil \Rightarrow \lceil P \wedge e \rceil\ \&\ \lceil P \wedge \neg e \rceil$, that is, if $\mu \vDash P$, then μ can be separated into two portions where e

is true and false, respectively. Due to idempotence, we can simplify the postconditions using the consequence $\psi \,\&\, \psi \Rightarrow \psi$.

$$\frac{\langle \lceil P \wedge e \rceil \rangle\ C_1\ \langle \psi \rangle \qquad \langle \lceil P \wedge \neg e \rceil \rangle\ C_2\ \langle \psi \rangle}{\langle \lceil P \rceil \rangle\ \texttt{if}\ e\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \psi \rangle}$$

Finally, we provide rules for analyzing the coin flip and nondeterministic selection syntactic sugar introduced in Section 3.2. The flip rule is derived using a straightforward application of Prob. The rule for nondeterministic selection is proven by induction on the size of S (recall that S is finite).

$$\frac{\varphi \Rightarrow \lceil e = p \rceil \qquad x \notin \mathsf{fv}(\varphi)}{\langle \varphi \rangle\ x \coloneqq \mathsf{flip}(e)\ \langle \varphi \wedge (\lceil x = \mathsf{true} \rceil \oplus_p \lceil x = \mathsf{false} \rceil) \rangle} \qquad \frac{}{\langle \lceil \mathsf{true} \rceil \rangle\ x \leftarrow S\ \langle \,\&_{v \in S} \lceil x = v \rceil \rangle}$$

Although the precondition of the nondeterministic selection rule is true, it can be used in conjunction with the rule of Constancy so that any basic assertion P can be the precondition. Beyond that, to extend to any precondition φ, de-structuring rules must be applied just like with the Nondet rule.

Remark 1 (Completeness). *We have not explored completeness of Demonic Outcome Logic, even for the loop-free fragment. One reason for this is that the derivations witnessed by the completeness proofs for similar probabilistic logics do not mimic the sort of derivations that one would produce by hand. For example, in Ellora [Barthe et al. 2018], the completeness proof involves quantifying over the (infinitely many) distributions that could satisfy the precondition, then showing that a derivation of the strongest postcondition is possible given any fixed one of those distributions. The complexity comes not only from loops, but also purely sequential constructs like probabilistic branching and if statements. To see examples of where this complexity arises, see Zilberstein [2024, Definition 3.5], or Dardinier and Müller [2024, Example 1].*

5 Analyzing Loops

In this section, we discuss proof rules for analyzing loops. We are inspired by work on weakest pre-expectations [Kaminski 2019; McIver and Morgan 2005; McIver et al. 2018], where probabilistic loop analysis has been studied extensively, but will argue in this section that our program logic approach has two advantages.

Fewer Conditions to Check. The weakest pre-expectation proof rules involve multiple checks, which include both *sub*-invariants and *super*-invariants and computing expected values of ranking functions. In contrast, in Demonic OL all the proof rules revolve around just one construct—outcome triples—and the premises of the rules can accordingly be consolidated.

Multiple Outcomes. It is often useful to specify programs in terms of their distinct outcomes, which we achieve using the assertions from Section 4.1. Pre-expectation calculi can only represent multiple outcomes by carrying out several distinct derivations, whereas Demonic OL can do so in one shot.

5.1 Almost Sure Termination

As we mentioned in Section 3.3, our semantics based on the Smyth powerdomain is suitable for total correctness—specifications implying that the program terminates. Since we are in a probabilistic setting, it makes sense to talk about a finer notion of termination—*almost sure termination*—meaning that the program terminates with probability 1. In terms of our program semantics, a program almost surely terminates if $\bot$ does not appear in the support of any of its resultant distributions.

Definition 5.1 (Almost Sure Termination). A program C *almost surely terminates* on input σ iff

$$\forall \mu \in [\![C]\!]\,(\sigma).\ \bot \notin \mathsf{supp}(\mu)$$

In addition, C *universally almost surely terminates* if it almost surely terminates on all $\sigma \in \Sigma$.

Going further, we show how almost sure termination is established in Demonic Outcome Logic.

Theorem 5.2. *A program C almost surely terminates starting from any state satisfying P if:*

$$\vDash \langle \lceil P \rceil \rangle\ C\ \langle \lceil \mathsf{true} \rceil \rangle$$

As a corollary, C universally almost surely terminates if $\vDash \langle \lceil \mathsf{true} \rceil \rangle\ C\ \langle \lceil \mathsf{true} \rceil \rangle$.

Proof. The triple $\vDash \langle \lceil P \rceil \rangle\ C\ \langle \lceil \mathsf{true} \rceil \rangle$ means that if $\sigma \in (\!|P|\!)$, that is, if $\delta_\sigma \vDash \lceil P \rceil$, then $\mathrm{supp}(\mu) \subseteq (\!|\mathsf{true}|\!) = \Sigma$ for all $\mu \in [\![C]\!]\,(\sigma)$. Since $\bot \notin \Sigma$, $\mathrm{supp}(\mu) \subseteq \Sigma$ iff $\bot \notin \mathrm{supp}(\mu)$. □

Following from Theorem 5.2, if $\langle \lceil P \rceil \rangle\ C\ \langle \psi \rangle$ holds, then C almost surely terminates as long as $\psi \Rightarrow \lceil \mathsf{true} \rceil$, which is simple to check in many cases. For example, if ψ is formed as a collection of atoms $Q_1, \ldots, Q_n$ joined by & and $\oplus_p$ connectives, then the program almost surely terminates since $\lceil Q_i \rceil \Rightarrow \lceil \mathsf{true} \rceil$ holds trivially, and connectives can be collapsed using idempotence of & and $\oplus_p$.

5.2 The Zero-One Law

McIver and Morgan [2005] showed that under certain conditions, probabilistic programs must terminate with probability either 0 or 1. In this circumstance, almost sure termination can be established simply by showing that the program terminates with nonzero probability.

The original rule of McIver and Morgan [2005, §2.6] used a propositional invariant P to describe all reachable states after each iteration of the loop. We generalize their rule by using an outcome assertion φ as the invariant, so that in addition to describing which states are reachable, we can also describe how those reachable states are distributed. Our version of the rule is stated below.

$$\frac{\varphi \Rightarrow \lceil e \rceil \quad \psi \Rightarrow \lceil \neg e \rceil \quad \langle \varphi \rangle\ C\ \langle \varphi\ \&\ \psi \rangle \quad \langle \varphi \rangle\ \mathbf{while}\ e\ \mathbf{do}\ C\ \langle \lceil \neg e \rceil \oplus_p \top \rangle \quad p > 0}{\langle \varphi \rangle\ \mathbf{while}\ e\ \mathbf{do}\ C\ \langle \psi \rangle}\ \textsc{Zero-One}$$

Given some loop **while** e **do** C, the first step of the Zero-One law is to come up with an invariant pair φ and ψ, where φ represents the distribution of states where the guard e remains true and ψ represents the distribution of states in which the loop has terminated. More precisely, $\varphi \Rightarrow \lceil e \rceil$ and $\psi \Rightarrow \lceil \neg e \rceil$. Next, we must prove that this is an invariant pair, by proving the following triple.

$$\langle \varphi \rangle\ C\ \langle \varphi\ \&\ \psi \rangle \tag{3}$$

That is, if the initial states are distributed according to φ, then loop body C will reestablish φ with some probability q and will terminate (in ψ) with probability $1 - q$. In the limit, ψ will describe the entire distribution of *terminating* outcomes [Zilberstein et al. 2024a, Lemma D.2], although we do not yet know if the loop almost surely terminates. We can establish almost sure termination if φ guarantees some nonzero probability of termination, as represented by the following triple.

$$\langle \varphi \rangle\ \mathbf{while}\ e\ \mathbf{do}\ C\ \langle \lceil \neg e \rceil \oplus_p \top \rangle \qquad \text{where} \qquad p > 0 \tag{4}$$

That is, for every $\mu \vDash \varphi$, the loop will always terminate (with $\neg e$ holding) with probability at least $p > 0$. From (3), we know that on the i^{th} iteration of the loop, there is some probability q_i of reestablishing the invariant φ, in which case the loop will continue to execute. This means that the total probability of nontermination is the product of all the q_i. In general, it is possible for such an infinite product to converge to a nonzero probability, however, from (4) we know that every tail of that product must be at most $1 - p$, which is strictly less than 1.

$$\mathbb{P}[\text{nonterm}] \quad = \quad q_1 \times q_2 \times q_3 \times \cdots \times \underbrace{q_n \times q_{n+1} \times \cdots}_{\leq 1-p} \quad = \quad 0$$

This implies that the product of the q_i must go to 0, so that the loop almost surely terminates. The full soundness proof is available in Zilberstein et al. [2024a, §D.1].

5.3 Proving Termination with Variants and Ranking Functions

Although the Zero-One law provides a means for proving almost sure termination, it can be difficult to use directly, because one still must establish a *minimum probability of termination*. In this section, we provide some inference rules derived from Zero-One that are easier to apply. The first rule uses a bounded family of *variants* $(\varphi_n)_{n=0}^N$, where $\varphi_0 \Rightarrow \neg e$ and $\varphi_n \Rightarrow e$ for $1 \le n \le N$. The index n can be thought of as a *rank*, so that we get closer to termination as n descends towards zero. The premise of the rule is that the rank must decrease by at least 1 with probability $p > 0$ on each iteration. We represent this formally using the following triple.

$$\langle \varphi_n \rangle\ C\ \langle (\&_{k=0}^{n-1} \varphi_k) \oplus_p (\&_{k=0}^{N} \varphi_k) \rangle$$

The assertion $\&_{k=0}^{n-1} \varphi_k$ is an aggregation of all the variants with rank strictly lower than n, essentially meaning that the states must be distributed according to those variants, but without specifying their relative probabilities. Note that, *e.g.*, $\varphi_{n-1} \Rightarrow \&_{k=0}^{n-1} \varphi_k$, so it is possible to establish this assertion if the rank always decreases by exactly 1. So, the postcondition states that with probability p the rank decreases by at least 1. That means that $\&_{k=0}^{N} \varphi_k$ must hold with probability $1 - p$, meaning that the rank can *increase* too, as long as that increase is not too likely.

In order to establish a minimum termination probability, we note that starting at φ_n, it takes at most N steps to reach rank 0, therefore the loop terminates with probability at least p^N, which is greater than 0 since $p > 0$ and N is finite. Putting this all together, we get the following rule.

$$\frac{\varphi_0 \Rightarrow \lceil \neg e \rceil \quad \forall n \in \{1, \dots, N\}.\ \varphi_n \Rightarrow \lceil e \rceil \quad \langle \varphi_n \rangle\ C\ \langle (\&_{k=0}^{n-1} \varphi_k) \oplus_p (\&_{k=0}^{N} \varphi_k) \rangle}{\langle \&_{k=0}^{N} \varphi_k \rangle\ \textbf{while}\ e\ \textbf{do}\ C\ \langle \varphi_0 \rangle}\ \textsc{Bounded Variant}$$

As a special case of the Bounded Variant rule, we can derive the variant rule of McIver and Morgan [2005, Lemma 7.5.1]. Instead of recording the rank with a family of outcome assertions, we will instead use an integer-valued expression e_{rank}. This can be thought of as a ranking function, since $[\![e_{\mathsf{rank}}]\!] : \Sigma \to \mathbb{Z}$ gives us a rank for each state $\sigma \in \Sigma$. In addition, the propositional invariant P describes the reachable states after each iteration of the loop. Finally, as long as the invariant holds and the loop guard is true, e_{rank} must be bounded between ℓ and h, in other words $\lceil P \wedge e \rceil \Rightarrow \lceil \ell \le e_{\mathsf{rank}} \le h \rceil$. The premise of the rule is that each iteration of the loop must strictly decrease the rank with probability at least $p > 0$. That is, for any n:

$$\langle \lceil P \wedge e \wedge e_{\mathsf{rank}} = n \rceil \rangle\ C\ \langle \lceil P \wedge e_{\mathsf{rank}} < n \rceil \oplus_p \lceil P \rceil \rangle$$

Given that the rank is integer-valued and strictly decreasing, it must fall below the lower bound ℓ within at most $h - \ell + 1$ steps, at which point e becomes false since $\lceil P \wedge e \rceil \Rightarrow \lceil \ell \le e_{\mathsf{rank}} \rceil$. So the loop terminates with probability at least $p^{h-\ell+1}$, and so by the Zero-One law, it almost surely terminates. In Zilberstein et al. [2024a, §D.2], we show how this rule is derived from Bounded Variant by letting $\varphi_0 \triangleq \lceil P \wedge \neg e \rceil$ and $\varphi_n \triangleq \lceil P \wedge e \wedge e_{\mathsf{rank}} = \ell + n - 1 \rceil$ for $1 \le n \le h - \ell + 1$. The full rule is shown below:

$$\frac{\lceil P \wedge e \rceil \Rightarrow \lceil \ell \le e_{\mathsf{rank}} \le h \rceil \quad \forall n.\ \langle \lceil P \wedge e \wedge e_{\mathsf{rank}} = n \rceil \rangle\ C\ \langle \lceil P \wedge e_{\mathsf{rank}} < n \rceil \oplus_p \lceil P \rceil \rangle}{\langle \lceil P \rceil \rangle\ \textbf{while}\ e\ \textbf{do}\ C\ \langle \lceil P \wedge \neg e \rceil \rangle}\ \textsc{Bounded Rank}$$

As an example application of the rule, recall the following resetting random walk, where the agent moves left with probability $\frac{1}{2}$, otherwise it resets to a position chosen by an adversary.

$$\begin{array}{l} \textbf{while}\ x > 0\ \textbf{do} \\ \quad (x := x - 1) \oplus_{\frac{1}{2}} (x \leftarrow \{1, \dots, 5\}) \end{array}$$

When starting in a state where $0 \le x \le 5$, this program almost surely terminates. Although there are uncountably many nonterminating traces, the probability of nontermination is zero. Even in the worst case in which the adversary always chooses 5, the agent eventually moves left in five consecutive iterations with probability 1. In fact, the program terminates in finite expected time, as it is a Bernoulli process. Using the invariant $P \triangleq 0 \le x \le 5$, the ranking function $e_{\mathsf{rank}} = x$, and the probability $p = \frac{1}{2}$, the premise of Bounded Rank is simply the following triple, which is easy to prove using the Prob and Assign rules.

$$\langle\lceil 0 < x \le 5 \wedge x = n\rceil\rangle\ (x \coloneqq x - 1) \oplus_{\frac{1}{2}} (x \leftarrow \{1, \ldots, 5\})\ \langle\lceil 0 \le x < n\rceil \oplus_{\frac{1}{2}} \lceil 0 \le x \le 5\rceil\rangle$$

McIver and Morgan [2005, Lemma 7.6.1] showed that Bounded Rank is complete for proving almost sure termination if the state space is finite. While we did not assume a finite state space for our language, this result nonetheless shows that the rule is broadly applicable. In addition, our new Bounded Variant rule is more expressive, as it allows the invariants to have multiple outcomes. We will see an example of how this is useful in Section 6.2.

More sophisticated rules are possible in which the rank need not be bounded. One such rule is shown below, based on that of McIver et al. [2018]. Instead of bounding the rank, we now require that the *expected* rank decreases each iteration, which is guaranteed in our rule by bounding the amount that it can increase in the case that an increase occurs.

$$\frac{\langle\lceil P \wedge e \wedge e_{\mathsf{rank}} = k\rceil\rangle\ C\ \langle\lceil P \wedge e_{\mathsf{rank}} \le k - d\rceil \oplus_p \lceil P \wedge e_{\mathsf{rank}} \le k + \frac{p}{1-p}d\rceil\rangle}{\langle\lceil P\rceil\rangle\ \textbf{while}\ e\ \textbf{do}\ C\ \langle\lceil P \wedge \neg e\rceil\rangle}\ \textsc{Progressing Rank}$$

We can use this rule to prove almost sure termination of the following demonically fair random walk, in which the agent steps towards the origin with probability $\frac{1}{2}$, otherwise an adversary can choose whether or not the adversary steps away from the origin.

$$\begin{array}{l}\textbf{while}\ x > 0\ \textbf{do}\\ \quad x \coloneqq x - 1 \oplus_{\frac{1}{2}} (x \coloneqq x + 1\ \&\ \textbf{skip})\end{array}$$

We instantiate Progressing Rank with $P \triangleq x \ge 0$, $e_{\mathsf{rank}} \triangleq x$, $p \triangleq \frac{1}{2}$, and $d \triangleq 1$ to get the following premise, which is easily proven using Prob, Nondet, Assign, and basic propositional reasoning.

$$\langle\lceil x = k > 0\rceil\rangle\ x \coloneqq x - 1 \oplus_{\frac{1}{2}} (x \coloneqq x + 1\ \&\ \textbf{skip})\ \langle\lceil 0 \le x \le k - 1\rceil \oplus_{\frac{1}{2}} \lceil 0 \le x \le k + 1\rceil\rangle$$

The full derivation for the demonic random walk and soundness proof for a more general version of the Progressing Rank rule—where p and d do not have to be constants—appear in Zilberstein et al. [2024a, §D.3]. Compared to the original version of this rule due to McIver et al. [2018], our rule consolidates three premises into just one.

6 Case Studies

In this section, we present three case studies in how our logic can be used to analyze programs that contain both probabilistic and nondeterministic operations.

6.1 The Monty Hall Problem

The Monty Hall problem is a classic paradox in probability theory in which a game show contestant tries to win a car by guessing which door it is behind. The player has three initial choices; the car is behind one door and the other two contain goats. After choosing a door, the host reveals one of the goats among the unopened doors and the player chooses to *stick* with the original door or *switch—which strategy is better?*

We model this problem with the **Game** program on the left side of Figure 4. First, the car is randomly placed behind a door. Next, the player chooses a door. Without loss of generality, we say

$$\textbf{Game} \triangleq \begin{cases} \mathsf{car} := 1 \oplus_{\frac{1}{3}} (\mathsf{car} := 2 \oplus_{\frac{1}{2}} \mathsf{car} := 3) \fatsemi \\ \mathsf{pick} := 1 \fatsemi \\ \textbf{if } \mathsf{car} = 1 \textbf{ then} \\ \quad \mathsf{open} \leftarrow \{2,3\} \\ \textbf{else if } \mathsf{car} = 2 \textbf{ then} \\ \quad \mathsf{open} := 3 \\ \textbf{else} \\ \quad \mathsf{open} := 2 \end{cases} \qquad \textbf{Switch} \triangleq \begin{cases} \textbf{if } \mathsf{open} = 2 \textbf{ then} \\ \quad \mathsf{pick} := 3 \\ \textbf{else} \\ \quad \mathsf{pick} := 2 \end{cases}$$

Fig. 4. Left: the Monty Hall program. Right: additional program to switch doors

that the player always chooses door 1. We could have instead universally quantified the choice of the player to indicate that the claim holds for any deterministic strategy. In that case, the proof would be largely the same, although with added cases so that the host does not open the player's door; we instead fix the player's choice to be door 1 for simplicity. Finally, the host *nondeterministically* chooses a door to open, which is neither the player's pick, nor the car. We now use Demonic OL to determine the probability of winning (that is, pick = car) in both the stick strategy (**Game**) and the switch strategy (**Game** $\fatsemi$ **Switch**, where **Switch** is the program representing the player switching doors, presented on the right of Figure 4). We derive the following triple for the **Game** program:

$$\langle \lceil \mathsf{true} \rceil \rangle$$
$$\textbf{Game}$$
$$\langle (\lceil \mathsf{car} = 1 \rceil \wedge (\lceil \mathsf{open} = 2 \rceil \,\&\, \lceil \mathsf{open} = 3 \rceil)) \oplus_{\frac{1}{3}} (\lceil \mathsf{car} = 2 \wedge \mathsf{open} = 3 \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{car} = 3 \wedge \mathsf{open} = 2 \rceil) \rangle$$

The derivation, using the rules from Figure 3, is shown in Figure 5. Note that to analyze the if statement, we first use the rule of Constancy with pick = 1 and then de-structure the remaining assertion with two applications of Prob Split. Below, we show manipulation of the postcondition of **Game** to give us the probability of winning using the *stick* strategy. Irrelevant information about the opened door is first removed. Next, since pick = 1 in all cases, we weaken car = 1 to pick = car, and use pick $\neq$ car in the other outcomes. Finally, we use idempotence of $\oplus_{\frac{1}{2}}$ in the last step.

$$\begin{aligned} &\lceil \mathsf{pick} = 1 \rceil \wedge ((\lceil \mathsf{car} = 1 \rceil \wedge (\lceil \mathsf{open} = 2 \rceil \,\&\, \lceil \mathsf{open} = 3 \rceil)) \\ &\qquad \oplus_{\frac{1}{3}} (\lceil \mathsf{car} = 2 \wedge \mathsf{open} = 3 \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{car} = 3 \wedge \mathsf{open} = 2 \rceil)) \\ &\implies \lceil \mathsf{pick} = 1 \rceil \wedge (\lceil \mathsf{car} = 1 \rceil \oplus_{\frac{1}{3}} (\lceil \mathsf{car} = 2 \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{car} = 3 \rceil)) \\ &\implies \lceil \mathsf{pick} = \mathsf{car} \rceil \oplus_{\frac{1}{3}} (\lceil \mathsf{pick} \neq \mathsf{car} \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{pick} \neq \mathsf{car} \rceil) \\ &\implies \lceil \mathsf{pick} = \mathsf{car} \rceil \oplus_{\frac{1}{3}} \lceil \mathsf{pick} \neq \mathsf{car} \rceil \end{aligned}$$

So, the player wins with probability $\frac{1}{3}$ in the stick strategy. Now, for the *switch* strategy, we can compositionally reason by appending the **Switch** program to the end of the previous derivation and then continue using the derivation rules. We again must de-structure to analyze the if statement, this time using both Prob Split and also ND Split.

$$\langle \lceil \mathsf{true} \rceil \rangle$$
$$\textbf{Game} \fatsemi$$
$$\langle (\lceil \mathsf{car} = 1 \rceil \wedge (\lceil \mathsf{open} = 2 \rceil \,\&\, \lceil \mathsf{open} = 3 \rceil)) \oplus_{\frac{1}{3}} (\lceil \mathsf{car} = 2 \wedge \mathsf{open} = 3 \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{car} = 3 \wedge \mathsf{open} = 2) \rceil \rangle$$
$$\textbf{if } \mathsf{open} = 2 \textbf{ then } \mathsf{pick} := 3 \textbf{ else } \mathsf{pick} := 2$$
$$\langle (\lceil \mathsf{car} = 1 \rceil \wedge (\lceil \mathsf{pick} = 3 \rceil \,\&\, \lceil \mathsf{pick} = 2 \rceil)) \oplus_{\frac{1}{3}} (\lceil \mathsf{car} = 2 \wedge \mathsf{pick} = 2 \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{car} = 3 \wedge \mathsf{pick} = 3 \rceil) \rangle$$
$$\langle \lceil \mathsf{pick} \neq \mathsf{car} \rceil \oplus_{\frac{1}{3}} (\lceil \mathsf{pick} = \mathsf{car} \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{pick} = \mathsf{car} \rceil) \rangle$$
$$\langle \lceil \mathsf{pick} \neq \mathsf{car} \rceil \oplus_{\frac{1}{3}} \lceil \mathsf{pick} = \mathsf{car} \rceil \rangle$$

$$
\begin{array}{l}
\langle \lceil \mathsf{true} \rceil \rangle \\
\ \mathsf{car} \coloneqq 1 \oplus_{\frac{1}{3}} (\mathsf{car} \coloneqq 2 \oplus_{\frac{1}{2}} \mathsf{car} \coloneqq 3) \fatsemi \\
\langle \lceil \mathsf{car} = 1 \rceil \oplus_{\frac{1}{3}} (\lceil \mathsf{car} = 2 \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{car} = 3 \rceil) \rangle \\
\ \mathsf{pick} \coloneqq 1 \fatsemi \\
\langle \lceil \mathsf{pick} = 1 \rceil \wedge (\lceil \mathsf{car} = 1 \rceil \oplus_{\frac{1}{3}} (\lceil \mathsf{car} = 2 \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{car} = 3 \rceil)) \rangle \\
\ \textbf{if}\ \mathsf{car} = 1\ \textbf{then} \\
\ \ \langle \lceil \mathsf{car} = 1 \rceil \rangle \\
\ \ \ \mathsf{open} \leftarrow \{2, 3\} \\
\ \ \langle \lceil \mathsf{car} = 1 \rceil \wedge (\lceil \mathsf{open} = 2 \rceil \ \&\ \lceil \mathsf{open} = 3 \rceil) \rangle \\
\ \textbf{else if}\ \mathsf{car} = 2\ \textbf{then} \\
\ \ \langle \lceil \mathsf{car} = 2 \rceil \rangle \\
\ \ \ \mathsf{open} \coloneqq 3 \\
\ \ \langle \lceil \mathsf{car} = 2 \wedge \mathsf{open} = 3 \rceil \rangle \\
\ \textbf{else} \\
\ \ \langle \lceil \mathsf{car} = 3 \rceil \rangle \\
\ \ \ \mathsf{open} \coloneqq 2 \\
\ \ \langle \lceil \mathsf{car} = 3 \wedge \mathsf{open} = 2 \rceil \rangle \\
\left\langle \lceil \mathsf{pick} = 1 \rceil \wedge \left(\begin{array}{l} (\lceil \mathsf{car} = 1 \rceil \wedge (\lceil \mathsf{open} = 2 \rceil \ \&\ \lceil \mathsf{open} = 3 \rceil)) \\ \quad \oplus_{\frac{1}{3}} (\lceil \mathsf{car} = 2 \wedge \mathsf{open} = 3 \rceil \oplus_{\frac{1}{2}} \lceil \mathsf{car} = 3 \wedge \mathsf{open} = 2 \rceil) \end{array} \right) \right\rangle
\end{array}
$$

Fig. 5. Derivation for the **Game** program from Figure 4.

Note that the last two lines of the derivation are obtained by weakening the postcondition with the rule of Consequence. Just like in the previous case, we weaken the postcondition to only assert whether pick = car or pick ≠ car, and then collapse two outcomes using idempotence of $\oplus_{\frac{1}{2}}$. This time the player wins with probability $\frac{2}{3}$, meaning that switching doors is the better strategy.

6.2 The Adversarial von Neumann Trick

The von Neumann [1951] trick is a protocol for simulating a fair coin using a coin of unknown bias p. To do so, the coin is flipped twice. If the outcome is heads, tails—occurring with probability $p(1-p)$—then we consider the result to be heads. If the outcome is tails, heads—which also occurs with probability $p(1-p)$—then we consider to result to be tails. Otherwise, we try again.

In this case study, we work with an *adversarial* version of the von Neumann trick in which an adversary can alter the bias of the coin on each round, as long as the bias is between ε and $1-\varepsilon$ for some fixed $0 < \varepsilon \leq \frac{1}{2}$. We will show that just like in the original von Neumann trick, and somewhat surprisingly, the simulated coin is fair in the presence of an adversarial bias. To model this protocol, we let the set $[\varepsilon, 1-\varepsilon]_N$ be a finite subset of the interval of rational numbers $[\varepsilon, 1-\varepsilon] \subseteq \mathbb{Q}$, formally defined as $[\varepsilon, 1-\varepsilon]_N \triangleq \{\varepsilon + \frac{k(1-2\varepsilon)}{N} \mid k = 0 \ldots N\}$. The program is shown below.

$$
\textbf{AdvVonNeumann} \triangleq \left\{ \begin{array}{l} x \coloneqq \mathsf{false} \fatsemi y \coloneqq \mathsf{false} \fatsemi \\ \textbf{while}\ x = y\ \textbf{do} \\ \quad p \leftarrow [\varepsilon, 1-\varepsilon]_N \fatsemi \\ \quad x \coloneqq \mathsf{flip}(p) \fatsemi \\ \quad y \coloneqq \mathsf{flip}(p) \end{array} \right.
$$

So, the program will terminate once $x \neq y$, meaning that one heads and one tails were flipped. We wish to prove that this program almost surely terminates, and that $x = \mathsf{true}$ and $x = \mathsf{false}$ occur with equal probability, meaning that we have successfully modeled a fair coin. More formally, we

$$
\begin{array}{l}
\langle \lceil \mathsf{true} \rceil \rangle \\
\quad x \coloneqq \mathsf{false} \\
\langle \lceil \neg x \rceil \rangle \\
\quad y \coloneqq \mathsf{false} \\
\langle \lceil \neg x \wedge \neg y \rceil \rangle \implies \\
\langle \lceil x = y \rceil \rangle \\
\quad \textbf{while } x = y \textbf{ do} \\
\qquad p \leftarrow [\varepsilon, 1-\varepsilon]_N \fatsemi \\
\qquad x \coloneqq \mathsf{flip}(p) \fatsemi \\
\qquad y \coloneqq \mathsf{flip}(p) \\
\langle \lceil x = \mathsf{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \mathsf{false} \rceil \rangle
\end{array}
\qquad
\begin{array}{l}
\langle \lceil x = y \rceil \rangle \\
\quad p \leftarrow [\varepsilon, 1-\varepsilon]_N \fatsemi \\
\langle \&_{q \in [\varepsilon, 1-\varepsilon]_N} \lceil p = q \rceil \rangle \\
\quad x \coloneqq \mathsf{flip}(p) \fatsemi \\
\langle \&_{q \in [\varepsilon, 1-\varepsilon]_N} \lceil p = q \rceil \wedge (\lceil x = \mathsf{true} \rceil \oplus_q \lceil x = \mathsf{false} \rceil) \rangle \\
\quad y \coloneqq \mathsf{flip}(p) \\
\left\langle \&_{q \in [\varepsilon, 1-\varepsilon]_N} \begin{array}{l} (\lceil x = \mathsf{true} \rceil \wedge (\lceil x = y \rceil \oplus_q \lceil x \neq y \rceil)) \oplus_q \\ (\lceil x = \mathsf{false} \rceil \wedge (\lceil x \neq y \rceil \oplus_q \lceil x = y \rceil)) \end{array} \right\rangle \\
\langle \&_{q \in [\varepsilon, 1-\varepsilon]_N} \varphi_0 \oplus_{2q(1-q)} \varphi_1 \rangle \\
\langle \varphi_0 \oplus_{2\varepsilon(1-\varepsilon)} (\varphi_0 \,\&\, \varphi_1) \rangle
\end{array}
$$

Fig. 6. Derivation of the von Neumann trick program.

will prove that $\langle \lceil \mathsf{true} \rceil \rangle$ **AdvVonNeumann** $\langle \lceil x = \mathsf{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \mathsf{false} \rceil \rangle$. We will use the Bounded Variant rule to analyze the main loop, with the following variants.

$$\varphi_0 \triangleq (\lceil x = \mathsf{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \mathsf{false} \rceil) \wedge \lceil x \neq y \rceil \qquad \varphi_1 \triangleq \lceil x = y \rceil$$

The variant φ_1 with the higher rank simply states that $x = y$, meaning that the loop will continue to execute. The lower-ranked variant φ_0 states both that $x \neq y$—the loop will terminate—and that $x = \mathsf{true}$ and $x = \mathsf{false}$ both occur with probability $\frac{1}{2}$. This is an example of a variant with multiple outcomes that is not supported in pre-expectation reasoning, as mentioned in Section 5.3.

Each execution of the loop body will reduce the rank of the variant from 1 to 0 with probability $2p(1-p)$, where p is chosen by the adversary. The *worst case* is that the adversary chooses either $p = \varepsilon$ or $p = 1 - \varepsilon$, in which case the probability of terminating the loop is $2\varepsilon(1-\varepsilon)$. Given that there are only two variants, the Bounded Variant rule simplifies to:

$$\frac{\langle \varphi_1 \rangle \; C \; \langle \varphi_0 \oplus_{2\varepsilon(1-\varepsilon)} (\varphi_0 \,\&\, \varphi_1) \rangle}{\langle \varphi_1 \rangle \; \textbf{while } e \textbf{ do } C \; \langle \varphi_0 \rangle}$$

The main derivation is shown on the left of Figure 6, and the premise of the Bounded Variant rule is on the right. After the two flips, all four probabilistic outcomes are enumerated. This is simplified using the associativity and commutativity rules from Section 4.1 to conclude that $\varphi_0 \oplus_{2q(1-q)} \varphi_1$ for each q. As mentioned before, since we know that $2q(1-q) \geq 2\varepsilon(1-\varepsilon)$, we can weaken this to be $\varphi_0 \oplus_{2\varepsilon(1-\varepsilon)} (\varphi_0 \,\&\, \varphi_1)$. Now, since the assertion no longer depends on q, we use idempotence to remove the outer &. In the end, we get the postcondition $\lceil x = \mathsf{true} \rceil \oplus_{\frac{1}{2}} \lceil x = \mathsf{false} \rceil$, as desired.

6.3 Probabilistic SAT Solving by Partial Rejection Sampling

Rejection sampling is a standard technique for generating random samples from certain distributions. A basic version of rejection sampling can be used when a program has a way to generate random samples uniformly from a set X, and needs to generate uniform random samples from a set S, where $S \subseteq X$. To do so, a simple rejection sampling procedure will draw a sample x from X and then check whether $x \in S$. If $x \in S$, the rejection sampler is said to *accept* x, and returns it. However, if $x \notin S$, the sampler is said to *reject* x, and repeats the process with a fresh sample from X.

In some situations, the set X is a product of sets $X_1 \times \cdots \times X_n$, and a sample $x = (x_1, \ldots, x_n)$ from X is generated by independently drawing samples $x_1, \ldots, x_n$, where each $x_i \in X_i$. In this case, when x is rejected, rather than redrawing *all* of the x_i to form a new sample from X, one might consider instead trying to *partially* resample the components of x. In particular if x is *close* to being

```
Solve ≜
  b := Eval ⨟
  while ¬b do
    SelectClause ⨟
    SampleClause ⨟
    b := Eval

SampleClause ≜
  x[cv[s][1]] := flip (1/2) ⨟
  x[cv[s][2]] := flip (1/2) ⨟
  x[cv[s][3]] := flip (1/2)
```

```
SelectClause ≜
  s := −1 ⨟
  i := 1 ⨟
  while i ≤ M do
    if ¬EvalClause(i) then
      if s = −1 then
        s := i
      else
        skip & s := i ⨟
    i := i + 1 ⨟
```

```
EvalClause(i) ≜
  (cs[i][1] ⊙ x[cv[i][1]]) ∨
  (cs[i][2] ⊙ x[cv[i][2]]) ∨
  (cs[i][3] ⊙ x[cv[i][3]])

Eval ≜
  EvalClause(1) ∧
  EvalClause(2) ∧
        ⋮
  EvalClause(M)
```

Fig. 7. SAT solving via rejection sampling, split into subroutines.

in S, then one might try to only redraw some subset of components $x_{j_1}, \ldots, x_{j_k}$, and re-use the other components of x to form a new sample x' to test for membership in S.

In general, partial resampling can result in drawing samples that are not *uniformly* distributed over the set S. However, Guo et al. [2019] observed that under certain conditions on the set S and X, a partial rejection sampling procedure *does* generate uniform samples from S. In particular, when the x_i are boolean variables, and the test for $(x_1, \ldots, x_n) \in S$ can be encoded as a boolean formula φ over these variables, then it suffices for φ to be a so-called *extremal* formula. Guo et al. [2019] showed that many algorithms for sampling combinatorial structures can be formulated in terms of sampling a satisfying assignment to an extremal formula. In this example, we consider a partial rejection sampler for generating a random satisfying assignment for a formula in 3-CNF form. We will prove that the sampler almost surely terminates if the formula has a satisfying assignment[1].

Figure 7 shows the solver program, **Solve**, broken up into subroutines[2]. The clauses are encoded using two 2-dimensional lists, cv and cs, each of size $M \times 3$, where M is the number of clauses. See Zilberstein et al. [2024a, §E.1] for the semantics of list operations. The entry $\mathrm{cv}[i][j]$ gives the variable of the j^{th} variable in clause i, and $\mathrm{cs}[i][j]$ is 0 if this variable occurs in negated form, and is 1 otherwise. The $\odot$ operation is *xnor*, so $1 \odot 0 = 0 \odot 1 = 0$ and $0 \odot 0 = 1 \odot 1 = 1$. The program stores its current truth-value assignment for each variable in the list x.

Each iteration of the loop in **Solve** starts by nondeterministically selecting an unsatisfied clause s to resample via the **SelectClause** subroutine. To do so, it iterates over the clauses, checking if each one is satisfied using **EvalClause**. When an unsatisfied clause is found, s is nondeterministically either updated to i or left as is (unless $s = -1$, in which case s is updated to i, to ensure that some unsatisfied clause is picked). Nondeterminism allows us to under-specify *how* the sampler selects a clause to resample, which in practice might be based on various heuristics. By proving almost-sure termination for this non-deterministic version, we establish almost-sure termination no matter which heuristics are used, including randomized ones. Presuming that the formula is not yet satisfied (**Eval** = false), the **SelectClause** routine selects an s such that $1 \le s \le M$ and **EvalClause**(s) = false, which is captured by the following specification and proven in Zilberstein et al. [2024a, §E].

$$\langle\lceil \textbf{Eval} = \text{false}\rceil\rangle\ \textbf{SelectClause}\ \langle\lceil 1 \le s \le M \wedge \textbf{EvalClause}(s) = \text{false}\rceil\rangle$$

[1]Since this termination property holds even if the formula does not satisfy the extremal property, we will not formally define the extremal property or assume it as a precondition.

[2]Note that our language does not include subroutines, but these routines are interpreted as macros and are inlined into the main program. We only separate them for readability.

$$
\begin{array}{l}
\langle\lceil \mathsf{true} \rceil\rangle \\
\quad b \coloneqq \mathbf{Eval} \fatsemi \\
\langle\lceil b = \mathbf{Eval} \rceil\rangle \\
\quad \mathbf{while}\ \neg b\ \mathbf{do} \\
\qquad \langle\lceil b = \mathbf{Eval} \wedge \neg b \wedge [\neg \mathbf{Eval}] \cdot \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) = k \rceil\rangle \\
\qquad \langle\lceil \mathbf{Eval} = \mathsf{false} \wedge \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) = k \rceil\rangle \\
\qquad\quad \mathbf{SelectClause} \fatsemi \\
\qquad \langle\lceil 0 \le s < M \wedge \mathbf{EvalClause}(s) = \mathsf{false} \wedge \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) = k \rceil\rangle \\
\qquad\quad \mathbf{SampleClause} \fatsemi \\
\qquad \langle\lceil \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) < k \rceil \oplus_{\frac{1}{8}} \lceil \mathsf{true} \rceil\rangle \\
\qquad\quad b \coloneqq \mathbf{Eval} \\
\qquad \langle\lceil b = \mathbf{Eval} \wedge \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) < k \rceil \oplus_{\frac{1}{8}} \lceil b = \mathbf{Eval} \rceil\rangle \\
\qquad \langle\lceil b = \mathbf{Eval} \wedge [\neg \mathbf{Eval}] \cdot \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) < k \rceil \oplus_{\frac{1}{8}} \lceil b = \mathbf{Eval} \rceil\rangle \\
\langle\lceil \mathbf{Eval} = \mathsf{true} \rceil\rangle
\end{array}
$$

Fig. 8. Derivation of the **Solve** program, where x^* is a known satisfying assignment.

Next, the three variables in the selected clause are resampled. In order to prove that the program almost surely terminates, we need to show that the resampling operation brings the process closer to termination with nonzero probability. To do this, we measure how close the candidate solution is to some satisfying assignment x^* (recall we assumed that at least one such satisfying assignment exists). Closeness is measured via the Hamming distance, computed as follows, where the Iverson brackets $[e]$ evaluates to 1 if e is true and 0 is e is false, and N is the number of variables.

$$\mathsf{dist}(x, y) \triangleq \textstyle\sum_{i=1}^{N} \big[x[i] \neq y[i]\big]$$

Now, we can give a specification for **SampleClause** in terms of the Hamming distance. That is, if $\mathsf{dist}(\mathsf{x}, \mathsf{x}^*)$ is initially k and clause s is not satisfied, then resampling s will strictly reduce the Hamming distance with probability at least $\frac{1}{8}$. The reason for this is that before resampling, x and x^* must disagree on at least one of the variables in clause s, since clause s is not satisfied by x. After resampling, there is at least a $\frac{1}{8}$ probability that all 3 resampled variables agree with x^*, in which case the Hamming distance is reduced by at least 1. The full proof is in Zilberstein et al. [2024a, §E].

$$\langle\lceil \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) = k \wedge \mathbf{EvalClause}(s) = \mathsf{false} \rceil\rangle\ \mathbf{SampleClause}\ \langle\lceil \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) < k \rceil \oplus_{\frac{1}{8}} \lceil \mathsf{true} \rceil\rangle$$

Using these specifications, we now prove that **Solve** almost surely terminates. The derivation is shown in Figure 8. We instantiate Bounded Rank to analyze the loop with the following parameters:

$$P \triangleq b = \mathbf{Eval} \qquad e_{\mathsf{rank}} \triangleq [\neg \mathbf{Eval}] \cdot \mathsf{dist}(\mathsf{x}, \mathsf{x}^*) \qquad p \triangleq \frac{1}{8}$$

The invariant P simply states that b indicates whether the current assignment of variables satisfies the formula. The ranking function is equal to the Hamming distance between x and the sample solution x^* if the the formula is not yet satisfied, otherwise it is zero, which accounts for the fact that the program may find a solution other than x^*. We also remark that e_{rank} is bounded between 1 and N (where N is the total number of variables) as long as the formula is not yet satisfied. As we saw in the specification for **SampleClause**, the probability of reducing the rank is $\frac{1}{8}$.

Entering the loop, we see that **Eval** must be false, so the rank is just $\mathsf{dist}(\mathsf{x}, \mathsf{x}^*)$. Applying the specifications for the two subroutines, we prove that the Hamming distance strictly decreases with probability at least p. The assignment to b then reestablishes the invariant. When the Hamming distance has decreased, we also have that e_{rank} decreased, as multiplying by $[\neg \mathbf{Eval}]$ can only make

the term smaller. Upon exiting the loop, we have that b = true and hence the final postcondition **Eval** = true, meaning that the formula is satisfied and the program almost surely terminates.

Aguirre et al. [2024] prove termination of a similar randomized SAT solving technique using a separation logic for reasoning about upper bounds on probabilities of non-termination. Because the language they consider does not have non-determinism, they fix a particular strategy for selecting clauses to resample, whereas the use of nondeterministic choice in the proof above implies termination for any strategy that selects an unsatisfied clause. Their proof essentially shows that for any $\varepsilon > 0$, after some number of iterations, the Hamming distance will decrease with probability at least $1 - \varepsilon$. The Bounded Rank rule effectively encapsulates this kind of reasoning in our proof.

7 Related Work

Program Logics. Demonic Outcome Logic takes inspiration from program logics for reasoning about purely probabilistic programs, such as Probabilistic Hoare Logic [Corin and den Hartog 2006; den Hartog 1999, 2002], VPHL [Rand and Zdancewic 2015], Ellora [Barthe et al. 2018], and Outcome Logic [Zilberstein 2024; Zilberstein et al. 2023, 2024b]. Those logics provide means to prove properties about the distributions of outcomes in probabilistic programs, to which we added the ability to also reason about demonic nondeterminism.

Although this paper introduces the first logic for reasoning about the *outcomes* in demonic probabilistic programs, there is some prior work on other styles of analysis. Building on the work of Varacca [2002, 2003], Polaris is a relational separation logic for reasoning about concurrent probabilistic programs [Tassarotti 2018; Tassarotti and Harper 2019]. Specifications take the form of refinements, where a complex program is shown to behave equivalently to an idealized version. Probabilistic analysis can then be done on the idealized program to determine its expected behavior, but it is external to the program logic. Polaris also does not support unbounded looping, and therefore it cannot be used to analyze our last two case studies.

Weakest Pre-Expectations. Weakest pre-expectation (wp) transformers are calculi for reasoning about probabilistic programs in terms of expected values [Morgan et al. 1996a]. They were inspired by propositional weakest precondition calculi [Dijkstra 1975, 1976], Probabilistic Propositional Dynamic Logic [Kozen 1983], and probabilistic predicate transformers [Jones 1990]. Refer to Kaminski [2019] for a thorough overview of this technique.

From the start, wp supported nondeterminism; in fact, wp emerged from a line of work on semantics for randomized nondeterministic programs [He et al. 1997; McIver and Morgan 2001; Morgan et al. 1996b]. Nondeterminism is handled by lower-bounding expectations, corresponding to larger expected values being *better*. An *angelic* variant can alternatively be used for upper bounds.

Work on wp has intersected with termination analysis for probabilistic programs. Some of this work uses martingales [Chakarov and Sankaranarayanan 2013] to show that programs terminate with finite expected running time. More sophisticated techniques exist for almost sure termination too [Kaminski 2019; McIver and Morgan 2005; McIver et al. 2018].

As noted by Kaminski [2019, §2.3.3], the choice of either upper or lower bounding the expected values is *"extremal"*—it forces a view where expectations must be either maximized or minimized, as opposed to our approach where multiple outcomes can be represented in one specification. However, reasoning about outcomes and expectations are not mutually exclusive; Barthe et al. [2018, Theorem 1] showed how to embed a wp calculus in a probabilistic program logic. A similar construction is possible in Demonic Outcome Logic.

Powerdomains for Probabilistic Nondeterminism. Powerdomains are a well-studied domain-theoretic tool for reasoning about looping nondeterministic programs, providing a means for defining a continuous domain in which loops can be interpreted as fixed points. This revolves around defining

appropriate orders over sets of states to show that iterated actions eventually converge. Given a partially ordered domain of program states $\langle \Sigma, \leq \rangle$, there are three typical choices for orders over sets of states, known as the Hoare, Smyth [1978], and Egli-Milner orders, defined below:

$$\begin{array}{lll} S \sqsubseteq_{\mathsf{H}} T & \text{iff} & \forall \sigma \in S.\ \exists \tau \in T.\ \ \sigma \leq \tau \\ S \sqsubseteq_{\mathsf{S}} T & \text{iff} & \forall \tau \in T.\ \exists \sigma \in S.\ \ \sigma \leq \tau \\ S \sqsubseteq_{\mathsf{EM}} T & \text{iff} & S \sqsubseteq_{\mathsf{H}} T \ \text{ and } \ S \sqsubseteq_{\mathsf{S}} T \end{array}$$

In general, none of these relations are antisymmetric, making them *preorders*, whereas domain theoretic tools for finding fixed points operate on *partial orders*. So the sets representing the program semantics must be closed in order to obtain a proper domain. This closure operation loses precision of the semantics, incorporating additional possibilities which are not always intuitive.

The Hoare order requires a down-closure, essentially meaning that nontermination may always be an option. This makes it a good choice for partial correctness, where we only wish to determine what happens *if* the program terminates, as in Hoare Logic. The Smyth [1978] order, which we use in this paper, requires an upwards closure, so that nontermination becomes erratic behavior. This makes it a good choice for total correctness [Manna and Pnueli 1974], which is concerned only with terminating programs where erratic behavior does not arise.

In the Egli-Milner case—and the associated Plotkin Powerdomain [1976]—the more precise, but also less intuitive Egli-Milner closure is used. McIver and Morgan [2001] created a denotational model where fixed points are taken with respect to the Egli-Milner order rather than the Smyth [1978] one. As such, they require the domain of computation to be Egli-Milner closed, which means that $S = \uparrow S \cap \downarrow S$. Unlike up-closedness (required for the Smyth approach), which is preserved by all the operations in Figure 1, the semantics of McIver and Morgan [2001] must take the Egli-Milner closure after nondeterministic and probabilistic choice and after sequential composition, making the model more complex and adding outcomes that do not have an obvious operational meaning. Refer to Keimel and Plotkin [2017]; Tix et al. [2009] for a more complete exploration of that approach.

Let us examine the semantics of a coin flip in order to demonstrate why the Smyth order is preferable to Hoare. The variable x is assigned the values true or false each with probability $\frac{1}{2}$. So, the result of running the program is a singleton set containing the aforementioned distribution.

$$[\![x \coloneqq \mathsf{flip}\left(\tfrac{1}{2}\right)]\!]\,(\sigma) = \left\{ \begin{array}{lcl} \sigma[x \coloneqq \text{true}\,] & \mapsto & \frac{1}{2} \\ \sigma[x \coloneqq \text{false}] & \mapsto & \frac{1}{2} \end{array} \right\}$$

If we were to use the Hoare powerdomain, then we would need to down-close this set, adding all smaller distributions too. This not only means that nontermination is possible, but we would not even be able to determine that $x = \text{true}$ and $x = \text{false}$ occur with equal probability.

$$[\![x \coloneqq \mathsf{flip}\left(\tfrac{1}{2}\right)]\!]\,(\sigma) = \left\{ \begin{array}{lcl} \sigma[x \coloneqq \text{true}\,] & \mapsto & p \\ \sigma[x \coloneqq \text{false}] & \mapsto & q \\ \bot & \mapsto & 1-p-q \end{array} \;\middle|\; p \leq \frac{1}{2},\ q \leq \frac{1}{2} \right\}$$

This was the approach taken by Varacca [2002], and the loss of precision is reflected in the adequacy theorems of that work. In particular, Varacca's Proposition 6.10 shows that the denotational model includes outcomes that may not be possible according to the associated operational model. By contrast, the up-closure—required by the Smyth order—adds nothing for this program; the semantics is already a full distribution and therefore there are no distributions larger than it. We can therefore conclude that the two outcomes occur with probability exactly $\frac{1}{2}$, as desired.

This example demonstrates that the notion of partial correctness (as embodied by the Hoare order) does not make much sense in probabilistic settings, since it translates to uncertainty about the minimum probability of an event. Total correctness, on the other hand, does make sense, and corresponds to the notion of *almost sure termination*, which is a property of great interest in probabilistic program analysis [Chakarov and Sankaranarayanan 2013; McIver et al. 2018].

The problem with the Smyth order is that a semantics based on it is not Scott [1972] continuous in the presence of unbounded nondeterminism [Apt and Plotkin 1986; Søndergaard and Sestoft 1992]. This is the reason why He et al. [1997] instead use the Knaster-Tarski theorem to guarantee the fixed point existence via transfinite iteration, which only requires monotonicity and not Scott continuity. The main shortcoming of He et al.'s approach is that it did not guarantee non-emptiness of the set of result distributions, meaning that some programs may have vacuous semantics.

To address this, Morgan et al. [1996a] added the additional requirement that domain only include topologically closed sets (Morgan et al. called this property *Cauchy closure*). As we mentioned in Section 3.3 and proved in Zilberstein et al. [2024a, §B.3], closure ensures that no programs are modeled as empty sets, but it also prevents commands from exhibiting unbounded nondeterminism. For example, it is not possible to represent a program $x := \bigstar$, which nondeterministically selects a value for x from the natural numbers—the set $\mathbb{N}$ is not closed since it does not contain a limit point.

McIver and Morgan [2005] suggested that topological closure opens up the possibility of Scott continuity. In addition, there has been work to combine classical powerdomains for nondeterminism [Plotkin 1976; Smyth 1978] with the probabilistic powerdomain of Jones [1990]; Jones and Plotkin [1989]. This was first pursued by Tix [1999], and was later refined in Keimel and Plotkin [2017]; Tix [2000]; Tix et al. [2009]. They obtain a Scott continuous composition operation (which they call $\widehat{f}$) via a universal property, as opposed to the direct construction of Jacobs [2008] that we use.

Monads for Probabilistic Nondeterminism. Varacca [2002, 2003] introduced powersets of *indexed valuations*. An indexed valuation behaves similarly to a distribution, but the idempotence property is removed, so that $X \oplus_p X \neq X$. As shown by Varacca and Winskel [2006], a powerset of indexed valuations has a Beck [1969] distributive law, and is therefore a monad. However, indexed valuations are difficult to work with since equivalence is taken modulo renaming of the indices.

Varacca [2002, Theorem 6.5] proved that denotational models based on indexed valuations are equivalent to operational models in which a *deterministic* scheduler resolves the nondeterminism. Given our goals of modeling *adversarial* nondeterminism, we opted to use convex sets, which model a more powerful probabilistic scheduler, giving us robust guarantees in a stronger threat model.

An alternative approach is to flip the order of composition and work instead with distributions of nondeterministic outcomes. While the distribution monad does not compose with powerset, it does compose with multiset, as shown by Jacobs [2021] and further explored by Kozen and Silva [2024]. The barrier to this approach is that the multisets must be finite, but it is easy to construct programs that reach infinitely many nondeterministic outcomes via while loops. So this model cannot be used to represent arbitrary programs from the language in Section 3. The use of multiset instead of powerset is again an instance of removing an idempotence law, this time for nondeterminism: $X \,\&\, X \neq X$. Indeed, idempotence is the key reason why no distributive law exists in both cases [Parlant 2020; Zwart 2020; Zwart and Marsden 2019].

Other Semantic Approaches. Segala [1995] created a model in which a tree of alternating probabilistic and nondeterministic choices is collapsed into a set of distributions collected from all combinations of nondeterministic choices. However, this model does not lead to a compositional semantics. Additional operational models of probabilistic nondeterminism have been studied through the lens of process algebras [den Hartog 1998, 2002; den Hartog and de Vink 1999; Mislove et al. 2004]. In addition, coalgebraic methods have been used to define trace semantics and establish bisimilarity of randomized nondeterministic automata [Bonchi et al. 2021a, 2019, 2021b, 2022; Jacobs 2008].

Aguirre and Birkedal [2023] note the difficulties of building denotational models that combine probabilistic and nondeterministic choice with other challenging semantic features. Instead, they start with an operational semantics for probabilistic and nondeterministic choice and then construct

a step-indexed logical relations model for a typed, higher-order language with polymorphism and recursive types. Using this logical relations model, they derive an equational theory for contextual equivalence and show that it validates many of the equations found in denotational models.

8 Conclusion

This paper introduced *Demonic Outcome Logic*, a logic for outcome based reasoning about programs that are both randomized and nondeterministic, a combination that presents many challenges for program semantics and analysis. The logic includes several novel features, such as equational laws for manipulating pre- and postconditions and rules for loops that both establish termination and quantify the distribution of final outcomes from a single premise. We build on a large body of work on semantics for probabilistic nondeterminism [He et al. 1997; Jacobs 2008; Morgan et al. 1996a,b; Tix et al. 2009; Varacca 2002], and also draw inspiration from Outcome Logic [Zilberstein 2024; Zilberstein et al. 2023, 2024b] and weakest pre-expectation calculi [Kaminski 2019; Morgan et al. 1996a; Zhang et al. 2024]. The resulting logic contains rules that enable effective reasoning about distributions of outcomes in randomized nondeterministic programs, as illustrated through the three presented case studies. The simplicity of the rules is enabled by a carefully chosen denotational semantics that allows us to hide the complex algebraic properties of the domain in the proof of their soundness. Compared to weakest pre-expectation reasoning, the propositional approach afforded by Demonic Outcome Logic enables reasoning about multiple outcomes in tandem, leading to more expressive specifications, and the loop rules rely on fewer, simpler premises.

Moving forward, we want to go beyond standard nondeterminism and extend the logic for reasoning about probabilistic fine-grain concurrency with shared memory. This will require fundamental changes to the denotational semantics and inference rules, although prior work on Concurrent Separation Logic [O'Hearn 2004], Outcome Separation Logic [Zilberstein et al. 2024b], and Concurrent Kleene Algebra [Hoare et al. 2011] will provide a good source of inspiration. Using the resulting logic, we will verify concurrent algorithms such as distributed cryptographic protocols, for which state of the art techniques use limited models of concurrency and operate by establishing observational equivalence and then separately proving properties of an idealized program [Gancher et al. 2023]. By contrast, we plan to develop a logic based on a fine-grain concurrency model, which can prove direct specifications involving probabilistic outcomes. We also plan to explore a mechanized implementation of the logic, building on existing frameworks for (concurrent) separation logic such as Iris [Jung et al. 2015].

Acknowledgments

This work was partially supported by ERC grant Autoprobe (no. 101002697), ARIA's Safeguarded AI programme, and NSF grant CCF-2008083. Some work on this paper was completed during a workshop at the Bellairs Research Institute of McGill University; we thank Prakash Panangaden for the invitation and the institute and their staff for providing a wonderful research environment.

References

Alejandro Aguirre and Lars Birkedal. 2023. Step-Indexed Logical Relations for Countable Nondeterminism and Probabilistic Choice. *Proc. ACM Program. Lang.* 7, POPL (2023), 33–60. https://doi.org/10.1145/3571195

Alejandro Aguirre, Philipp G. Haselwarter, Markus de Medeiros, Kwing Hei Li, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2024. Error Credits: Resourceful Reasoning about Error Bounds for Higher-Order Probabilistic Programs. *Proc. ACM Program. Lang.* 8, ICFP, Article 246 (Aug. 2024), 33 pages. https://doi.org/10.1145/3674635

Krzysztof Apt and Gordon Plotkin. 1986. Countable nondeterminism and random assignment. *J. ACM* 33, 4 (aug 1986), 724–767. https://doi.org/10.1145/6490.6494

Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. An Assertion-Based Program Logic for Probabilistic Programs. In *Programming Languages and Systems*. Springer International Publishing, Cham, 117–144. https://doi.org/10.1007/978-3-319-89884-1_5

Jon Beck. 1969. Distributive laws. In *Seminar on Triples and Categorical Homology Theory*, B. Eckmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 119–140. https://doi.org/10.1007/BFb0083084

Filippo Bonchi, Alexandra Silva, and Ana Sokolova. 2021a. Distribution Bisimilarity via the Power of Convex Algebras. *Logical Methods in Computer Science* Volume 17, Issue 3 (July 2021). https://doi.org/10.46298/lmcs-17(3:10)2021

Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. 2019. The Theory of Traces for Systems with Nondeterminism and Probability. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–14. https://doi.org/10.1109/lics.2019.8785673

Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. 2021b. Presenting Convex Sets of Probability Distributions by Convex Semilattices and Unique Bases. In *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 211)*, Fabio Gadducci and Alexandra Silva (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 11:1–11:18. https://doi.org/10.4230/LIPIcs.CALCO.2021.11

Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. 2022. The Theory of Traces for Systems with Nondeterminism, Probability, and Termination. *Logical Methods in Computer Science* Volume 18, Issue 2 (June 2022). https://doi.org/10.46298/lmcs-18(2:21)2022

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

Ricardo Corin and Jerry den Hartog. 2006. A Probabilistic Hoare-style Logic for Game-Based Cryptographic Proofs. In *Automata, Languages and Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 252–263.

Thibault Dardinier and Peter Müller. 2024. Hyper Hoare Logic: (Dis-)Proving Program Hyperproperties. *Proc. ACM Program. Lang.* 8, PLDI, Article 207 (jun 2024), 25 pages. https://doi.org/10.1145/3656437

Jerry den Hartog. 1998. *Comparative semantics for a process language with probabilistic choice and non-determinism*. Vrije Universiteit, Netherlands. Imported from DIES.

Jerry den Hartog. 1999. Verifying Probabilistic Programs Using a Hoare like Logic. In *Advances in Computing Science — ASIAN'99*, P. S. Thiagarajan and Roland Yap (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 113–125.

Jerry den Hartog. 2002. *Probabilistic Extensions of Semantical Models*. Ph.D. Dissertation. Vrije Universiteit Amsterdam. https://core.ac.uk/reader/15452110

Jerry den Hartog and Erik de Vink. 1999. Mixing Up Nondeterminism and Probability: a preliminary report. *Electronic Notes in Theoretical Computer Science* 22 (1999), 88–110. https://doi.org/10.1016/S1571-0661(05)82521-6 PROBMIV'98, First International Workshop on Probabilistic Methods in Verification.

Edsger W. Dijkstra. 1975. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Commun. ACM* 18, 8 (Aug 1975), 453–457. https://doi.org/10.1145/360933.360975

Edsger W. Dijkstra. 1976. *A Discipline of Programming*. Prentice-Hall. I–XVII, 1–217 pages.

Joshua Gancher, Kristina Sojakova, Xiong Fan, Elaine Shi, and Greg Morrisett. 2023. A Core Calculus for Equational Proofs of Cryptographic Protocols. *Proc. ACM Program. Lang.* 7, POPL, Article 30 (jan 2023), 27 pages. https://doi.org/10.1145/3571223

Heng Guo, Mark Jerrum, and Jingcheng Liu. 2019. Uniform Sampling Through the Lovász Local Lemma. *J. Acm* 66, 3, Article 18 (apr 2019), 31 pages. https://doi.org/10.1145/3310131

Jifeng He, Karen Seidel, and Annabelle McIver. 1997. Probabilistic models for the guarded command language. *Science of Computer Programming* 28, 2 (1997), 171–192. https://doi.org/10.1016/S0167-6423(96)00019-6 Formal Specifications: Foundations, Methods, Tools and Applications.

Charles Antony Richard Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580. https://doi.org/10.1145/363235.363259

Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. 2011. Concurrent Kleene Algebra and its Foundations. *J. Log. Algebraic Methods Program.* 80, 6 (2011), 266–296. https://doi.org/10.1016/J.JLAP.2011.04.005

Bart Jacobs. 2008. Coalgebraic Trace Semantics for Combined Possibilitistic and Probabilistic Systems. *Electronic Notes in Theoretical Computer Science* 203, 5 (2008), 131–152. https://doi.org/10.1016/j.entcs.2008.05.023 Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science (CMCS 2008).

Bart Jacobs. 2021. From Multisets over Distributions to Distributions over Multisets. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science* (Rome, Italy) *(LICS '21)*. Association for Computing Machinery, New York, NY, USA, Article 39, 13 pages. https://doi.org/10.1109/LICS52264.2021.9470678

Claire Jones. 1990. *Probabilistic Non-determinism*. Ph.D. Dissertation. University of Edinburgh. http://hdl.handle.net/1842/413

Claire Jones and Gordon Plotkin. 1989. A Probabilistic Powerdomain of Evaluations. In *Fourth Annual Symposium on Logic in Computer Science*. 186–195. https://doi.org/10.1109/lics.1989.39173

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Mumbai, India) *(POPL '15)*. Association for Computing Machinery, New York, NY, USA, 637–650. https://doi.org/10.1145/2676726.2676980

Benjamin Lucien Kaminski. 2019. *Advanced weakest precondition calculi for probabilistic programs*. Dissertation. RWTH Aachen University, Aachen. https://doi.org/10.18154/RWTH-2019-01829

Klaus Keimel and Gordon Plotkin. 2017. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science* Volume 13, Issue 1 (Jan. 2017). https://doi.org/10.23638/LMCS-13(1:2)2017

Dexter Kozen. 1983. A Probabilistic PDL. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC '83)*. Association for Computing Machinery, New York, NY, USA, 291–297. https://doi.org/10.1145/800061.808758

Dexter Kozen. 1997. Kleene Algebra with Tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (May 1997), 427–443. https://doi.org/10.1145/256167.256195

Dexter Kozen and Alexandra Silva. 2024. *Multisets and Distributions*. Springer Nature Switzerland, Cham, 168–187. https://doi.org/10.1007/978-3-031-61716-4_11

Zohar Manna and Amir Pnueli. 1974. Axiomatic Approach to Total Correctness of Programs. *Acta Inf.* 3, 3 (sep 1974), 243–263. https://doi.org/10.1007/BF00288637

Annabelle McIver and Carroll Morgan. 2001. Partial correctness for probabilistic demonic programs. *Theoretical Computer Science* 266, 1 (2001), 513–541. https://doi.org/10.1016/S0304-3975(00)00208-5

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer. https://doi.org/10.1007/b138392

Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2018. A New Proof Rule for Almost-Sure Termination. *Proc. ACM Program. Lang.* 2, POPL, Article 33 (Jan 2018), 28 pages. https://doi.org/10.1145/3158121

Matteo Mio and Valeria Vignudelli. 2020. Monads and Quantitative Equational Theories for Nondeterminism and Probability. In *31st International Conference on Concurrency Theory (CONCUR 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 171)*, Igor Konnov and Laura Kovács (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 28:1–28:18. https://doi.org/10.4230/LIPIcs.CONCUR.2020.28

Michael Mislove. 2000. Nondeterminism and Probabilistic Choice: Obeying the Laws. In *CONCUR 2000 — Concurrency Theory*, Catuscia Palamidessi (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 350–365. https://doi.org/10.1007/3-540-44618-4_26

Michael Mislove, Joël Ouaknine, and James Worrell. 2004. Axioms for Probability and Nondeterminism. *Electronic Notes in Theoretical Computer Science* 96 (2004), 7–28. https://doi.org/10.1016/j.entcs.2004.04.019 Proceedings of the 10th International Workshop on Expressiveness in Concurrency.

Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996a. Probabilistic Predicate Transformers. *ACM Trans. Program. Lang. Syst.* 18, 3 (may 1996), 325–353. https://doi.org/10.1145/229542.229547

Carroll Morgan, Annabelle McIver, Karen Seidel, and J. W. Sanders. 1996b. Refinement-oriented probability for CSP. *Form. Asp. Comput.* 8, 6 (nov 1996), 617–647. https://doi.org/10.1007/bf01213492

Peter W. O'Hearn. 2004. Resources, Concurrency and Local Reasoning. In *CONCUR 2004 - Concurrency Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 49–67. https://doi.org/10.1016/j.tcs.2006.12.035

Louis Parlant. 2020. *Monad Composition via Preservation of Algebras*. Ph.D. Dissertation. University College London. https://discovery.ucl.ac.uk/id/eprint/10112228/

Gordon Plotkin. 1976. A Powerdomain Construction. *SIAM J. Comput.* 5, 3 (1976), 452–487. https://doi.org/10.1137/0205035 arXiv:https://doi.org/10.1137/0205035

Robert Rand and Steve Zdancewic. 2015. VPHL: A Verified Partial-Correctness Logic for Probabilistic Programs. In *Electronic Notes in Theoretical Computer Science*, Vol. 319. 351–367. https://doi.org/10.1016/j.entcs.2015.12.021 The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

Dana Scott. 1972. Continuous lattices. In *Toposes, Algebraic Geometry and Logic*, F. W. Lawvere (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 97–136. https://doi.org/10.1007/BFb0073967

Roberto Segala. 1995. *Modeling and verification of randomized distributed real-time systems*. Ph.D. Dissertation. USA. https://groups.csail.mit.edu/tds/papers/Segala/phd1.pdf

Roberto Segala and Nancy Lynch. 1994. Probabilistic simulations for probabilistic processes. In *CONCUR '94: Concurrency Theory*, Bengt Jonsson and Joachim Parrow (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 481–496. https://doi.org/10.1007/BFb0015027

Michael Smyth. 1978. Power domains. *J. Comput. System Sci.* 16, 1 (1978), 23–36. https://doi.org/10.1016/0022-0000(78)90048-X

Harald Søndergaard and Peter Sestoft. 1992. Non-determinism in Functional Languages. *Comput. J.* 35, 5 (10 1992), 514–523. https://doi.org/10.1093/comjnl/35.5.514 arXiv:https://academic.oup.com/comjnl/article-pdf/35/5/514/1125580/35-5-514.pdf

Joseph Tassarotti. 2018. *Verifying Concurrent Randomized Algorithms*. Ph.D. Dissertation. Carnegie Mellon University. https://csd.cmu.edu/academics/doctoral/degrees-conferred/joseph-tassarotti

Joseph Tassarotti and Robert Harper. 2019. A Separation Logic for Concurrent Randomized Programs. *Proc. ACM Program. Lang.* 3, POPL, Article 64 (Jan 2019), 30 pages. https://doi.org/10.1145/3290377

Regina Tix. 1999. *Continuous D-cones: convexity and powerdomain constructions*. Ph.D. Dissertation. Darmstadt University of Technology, Germany. https://d-nb.info/957239157

Regina Tix. 2000. Convex Power Constructions for Continuous D-Cones. *Electronic Notes in Theoretical Computer Science* 35 (2000), 206–229. https://doi.org/10.1016/S1571-0661(05)80746-7 Workshop on Domains IV.

Regina Tix, Klaus Keimel, and Gordon Plotkin. 2009. Semantic Domains for Combining Probability and Non-Determinism. *Electronic Notes in Theoretical Computer Science* 222 (2009), 3–99. https://doi.org/10.1016/j.entcs.2009.01.002

Daniele Varacca. 2002. The powerdomain of indexed valuations. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*. 299–308. https://doi.org/10.1109/LICS.2002.1029838

Daniele Varacca. 2003. *Probability, Nondeterminism and Concurrency: Two Denotational Models for Probabilistic Computation*. Ph.D. Dissertation. University of Aarhus. https://www.brics.dk/DS/03/14/

Daniele Varacca and Glynn Winskel. 2006. Distributing probability over non-determinism. *Mathematical Structures in Computer Science* 16, 1 (2006), 87–113. https://doi.org/10.1017/S0960129505005074

John von Neumann. 1951. Various techniques used in connection with random digits. In *Monte Carlo Method*, A.S. Householder, G.E. Forsythe, and H.H. Germond (Eds.). National Bureau of Standards Applied Mathematics Series, 12, Washington, D.C.: U.S. Government Printing Office, 36–38.

Linpeng Zhang, Noam Zilberstein, Benjamin Lucien Kaminski, and Alexandra Silva. 2024. Quantitative Weakest Hyper Pre: Unifying Correctness and Incorrectness Hyperproperties via Predicate Transformers. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 300 (oct 2024), 30 pages. https://doi.org/10.1145/3689740

Noam Zilberstein. 2024. Outcome Logic: A Unified Approach to the Metatheory of Program Logics with Branching Effects. arXiv:2401.04594 [cs.LO]

Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 93 (Apr 2023), 29 pages. https://doi.org/10.1145/3586045

Noam Zilberstein, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. 2024a. A Demonic Outcome Logic for Randomized Nondeterminism (Extended Version). arXiv:2410.22540 [cs.LO] https://arxiv.org/abs/2410.22540

Noam Zilberstein, Angelina Saliling, and Alexandra Silva. 2024b. Outcome Separation Logic: Local Reasoning for Correctness and Incorrectness with Computational Effects. *Proc. ACM Program. Lang.* 8, OOPSLA1 (Apr 2024). https://doi.org/10.1145/3649821

Maaike Zwart. 2020. *On the Non-Compositionality of Monads via Distributive Laws*. Ph.D. Dissertation. University of Oxford. https://ora.ox.ac.uk/objects/uuid:b2222b14-3895-4c87-91f4-13a8d046febb

Maaike Zwart and Dan Marsden. 2019. No-Go Theorems for Distributive Laws. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–13. https://doi.org/10.1109/lics.2019.8785707

Received 2024-07-11; accepted 2024-11-07