Modular Specifications and Implementations of Random Samplers in Higher-Order Separation Logic

Virgil Marionneau
ENS Rennes
Rennes, France
virgil.marionneau@ens-rennes.fr

Félix Sassus Bourda
ENS Paris-Saclay
Gif-sur-Yvette, France
felix.sassus-bourda@ens-paris-saclay.fr

Alejandro Aguirre
Aarhus University
Aarhus, Denmark
alejandro@cs.au.dk

Lars Birkedal
Aarhus University
Aarhus, Denmark
birkedal@cs.au.dk

Abstract

Probabilistic programs have a myriad of applications, from randomized algorithms to statistical modeling, and as such have inspired a long tradition of probabilistic program logics to verify their correctness. One essential use of probabilistic programs is to program new samplers from more primitive samplers, e.g., to generate samples from more complex distributions only given a primitive uniform sampler. Such samplers are an ideal case study for probabilistic program logics, to ensure that they implement the target distributions correctly. But proving correctness is often not enough, one also wants to reason about clients of these samplers, which require their specifications to be expressive and reusable.

In this work, we propose a methodology for giving specifications to samplers that are detailed enough to prove that they are correct, and expressive enough to reason about their clients. We propose our methodology for Eris, a recent probabilistic program logic based on the Iris separation logic. We identify what makes the proof rules and reasoning principles for primitive distributions in Eris work, and we distill them into a distribution typeclass. This presents at an abstract level the requirements that a concrete implementation of a target distribution should satisfy, and provides reasoning principles for clients of the interface. Working at this level of abstraction allows us to prove correctness results, as well as to derive additional reasoning principles for all implementations that adhere to the typeclass interface. We instantiate this approach to a variety of samplers for classical distributions, such as binomials, geometrics and beta-binomials.

CPP '26, Rennes, France

ACM ISBN 979-8-4007-2341-4/2026/01
https://doi.org/10.1145/3779031.3779109

CCS Concepts: • **Theory of computation** → **Separation logic**; **Logic and verification**; **Probabilistic computation**; **Program verification**.

Keywords: Probabilistic Programming, Separation Logic, Formal Verification

ACM Reference Format:
Virgil Marionneau, Félix Sassus Bourda, Alejandro Aguirre, and Lars Birkedal. 2026. Modular Specifications and Implementations of Random Samplers in Higher-Order Separation Logic. In *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '26), January 12–13, 2026, Rennes, France.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3779031.3779109

1 Introduction

Randomized sampling in programs is prevalent in many areas of computer science. For example, randomized algorithms [Motwani and Raghavan 1995] use sampling to traverse large search spaces faster, leading to an improved performance at the cost of a small probability of errors. Other examples come from the fields of computer security and cryptography, where randomization is crucial for having strong security guarantees [Goldwasser and Micali 1982]. However, randomization leads to unintuitive behaviors in programs that are hard to reason about.

This has led to a long tradition of developing principles to reason about probabilistic programs, using a variety of techniques such as predicate transformers [Batz et al. 2019; McIver and Morgan 2005]; different kinds of unary and relational program logics [Aguirre et al. 2024; Bao et al. 2025; Barthe et al. 2018, 2015, 2016; Li et al. 2023; Zilberstein et al. 2025]; or techniques inspired by model checking or Markov chain analysis [Chakarov and Sankaranarayanan 2013]. These have different strengths and weaknesses in terms of the base language they target, the ambient logic in which they work on, and the classes of properties they can express and reason about, we discuss them in more detail at the end of the paper.

In most of these approaches, one assumes that the target language only comes equipped with some simple random primitives (e.g. just a uniform bit sampling operation, or a fair choice operator), which simplifies the presentation of the logics. Since it is known that this suffices to represent all distributions of interest [Dal Lago et al. 2014], then one can argue that these primitives and the rules to reason about them are sufficient, and that the approach can be extended to support more distributions if one so wanted to.

This is analogous to the way in which random sampling libraries are implemented in mainstream programming languages. From a primitive source of randomness provided, e.g., by the operating system or the hardware, library designers can then program samplers that implement more complex distributions. When verifying such a library, we have two objectives in mind: first, we want to make sure that the samplers actually implement the target distribution correctly. Second, we want to ensure that the specifications that we prove for the samplers are expressive enough to use them in proofs of larger programs that are clients of the sampling library.

In this paper, we propose a methodology to realize these objectives. We extend Eris [Aguirre et al. 2024], a recent separation logic for higher-order probabilistic programs, with a new library for random samplers. These are implemented in an expressive higher-order language with primitive support for uniform sampling over a finite range of integers. We use the facilities provided by Eris, and a novel proof technique, to give and prove expressive specifications for our samplers, that can be later used in other proofs as if they were primitives. Moreover, we showcase that the abstract reasoning principles that Eris provides for the primitive distributions can also be extended to the distributions implemented in the library. The entire development is mechanized using Rocq [The Rocq Development Team 2025] and Iris [Jung et al. 2015], and the accompanying artifact can be consulted in [Marionneau et al. 2025].

2 Overview of the Approach

In this section we give an informal overview of our approach to specifying and reasoning about random samplers. We present more precise definitions and details in the coming sections.

Eris [Aguirre et al. 2024] is a program logic designed to prove upper bounds on the probability of events considered over the final result and state after a program execution. The logic is implemented as an extension of Iris [Jung et al. 2015]. The key reasoning tool in Eris is a so-called error credit, a separation logic predicate (aka separation logic resource) that can be spent to avoid certain outcomes of probabilistic choices during program execution. Ownership of $\varepsilon \in [0,1]$ error credits is denoted $\lightning(\varepsilon)$. This intuition is realized through the adequacy theorem of Eris: if the triple $\{\lightning(\varepsilon)\}\ e\ \{\phi\}$ is valid, then the probability that executing e will result in a final result that does *not* satisfy ϕ is at most ε. In particular, note that error credits have the property $\lightning(1) \vdash \mathrm{False}$, which corresponds to the fact that the probability of any event is bounded from above by 1.

Error credits are manipulated in proofs through the laws dictated by their resource algebra representation (detailed in Section 3.3) and a rule for sampling from a uniform distribution over the integers $\{0,\dots,N\}$:

$$\frac{\sum_{i=0}^{N} \frac{\mathcal{E}_2(i)}{N+1} = \varepsilon_1}{\vdash \{\lightning(\varepsilon_1)\}\ \mathsf{rand}\, N\ \{v\ .\ \lightning(\mathcal{E}_2(v))\}}\ \textsc{ht-rand-exp}$$

Morally this rule says that if we own ε_1 error credits, we can assume that we will sample some v and own $\mathcal{E}_2(v)$ error credits afterwards as long as the expected value of $\mathcal{E}_2$ is ε_1. Note that in some of the outcomes the amount of credits may be larger than the initial ε_1. One can discard any outcome with 1 error credit using the rule $\lightning(1) \vdash \mathrm{False}$. From this, one can derive the rule

$$\frac{0 \le n \le N}{\vdash \left\{\lightning\left(\tfrac{1}{N+1}\right)\right\}\ \mathsf{rand}\, N\ \{v\ .\ v \neq n\}}\ \textsc{ht-rand-avoid}$$

which allows us to spend $\lightning\left(\frac{1}{N+1}\right)$ to avoid a concrete outcome n, recovering the primary intuition behind error credits.

In addition to the HT-RAND-EXP above, Eris also has so-called presampling rules, shown in Figure 3. These allow the user to generate randomness in advance at the logical level, which will be used later physically by the program. Presampling is a useful reasoning principle, e.g., in proofs of almost sure termination. Presampling requires ownership of a *presampling tape*, a separation logic resource that holds random samples that will be used in the execution. The separation logic predicate $\iota \hookrightarrow (N, \vec{n})$ denotes ownership of a tape with label ι containing a sequence $\vec{n}$ of samples uniformly distributed in $\{0,\dots,N\}$. Tapes can be allocated using similar syntax as standard references, following the rule ALLOC-TAPE. The rule PRESAMPLE-EXP appends a fresh uniform sample to a tape and, analogously to HT-RAND-EXP, allows the user to distribute error credits across the outcomes. Sampling instructions can then be annotated with a tape label, and will deterministically read the first element of the corresponding tape, as specified by LOAD-TAPE. A more involved, *planner rule* (PRESAMPLE-PLANNER) allows users to presample at once a large sequence of samples ending on a particular suffix.

Now suppose we have implemented a sampler bern $p\ q$ which returns true with probability p/q and false with probability $(q-p)/q$, for $0 < p \le q$, and we want to prove it is correct. Following the intuition behind the meaning of the triples, one could think of the following naïve specification:

$$\forall b\colon \mathbb{B},\ \{\lightning(\text{if } b \text{ then } (q-p)/q \text{ else } p/q)\}\ \mathsf{bern}\ p\ q\ \{v\ .\ v = b\}$$

This specification allows us to spend $\lightning((q-p)/q)$ to ensure that the result of sampling is true, or to use $\lightning(p/q)$ to ensure the result is false, which is precisely how a Bernoulli sampler should behave. This specification does indeed prove correctness of the sampler. However, we argue that this specification is *not strong enough* to be used in clients of the sampler, as shown by the example below:

Example 2.1. Consider the program below:

$$e \triangleq \text{if bern 2 3 then } () \text{ else if bern 2 3 then } () \text{ else fail}$$

This program returns safely with probability $8/9$, otherwise it crashes. Therefore, we would hope to be able to show the specification

$$\{\lightning(1/9)\}\ e\ \{v\ .\ \text{True}\}$$

However, the specification

$$\forall b : \mathbb{B},\ \{\lightning(\text{if } b \text{ then } 1/3 \text{ else } 2/3)\}\ \text{bern 2 3}\ \{v\ .\ v = b\}$$

is not enough to prove this, because we need to supply $\lightning(1/3)$ to resolve the first bern to true, but we only have $\lightning(1/9)$.

The key observation behind our approach is that the HT-RAND-EXP rule captures the fact that rand N yields a uniformly distributed value by describing not how it consumes error credits, but how it can *transform* them. To characterize a sampler for another distribution μ, we have to describe how it can transform credits. We can do this by showing a rule similar to HT-RAND-EXP, but where the expectation is taken over μ.

We propose then to give the following specification to the Bernoulli sampler:

$$\forall \mathcal{E}_2.\ \left\{\left(\frac{p}{q}\cdot\mathcal{E}_2(\text{true}) + \frac{q-p}{q}\cdot\mathcal{E}_2(\text{false})\right) = \varepsilon_1 * \lightning(\varepsilon_1)\right\}\ \text{bern } p\ q\ \{b\ .\ \lightning(\mathcal{E}_2(b))\}$$

This specification states that if we begin by owning ε_1 error credits, then we will sample some b and own $\mathcal{E}_2(b)$ error credits, as long as the expected value of $\mathcal{E}_2$ under the corresponding Bernoulli distribution (i.e. $(p/q)\cdot\mathcal{E}_2(\text{true}) + ((q-p)/q)\cdot\mathcal{E}_2(\text{false})$) is equal to ε_1. In particular, the previous naïve specification is derivable from this one.

Example 2.1 (continued). *With this specification we can now finish our proof. For the first sampling in e we instantiate the specification of* bern 2 3 *with*

$$\mathcal{E}_2(b) \triangleq \text{if } b \text{ then } 0 \text{ else } 1/3$$

It is easy to check that the precondition is satisfied, since

$$(2/3)\cdot 0 + (1/3)\cdot(1/3) = (1/9)$$

Then we will either be in the then branch, owning $\lightning(0)$, and conclude immediately, or in the else branch, owning $\lightning(1/3)$. In the latter case, we use the specification of bern 2 3 *again, this time with*

$$\mathcal{E}_2(b) \triangleq \text{if } b \text{ then } 0 \text{ else } 1$$

In the then branch, we conclude immediately. Otherwise, we will own $\lightning(1)$, and since $\lightning(1) \vdash$ False, we can also conclude.

The soundness of our approach is proven through a novel *distribution adequacy* theorem, which is stated using total correctness Hoare triples.[1]

Theorem 2.2. *Let e be an expression, and μ a probability distribution over values. Suppose the statement below is derivable in Eris:*

$$\vdash \forall \varepsilon_1, \mathcal{E}_2.\left[\sum_v \mu(v)\cdot\mathcal{E}_2(v) = \varepsilon_1 * \lightning(\varepsilon_1)\right] e\ [v.\lightning(\mathcal{E}_2(v))]$$

Then, the final value of e distributes as μ.

Using the expressive program logic features of Eris (inherited from Iris), we further prove presampling rules for the implemented samplers. For example, ownership of a Bernoulli tape can be expressed in terms of ownership of a standard (uniform) tape and a condition on its contents that establishes how each Bernoulli sample is translated into a uniform sample.

To provide a unified interface for the newly implemented samplers, we introduce an abstract distribution typeclass, which states that a program e implements a meta-level distribution μ. The interface imposes a series of Eris proof obligations on the user, and provides a set of proof rules that can be used to reason about clients of the samplers. Crucially, we show that a generic planner rule can be derived for any sampler as a consequence of the typeclass definition, which allows us to develop expressive reasoning principles for complex distributions from a single proof.

3 Preliminaries

3.1 Probability Theory

Definition 3.1 (Mass). *For a countable set A and a function $\mu : A \to [0,1]$, we write $|\mu|$ for the sum $\sum_{a\in A}\mu(a)$ when it is finite.*

Definition 3.2 (Subdistribution). *A subdistribution over a countable set A is a function $\mu : A \to [0,1]$ such that $|\mu| \le 1$. We let $\mathcal{D}(A)$ denote the set of all subdistributions over A. We call the set $\operatorname{supp}\mu \triangleq \{a \in A \mid 0 < \mu(a)\}$ the support of μ.*

We need subdistributions instead of full distributions to account for non-terminating programs. This does not come at a high cost as much of the structure of interest when working with distributions is still present for subdistributions.

Proposition 3.3. *The operation $A \mapsto \mathcal{D}(A)$ admits monadic structure.*

[1] We use curly brackets for partial correctness Hoare triples and square brackets for total correctness Hoare triples as detailed in the next section.

Proof. Define the action of $\mathcal{D}$ on $f : A \to B$ as $\mathcal{D}(f)(\mu)(b) \triangleq \sum_{a \in A} \delta_b(f(a)) \cdot \mu(a)$, where δ_x is the mass function of the Dirac distribution, which maps x to 1, and any $x' \neq x$ to 0; and define the unit and bind operations as $\mathrm{ret}_A(a)(b) = \delta_a b$ and $(\mu >\!\!>\!\!= f)(b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b)$. Functoriality of $\mathcal{D}$ and the monadic laws are easily checked. □

Remark 3.4. *While $\mathcal{D}$ admits monadic operations, it is not itself a monad, since, for a non-empty countable set A, $\mathcal{D}(A)$ is not itself countable. Thus $\mathcal{D}$ is not an endofunctor. Instead, it is an example of a relative monad [Altenkirch et al. 2015].*

Definition 3.5 (Restriction)**.** *For a subdistribution μ, we write $\mu_{|P}(x)$ for the subdistribution defined by:*

$$\mu_{|P}(x) = \begin{cases} \mu(x) & \text{if } P(x) \text{ holds} \\ 0 & \text{otherwise} \end{cases}$$

We call it the restriction of μ to P. The mass $|\mu_{|P}|$ of the restriction of μ to P is called the probability of P with respect to μ. We write it as $\Pr_\mu[P]$.

3.2 Programming Language and Semantics

The language we consider is RandML [Gregersen et al. 2024], a variant of sequential ML with higher-order dynamic store and probabilistic sampling. Its syntax, detailed in Figure 1, is mostly standard. However, we call attention to the rand and tape constructs. RandML supports two kinds of primitive sampling commands: unlabeled and labeled. The former takes a single integer argument N and evaluates to a uniformly distributed integer between 0 and N, both included. The second takes two arguments, an integer N and a tape label ι, and is used to support the presampling rules. Tapes are a piece of state and behave similarly to references: the tape N construct allocates an empty tape that will be used to contain uniform samples between 0 and N, and returns a label. The command rand $N\ \iota$ will then check that the tape ι is of the right type, return its first element deterministically, and remove it from the tape. Notice that there is no language construct that adds values onto the tape, we will later explain how they are populated at the logical level. The interested reader can also consult [Gregersen et al. 2024] for mode details on how tapes are used, and for a proof of why it is sound to add them to the language.

The language is equipped with mostly standard probabilistic small-step operational semantics. A single step of reduction will be modeled by a function $\mathrm{step} : \mathit{Cfg} \to \mathcal{D}(\mathit{Cfg})$, most of the expressions in the language reduce deterministically and so for those the reduction is simply a Dirac distribution. Some of the more interesting cases are detailed in Figure 2. In particular, note how the reduction for the labeled sampling is deterministic, except in the case where the corresponding tape is empty, where a fresh sample is produced.

After defining single steps of computations, we define $\mathrm{exec}_n : \mathit{Cfg} \to \mathcal{D}(\mathit{Val})$ to represent the subdistribution of values reached after n steps of execution:

$$\mathrm{exec}_n(e,\sigma) \triangleq \begin{cases} \mathrm{ret}(e) & \text{if } e \in \mathit{Val} \\ \mathrm{step}(e,\sigma) >\!\!>\!\!= \mathrm{exec}_{n-1} & \text{if } e \notin \mathit{Val} \text{ and } n > 0 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

By taking the limit of $\mathrm{exec}_n(\rho)$ as n goes to infinity, we obtain the subdistribution representing the results of a full execution of ρ, which we denote by $\mathrm{exec}(\rho)$. The probability of termination of a full execution of ρ is given by $|\mathrm{exec}(\rho)|$. We call a configuration almost-surely terminating if its probability of termination is 1.

3.3 The Eris Program Logic

Eris is a probabilistic variant of Iris [Jung et al. 2015] and it inherits most of its syntax and derivation rules from it. In this paper, we focus on the main concepts of Eris required to follow our contributions; we refer the interested reader to Aguirre et al. [2024] for further details.

The syntax of propositions in Eris is outlined below. It includes standard separation logic connectives, as well as Iris-specific modalities such as persistently ($\Box$) and later ($\triangleright$). Notably, it is a higher-order program logic, so Hoare triples $\{P\}\ e\ \{Q\}$ and $[P]\ e\ [Q]$ are also propositions in Eris. The former indicates partial correctness, while the latter indicates total correctness.

$$\begin{aligned} &P, Q \in \mathit{iProp} ::= \\ &\quad \mathsf{True} \mid \mathsf{False} \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q \mid \forall x.\, P \mid \exists x.\, P \\ &\quad \mid P * Q \mid P \mathrel{-\!\!*} Q \mid \Box P \mid \triangleright P \mid \ell \mapsto v \\ &\quad \mid \lightning(\varepsilon) \mid \iota \hookrightarrow (N, xs) \mid \{P\}\ e\ \{Q\} \mid [P]\ e\ [Q] \mid \ldots \end{aligned}$$

The main novelty of Eris is the addition of a new resource called "error credits", whose rules are outlined below:

$$\vdash \lightning(0) \qquad \lightning(\varepsilon_1) * \lightning(\varepsilon_2) \dashv\vdash \lightning(\varepsilon_1 + \varepsilon_2) \qquad \lightning(1) \vdash \mathsf{False}$$

Ownership of ε error credits (denoted $\lightning(\varepsilon)$) can be seen as an allowance to fail to prove the specification under scrutiny with at most ε probability. Hence, owning $\lightning(1)$ allows us to prove any conclusion. This is made precise by the following theorem for partial correctness triples:

Theorem 3.6 (Adequacy)**.** *For any state σ, if $\vdash \{\lightning(\varepsilon)\}\ e\ \{\phi\}$ then $\Pr_{\mathrm{exec}(e,\sigma)}[\neg\phi] \leq \varepsilon$. Moreover, the probability of e getting stuck is at most ε.*

Notice that $\mathrm{exec}(e,\sigma)$ is a subdistribution and as such this theorem does not imply the stronger fact that $\Pr_{\mathrm{exec}(e,\sigma)}[\phi] \geq 1-\varepsilon$. In order to derive such an inequality, one needs to use total correctness triples instead. The difference between them amounts to considering non-termination as a valid behavior for partial correctness, but not for total correctness. Total correctness triples can therefore be weakened to partial ones. For total correctness triples, the following holds:

$$
\begin{aligned}
v \in \mathit{Val} &\triangleq z \in \mathbb{Z} \mid b \in \mathbb{B} \mid () \mid \ell \in \mathit{Loc} \mid \iota \in \mathit{Label} \mid \mathsf{rec}\ \mathsf{f}\ \mathsf{x} = e \mid (v, w) \mid \mathsf{inl}\, v \mid \mathsf{inr}\, v \\
e \in \mathit{Expr} &\triangleq v \mid \mathsf{x} \mid e_1\ e_2 \mid e_1 + e_2 \mid e_1 - e_2 \mid \ldots \mid \\
&\quad \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid (e_1, e_2) \mid \mathsf{fst}\, e \mid \mathsf{snd}\, e \mid \mathsf{inl}(e) \mid \mathsf{inr}(e) \mid \mathsf{match}\ e\ \mathsf{with}\ \mathsf{inl}\, v \Rightarrow e_1 \mid \mathsf{inr}\, w \Rightarrow e_2\ \mathsf{end} \mid \\
&\quad \mathsf{allocn}\ e_1\ e_2 \mid\ !\, e \mid e_1 \leftarrow e_2 \mid \mathsf{rand}\, e \mid \mathsf{rand}\, e_1\ e_2 \mid \mathsf{tape}\, e \\
K \in \mathit{Ectx} &\triangleq\ - \mid e\, K \mid K\, v \mid \mathsf{allocn}\, K \mid\ !\, K \mid e \leftarrow K \mid K \leftarrow v \mid \mathsf{rand}\, K \mid \ldots \\
t \in \mathit{Tape} &\triangleq \{(N, \vec{n}) \mid N \in \mathbb{N} \wedge \vec{n} \in \mathsf{List}\, \mathbb{N}_{\leq N}\} \\
\sigma \in \mathit{State} &\triangleq (\mathit{Loc} \xrightarrow{\mathsf{fin}} \mathit{Val}) \times (\mathit{Label} \xrightarrow{\mathsf{fin}} \mathit{Tape}) \qquad \rho \in \mathit{Cfg} \triangleq \mathit{Expr} \times \mathit{State}
\end{aligned}
$$

Figure 1. Syntax of RandML

$$
\mathrm{step}(\mathsf{rand}\ N, \sigma) \triangleq \begin{cases} \mu_{\mathcal{U}(\{0,\ldots,N\}\times\{\sigma\})} & \text{if } N \in \mathbb{N} \\ \mathbf{0} & \text{otherwise} \end{cases} \qquad \mathrm{step}(\mathsf{tape}\, N, \sigma) \triangleq \begin{cases} \delta_{(\iota, \sigma[\iota \mapsto (N,\epsilon)])} & \text{if } N \in \mathbb{N}, \text{ with } \iota \text{ fresh in } \sigma \\ \mathbf{0} & \text{otherwise} \end{cases}
$$

$$
\mathrm{step}(\mathsf{rand}\ \iota\ N, \sigma) \triangleq \begin{cases} \delta_{(n, \sigma[\iota \mapsto (N,\vec{n})])} & \text{if } N \in \mathbb{N} \text{ and } \iota \mapsto (N, n :: \vec{n}) \in \sigma \\ \mu_{\mathcal{U}(\{0,\ldots,N\}\times\{\sigma\})} & \text{if } N \in \mathbb{N} \text{ and } \iota \mapsto (N, n :: \vec{n}) \notin \sigma \\ \mathbf{0} & \text{otherwise} \end{cases}
$$

where $\mu_{\mathcal{U}(A)}$ is the mass function of the uniform distribution on a finite set A.

Figure 2. Selected reduction rules of RandML

Theorem 3.7 (Total Adequacy). *If* $\vdash [\lightning(\varepsilon)]\ e\ [\phi]$ *then* $\Pr_{\mathrm{exec}(e,\sigma)}[\phi] \geq 1 - \varepsilon$, *for any state* σ. *In particular, the probability of termination of* (e, σ) *is at least* $1 - \varepsilon$.

Partial or total correctness triples can be proven in the logic using familiar derivation rules directed by the program syntax, which we omit here to focus on the rules that manipulate error credits. As mentioned in §2, one of the key proof rules is the following sampling rule (here displayed for total correctness triples):

$$
\frac{\sum_{i=0}^{N} \frac{\mathcal{E}_2(i)}{N+1} = \varepsilon_1}{\vdash [\lightning(\varepsilon_1)]\ \mathsf{rand}\, N\ [n\ .\ \lightning(\mathcal{E}_2(n))]}\ \textsc{ht-rand-exp-t}
$$

Ownership of a tape is denoted by the predicate $\iota \hookrightarrow (N, \vec{n})$, specifying that $\iota \mapsto (N, \vec{n})$ is in the current state of the program. Rules governing tapes can be seen in Figure 3. In particular, Eris provides the user with a ghost operation PRESAMPLE-EXP which allows to change the state of the tape through a logical update.

Tapes provide a way to decouple reasoning about error credits from program reduction, thus making for cleaner proofs. For example, sometimes it is useful to pre-populate the tape in advance and then reason about the program as if it was deterministic. This does not lose expressivity, since HT-RAND-EXP-T and PRESAMPLE-EXP behave in the same way.

Furthermore, tapes allow for one more powerful principle. As they are not tied to program execution, one can keep presampling until the tape ends in a predetermined suffix. This is captured by the PRESAMPLE-PLANNER rule, sometimes referred to as the "planner rule". The rule states that, assuming we own a tape ι with contents xs, we can keep presampling randomness to the tape until it is of the form $xs \cdot ys \cdot s(xs \cdot ys)$, where ys is some unknown sequence of samples and $s(xs \cdot ys)$ is a pre-chosen suffix that might depend on both $xs \cdot ys$. We refer interested readers to [Aguirre et al. 2024] for details on why the rule is sound and for examples of how it can, e.g., be used to show almost-sure termination; we will also show an example in §5. The intuition is that the valid suffixes have finite length at most L, so with probability 1 the presampling process will eventually terminate and produce a valid suffix.

4 Specifying and Implementing New Samplers in Eris

The rules provided by Eris to reason about rand form a powerful and modular framework to prove properties of programs sampling from a uniform distribution. However the uniform distribution is not the only distribution one might wish to investigate. Indeed many algorithms rely on Bernoulli sampling and many real life situations can be modeled by more complex distributions such as the binomial distribution. Many of those distributions can be implemented inside RandML. We want to prove that the samplers that we implement are correct, that is, their output distributes as the target probability distribution. But we also want to have abstract reasoning principles for them that allow us to use them in larger programs and reason about them.

To address this challenge, we leverage the expressivity of the ambient logic (Rocq) and of Eris itself. We define a distribution typeclass that implemented samplers should

$$\frac{}{\vdash [\mathsf{True}]\ \mathsf{tape}\, N\ [\iota.\, \iota \hookrightarrow (N, \epsilon)]}\ \textsc{alloc-tape} \qquad \frac{}{\vdash [\iota \hookrightarrow (N, n :: \vec{n})]\ \mathsf{rand}\, N\, \iota\ [x.\, x = n * \iota \hookrightarrow (N, \vec{n})]}\ \textsc{load-tape}$$

$$\frac{e \notin Val \qquad \sum_{i=0}^{N} \frac{\mathcal{E}_2(i)}{N+1} = \varepsilon_1 \qquad \vdash \forall n.\, [\iota \hookrightarrow (N, \vec{n} \cdot [n]) * \lightning(\mathcal{E}_2(n))]\ e\ [P]}{\vdash [\iota \hookrightarrow (N, \vec{n}) * \lightning(\varepsilon_1)]\ e\ [P]}\ \textsc{presample-exp}$$

$$\frac{\forall l.\, |s(l)| \le L \qquad \vdash [\exists ys.\, \iota \hookrightarrow (N, xs \cdot ys \cdot s(xs \cdot ys))]\ e\ [P]}{\vdash [\iota \hookrightarrow (N, xs)]\ e\ [P]}\ \textsc{presample-planner}$$

Figure 3. Rules for presampling tapes

adhere to. To satisfy the typeclass constraints, implementers have to prove that their sampler satisfies a series of reasoning principles that can then be used abstractly by clients of the sampler. The advantages of enforcing the typeclass interface are multiple. First, it gives implementers a clear guideline to follow. Second, one can derive reasoning principles for *all* implemented samplers, independently of the distribution that they implement. Finally, one can prove a correctness result once and for all implemented samplers. This section describes the distribution typeclass and the methodology around it.

4.1 The Distribution Typeclass

The distribution typeclass is defined in Figure 4. The typeclass distrImpl ($\mu\colon \mathcal{D}(Val)$) characterizes an implementation of a sampler for a distribution μ over values. It consists of a program that implements the distribution μ, propositions to denote ownership of abstract tapes for the implemented distribution, as well as proofs of propositions that characterize how the sampling expression is used, and how it interacts with tapes and credits. Note that these are actually abstract versions of the rules for the primitive uniform distribution.

The program sample is meant to represent the main program implementing the distribution. It takes as argument the label of an abstract tape, which can be the null label (unitLoc). The typeclass also provides a rule HT-DISTR-EXP that characterizes the sampler. This rule states that, whenever we sample from sample we can distribute $\lightning(\varepsilon_1)$ error credits, assigning $\lightning(\mathcal{E}_2(v))$ to each outcome v as long as the expected value of $\mathcal{E}_2$ over μ is ε_1. It is this rule where the concrete implementation sample is connected to the meta-(Rocq-)level distribution μ. Note how this generalizes the HT-RAND-EXP-T rule for the uniform distribution, except that now we take expected values over the target distribution μ.

The typeclass also includes propositions to assert ownership of abstract tapes containing samples from μ, as well as rules to manipulate them. Notice that, operationally, the language only supports tapes containing uniform samples. However, one can implement abstract tapes for other distributions in terms of uniform tapes, analogously to the way one implements samplers for other distributions in terms of the uniform sampler. Later in the paper we will see how this is achieved for some example distributions.

Thus, the predicate ownTape $\alpha\ l$ asserts ownership of an abstract tape with an abstract label $\alpha\colon$ absLoc, and contents l. Abstract tapes are allocated via a program allocTape, which returns a label and an empty tape, as specified by ALLOC-DISTR-TAPE. A labeled sampling instruction sample α requires ownership of α and reads its first element and removes it, as specified by LOAD-DISTR-TAPE. Finally, the rule PRESAMPLE-DISTR-EXP allows us to presample elements to the abstract tape and distribute error credits across all possible outcomes, again requiring their expected value over μ to be equal to the initial amount.

4.2 Distribution Adequacy

The first application of the distribution typeclass is that we can prove a single correctness result for all expressions that implement it. Suppose that e implements the typeclass distrImpl μ for some $\mu\colon \mathcal{D}(Val)$. This means, in particular, that e satisfies this rule:

$$\frac{\forall v \in Val\,.\ 0 \le \mathcal{E}_2(v) \le 1 \qquad \sum_{v \in Val} \mu(v) \cdot \mathcal{E}_2(v) = \varepsilon_1}{\vdash [\lightning(\varepsilon_1)]\ e\ [v.\ \lightning(\mathcal{E}_2(v))]}$$

Under those assumptions, set

$$\mathcal{E}_v^{eq}(w) \triangleq \begin{cases} 0 & \text{if } w = v \\ 1 & \text{otherwise} \end{cases} \qquad \mathcal{E}_v^{ne}(w) \triangleq \begin{cases} 1 & \text{if } w = v \\ 0 & \text{otherwise} \end{cases}$$

We can then easily check that both of those functions are positive and bounded above by 1 and that their expected values are as follows:

$$\sum_{w \in Val} \mu(w) \cdot \mathcal{E}_v^{eq}(w) = |\mu| - \mu(v), \quad \sum_{w \in Val} \mu(w) \cdot \mathcal{E}_v^{ne}(w) = \mu(v)$$

Hence we can derive the following two rules:

$$\frac{}{\vdash [\lightning(|\mu| - \mu(v))]\ e\ [w.\ w = v]}\ \textsc{ht-mu-eq}$$

$$\frac{}{\vdash [\lightning(\mu(v))]\ e\ [w.\ w \neq v]}\ \textsc{ht-mu-ne}$$

from which we can prove the following result, which asserts that the final result of evaluating e distributes as μ:

$$\boxed{\text{distrImpl}\ (\mu\colon \mathcal{D}(Val))}$$

$$\text{sample}\colon Val;\quad \text{absLoc}\colon Type;\quad \text{ownTape}\colon \text{absLoc} \to \text{List}\, Val \to iProp;$$

$$\text{isAbsLoc}\colon \text{absLoc} \to Val \to iProp;\quad \text{allocTape}\colon Val;\quad \text{unitLoc}\colon Val;$$

$$\frac{\forall v \in Val\,.\ 0 \le \mathcal{E}_2(v) \le 1 \qquad \sum_{v \in Val} \mu(v) \cdot \mathcal{E}_2(v) = \varepsilon_1}{[\lightning(\varepsilon_1)]\ \text{sample unitLoc}\ [v.\ \lightning(\mathcal{E}_2(v))]}\ \textsc{ht-distr-exp};$$

$$\frac{}{[\text{True}]\ \text{allocTape}\ ()\ [(\Delta\colon \text{absLoc})(\alpha\colon Val).\ \text{isAbsLoc}\ \Delta\ \alpha * \text{ownTape}\ \alpha\ []]}\ \textsc{alloc-distr-tape};$$

$$\frac{}{[\text{isAbsLoc}\ \Delta\ \alpha * \text{ownTape}\ \alpha\ (w \cdot l)]\ \text{sample}\ \alpha\ [v.\ v = w * \text{ownTape}\ \alpha\ l]}\ \textsc{load-distr-tape};$$

$$\frac{e \notin Val \qquad \forall v \in Val\,.\ 0 \le \mathcal{E}_2(v) \le 1 \qquad \sum_{v \in Val} \mu(v) \cdot \mathcal{E}_2(v) = \varepsilon_1 \qquad \forall v.\ [\text{ownTape}\ \alpha\ (l \cdot [v]) * \lightning(\mathcal{E}_2(v))]\ e\ [P]}{[\text{ownTape}\ \alpha\ l * \lightning(\varepsilon_1)]\ e\ [P]}\ \textsc{presample-distr-exp}$$

Figure 4. Definition of the distribution typeclass. We use proof rule notation for propositions in *iProp*, where the separating conjunction of all the premises implies (via separating implication) the conclusion

Theorem 4.1 (Distribution Adequacy)**.** *Under the conditions for e and μ outlined above, we have, for any $v \in Val$,*

$$\Pr_{\text{exec}(e)}[\lambda w, w = v] = \mu(v)$$

Proof. Applying Theorem 3.7 to HT-MU-EQ and Theorem 3.6 to HT-MU-NE, and using the fact $0 \le |\mu| \le 1$, we get the following bounds on $\Pr_{\text{exec}(e)}[\lambda w, w = v]$:

$$\mu(v) \le \mu(v) + (1 - |\mu|) \le \Pr_{\text{exec}(e)}[\lambda w, w = v] \le \mu(v)$$

Therefore $\Pr_{\text{exec}(e)}[\lambda w, w = v] = \mu(v)$. □

4.3 The Generic Planner Rule

In this section, we show that the rules in the distribution interface suffice to derive a planner rule, analogous to the PRESAMPLE-PLANNER rule. Note that this means that the planner rule can be proven once, just from the abstract interface, and then instantiated to all samplers that implement the typeclass.

The statement of the rule is the following:

$$\textsc{presample-distr-planner}\quad \frac{\begin{array}{l} e \notin Val \qquad R \text{ finite} \qquad \forall \vec{n}.\ a \in R \to 0 < \mu\ a \\ \forall \vec{n}.\ a \in s(\vec{n}) \to a \in R \qquad \forall \vec{n}.\ |s(\vec{n})| \le L \\ \vdash \forall v.\ \left[\text{ownTape}\ \alpha\ (\vec{n} \cdot \vec{j} \cdot s(\vec{n} \cdot \vec{j}))\right]\ e\ [P] \end{array}}{\vdash [\text{ownTape}\ \alpha\ \vec{n}]\ e\ [P]}$$

In order to support arbitrary distributions that might have non-finite support, we require that all elements appearing in suffixes must contained in some finite set R. However, we believe that this restriction can be lifted by adding other constraints on s, for instance by implementing some form of martingale-based termination condition [Majumdar and Sathiyanarayana 2025].

The proof of this rule is a generalization of the proof of the basic planner rule from [Aguirre et al. 2024]. At a high level, the proof of the planner rule proceeds by repeatedly applying the PRESAMPLE-EXP rule and choosing the coefficients $\mathcal{E}_2(i)$ in the premise depending on the current state. By choosing them appropriately, it is possible to ensure one out of two eventual outcomes: either we sample the desired suffix, or we increment our error credits until we reach $\lightning(1)$. Therefore it is sound to assume the former.

To prove the general version of the rule for the interface, we first prove an *abstract planner rule* which is phrased in terms of abstract predicates about programs and resources (in place of the Hoare triples and the tape ownership predicates) and it allows us to isolate the complicated credit arithmetic. We refer interested readers to the appendix for more details about the proof.

5 Case Studies

Now armed with results allowing us to derive great amounts of expressivity from simple prerequisites, we focus on showing that meeting those prerequisites is feasible. We do this by detailing various implementations of some probability distributions in RandML and showing how they satisfy the constraints of the distribution typeclass, in particular showing how the HT-DISTR-EXP and PRESAMPLE-DISTR-EXP rules

are proven for concrete implementations, as well as how the abstract tapes are defined for them. We first treat the Bernoulli distribution in detail before focusing on some of the core ideas of the subsequent distributions.

5.1 The Bernoulli Distribution

The Bernoulli distribution with parameter $\theta \in [0,1]$ models a weighted coin flip where the probability of the coin landing on heads is θ and, conversely, the probability of it landing on tails is $1-\theta$. We denote its mass function by $\mu_{\mathcal{B}(\theta)}$. We choose to restrict our attention to rational parameters as RandML does not support real numbers as a base type, and adding them is orthogonal to our goals. We thus assume that every parameter for the distribution is of the form $\theta = \frac{p}{q+1}$ with $p, q \in \mathbb{N}$ and $p \leq q+1$. In turn this makes it easy to represent the parameters of the distribution as pairs of natural numbers (p,q) with $p \leq q+1$. With this representation in mind, one can see the Bernoulli distribution in another light — it is the distribution of an urn model where one draws exactly one ball from an urn containing $q+1$ balls, p of which are red. Here drawing a red ball is analogous to the coin landing on heads. With this model laid out, an implementation arises almost immediately as the following program:

$$\begin{aligned}\mathsf{bern}\ p\ q\ \iota \triangleq\ & \mathsf{let}\ k = \mathsf{rand}\ q\ \iota\ \mathsf{in}\\ & \mathsf{if}\ k < p\ \mathsf{then}\ 1\ \mathsf{else}\ 0\end{aligned}$$

We will write $\mathsf{bern}\ p\ q$ for the same program where the rand operation is unlabeled; from now on we will always implicitly follow this convention. The first thing we do after defining this implementation is to prove that it admits a specification that is an instance of HT-DISTR-EXP. Given that the support of this particular distribution is simply $\{0,1\}$, we can formulate it in the following way:

HT-BERNOULLI-EXP

$$\frac{\varepsilon_1, \varepsilon_2 \geq 0 \qquad p \leq q+1 \qquad \varepsilon_1 \cdot (1 - \frac{p}{q+1}) + \varepsilon_2 \cdot \frac{p}{q+1} = \varepsilon}{\vdash [\lightning(\varepsilon)]\ \mathsf{bern}\ p\ q\ [v.\ v = 0 * \lightning(\varepsilon_1) \vee v = 1 * \lightning(\varepsilon_2)]}$$

We remark that this specification (as well as the ones for other distributions in this section) is proven internally in Eris by reasoning over the code of $\mathsf{bern}\ p\ q$. We first apply HT-RAND-EXP-T to the $\mathsf{rand}\ q$ statement, setting

$$\mathcal{E}_2(i) \triangleq \mathsf{if}\ i < p\ \mathsf{then}\ \varepsilon_2\ \mathsf{else}\ \varepsilon_1$$

This requires us to show $\sum_{i \in \{0,\dots,q\}} \mathcal{E}_2(i) = \varepsilon$, which follows from the assumptions. Then, we sample some k and own $\lightning(\mathcal{E}_2(k))$. Finally, we do a case distinction on whether $k < p$, letting us conclude.

With this taken care of, we can immediately make use of Theorem 4.1 to prove that this program indeed models the Bernoulli distribution with parameter $\theta = \frac{p}{q+1}$.

The next step is to define an abstract notion of tape $\iota \hookrightarrow_{\mathcal{B}} (p,q,\vec{n})$ expressing ownership and knowledge of the next outcomes of running $\mathsf{bern}\ p\ q\ \iota$ are described by $\vec{n}$. Since bern is implemented using the primitive rand, we can implement a tape for bern by owning a tape for rand, and then translating the uniform samples into Bernoulli samples. We achieve this through the use of the predicate $\vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m}$, where $\vec{n} \in \mathsf{List}\ \{0,\dots,q\}$ and $\vec{m} \in \mathsf{List}\ \{0,1\}$, inductively defined in Figure 5. The tape predicate is then simply defined as:

$$\iota \hookrightarrow_{\mathcal{B}} (p,q,\vec{m}) \triangleq \exists \vec{n} \in \mathsf{List}\ \{0,...,q\}.\ \vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m} * \iota \hookrightarrow (q,\vec{n})$$

$$\frac{}{\epsilon\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \epsilon}\ \text{BERNOULLI-TL-NIL}$$

$$\frac{n < p \qquad \vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m}}{n :: \vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ 1 :: \vec{m}}\ \text{BERNOULLI-TL-CONS-LT}$$

$$\frac{n \geq p \qquad \vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m}}{n :: \vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ 0 :: \vec{m}}\ \text{BERNOULLI-TL-CONS-GE}$$

Figure 5. Translation predicate for the Bernoulli distribution

Our next obligation is to define an expression that allocates empty Bernoulli tapes. This is realized by the program:

$$\mathsf{alloc}_{\mathcal{B}}\ p\ q \triangleq \mathsf{tape}\ q$$

While the first parameter of $\mathsf{alloc}_{\mathcal{B}}$ is ignored and could be dispensed with, we choose to have it explicitly to make the code clearer. We can prove the allocation rule for Bernoulli tapes by combining ALLOC-TAPE and BERNOULLI-TL-NIL:

$$\frac{}{\vdash [\mathsf{True}]\ \mathsf{alloc}_{\mathcal{B}}\ pq\ [\iota.\ \iota \hookrightarrow_{\mathcal{B}} (p,q,\epsilon)]}\ \text{BERNOULLI-ALLOC-TAPE}$$

The load rule for Bernoulli tapes follows by doing a case distinction on whether the first element is 0 or 1, applying either BERNOULLI-TL-CONS-GE or BERNOULLI-TL-CONS-LT to translate it back to a uniform tape, and finally executing bern symbolically. At the sampling point, we apply PRESAMPLE-LOAD to read the first element of the uniform tape, which ensures we will go to the correct branch. The rule we then obtain looks as follows:

LOAD-BERNOULLI-TAPE

$$\frac{}{\vdash [\iota \hookrightarrow_{\mathcal{B}} (p,q,b :: \vec{n})]\ \mathsf{bern}\ pq\ \iota\ [v.v = b * \iota \hookrightarrow_{\mathcal{B}} (p,q,\vec{n})]}$$

To prove the presampling rule for this tape predicate, we need the following two properties, which are easily shown by induction on the derivation of $\vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m}$:

$$\frac{n < p \qquad \vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m}}{\vec{n} \cdot [n]\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m} \cdot [1]}\ \text{BERNOULLI-TL-SNOC-LT}$$

$$\frac{n \geq p \qquad \vec{n}\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m}}{\vec{n} \cdot [n]\ \overset{q}{\mathcal{U}}\sim\overset{p,q}{\mathcal{B}}\ \vec{m} \cdot [0]}\ \text{BERNOULLI-TL-SNOC-GE}$$

We can then show the following rule by first unpacking the uniform tape predicate from the definition of $\iota \hookrightarrow_{\mathcal{B}} (p, q, \vec{n})$, and then using PRESAMPLE-EXP (with $\mathcal{E}_2$ defined identically as for the proof of HT-BERNOULLI-EXP) to sample a new value at the end of the tape, before using BERNOULLI-TL-SNOC-LT or BERNOULLI-TL-SNOC-GE to update the translation predicate and repackage everything into an abstract tape predicate:

PRESAMPLE-BERNOULLI-EXP

$$\frac{\begin{array}{c} e \notin \mathit{Val} \qquad \varepsilon_1 \geq 0 \qquad \varepsilon_2 \geq 0 \qquad p \leq q+1 \\ \varepsilon_1 \cdot (1 - \frac{p}{q+1}) + \varepsilon_2 \cdot \frac{p}{q+1} = \varepsilon \\ \vdash \forall i \in \{0,1\} [\iota \hookrightarrow_{\mathcal{B}} (p, q, \vec{n} \cdot [i]) * \lightning(\varepsilon_i)]\ e\ [P] \end{array}}{\vdash [\iota \hookrightarrow_{\mathcal{B}} (p, q, \vec{n}) * \lightning(\varepsilon)]\ e\ [P]}$$

All these results together form an instance of the typeclass $\mathsf{distrImpl}(\mu_{\mathcal{B}(\frac{p}{q+1})})$. In particular, this means that we immediately obtain a correctness result for the implementation through Theorem 4.1, as well as the instance of the planner rule for the Bernoulli sampler below:

PRESAMPLE-BERNOULLI-PLANNER

$$\frac{\begin{array}{c} e \notin \mathit{Val} \qquad p \leq q+1 \qquad \forall \vec{m}\ .\ |s(\vec{m})| \leq L \\ \forall \vec{m}\ .\ m \in s(\vec{m}) \to \mu_{\mathcal{B}(p/(q+1))}(m) > 0 \\ \vdash \left[\exists \vec{j}.\, \iota \hookrightarrow_{\mathcal{B}} (p, q, \vec{n} \cdot \vec{j} \cdot s(\vec{n} \cdot \vec{j}))\right]\ e\ [P] \end{array}}{\vdash [\iota \hookrightarrow_{\mathcal{B}} (p, q, \vec{n})]\ e\ [P]}$$

Application: Martingale betting on a biased coin. To illustrate how the sampler can be used in practice, we consider an example of a martingale gambling strategy. Consider a game in which, on every round, we can place a bet b on the result of a coin flip. If it is heads, we receive a total amount of $2b$, otherwise we lose our bet. The martingale strategy consists in starting with an initial bet b, and doubling it every round until the first time we observe heads. Assuming an unbounded pool for betting and unbounded number of rounds, this guarantees to terminate with probability 1, with a profit of b. The strategy works even if the coin is biased, as long as the probability of heads is non-zero.

We model this process as the program below, where we model the possibly biased coin as a Bernoulli distribution:

$$\begin{array}{l} \mathsf{martingale}\ w\ b\ \iota \triangleq \\ \quad \mathsf{let}\ b = \mathsf{bern}\ p\ q\ \iota\ \mathsf{in} \\ \quad \mathsf{if}\ b = 0\ \mathsf{then}\ \mathsf{martingale}\ (w - b)\ (2 * b)\ \iota\ \mathsf{else}\ \ w + b \end{array}$$

The parameter w represents the current earnings (possibly negative), while the parameter b represents the current bet. We can prove the following specification:

$$\forall w \in \mathbb{Z}, b \in \mathbb{N}.\ [\iota \hookrightarrow_{\mathcal{B}} (p, q, \epsilon)]\ \mathsf{martingale}\ w\ b\ [v.v = b + w]$$

While this program seems quite simple, note that in order for this specification to be valid, we also have to prove that the program is almost-surely terminating, so induction on the recursive call is not sound.

Our proof instead uses PRESAMPLE-BERNOULLI-PLANNER, which allows us to presample a list of Bernoulli samples that ends in 1, and then we can proceed to prove by induction on the list $\vec{n}$ that

$$\forall \vec{n} \in \mathsf{List}\ \{0,1\}, w \in \mathbb{Z}, b \in \mathbb{N}.\ [\iota \hookrightarrow_{\mathcal{B}} (p, q, \vec{n} \cdot [1])]\ \mathsf{martingale}\ w\ b\ [v.v = b + w]$$

In the base case, $\vec{n} = \epsilon$, so $\mathsf{bern}\ p\ q\ \iota$ will read 1 off ι and the program terminates immediately returning $w + b$. In the inductive case, we do a case distinction on the first element of the tape. If it is 1, we are in the same situation as before. Otherwise, we will consume the first element of the tape and make the recursive call, but now we will have an inductive hypothesis available, which we can use to conclude.

5.2 The Binomial Distribution

Using the Bernoulli distribution as a base, we can then construct the binomial distribution $\mathcal{B}in(\theta, n)$ with $0 \leq \theta \leq 1$ and $0 \leq n$, which is the distribution of the number of successes in a sequence of n samples from $\mathcal{B}(\theta)$. Recall that its mass function is given by the expression

$$\mathcal{B}in(\theta, n) \triangleq \lambda k. \binom{n}{k} \cdot \theta^k \cdot (1-\theta)^{n-k}$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the binomial coefficient (by convention, if $n < k$ or $k < 0$ then $\binom{n}{k} = 0$). The description of the binomial process can be turned into the following RandML program, where again we assume that the θ parameter is a rational number $p/(q+1)$ with $p \leq q+1$.

$$\begin{array}{l} \mathsf{binom}\ p\ q\ n\ \iota \triangleq \mathsf{if}\ n = 0\ \mathsf{then}\ 1 \\ \qquad\qquad\qquad\qquad \mathsf{else}\ \mathsf{bern}\ p\ q\ \iota + \mathsf{binom}\ p\ q\ (n-1)\ \iota \end{array}$$

We can prove that the program satisfies the appropriate instantiation of the HT-DISTR-EXP rule. The proof exploits the modularity of our approach by using the specification HT-BERNOULLI-EXP of the Bernoulli distribution, which allows to distribute the error credits appropriately. In particular, the proof does not need to inspect the concrete implementation of the Bernoulli sampler, and relies only on the fact that it implements the distribution typeclass for $\mu_{\mathcal{B}(\frac{p}{q+1})}$.

In order to implement the typeclass interface we also need to define a notion of tape for the binomial distribution. While the Bernoulli tapes can be encoded one-to-one into primitive uniform tapes, each binomial sample consists of multiple, but always the same fixed amount of Bernoulli samples. We can therefore encode the abstract binomial tapes in terms of abstract Bernoulli tapes. The translation predicate $\vec{b} \overset{p,q}{\mathcal{B}} \sim \overset{p,q,n}{\mathcal{B}in} \vec{k}$ is defined in Figure 6 and the predicate denoting ownership of an abstract binomial tape is defined below:

$$\begin{array}{l} \iota \hookrightarrow_{\mathcal{B}in} (p, q, n, \vec{k}) \triangleq \\ \quad \exists\, \vec{b} \in \mathsf{List}\ \{0,1\}\ .\ \vec{b} \overset{p,q}{\mathcal{B}} \sim \overset{p,q,n}{\mathcal{B}in} \vec{k} * \iota \hookrightarrow_{\mathcal{B}} (p, q, \vec{b}) \end{array}$$

$$\frac{}{\epsilon \;{}^{p,q}_{\mathcal{B}}\!\sim^{p,q,n}_{\mathcal{B}in}\; \epsilon}\ \textsc{binom-tl-nil}$$

$$\frac{\vec{b}_2 \;{}^{p,q}_{\mathcal{B}}\!\sim^{p,q,n}_{\mathcal{B}in}\; \vec{k}_2 \qquad \mathsf{length}(\vec{b}_1) = n \qquad \mathsf{sum}(\vec{b}_1) = k}{\vec{b}_1 \cdot \vec{b}_2 \;{}^{p,q}_{\mathcal{B}}\!\sim^{p,q,n}_{\mathcal{B}in}\; k :: \vec{k}_2}\ \textsc{binom-tl-cons}$$

Figure 6. Translation predicate for the Binomial distribution

Without showing the details, we prove that these tapes satisfy all the appropriate rules to implement the interface $\mathsf{distrImpl}(\mathcal{B}in(\frac{p}{q+1}, n))$.

5.3 The Geometric Distribution

The next distribution we consider is the geometric distribution. The distribution $\mathcal{G}(\theta)$ counts the number of failed $\mathcal{B}(\theta)$ trials before the first success. Its mass function is thus given by $\mathcal{G}(\theta) \triangleq \lambda k.(1-\theta)^k \cdot \theta$. We note that as opposed to the distributions we have seen so far, including the primitive uniform distribution, the geometric distribution does not have finite support. The implementation again considers only rational parameters. The program that describes the geometric process is:

$$\begin{aligned}\mathsf{geo}\ p\ q\ \iota \triangleq\ & \mathsf{let}\ b = \mathsf{bern}\ p\ q\ \iota\ \mathsf{in}\\ & \mathsf{if}\ b = 1\ \mathsf{then}\ 0\\ & \mathsf{else}\ 1 + \mathsf{geo}\ p\ q\ \iota\end{aligned}$$

This program is almost surely terminating, assuming $0 < p$, but it may have arbitrarily long executions. The proof of the instance of HT-DISTR-EXP for the geometric distribution relies on a principle known as *error induction* [Aguirre et al. 2024], which allows us to prove total correctness triples for recursive probabilistic functions.

Defining and encoding tapes for the geometric distribution in terms of the underlying Bernoulli distribution is challenging as well. Notice that intuitively, a geometric sample should correspond to a sequence of failed Bernoulli samples, terminated by a single successful Bernoulli sample. However, note that now not only are there multiple Bernoulli samples corresponding to each single geometric sample but, in fact, the amount varies, and potentially it can be unbounded.

With this in mind, we can define a predicate denoting ownership of a geometric tape as follows:

$$\iota \hookrightarrow_{\mathcal{G}} (p, q, \vec{k}) \triangleq \exists \vec{b} \in \mathsf{List}\,\{0,1\}.\ \vec{b}\ {}^{p,q}_{\mathcal{B}}\!\sim^{p,q}_{\mathcal{G}}\ \vec{k} * \iota \hookrightarrow_{\mathcal{B}} (p, q, \vec{b})$$

where the predicate $\vec{b}\ {}^{p,q}_{\mathcal{B}}\!\sim^{p,q}_{\mathcal{G}}\ \vec{k}$, defined in Figure 7, captures the encoding from a geometric tape $\vec{k}$ to a Bernoulli tape $\vec{b}$. The proof of the PRESAMPLE-DISTR-EXP for the geometric distribution below relies follows from applying the PRESAMPLE-BERNOULLI-EXP and using credit arithmetic to choose the

$$\frac{}{\epsilon \;{}^{p,q}_{\mathcal{B}}\!\sim^{p,q}_{\mathcal{G}}\; \epsilon}\ \textsc{geometric-tl-nil}$$

$$\frac{\vec{b}\ {}^{p,q}_{\mathcal{B}}\!\sim^{p,q}_{\mathcal{G}}\ \vec{k} \qquad \mathsf{length}(\vec{z}) = n \qquad \forall i, \vec{z}_i = 0}{\vec{z} \cdot (1 :: \vec{b})\ {}^{p,q}_{\mathcal{B}}\!\sim^{p,q}_{\mathcal{G}}\ n :: \vec{k}}\ \textsc{geometric-tl-cons}$$

Figure 7. Translation predicate for the Geometric distribution

appropriate way to distribute credits.

PRESAMPLE-GEO-EXP

$$\frac{\forall v.\mathcal{E}_2(v) \geq 0 \qquad p \leq q+1 \qquad \displaystyle\sum_{k=0}^{\infty} \left(\frac{p}{q+1}\right) \cdot \left(\frac{q-p+1}{q+1}\right)^k \cdot \mathcal{E}_2(k) = \varepsilon_1 \qquad \forall k \in \{0,\dots,n\} \vdash \left[\iota \hookrightarrow_{\mathcal{G}} (p, q, \vec{n} \cdot [k]) * \lightning(\mathcal{E}_2(k))\right] e\ [P]}{\vdash \left[\iota \hookrightarrow_{\mathcal{G}} (p, q, \vec{n}) * \lightning(\varepsilon_1)\right] e\ [P]}$$

5.4 The Negative Binomial Distribution

The negative binomial distribution $\mathcal{NB}(\theta, n)$ can be understood as the sum of n independent geometric samples from $\mathcal{G}(\theta)$ or, equivalently, as the number of failed Bernoulli samples from $\mathcal{B}(\theta)$ we observe before observing the n-th successful one. Its mass function is:

$$\mu_{\mathcal{NB}(\theta,n)} \triangleq \lambda k. \binom{k+n-1}{k} \cdot \theta^n \cdot (1-\theta)^k$$

We use the process described above to implement a negative binomial sampler assuming a rational parameter $p/(q+1)$:

$$\begin{aligned}\mathsf{nbin}\ \ p\ q\ n\ \iota \triangleq\ & \mathsf{if}\ n = 0\ \mathsf{then}\ 0\\ & \mathsf{else\ let}\ b = \mathsf{bern}\ p\ q\ \iota\ \mathsf{in}\\ & \quad \mathsf{if}\ b = 0\ \mathsf{then}\ 1 + \mathsf{nbin}\ \ p\ q\ n\\ & \quad \mathsf{else\ nbin}\ \ p\ q\ (n-1)\end{aligned}$$

As was the case with the geometric distribution, the negative binomial also has infinite support. Note that in the recursive call the last argument decreases only with probability $\mathcal{B}(p/(q+1))$, which means that we may have arbitrarily long executions. Nonetheless, the program is still almost surely terminating and Eris is expressive enough to prove the instantiation of HT-DISTR-EXP.

By generalizing the ideas for the geometric distribution, we can define a notion of tape $\iota \hookrightarrow_{\mathcal{NB}} (p, q, r, \vec{n})$ for the negative binomial distribution, by a translation into Bernoulli tapes, for which we can prove a presampling rule.

5.5 The Beta-Binomial Distribution

The beta-binomial distribution $\mathcal{BB}(n, a, b)$ corresponds to a sum of n Bernoulli samples where the parameter at each sample is itself randomized and drawn from a beta distribution, whose parameter depends on the previous samples.

It can also be understood through an urn model, due to Pólya. Suppose we start with an urn containing a red balls and b black balls, and we repeat the following process n times: we draw a ball uniformly at random, we note its color, and we put it back, together with a duplicate ball of the same color. The total number of red balls observed at the end of the process will then distribute according to $\mathcal{BB}(n,a,b)$. Its mass function is given by

$$\mu_{\mathcal{BB}(n,a,b)} \triangleq \lambda k. \binom{n}{k} \cdot \frac{\mathbf{B}(k+a, n-k+b)}{\mathbf{B}(a,b)}$$

where $N \in \mathbb{N}, a, b \in \mathbb{R}^{>0}$ and $\mathbf{B}(x,y) \triangleq \Gamma(x)\cdot\Gamma(y)/\Gamma(x+y)$ denotes the beta function, defined in terms of the gamma function Γ. In our case, where a, b are positive integers, we can use the well-known identity $\Gamma(n+1) = n!$, for $n \in \mathbb{N}$.

The urn model immediately suggests the following implementation of a beta-binomial sampler in RandML:

```
betabin N a b ⟨?⟩ ≜
  if N = 0 then 0
  else let x = bern a (a + b − 1) ⟨?⟩ in
    if x = 0 then
      betabin (N − 1) a (b + 1) ⟨?⟩
    else 1 + betabin (N − 1) (a + 1) b ⟨?⟩
```

The challenge here is to figure out what to put in the place of the placeholders $\langle ? \rangle$. In previous distributions we used plain tape labels because ultimately there was a one-to-one correspondence between the abstract tape of the target distribution, and a primitive uniform tape from which all randomness came from. However, here we make successive calls to Bernoulli distributions with changing parameters, which are obtained from different underlying uniform distributions. Furthermore, the parameters of those distributions depend on previous values sampled during the current call to betabin. Thus we need a more complex notion of location if we are to develop a tape predicate for this distribution.

To make what we need clearer, we need to know what parameters can appear in front of bern during a call to betabin $N\ a\ b$. It is clear that the first Bernoulli trial is of parameter $p = \frac{a}{a+b}$, then each subsequent trial is done after incrementing either a or b and the total number of increments is N (note that the values after the N^{th} increment are not used for a trial). Thus we conclude that the set of possible parameters is:

$$\left\{ \frac{a+i}{(a+i)+(b+j)} \;\middle|\; i,j \in \mathbb{N} \wedge 0 \le i+j < N \right\}$$

We therefore need a location for each pair of natural numbers i, j such that $i + j < N$. If we think of them as coordinates over a 2D plane, this describes a set of indices that arrange themselves in the shape of a triangle. As such we need to hold onto a tape with the proper parameter for each of these indices. To better match the behavior of the program which has as main decreasing argument the number of remaining balls to be drawn, we choose to rearrange the indices as a pair (k, i) of the number k of balls drawn so far and the number i of those balls that were red under the the constraints that $0 \le i \le k < N$. To keep track of the values of the tapes in a way that is more amenable to reasoning about them, we use the data structure inductively generated by the following constructors:

$$\frac{}{\epsilon_t : \text{triangle } A\ 0}\ \textsc{trig-nil}$$

$$\frac{\tau : \text{triangle } A\ N \qquad l : \text{vector } A\ (N+1)}{\tau \odot l : \text{triangle } A\ (N+1)}\ \textsc{trig-snoc}$$

The elements of triangle $A\ N$ are discrete triangles of elements of A of height N. There is always an empty triangle of height 0 for any set A and one can get a triangle of height $N+1$ by gluing a column of height $N+1$ at then end of a triangle of height N. For τ : triangle $A\ N$, we will use $\tau_{k,i}$ to denote the i-th element of the k-th vector in τ, where $0 \le k < N$ and $0 \le i < k$.

We will use a τ : triangle (List $\{0,1\}$) N to hold the values contained in Bernoulli tapes for each pair of indices needed for a beta-binomial distribution. We do this by taking a triangle τ in which the list in $\tau_{k,i}$ corresponds to a Bernoulli tape of parameter $p = \dfrac{a+i}{a+b+k}$.

Besides the projections induced by the inductive definition, we consider two other projection functions:

$$\text{bottom-trig} : \text{triangle } A\ (N+1) \to \text{triangle } A\ N$$
$$\text{right-trig} : \text{triangle } A\ (N+1) \to \text{triangle } A\ N$$

Intuitively, bottom-trig τ is the result of discarding the first element from every vector in τ, and right-trig is the result of discarding the last element from every vector in τ; in both cases we also discard the now empty first vector. This is depicted in Figure 8.

To talk about locations we then take Δ : triangle $Loc\ N$ and we form the following predicate stating that each tape in τ is owned at the corresponding location in Δ:

$$\text{ownTrig}(a,b,N,\Delta,\tau) \triangleq \mathop{\Large *}_{0\le i\le k<N} \Delta_{k,i} \hookrightarrow_{\mathcal{B}} (a+i, a+b+k, \tau_{k,i})$$

Next, we need to define the translation relation between a beta-binomial tape and a triangle of Bernoulli tapes. Note that every call to betabin $a\ b\ N$ consumes a total of N Bernoulli samples. We define an auxiliary function:

$$\text{hsup} : \text{triangle (List } \{0,1\})\ N \times \text{vector } \{0,1\}\ N \to \text{triangle (List } \{0,1\})\ N$$

Intuitively hsup(τ, l) starts at $\tau_{0,0}$ and traverses τ depending on the elements of l. An example execution of hsup is displayed in Figure 9. At every step, the traversal will be at an element $\tau_{k,i}$, and will consider the list $b :: \vec{b}$. Then it appends

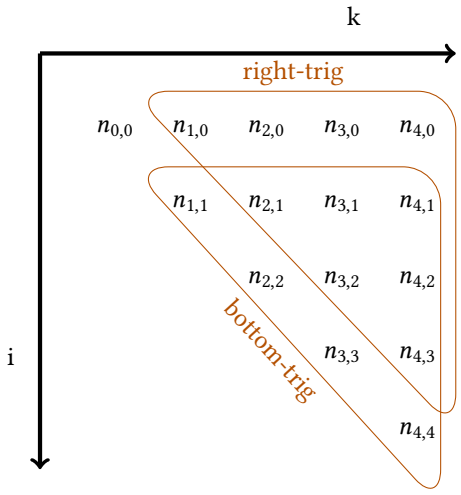

Figure 8. Graphic representation of an element of triangle $\mathbb{N}\ 5$ and its projections

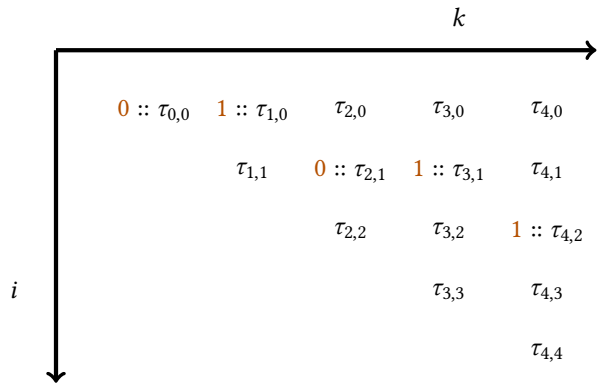

Figure 9. Graphic representation of hsup $\tau\ [0, 1, 0, 1, 1]$

b to $\tau_{k,i}$, and continues the traversal with the list $\vec{b}$ from $\tau_{k+1,i}$ if $b = 0$, or from $\tau_{k+1,i+1}$ if $b = 1$. This mimics a run of b in which the sequence of colors drawn corresponds to l, the 1s representing red balls and the 0s black balls by placing the corresponding draws on the correct tapes. Finally we define encode : List (vector $\{0,1\}\ N$) $\to$ triangle (List $\{0,1\}$) N where encode(l) successively applies hsup with the elements of l starting with a triangle containing only empty lists.

We can now define the translation predicate

$$\tau\ {}_{\mathcal{B}}{\sim}^{a,b,N}_{\mathcal{B}\mathcal{B}}\ \vec{n} \triangleq \exists\, \vec{m}\ .\ \tau = \text{encode}(\vec{m}) * \vec{n} = (l \mapsto \sum l)\ \langle\$\rangle\ \vec{m}$$

which states that τ is the encoding of a series of runs corresponding to calls to betabin in such a way that they result in the successive values in $\vec{n}$. Now, we can finally define the tape predicate:

$$\Delta \hookrightarrow_{\mathcal{B}\mathcal{B}}\ (a, b, N, \vec{n}) \triangleq$$
$$\exists\, \tau\ .\ \text{ownTrig}(a, b, N, \Delta, \tau) * \tau\ {}_{\mathcal{B}}{\sim}^{a,b,N}_{\mathcal{B}\mathcal{B}}\ \vec{n}$$

From here, induction over N allows us to prove the PRESAMPLE-BETA-EXP rule in Figure 10.

Remark 5.1. *While we omit the details about the planner rule for this distribution, we call attention to the benefit we get from §4.3 and the general derivation of the planner rule. In principle, it would be possible to derive a planner rule individually for all previous distributions through an intricate translation to and from the planner rules for the underlying distributions used to formulate them. However, in this case, this approach would be overly complex because we would need to keep track of all randomness presampled to multiple tapes and the interdependencies between them. Using the generic planner rule, we can derive a planner rule for the beta-binomial directly from its presampling rule and treating the tape predicates abstractly.*

The last rule we will prove is the load rule for beta-binomial tapes. In this case, it is worth spelling out the concrete definition of isAbsLoc, which ties an abstract location Δ with an expression in RandML:

$$\text{isAbsLoc}(a, b, N, \Delta, \delta) \triangleq [0 \le i \le k < N]\ \delta\ k\ i\ \left[\iota\ .\ \iota = \Delta_{k,i}\right]$$

which simply states that for valid indices the abstract location δ should always return the locations held in Δ. When reading an element off a beta-binomial tape, we will read the head element from a series of Bernoulli tapes in τ, starting at the apex of the triangle, i.e., $\tau_{0,0}$. Notice that depending on the result of this Bernoulli trial, an entire set of parameters can never be encountered in the rest of the call to betabin. If the value sampled is a 1 then we can never again have only a red balls and if it is a 0, we will never draw from a box containing $a + n - 1$ red balls. These cases correspond to the two projections bottom-trig and right-trig respectively, which can be reflected into RandML as:

$$\text{bottom-loc}\ \delta \triangleq \lambda\, k\, i\ .\ \delta\ (k+1)\ i$$
$$\text{right-loc}\ \delta \triangleq \lambda\, k\, i\ .\ \delta\ (k+1)\ (i+1)$$

These two programs are tied to bottom-trig and right-trig by their specifications shown in Figure 10.

We can now finally complete our definition of betabin:

```
betabin N a b δ ≜
  if N = 0 then 0
  else let x = bern a (a + b − 1) (δ 0 0) in
    if x = 0 then
      betabin (N − 1) a (b + 1) (right-loc δ)
    else 1 + betabin (N − 1) (a + 1) b (bottom-loc δ)
```

Finally, leveraging the machinery we have put into place, we can prove the LOAD-BETA-TAPE rule in Figure 10 by induction over N. With this, we prove that the sampler adheres to the distrImpl typeclass interface.

6 Related Work

There is a wide variety of techniques used to prove correctness of probabilistic programs. Here we focus on program

$$\frac{e \notin \mathit{Val} \qquad a, b > 0 \qquad \sum_{0 \le k \le N} \mu_{\mathcal{BB}(a,b,N)}(k) \cdot \mathcal{E}_2(k) = \varepsilon_1 \qquad \vdash \forall k \in \{0, \ldots, N\}.\ [\Delta \hookrightarrow_{\mathcal{BB}} (a, b, N, \vec{n} \cdot [k]) * \lightning(\mathcal{E}_2(k))]\ e\ [P]}{\vdash\ [\Delta \hookrightarrow_{\mathcal{BB}} (a, b, N, \vec{n}) * \lightning(\varepsilon_1)]\ e\ [P]}\ \textsc{presample-beta-exp}$$

$$\frac{}{\vdash\ [\mathsf{isAbsLoc}(a, b, N+1, \Delta, \delta)]\ \textsf{bottom-loc}\ \delta\ [\delta'\ .\ \mathsf{isAbsLoc}(a, b+1, N, \textsf{bottom-trig}(\Delta), \delta')]}\ \textsc{bottom-trig-loc}$$

$$\frac{}{\vdash\ [\mathsf{isAbsLoc}(a, b, N+1, \Delta, \delta)]\ \textsf{right-loc}\ \delta\ [\delta'\ .\ \mathsf{isAbsLoc}(a+1, b, N, \textsf{right-trig}(\Delta), \delta')]}\ \textsc{right-trig-loc}$$

$$\frac{a, b > 0 \qquad \mathsf{isAbsLoc}(a, b, N, \Delta, \delta)}{\vdash\ [\Delta \hookrightarrow_{\mathcal{BB}} (a, b, N, n :: \vec{n})]\ \textsf{betabin}\ a\ b\ N\ \delta\ [k\ .\ k = n * \Delta \hookrightarrow_{\mathcal{BB}} (a, b, N, \vec{n})]}\ \textsc{load-beta-tape}$$

Figure 10. Specification of the beta-binomial sampler

logics, which can be further divided depending on the underlying class of logical assertions that the logic considers. We remark that our approach, to the best of our knowledge, is the first which simultaneously (1) allows one to prove correctness of implementations of complex samplers, (2) provides specifications that are usable by clients, and (3) can be formalized within a mechanized program logic.

The work on expectation transformers [McIver and Morgan 2005] originates from the idea of using as assertions predicates whose truth value is a real number (known as expectations). They can be used to reason about properties such as expected runtime [Kaminski et al. 2016] or correctness of pointer programs [Batz et al. 2019], and can be semi-automated [Schröer et al. 2023], but they have not been mechanized. Their work is closely related to Eris in the sense that the rule for primitive sampling also requires the pre-expectation to be the expected value of the post-expectations. The logic is not higher-order, so quantification over expectations is not expressible in their logic, but we believe specifications similar to ours could be proven by exploiting quantification at the meta-level.

Other logics consider assertions that represent sets of distributions over states or heaps. One example is Ellora [Barthe et al. 2018], a Hoare-style logic for probabilistic programs which has a prototype implementation in Easycrypt [Barthe et al. 2014]. The class of assertions is presented from a semantic point of view and the logic has a relative completeness result, so in principle it could be used to prove correctness of samplers. However, to our knowledge, no significant effort in that direction has been carried out.

Probabilistic Concurrent Outcome Logic (PCOL) [Zilberstein et al. 2025] is an expressive logic to reason about concurrent probabilistic programs. Its assertion language includes "distribute-as" assertions, which can express that a program variable distributes as a concrete distribution (Bernoulli, geometric, uniform). However, reasoning about general distributions would require assertions that express that a variable distributes as an arbitrary distribution with a user-chosen pdf. or cdf. The authors mention future plans to mechanize PCOL, but a mechanization does not currently exist.

Lilac [Li et al. 2023] is a program logic for probabilistic programs inspired by separation logic, with the twist that the separating conjunction is used to represent probabilistic independence. As opposed to Eris, Lilac considers the more complex setting of continuous sampling. Lilac supports "distribute-as" assertions for arbitrary distributions, and as one of its case studies proves correctness of a weighted sampling algorithm. Lilac supports only bounded loops, so samplers for distributions such as the geometric or the negative binomial, as well as rejection samplers in general would not be expressible in the system. The logic does not currently have a mechanization.

Bluebell [Bao et al. 2025] is a recent logic for probabilistic programs that combines ideas from different sources, including separations logics for independence and coupling-based logics, through the use of a joint conditioning modality. By properly instantiating this modality, one can recover the reasoning principles of many of the previously mentioned logics, which equips Bluebell with a lot of expressivity. However, Bluebell has a complex semantic model, and does not support unbounded loops, and has not been mechanized.

Other logics consider plain assertions (i.e., sets of states) and instead introduce quantitative aspects into the logic through other means. Closely related to Eris, approximate Hoare logic (aHL) [Barthe et al. 2016] has as judgments Hoare triples indexed with a real number which corresponds to the probability that the postcondition will fail to hold. Error credits can be understood as an resourceful representation of these annotations. However, aHL does not support making these indices dependent on the input or output state. One can

prove correctness of a sampler by quantifying at the meta level over the outputs, and proving an analogous statement to our naïve specification in §2, but this will not be usable for clients for the same reasons as ours was not.

Logics based on probabilistic couplings [Barthe et al. 2009, 2012; Gregersen et al. 2024] adapt techniques from the Markov chain literature to reason about relational properties of probabilistic programs, in particular equivalence, statistical dominance or differential privacy. It is possible to express correctness of a sampler as a relational judgment, e.g. by stating equivalence of an implemented sampler for a target distribution and a primitive sampler for the same distribution. For example, rejection samplers can be proven correct in this manner [Avanzini et al. 2025; Haselwarter et al. 2025]. However, proving these specifications would in general require adding primitive samplers for complex distributions to the language, and ad-hoc rules to the logic to reason about them.

Staton et al. [2018] propose a semantic method to verify a sampler for a beta-bernoulli distribution. They define an abstract module for the sampler operations, and then proposing two implementations: one relying on a Polya urn model, and one directly sampling from a beta distributions. The two can be proven contextually equivalent using an equational theory for probabilistic programs.

7 Conclusions

In this work we have shown a general methodology to implement and specify samplers for probability distributions starting from only a uniform sampler. Through the use of a common abstract distribution interface, we can separate the proof of correctness for individual sampler, and then prove properties about their clients that are independent from the concrete implementation of the sampler. The abstract interface also allows us to obtain correctness proofs for the samplers as well as to derive novel reasoning principles for all samplers at once. Although we only consider discrete distributions in this paper, we believe that our methods can be generalized to reasoning about correctness of samplers that either draw randomness from a primitive distribution that is continuous (e.g., a uniform over the unit interval), or that themselves implement continuous distributions.

Acknowledgments

This work was supported in part by a Villum Investigator grant, no. 25804, Center for Basic Research in Program Verification (CPV), from the VILLUM Foundation, and the European Union (ERC, CHORDS, 101096090). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

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. doi:10.1145/3674635

Thosten Altenkirch, James Chapman, and Tarmo Uustalu. 2015. Monads need not be endofunctors. *Logical Methods in Computer Science* Volume 11, Issue 1 (March 2015), 928. doi:10.2168/LMCS-11(1:3)2015

Martin Avanzini, Gilles Barthe, Davide Davoli, and Benjamin Grégoire. 2025. A Quantitative Probabilistic Relational Hoare Logic. *Proceedings of the ACM on Programming Languages* 9, POPL (Jan. 2025), 1167–1195. doi:10.1145/3704876

Jialu Bao, Emanuele D'Osualdo, and Azadeh Farzan. 2025. Bluebell: An Alliance of Relational Lifting and Independence for Probabilistic Reasoning. *Proceedings of the ACM on Programming Languages* 9, POPL (Jan. 2025), 1719–1749. doi:10.1145/3704894

Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. 2014. *EasyCrypt: A Tutorial.* Springer International Publishing, Cham, 146–166. doi:10.1007/978-3-319-10082-1_6

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*, Amal Ahmed (Ed.). Vol. 10801. Springer International Publishing, Cham, 117–144. doi:10.1007/978-3-319-89884-1_5 Series Title: Lecture Notes in Computer Science.

Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanesco, and Pierre-Yves Strub. 2015. Relational Reasoning via Probabilistic Coupling. In *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings.* doi:10.1007/978-3-662-48899-7_27

Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. A Program Logic for Union Bounds. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 55)*, Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 107:1–107:15. doi:10.4230/LIPIcs.ICALP.2016.107

Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. *SIGPLAN Not.* 44, 1 (Jan 2009), 90–101. doi:10.1145/1594834.1480894

Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. 2012. Probabilistic relational reasoning for differential privacy. *SIGPLAN Not.* 47, 1 (Jan 2012), 97–110. doi:10.1145/2103621.2103670

Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Thomas Noll. 2019. Quantitative separation logic: a logic for reasoning about probabilistic pointer programs. *Proc. ACM Program. Lang.* 3, POPL, Article 34 (Jan 2019), 29 pages. doi:10.1145/3290347

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

Ugo Dal Lago, Sara Zuppiroli, and Maurizio Gabbrielli. 2014. Probabilistic Recursion Theory and Implicit Computational Complexity. *Scientific Annals of Computer Science* 24, 2 (2014), 177–216. doi:10.7561/SACS.2014.2.177

Shafi Goldwasser and Silvio Micali. 1982. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing (STOC '82)*. Association for Computing Machinery, New York, NY, USA, 365–377. doi:10.1145/800070.802212

Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2024. Asynchronous Probabilistic

Couplings in Higher-Order Separation Logic. *Proc. ACM Program. Lang.* 8, POPL (2024), 753–784. doi:10.1145/3632868

Philipp G. Haselwarter, Kwing Hei Li, Alejandro Aguirre, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2025. Approximate Relational Reasoning for Higher-Order Probabilistic Programs. *Proceedings of the ACM on Programming Languages* 9, POPL (Jan. 2025), 1196–1226. doi:10.1145/3704877

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, POPL 2015, Mumbai, India, January 15-17, 2015*. 637–650. doi:10.1145/2676726.2676980

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run–Times of Probabilistic Programs. In *Programming Languages and Systems*, Peter Thiemann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 364–389.

John M. Li, Amal Ahmed, and Steven Holtzen. 2023. Lilac: A Modal Separation Logic for Conditional Probability. *Proc. ACM Program. Lang.* 7, PLDI, Article 112 (Jun 2023), 24 pages. doi:10.1145/3591226

Rupak Majumdar and V.R. Sathiyanarayana. 2025. Sound and Complete Proof Rules for Probabilistic Termination. *Proceedings of the ACM on Programming Languages* 9, POPL (Jan. 2025), 1871–1902. doi:10.1145/3704899

Virgil Marionneau, Félix Sassus Bourda, Alejandro Aguirre, and Lars Birkedal. 2025. Artifact for Modular Specifications and Implementations of Random Samplers in Higher-Order Separation Logic. doi:10.5281/zenodo.17800602

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer.

Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press, Cambridge. doi:10.1017/CBO9780511814075

Philipp Schröer, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2023. A Deductive Verification Infrastructure for Probabilistic Programs. *Reproduction Package for Article 'A Deductive Verification Infrastructure for Probabilistic Programs'* 7, OOPSLA2 (Oct. 2023), 294:2052–294:2082. doi:10.1145/3622870

Sam Staton, Dario Stein, Hongseok Yang, Nathanael L. Ackerman, Cameron E. Freer, and Daniel M. Roy. 2018. The Beta-Bernoulli process and algebraic effects. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 141:1–141:15. doi:10.4230/LIPIcs.ICALP.2018.141 ISSN: 1868-8969.

The Rocq Development Team. 2025. *The Rocq Prover*. doi:10.5281/zenodo.15149629

Noam Zilberstein, Alexandra Silva, and Joseph Tassarotti. 2025. Probabilistic Concurrent Reasoning in Outcome Logic: Independence, Conditioning, and Invariants. doi:10.48550/arXiv.2411.11662 arXiv:2411.11662 [cs].

Received 2025-09-12; accepted 2025-11-13