Bayesian Separation Logic

A Logical Foundation and Axiomatic Semantics for Probabilistic Programming

SHING HIN HO, Imperial College London, UK
NICOLAS WU, Imperial College London, UK
AZALEA RAAD, Imperial College London, UK

Bayesian probabilistic programming languages (BPPLs) let users denote statistical models as code while the interpreter infers the posterior distribution. The semantics of BPPLs are usually mathematically complex and unable to reason about desirable properties such as expected values and independence of random variables. To reason about these properties in a non-Bayesian setting, probabilistic separation logics such as PSL and Lilac interpret separating conjunction as probabilistic independence of random variables. However, no existing separation logic can handle Bayesian updating, which is the key distinguishing feature of BPPLs.

To close this gap, we introduce Bayesian separation logic (BASL), a probabilistic separation logic that gives semantics to BPPL. We prove an internal version of Bayes' theorem using a result in measure theory known as the Rokhlin-Simmons disintegration theorem. Consequently, BASL can model probabilistic programming concepts such as Bayesian updating, unnormalised distribution, conditional distribution, soft constraint, conjugate prior and improper prior while maintaining modularity via the frame rule. The model of BASL is based on a novel instantiation of Kripke resource monoid via σ-finite measure spaces over the Hilbert cube, and the semantics of Hoare triple is compatible with an existing denotational semantics of BPPL based on the category of s-finite kernels. Using BASL, we then prove properties of statistical models such as the expected value of Bayesian coin flip, correlation of random variables in the collider Bayesian network, the posterior distributions of the burglar alarm model, a parameter estimation algorithm, and the Gaussian mixture model.

CCS Concepts: • **Theory of computation** → **Separation logic**; **Probabilistic computation**.

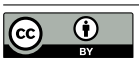
Additional Key Words and Phrases: Probabilistic programming, Bayesian inference, Axiomatic semantics

ACM Reference Format:
Shing Hin Ho, Nicolas Wu, and Azalea Raad. 2026. Bayesian Separation Logic: A Logical Foundation and Axiomatic Semantics for Probabilistic Programming. *Proc. ACM Program. Lang.* 10, POPL, Article 54 (January 2026), 29 pages. https://doi.org/10.1145/3776696

1 Introduction

Statistical techniques have become increasingly prevalent with widespread applications across a multitude of domains from computer vision and data science to computational biology and social science. As a result, there is a pressing need for developing secure and explainable statistical models. A way of ensuring correctness of such systems is with *formal methods*, a collection of techniques that allows computer scientists to prove properties of algorithms/programming languages by modelling the objects of interest mathematically. In fact, researchers are increasingly applying formal methods to probabilistic/machine learning algorithms in order to provide correctness guarantees on desirable statistical properties [Cruz and Shoukry 2022; Slusarz et al. 2022]. We are interested in formal and

Authors' Contact Information: Shing Hin Ho, Imperial College London, London, UK, shinghin.ho21@imperial.ac.uk; Nicolas Wu, Imperial College London, London, UK, n.wu@imperial.ac.uk; Azalea Raad, Imperial College London, London, UK, azalea.raad@imperial.ac.uk.



ACM 2475-1421/2026/1-ART54
https://doi.org/10.1145/3776696

compositional reasoning techniques for *Bayesian probabilistic programming languages* – a class of statistical programming languages that implement algorithms for *Bayesian inference*.

Bayesian inference is a statistical technique that allows users to infer unknown distributions using a statistical model and Bayes' theorem. It has a wide range of applications ranging from medicine [Muehlemann et al. 2023] to computer vision [Geman and Geman 1984]. For instance, problems such as clustering and regression can be solved by reframing them as Bayesian inference problems. Computationally, *probabilistic programming* is the programming paradigm that implements Bayesian inference by using general inference algorithms such as Hamiltonian Monte Carlo and Gibbs sampling [Brooks et al. 2011], while providing a simple interface for users to specify their statistical models [van de Meent et al. 2021]. *Bayesian probabilistic programming languages* (BPPLs) such as Stan [Carpenter et al. 2017], Anglican [Tolpin et al. 2016] and Gen [Cusumano-Towner et al. 2019] are implementations of the programming paradigm and have specialised constructs for users to easily express statistical models.

Goal. Our goal is to develop a logical foundation that allows us to reason about statistical properties of BPPL programs (e.g. independence, expected value, correlation) in a *compositional* manner. To achieve this, we use *separation logic* [Reynolds 2002], a logical system designed to allow compositional reasoning of computational resources, which in our case, are the random variables produced by the probabilistic program. In particular, we develop *Bayesian separation logic* (BASL, pronounced 'basil'), a logical system for Bayesian reasoning based on probabilistic separation logic. From BASL, we derive the first Hoare logic for BPPLs, which then allows us to prove the correctness/properties of statistical models such as the *Bayesian coin flip* model, the *collider network*, a *parameter estimation algorithm*, and a *Gaussian-mixture-based clustering model*.

Bayesian Conditioning. Unlike *randomised* programming languages, which are languages with a sampling construct, **sample**, implemented via a pseudo-random number generator (e.g. in C and Python), a BPPL has an additional *conditioning* construct, **observe**[1], that allows users to express *conditional probability*. For example, consider an experiment where we toss a fair die X, then condition on the event $X > 4$. This can be expressed by the BPPL program in Figure 1, which computes the distribution of X given that $X > 4$:

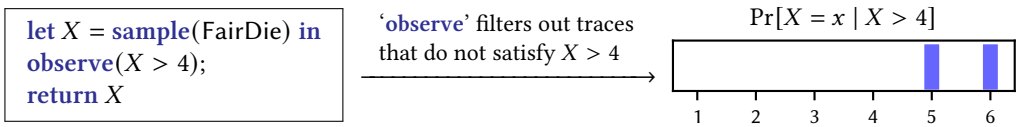


Fig. 1. Conditioning as the computational effect **observe**

Probabilistic Separation Logics. The reasoning principles for *randomised* languages have been studied in *probabilistic separation logics* [Bao et al. 2021a,b; Barthe et al. 2019; Li et al. 2023; Tassarotti and Harper 2019] through a set of axiomatic rules for deriving *Hoare triples*. A Hoare triple, $\{P\}\ M\ \{X.Q\}$, consists of four components: the program M, the precondition P, the return variable X and the postcondition Q. A Hoare triple $\{P\}\ M\ \{X.Q\}$ is *valid* when executing the program M assuming P produces a variable X and the proposition Q holds. For instance, consider the following Hoare triple (using the notation of Lilac [Li et al. 2023]):

$$\{\top\}\ \textbf{sample}(\text{FairDie})\ \{X.X \sim \text{Unif}(\{1, \ldots, 6\})\}$$

[1]For readers familiar with probabilistic programming, **observe** is implemented using the soft-constraint construct **score**.

The triple above describes the pre/postcondition for executing **sample**(FairDie) for rolling a fair die. Assuming a precondition with no knowledge of variables ($\top$), executing **sample**(FairDie) yields a random variable X that has a uniform distribution on $\{1, ..., 6\}$. Moreover, since the underlying logic of the Hoare triple is based on probabilistic *separation* logic, we can freely add independent random variables in our pre/postconditions via the *probabilistic separating conjunction* '$*$'. For example, the Hoare triple below states that given a random variable Y with distribution $\mathbb{P}$, executing **sample**(FairDie) produces an X that is *probabilistically independent* of Y:

$$\{Y \sim \mathbb{P}\}\ \textbf{sample}(\mathsf{FairDie})\ \{X.X \sim \mathsf{Unif}(\{1, ..., 6\}) * Y \sim \mathbb{P}\}$$

This is useful due to the nature of probabilistic reasoning, where a key part is to determine which random variables are independent of each other. Apart from the property of *distribution* (propositions of the form $X \sim \mathbb{P}$), probabilistic separation logic can also reason about different probabilistic properties. For example, the logical entailment

$$X \sim \mathcal{N}(0,1) * Y \sim \mathcal{N}(0,1) \vdash (\mathbb{E}[X] = 0 * \mathbb{E}[Y] = 0) \wedge \mathrm{Cov}[X, Y] = 0$$

states that if X and Y are independent, normally-distributed random variables with mean 0 and standard deviation 1, then they both have *expected value*[2] zero ($\mathbb{E}[X] = 0 * \mathbb{E}[Y] = 0$) and X, Y are uncorrelated (since the covariance is zero $\mathrm{Cov}[X, Y] = 0$). Having statistical propositions and combining them using the separating conjunction '$*$' allows formal and convenient statistical reasoning in a formal setting.

Limitation of Existing Work: No Support for Bayesian Conditioning. While probabilistic separation logics are a rich field of study with plenty of variations, existing probabilistic separation logics cannot reason about the **observe** construct. For instance, following the intuition in Figure 1, we *should* ideally have a Hoare triple specification as follows:

$$\{X \sim \mathsf{Unif}(\{1, ..., 6\})\}\ \textbf{observe}(X > 4)\ \{X \sim \mathsf{Unif}(\{5, 6\})\}$$

This technique is called *Bayesian updating*/*Bayesian conditioning*, where we 'update' our distributions based on observations. BaSL is the *first* probabilistic separation logic that allows reasoning of Bayesian updating, and we do this by drawing an analogy between mutation of memory in standard separation logic. We achieve this by proving an *internal Bayes' theorem* (Theorem 4.11) using a result in measure theory known as the *Rokhlin-Simmons disintegration theorem*.

Limitation of Existing Work: Dependencies of Random Variables. Existing probabilistic separation logics such as Lilac and BlueBell can reason about dependent random variable via the *conditioning modality*. For instance, the proposition $x \leftarrow X \mid Y \sim \mathcal{N}(x, 1)$ means Y has distribution $\mathcal{N}(x, 1)$, conditioning on $X = x$ for some $x \in \mathbb{R}$. BaSL extends the conditioning modality to support conditional reasoning in the presence of Bayesian updating. Furthermore, the Hoare logic in BaSL supports *conditional sampling*. For example, the following Hoare triple is provable in BaSL:

$$\{X \sim \mathsf{Unif}(0,1)\}\ \textbf{sample}(\mathsf{Normal}(X, 1))\ \{Y.x \xleftarrow{\mathsf{Unif}(0,1)} X \mid Y \sim \mathcal{N}(x, 1)\}$$

It states that assuming $X \sim \mathsf{Unif}(0, 1)$, sampling a normal with mean X yields a random variable Y, which has a *conditional distribution* $Y \sim \mathcal{N}(x, 1)$ when $X = x$ for almost all $x \in [0, 1]$.

Proving Properties of the Bayesian Coin Flip Model. By supporting the above features, together with the compositionality afforded by the frame rule, BaSL can serve as a logical foundation for symbolic reasoning probabilistic programming. To demonstrate BaSL and its associated Hoare logic, we first consider a simple statistical problem known as *Bayesian coin flip*.

[2]The expected value $\mathbb{E}[X]$ of a random variable X is the average of the distribution of X.

$X \sim \mathsf{Unif}(0,1)$

$Flip_1 \sim \mathsf{Bern}(x) \quad \text{when } X = x$

$Flip_2 \sim \mathsf{Bern}(x) \quad \text{when } X = x$

observation: $Flip_1 = 1$

observation: $Flip_2 = 0$

Fig. 2. Bayesian solution of Problem 1

Precondition: $\top$

Postcondition: return value X has expected value $1/2$

```
let X = sample(Unif(0, 1)) in    // specify prior belief
let Flip_1 = sample(Bern(X)) in  // toss the coin
observe(Flip_1 = 1);             // assert it comes up heads
let Flip_2 = sample(Bern(X)) in  // toss again
observe(Flip_2 = 0);             // assert it comes up tails
return X                         // what is our belief now?
```

Fig. 3. BayesCoin

Problem 1. *We have a coin and we want to know if it is fair. We toss it once, it comes up heads. We toss it again, it comes up tails. Is it a fair coin?*

The problem, while simple, encapsulates the three main steps of the Bayesian method:

(1) We *assume* a *prior belief* (or *prior* in short) regarding the problem. In our case, we assume a prior regarding whether the coin is fair.
(2) We *observe* real-life data. In our case, we have two observations of coin flip.
(3) We *update* our belief based on observations. In our case, we update our belief based on *observations* using *Bayes' theorem*.

We defer the explanation of the statistical model to §2. For now, we describe the problem using statistical notation in Figure 2, and its programmatic counterpart in Figure 3. Figure 2 states that X has (prior) distribution $\mathsf{Unif}(0,1)$, $Flip_i$ has distribution $\mathsf{Bern}(x)$ when $X = x$ (the Bernoulli distribution), which is a distribution that returns 1 with x probability and 0 with $1 - x$ probability, and we have two observations. Using a BPPL, we can express the same model programmatically as shown in the BayesCoin program in Figure 3. Programmatically, the underlying interpreter for the language is performing *rejection sampling*, which, to a first approximation, executes the program many times and filters out traces that do *not* satisfy the required observations, i.e. when $Flip_1 \neq 1$ or $Flip_2 \neq 0$. Intuitively, since we observed a head and a tail, the expected value (average) of X should remain $1/2$. As we show in §2, using BASL we can encode and prove this specification easily (Figure 4), and our proof derivation *formally* describes how the distributions of the random variables evolve.

Application: Justifying Probabilistic Program Rewrites. It is well-known that probabilistic programming languages are computationally expensive to execute. For example, consider the two programs below (which we will explain in detail later in §3.7):

```
let X = sample(N(0, 1)) in
let Y = sample(Beta(m, n)) in
for I in [1, ..., N] do
  observe x_I from N(M, 1);
  observe c_I from Bern(W)
```

```
let X = sample(N(sum([x_1,...,x_N])/(N+1), 1)) in
let Y = sample(Beta(n + #(c = 1), m + #(c = 0))) in
return (X, Y)
```

The two programs have *equivalent* distributions and can be considered to be equal. However, the program on the left is computationally costly to sample from due to the existence of two conditioning (**observe-from**) constructs and a loop, while the one on the right is easy to sample from. As we show later in §3.7, using BASL we can justify *rewriting* the program on the left to the one on the right by showing that we can prove the *same* Hoare triple (with the same pre- and post-conditions) for both programs, establishing their equivalent distributions.

Contributions and Roadmap. Our contributions are as follows:

- We explain how BayesCoin can be specified in BaSL (§2).
- We prove properties of five standard, but non-trivial Bayesian statistical models such as Bayesian networks (§3.3, §3.4), parameter estimation algorithm (§3.2), and the Gaussian mixture model (§3.6). The models have non-trivial features such as conditional dependencies, soft constraint, conjugate prior and improper prior (§3.5), which no existing probabilistic separation logic can verify (§3).
- For the semantics of BaSL, we describe intuitively the existing resource-theoretic model of Lilac for modelling randomness (§4.1) and motivate the need for a measure-theoretic semantics to model randomness in a Bayesian setting (§4.2).
- We develop a novel *Kripke resource monoid* for modelling randomness that supports *Bayesian updating* by using σ-finite measure spaces over the Hilbert cube (§4.3).
- We show that the Kripke resource model [Galmiche et al. 2005] of BaSL is compatible with *partially affine separation logic* [Charguéraud 2020] (§4.3).
- We generalise Lilac's *disintegration modality* so that conditional reasoning can be performed in the presence of Bayesian updating (§4.4).
- We design new logical propositions to encode the concept of *likelihood* and *normalising constants*, two key concepts in Bayesian probability (§4.4).
- We prove an internal version of Bayes' theorem and show that the concept can be encoded as logical propositions in BaSL by combining the disintegration modality and the likelihood proposition (Theorem 4.11).
- We develop the *BaSL proof system* (a set of Hoare triples) and prove that it is *sound* with respect to our Kripke resource model of BaSL (§4.4).

Finally, we discuss related and future work and conclude (§5).

2 Overview

To give an intuition of how BaSL can be used to prove properties of statistical models, we demonstrate its proof rules via BayesCoin (Problem 1). Recall that $\mathsf{Bern}(0.5)$ represents a fair coin as it returns 1 and 0 with equal probability, while $\mathsf{Bern}(0.9)$ represents a biased coin that comes up heads 90% of the time. Hence, we model our belief about whether our coin is fair via a random variable X that takes a value in $[0, 1]$.

Since we do not have additional information about the coin (X), we assume $X \sim \mathsf{Unif}(0, 1)$ as our prior distribution and write the program described in Figure 3. To semantically deduce that the return value X has expected value $1/2$ (see the postcondition of Figure 3), we use BaSL to describe BayesCoin axiomatically via pre/postcondition style reasoning rules. We present the BaSL proof sketch of BayesCoin in Figure 4. We proceed with a detailed but informal explanation of our proof.

Lines 1-3. Initially, we have no random variables, as captured by the *trivial* precondition $\top_1$, where 1 in $\top_1$ is the current *normalising constant* (we explain this on Page 7). After executing line 2 (of Figure 4), the interpreter produces a random variable X with the desired distribution. To reason about this, we apply the H-Sample rule below at line 2, which states that sampling from a distribution $\mathbb{P}$ of type $\mathbb{R}$ returns a random variable X of type $\mathbb{R}$ distributed according to $\mathbb{P}$ (for line 2, $\mathbb{P}$ is instantiated to $\mathsf{Unif}(0, 1)$):

$$\vdash \{\top_1\}\ \mathbf{sample}(\mathbb{P})\ \{X : \mathbb{R}.\ X \sim \mathbb{P}\}\ \ \textsc{H-Sample}$$

To 'chain' the postcondition to the rest of the program, we apply the sequencing rule H-Let below, chaining the postcondition of the first program to the precondition of the second program, thus

```
1  {⊤₁}
2    let X = sample(Unif(0, 1)) in
3  {X ~ Unif(0, 1)}
4    let Flip₁ = sample(Bern(X)) in
5  {x ←Unif(0,1)— X | Flip₁ ~ Bern(x)}
6    observe(Flip₁ = 1);
7  {x ←Unif(0,1)— X | Flip₁ ~ ℓ=1 · Bern(x)}
8  {x ←Unif(0,1)— X | L(x)} // likelihood of X = x is x
9  {X ~ Beta(2, 1) ∗ NormConst}
10   {X ~ Beta(2, 1)}
11     let Flip₂ = sample(Bern(X)) in
12   {x ←Beta(2,1)— X | Flip₂ ~ Bern(x)}
13     observe(Flip₂ = 0);
14   {x ←Beta(2,1)— X | Flip₂ ~ ℓ=0 · Bern(x)}
15   {X ~ Beta(2, 2) ∗ NormConst}
   H-Frame
16 {X ~ Beta(2, 2) ∗ NormConst ∗ NormConst}
17 {X ~ Beta(2, 2) ∗ NormConst}
18   return X
19 {X ~ Beta(2, 2) ∗ NormConst}
20 {𝔼[X] = 1/2 ∗ NormConst}
```

Fig. 4. An axiomatic description of BayesCoin

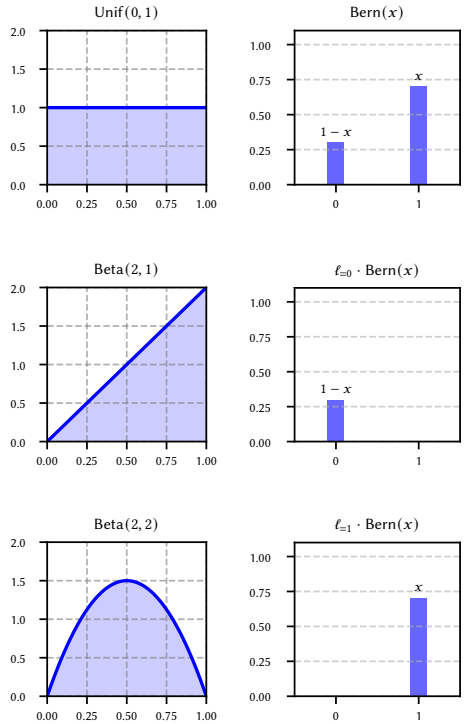


Fig. 5. Visualisation of distributions

obtaining $X \sim \mathsf{Unif}(0,1)$ on line 3.

$$\frac{\vdash \{P\}\ M\ \{X : \mathcal{A}.Q\} \qquad X : \mathcal{A} \vdash \{Q\}\ N\ \{Y : \mathcal{B}.R\}}{\vdash \{P\}\ \mathbf{let}\ X = M\ \mathbf{in}\ N\ \{Y : \mathcal{B}.R\}}\ \text{H-Let}$$

Lines 4-5. At line 4 we sample a Bernoulli variable according to our sampled X. While this line certainly typechecks, it is more challenging to verify: it is ambiguous statistically, as it semantically describes a *conditional distribution*. Given $X \sim \mathsf{Unif}(0,1)$, we write $Flip_1 | X = x \sim \mathsf{Bern}(x)$ to mean that conditioning on $X = x$ for almost all $x \in [0,1]$, the random variable $Flip_1$ has distribution $\mathsf{Bern}(x)$. To reason about this in BaSL, we introduce our conditional sampling axiom:

$$\vdash \{X \sim \mathbb{P}\}\ \mathbf{sample}(p(X))\ \{Y : \mathbb{R}.x \overset{\mathbb{P}}{\leftarrow} X \mid Y \sim p(x)\}\ \text{H-CondSample}$$

The arrow/bar notation $x \overset{\mathbb{P}}{\leftarrow} X \mid P$ is our novel *conditioning modality*, which assumes X has distribution $\mathbb{P}$ and binds it to a deterministic name x, while the proposition P on the right is a proposition on the conditioned space after conditioning $X = x$. Specifically, $x \overset{\mathbb{P}}{\leftarrow} X \mid Y \sim p(x)$ in the postcondition is read as follows: assuming $X \sim \mathbb{P}$, if we condition on $X = x$ for some deterministic x, then the sampled Y has distribution $p(x)$. Instantiating the axiom above, line 5 says $Flip_1$ has distribution $\mathsf{Bern}(x)$ whenever $X = x$.

Lines 6-8. Line 6 asserts $Flip_1 = 1$ and rejects all program traces where $Flip_1 = 0$. Semantically, this means we apply a *likelihood function* $\ell_{=1} : \{0,1\} \to [0,\infty)$ to $\mathsf{Bern}(x)$,

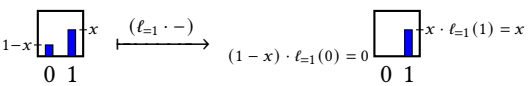


Fig. 6. Applying likelihood function

where $\ell_{=1}(1) \coloneqq 1$ and $\ell_{=1}(0) \coloneqq 0$. Specifically, since $\mathit{Flip}_1 \sim \mathsf{Bern}(x)$ (as depicted on the left side of Figure 6), applying the likelihood function $\ell_{=1}$ to $\mathsf{Bern}(x)$ *updates* the distribution to the one on the right side of Figure 6. Notice that this operation is done in the conditioned space conditioning on $X = x$. In light of this, we introduce the H-CondObserve axiom below for *conditional observation*:

$$\vdash \{x \overset{\pi}{\leftarrow} X \mid Y \sim p(x)\}\ \mathbf{observe}(P(Y))\ \{x \overset{\pi}{\leftarrow} X \mid Y \sim \ell_P \cdot p(x)\}\ \textsc{H-CondObserve}$$

where P is a Boolean predicate on Y. The axiom states that assuming $Y \sim p(x)$ when conditioning on $X = x$, where $X \sim \pi$ in the unconditioned space, then observing $P(Y)$ updates the distribution from $p(x)$ to $\ell_P \cdot p(x)$, where ℓ_P is the likelihood function with $\ell_P(x) \coloneqq 1$ if $P(x)$ holds, and 0 otherwise. In our case, $Y = \mathit{Flip}_1$ and $P(\mathit{Flip}_1)$ is defined to be $(\mathit{Flip}_1 = 1)$, which yields the postcondition on line 7. For readers familiar with probabilistic programming, **observe** is implemented using the *soft constraint* construct **score**, and there is a more general rule H-CondScore as we explain in §4.

Since $\mathbf{observe}(\mathit{Flip}_1 = 1)$ multiplies the likelihood of $\mathit{Flip}_1 = 1$ by 1, the likelihood of $\mathit{Flip}_1 = 1$ is $x \cdot 1 = x$ when $X = x$. Similarly, $\mathbf{observe}(\mathit{Flip}_1 = 1)$ multiplies the likelihood of $\mathit{Flip}_1 = 0$ by 0; i.e. $\mathit{Flip}_1 = 0$ has likelihood $(1 - x) \cdot 0 = 0$ when $X = x$. In other words, this describes traces where $\mathit{Flip}_1 \sim \mathsf{Bern}(x)$ has likelihood x. Intuitively this makes sense: when $X = 0.1$, the likelihood of $\mathit{Flip}_1 = 1$ is 0.1, which is lower than the likelihood when $X = 0.99$. To reason about this, we introduce a *likelihood proposition* $\mathsf{L}(x)$, which denotes that the likelihood of our current state is x. Since $\mathit{Flip}_1 \sim \mathsf{Bern}(x)$ has likelihood x, the entailment $\mathit{Flip}_1 \sim \ell_{=1} \cdot \mathsf{Bern}(x) \vdash \mathsf{L}(x)$ holds, which results in the postcondition on line 8 using the standard rule of consequence H-Cons (see Figure 9).

Line 9. In light of our observation regarding how likely $X = x$ is, we can now *update* our belief. To achieve this, we apply an internal, logical version of *Bayes' theorem*. Recall that Bayes' theorem, in the context of Bayesian statistics, states the following:

$$\text{posterior} \propto \text{prior} \cdot \text{likelihood}$$

Or, equivalently, suppose $1/Z$ is the *normalising constant* for some $Z > 0$, then Bayes' theorem can be stated as unnormalised posterior $\cdot\ Z = \text{prior} \cdot \text{likelihood}$. Notice that line 8 has a similar structure to above – we have the prior $X \sim \mathsf{Unif}(0,1)$ and the likelihood $\mathsf{L}(x)$. The question is: can we find a suitable proposition that represents the (unnormalised) posterior? That is, finding a suitable proposition $?P$ such that the following entailment holds:

$$\overbrace{x \overset{\mathsf{Unif}(0,1)}{\longleftarrow} X}^{\text{prior}} \mid \overbrace{\mathsf{L}(x)}^{\text{likelihood}} \vdash \overbrace{?P}^{\text{posterior}}$$

Before explaining what the proposition looks like, let us first intuitively visualise how our belief regarding the coin has changed.

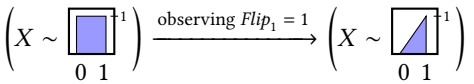


Fig. 7. Bayesian updating of $\mathsf{Unif}(0,1)$

After observing the coin landing on heads, we shift towards believing X being more likely to be larger. For instance, it is less likely for X to be close to zero, as this would make landing on heads less likely. Without doing the calculations (which can be found in a standard text on Bayesian methods, e.g. by McElreath [2020, §2.2]), the original distribution $\mathsf{Unif}(0,1)$ (left of Figure 7) is updated to $1/2 \cdot \mathsf{Beta}(2,1)$ (right of Figure 7). There is, however, a caveat: since we applied Bayes' theorem, the Beta distribution is *unnormalised*: the area under the triangle of the graph is $1/2 \neq 1$, as denoted by $1/2$ in $1/2 \cdot \mathsf{Beta}(2,1)$. To remedy this, we use the separating conjunction '$*$' to factor out the normalising constant. Specifically, for all $Z > 0$ (e.g. $Z = 1/2$ here), the entailment $X \sim Z \cdot \mathsf{Beta}(2,1) \vdash X \sim \mathsf{Beta}(2,1) * \mathsf{NormConst}$ holds. Intuitively, NormConst factors out and hides the normalising factor

and asserts the existence of a non-zero normalising constant, i.e. $\mathsf{NormConst} \coloneqq \exists k : (0, \infty).\ \mathsf{L}(k)$. This answers our question: the 'posterior proposition' $?P$ is of the form:

$$\overbrace{x \xleftarrow{\mathsf{Unif}(0,1)} X}^{\text{prior}} \mid \overbrace{\mathsf{L}(x)}^{\text{likelihood}} \ \vdash \overbrace{X \sim \mathsf{Beta}(2,1) * \mathsf{NormConst}}^{\text{posterior}}$$

As we will see in §4 (Theorem 4.11), the entailment is sound in BaSL via properties of *disintegration* (Lemma 4.10). Next, by applying the rule of consequence, we obtain the postcondition on line 9.

Lines 10-16. Note that lines 10-15 are similar to line 3-7: we perform another coin toss, observe it comes up tails, and return our updated belief. Intuitively, after observing the coin landing on tails, we 'shift back' our belief about X being likely to take on higher values – X is more likely to take on values in the middle (e.g. $0.4 \leq X \leq 0.6$ has higher probability than $X \geq 0.8$ or $X \leq 0.2$), i.e. we update the distribution to the one on the right of Figure 8. This distribution is known as the $\mathsf{Beta}(2,2)$ distribution. To prove this in BaSL, we first assume $X \sim \mathsf{Beta}(2,1)$ as our precondition of line 10, repeat the steps above and obtain the postcondition on line 15.

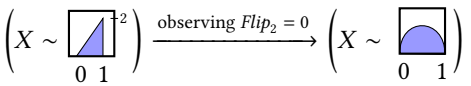


Fig. 8. Bayesian updating of $\mathsf{Beta}(2,1)$

We now enter another key step of our proof: using the *frame rule* of separation logic:

$$\frac{\vdash \{P\}\ M\ \{X.Q\}}{\vdash \{P * F\}\ M\ \{X.Q * F\}}\ (X \notin \mathsf{fv}(F))\ \textsc{H-Frame}$$

The H-Frame rule allows us to *frame off* (factor out) propositions that are probabilistically independent of our current resources. It states that in order to prove $\{P * F\}\ M\ \{X.Q * F\}$, where F is a proposition probabilistically independent of P and Q, it suffices to prove $\{P\}\ M\ \{X.Q\}$.

As the normalising constant proposition NormConst is 'separated' from $X \sim \mathsf{Beta}(2,1)$ on line 9, we can 'frame off' the normalising constant NormConst prior to line 10 and frame it back on after line 15 and obtain the postcondition on line 16.

Lines 17-20. At line 16 we have two normalising constants NormConst and NormConst, each created from an update of X. We now combine them: intuitively, if two independent unnormalised random variables have normalising constants Z and Z' respectively, the overall distribution has normalising constant $Z \cdot Z'$. This justifies the entailment $\mathsf{NormConst} * \mathsf{NormConst} \vdash \mathsf{NormConst}$. This means we can obtain the postcondition on line 17. Returning X yields the same postcondition (see H-Ret in Figure 9), which leads to our desired postcondition on line 19. Since from line 19 we know $X \sim \mathsf{Beta}(2,2)$ and a $\mathsf{Beta}(2,2)$-distributed random variable has expected value (average) $1/2$, the entailment $X \sim \mathsf{Beta}(2,2) * \mathsf{NormConst} \vdash \mathbb{E}[X] = 1/2 * \mathsf{NormConst}$ holds, and we obtain the postcondition on line 20 using the standard rule of consequence H-Cons (see Figure 9).

3 Verifying Statistical Models with BaSL

We demonstrate the expressivity and verification capability of BaSL by verifying five programs described below, each with distinct features. To this end, in §3.1 we first present the BaSL programming language, Bppl, a standard Bayesian probabilistic programming language, and then present the BaSL proof system as a set of Hoare triples (most of which we have described in §2). Specifically, we verify and prove probabilistic properties of six programs listed in Table 1.

Table 1. Statistical models considered in this section and their distinct features

§	Statistical model	Distinct feature(s)
§3.2	Parameter estimation algorithm	Soft constraint; conjugate priors
§3.3	The burglar alarm Bayesian network	Joint conditioning; Bayes' theorem
§3.4	The common effect Bayesian network	Conditional dependence and correlation
§3.5	The semantic Lebesgue measure	Improper prior; handling σ-finite measures
§3.6	Gaussian-mixture-based clustering	Intractable posterior; bounded loops
§3.7	Models with equivalent posterior	Verifying program rewriting

3.1 BaSL Programming Language and Proof System

The BaSL programming language, Bppl, is a typed, first-order language equipped with two effects: the *probabilistic sampling* effect $\mathbf{sample}(\mathbb{P})$, which samples from a distribution $\mathbb{P}$, and the *soft conditioning* effect $\mathbf{score}(\ell)$, which *scales* the current distribution according to a non-negative number ℓ. The Bppl *terms and types* are defined by the following grammar, where X ranges over a countably infinite set of names, $n \in \mathbb{N}$, $r \in \mathbb{R}$, and f ranges over (measurable) functions.

$$\begin{aligned}\mathsf{Term} \ni M ::= {} & () \mid X \mid n \mid r \mid f(M) \mid (M, M) \mid M.\mathbf{1} \mid M.\mathbf{2} \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\ M\ \mathbf{then}\ M\ \mathbf{else}\ M \\ & \mid \mathbf{sample}(M) \mid \mathbf{score}(M) \mid \mathbf{return}(M) \mid \mathbf{let}\ X = M\ \mathbf{in}\ M \\ \mathsf{Type} \ni \tau ::= {} & 1 \mid \mathbb{N} \mid \mathbb{R} \mid \mathbb{B} \mid \tau \times \tau \mid \mathbb{P}(\tau)\end{aligned}$$

Additional Encodings. We encode the following syntactic shorthands:

$$M; N \coloneqq \mathbf{let}\ \_ = M\ \mathbf{in}\ N \qquad \mathbf{observe}(M) \coloneqq \mathbf{score}(\mathbf{if}\ M\ \mathbf{then}\ 1\ \mathbf{else}\ 0)$$
$$\mathbf{observe}\ M\ \mathbf{from}\ \mathbb{P} \coloneqq \mathbf{score}(\mathsf{density}_{\mathbb{P}}(M))$$

The sequential composition (;) shorthand is standard; we describe the **observe** construct shortly below, and elaborate on the *soft constraint* construct '**observe from**' in §3.2.

Bppl Typing Judgements. We present the Bppl typing judgements in the technical appendix (Ho et al. [2025, §B]), where a term M is typed via a judgement $\vdash_{\mathsf{p}}$. Semantically, every open term $\Delta \vdash_{\mathsf{p}} M : \tau$ denotes an *s-finite kernel* $[\![M]\!] : [\![\Delta]\!] \rightsquigarrow [\![\tau]\!]$. We refer the reader to the work of Staton [2017] for a detailed explanation of the semantics as this is not essential for understanding BaSL.

BaSL Assertions. The BaSL *assertions* are defined by the following grammar:

$$\begin{aligned}P ::= {} & \top \mid \bot \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid \forall x : A.P \mid \exists x : A.P \mid P * P \mid P \mathrel{-\!\!*} P \\ & \mid E \sim \pi \mid \mathbb{E}[E] = e \mid \mathsf{own}\, E \mid (x \xleftarrow{\pi} E \mid P) \mid \mathsf{L}(e) \mid \forall_{\mathsf{rv}} X : \mathcal{A}.P \mid \exists_{\mathsf{rv}} X : \mathcal{A}.P \mid \{P\}\, M\, \{X : \mathcal{A}.P\}\end{aligned}$$

where π, E, e, M range over maps defined later in Figure 23, and $\top_1$, as seen in §2, represents states with normalising constant 1 and is defined to be $\mathsf{L}(1)$. Intuitively, E is a random expression, e is a deterministic expression, M is a Bppl term, and π is a measure (distribution). The first-order and separation logic assertions (first line of the grammar) are standard. For probabilistic assertions (second line), we have described most of them intuitively in §2, but for $\forall_{\mathsf{rv}}, \exists_{\mathsf{rv}}$ quantifiers, which quantify over *random variables*, and own, which asserts ownership of random expression E.

BaSL Proof System. We present the *BaSL axiomatic proof system* through a set of rules in Figure 9. We have intuitively described most of the axioms with the coin flip example in §2 except for axioms related to the **score** construct. To understand what **score** does intuitively, suppose we have a random variable $X \sim \mathsf{Unif}(0, 1)$. Executing $\mathbf{score}(\mathbf{if}\ X < 1/2\ \mathbf{then}\ 2\ \mathbf{else}\ 4)$ then increases the

H-Sample
$\{\top_1\}\ \mathbf{sample}(\mathbb{P})\ \{X : \mathbb{R}.X \sim \mathbb{P}\}$

H-CondSample
$\{X \sim \mathbb{P}\}\ \mathbf{sample}(p(X))\ \{Y : \mathbb{R}.x \overset{\mathbb{P}}{\leftarrow} X \mid Y \sim p(x)\}$

H-Score
$\{X \sim \pi\}\ \mathbf{score}(f(X))\ \{X \sim f \cdot \pi\}$

H-CondScore
$\{x \overset{\pi}{\leftarrow} X \mid Y \sim p(x)\}\ \mathbf{score}(f(Y))\{x \overset{\pi}{\leftarrow} X \mid Y \sim f \cdot p(x)\}$

H-Observe
$\{X \sim \pi\}\ \mathbf{observe}(P(X))\ \{X \sim \ell_P \cdot \pi\}$

H-CondObserve
$\{x \overset{\pi}{\leftarrow} X \mid Y \sim p(x)\}\ \mathbf{observe}(P(Y))\ \{x \overset{\pi}{\leftarrow} X \mid Y \sim \ell_P \cdot p(x)\}$

H-Return
$\{Q[[\![M]\!]/X]\}\ \mathbf{return}\ M\ \{X : \mathcal{A}.Q\}$

H-Frame
$$\frac{\vdash \{P\}\ M\ \{X : \mathcal{A}.Q\}}{\vdash \{P * F\}\ M\ \{X : \mathcal{A}.Q * F\}}\ (X \notin \mathsf{fv}(F))$$

H-Let
$$\frac{\vdash \{P\}\ M\ \{X : \mathcal{A}.Q\} \qquad \vdash \forall_{\mathsf{rv}} X : \mathcal{A}.\ \{Q\}\ N\ \{Y : \mathcal{B}.R\}}{\vdash \{P\}\ \mathbf{let}\ X = M\ \mathbf{in}\ N\ \{Y : \mathcal{B}.R\}}$$

H-Cons
$$\frac{P' \vdash P \qquad \vdash \{P\}\ M\ \{X : \mathcal{A}.Q\} \qquad Q \vdash Q'}{\vdash \{P'\}\ M\ \{X : \mathcal{A}.Q'\}}$$

Fig. 9. BaSL proof rules

likelihood of values $< 1/2$ being drawn by a factor of 2, and that of values $\geq 1/2$ by a factor of 4:

$$\left\{X \sim \begin{array}{c}\square^{1}\\ 0\ \ 1\end{array}\right\}\ \mathbf{score}(\mathbf{if}\ X < 1/2\ \mathbf{then}\ 2\ \mathbf{else}\ 4)\ \left\{X \sim \begin{array}{c}2\ \square^{4}\\ 0\ \ 1\end{array}\right\}$$

Note that the distribution of X in the postcondition is *unnormalised*, i.e. the shaded region has area (normalising constant) $2 \cdot 1/2 + 4 \cdot 1/2 = 3$. Moreover, when the distribution of X is normalised, we can calculate that $\Pr[X \geq 1/2] = 2 \cdot \Pr[X < 1/2]$. Statistically speaking, it is useful to think of **score** as a way for users to specify the likelihood of an observation, allowing the interpreter to 'mutate' the current distribution. Indeed, as we described above, we encode the hard conditioning $\mathbf{observe}(M)$ construct as $\mathbf{score}(\mathbf{if}\ M\ \mathbf{then}\ 1\ \mathbf{else}\ 0)$, setting to 0 the likelihood of observations where M does not hold, rendering them impossible.

3.2 Conjugate Priors as Hoare Triples: Verifying a Parameter Estimation Algorithm

We begin our journey of verifying statistical models by considering a useful language feature known as *soft constraint* in languages such as Stan and Anglican, which allows users to specify observations even when they are drawn from *continuous* distributions. BaSL is the *first* probabilistic separation logic that can reason about soft constraints and Bayesian updating.

As described above, we encode the *soft constraint* construct as $\mathbf{observe}\ x\ \mathbf{from}\ \mathbb{P} \coloneqq \mathbf{score}(\mathsf{density}_{\mathbb{P}}(x))$. The $\mathbf{observe}\ x\ \mathbf{from}\ \mathbb{P}$ denotes a distribution $\mathbb{P}$ that has density with respect to either the Lebesgue measure $\lambda_{\mathbb{R}}$ or the counting measure $\#_{\mathbb{N}}$ (which includes 'common' distributions such as normal, binomial, gamma, etc.). We write $\mathsf{density}_{\mathbb{P}}$ to denote the corresponding density function of $\mathbb{P}$ (see works of Vákár and Ong [2018, §7] and Staton [2020, §4]). Intuitively, the score is higher if the observed x is more likely to be generated from $\mathbb{P}$, and the score is 0 if the observation is not possible, e.g. $\mathbf{observe}\ 2\ \mathbf{from}\ \mathsf{Unif}(0,1)$ is tantamount to $\mathbf{score}(0)$.

$\mathbf{let}\ \Theta = \mathbf{sample}(\mathsf{Normal}(\theta_0, \sigma_0))\ \mathbf{in}$
$\mathbf{observe}\ x_1\ \mathbf{from}\ \mathsf{Normal}(\Theta, \sigma);$
$\mathbf{observe}\ x_2\ \mathbf{from}\ \mathsf{Normal}(\Theta, \sigma);$
$\mathbf{return}\ \Theta$

Fig. 10. The GaussParam program

We next consider the GaussParam example by Lee [2012, §2] in Figure 10, where we have a normally distributed population with a known standard deviation σ, and we would like to estimate

H-Norm-Conj-Norm
$\{\Theta \sim \mathcal{N}(\theta_0, \sigma_0)\}$ **observe** x **from** Normal(Θ, σ) $\{\Theta \sim \mathcal{N}(\theta^{\theta_0,\sigma_0}_{\text{new},x}, \sigma^{\theta_0,\sigma_0}_{\text{new}}) * \text{NormConst}\}$

H-Beta-Conj-Bern
$\{\Theta \sim \text{Beta}(m, n)\}$ **observe** 1 **from** Bern(Θ) $\{\Theta \sim \text{Beta}(m+1, n) * \text{NormConst}\}$

H-Gamma-Conj-Poisson
$\{\lambda \sim \Gamma(k, \theta)\}$ **observe** x **from** Poisson(λ) $\{\lambda \sim \Gamma(k + x, \theta/\theta+1) * \text{NormConst}\}$

$\{\Theta \sim \mathcal{N}(\theta_0, \sigma_0)\}$ // H-Sample
observe x **from** Normal(Θ, σ) // desugars to **score**(normal-pdf$(x_1 \mid \Theta, \sigma)$)
$\{\Theta \sim \text{normal-pdf}(x \mid -, \sigma) \cdot \mathcal{N}(\theta_0, \sigma_0)\}$ // H-Score
$\{\Theta \sim \mathcal{N}(\theta^{\theta_0,\sigma_0}_{\text{new},x}, \sigma^{\theta_0,\sigma_0}_{\text{new}}) * \text{NormConst}\}$ // H-Cons

Fig. 11. Examples of derived conjugate priors in BASL (above); a BASL derivation of H-Norm-Conj-Norm (below), where the //annotation denotes the BASL rule(s) applied to obtain the associated postcondition.

$\{\top_1\}$
let Θ = **sample**(Normal(θ_0, σ_0)) **in**
$\{\Theta \sim \mathcal{N}(\theta_0, \sigma_0)\}$ // H-Sample
observe x_1 **from** Normal(Θ, σ) // let $\theta_{i+1} \coloneqq \theta^{\theta_i,\sigma_i}_{\text{new},x_{i+1}}$; $\sigma_{i+1} \coloneqq \sigma^{\theta_i,\sigma_i}_{\text{new}}$ for $i \geq 0$
$\{\Theta \sim \mathcal{N}(\theta_1, \sigma_1) * \text{NormConst}\}$ // H-Norm-Conj-Norm
H-Frame: $\{\Theta \sim \mathcal{N}(\theta_1, \sigma_1)\}$ **observe** x_2 **from** Normal(Θ, σ) $\{\Theta \sim \mathcal{N}(\theta_2, \sigma_2) * \text{NormConst}\}$ // H-Frame and H-Norm-Conj-Norm
$\{\Theta \sim \mathcal{N}(\theta_2, \sigma_2) * \text{NormConst} * \text{NormConst}\}$
$\{\Theta \sim \mathcal{N}(\theta_2, \sigma_2) * \text{NormConst}\}$ // H-Cons
return Θ
$\{\Theta \sim \mathcal{N}(\theta_2, \sigma_2) * \text{NormConst}\}$ // H-Ret

Fig. 12. A BASL proof sketch of GaussParam.

its mean Θ. To do this, we assume a normal prior $\Theta \sim \mathcal{N}(\theta_0, \sigma_0)$, where θ_0, σ_0 are constants. Suppose we draw two samples $\{x_1, x_2\}$ from the dataset; we can then write **observe** x_i **from** Normal(Θ, σ) for $i \in \{1, 2\}$ to compute our updated belief of Θ, as shown in Figure 10. Intuitively, the mean of Θ shifts *up* when $x_i > \Theta$ and shifts down otherwise. Using Bayes' rule, we can compute the updated mean and standard deviation as follows [Lee 2012, §2.2.1]:

$$\theta^{\theta_0,\sigma_0}_{\text{new},x} \coloneqq (\sigma^{\theta_0,\sigma_0}_{\text{new}})^2 \left(\frac{\theta_0}{\sigma_0^2} + \frac{x}{\sigma^2}\right) \qquad \sigma^{\theta_0,\sigma_0}_{\text{new}} \coloneqq \sqrt{\frac{1}{\sigma_0^{-2} + \sigma^{-2}}}$$

Note that the updated distribution has a closed-form solution (which is not usually the case). Indeed, the Normal-prior/Normal-likelihood pair is an instance of a *conjugate distribution*, which is a prior/likelihood pair that leads to a closed-form posterior distribution via Bayes' rule.

In BASL we can obtain this fact using the *derived* H-Norm-Conj-Norm rule in Figure 11 (above), with its BASL derivation given at the bottom of Figure 11. In fact, the ability to derive conjugate priors in BASL allows us to perform symbolic reasoning. To this end, in Figure 11 we list a few

common conjugate priors represented as Hoare triples. The ability to derive conjugate priors in BASL, along with the frame rule, gives us a foundation for *modular and symbolic reasoning* of probabilistic programs: for **observe** x_1 **from** $\mathsf{Normal}(\Theta, \sigma)$, we apply the frame rule to 'frame out' the normalising constant, then apply the conjugate distribution triple, which then allows us to symbolically derive the posterior distribution for Θ in Figure 12.

3.3 Verifying the 'Hello World' of Probabilistic Programming: Burglar Alarm

With soft constraint conditioning defined, we now consider a classic example in Bayesian statistics, the BURGLARALARM program at the top of Figure 13. The problem is as follows: your home has a burglar alarm that activates when there is a burglar, but it also accidentally activates when there is an earthquake. There is a 1% probability that there is a burglar, and a 10% probability that there is an earthquake. When the alarm activates, there is a 99% probability of the phone ringing. Assuming that your phone is ringing now, what is the probability of there being an earthquake?

We use BASL in Figure 13 to prove that the probability of having an earthquake is roughly 84.7% by using a technique called *joint conditioning*. Note that random variable A takes a value in $\{0, 1\}$ depending on whether there is a burglar (captured by B) or an earthquake (E). By conditioning on the *joint* random variable $(B, E) = (b, e)$ for some $(b, e) \in \{0, 1\}^2$, we know A has distribution $\delta_{b \vee e}$. We then apply the rule of consequence H-CONS with our internal notion of Bayes' theorem E-BAYES (explained later in Theorem 4.11), we obtain a posterior distribution stating that there is an 84.7% probability of there being an earthquake, as shown below, with f defined in Figure 13:

$$\overbrace{(b,e) \xleftarrow{\mathsf{Bern}(0.01)\otimes\mathsf{Bern}(0.1)} (B,E)}^{\text{prior}} \mid \overbrace{\mathsf{L}(f(b,e))}^{\text{likelihood}} \vdash \overbrace{(B,E) \sim f \cdot (\mathsf{Bern}(0.01) \otimes \mathsf{Bern}(0.1))}^{\text{posterior}} \quad \text{(Theorem 4.11)}$$

$$\vdash E \sim \mathsf{Bern}(0.099/0.11682) \quad \text{(calculation)}$$

3.4 Reasoning about Independence and Correlations: The Collider Bayesian Network

We now consider a *Bayesian network* in which two random variables X and Y that are initially independent become negatively correlated after performing Bayesian conditioning. BASL is the *first* logic that can reason about such conditional dependence brought about by Bayesian updating.

A Bayesian network is a directed acyclic graph modelling relationship of random variables. A common structure in Bayesian networks is known as *colliders* (or *common effects*), where the distribution of a random variable Z is conditionally independent upon X and Y. For our example in Figure 14, we flip two coins X and Y (taking values in $\{0, 1\}$) and let Z be the maximum of X and Y. Before observing Z, X and Y are independent. That is, we can use BASL to prove $X \sim \mathsf{Bern}(1/2) * Y \sim \mathsf{Bern}(1/2)$ prior to observing Z. However, once we perform Bayesian conditioning by observing $Z = 1$, X and Y are no longer independent: knowing the value of Z gives us information about X and Y. For example, knowing $Z = 1$ and $X = 0$ gives us extra information $Y = 1$. In fact, not only X and Y are now dependent, they are also *negatively correlated*: when X has a higher value, Y is likely to be lower, and vice versa.

let X = **sample**($\mathsf{Bern}(1/2)$) **in**
let Y = **sample**($\mathsf{Bern}(1/2)$) **in**
let Z = **return**$(X \vee Y)$ **in**
observe$(Z = 1)$;
return (X, Y)

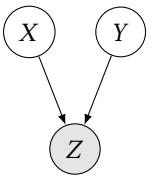


Fig. 14. COLLIDER and its Bayesian network

We use BASL to prove this negative correlation as shown in Figure 15. We first apply H-SAMPLE to sample X with distribution $\mathsf{Bern}(1/2)$, then apply H-FRAME and H-SAMPLE to obtain $Y \sim \mathsf{Bern}(1/2)$ and

```
let B = sample(Bern(0.01)) in                    // probability of burglary is 1%
let E = sample(Bern(0.1)) in                     // probability of earthquake is 10%
let A = return(B ∨ E) in                         // alarm activates in case of burglary or earthquake
observe 1 from Bern(if A then 0.99 else 0.01);   // observe the phone is ringing
return E                                         // is there an earthquake?
```

$\{\top_1\}$
  let B = sample(Bern(0.01)) in
$\{B \sim \mathsf{Bern}(0.01)\}$ // H-Sample
  let E = sample(Bern(0.1)) in
$\{B \sim \mathsf{Bern}(0.01) * E \sim \mathsf{Bern}(0.1)\}$ // H-Sample, H-Frame
$\{(B,E) \sim \mathsf{Bern}(0.01) \otimes \mathsf{Bern}(0.1)\}$ // H-Cons
  let A = return$(B \vee E)$ in
$\{(b,e) \xleftarrow{\mathsf{Bern}(0.01)\otimes\mathsf{Bern}(0.1)} (B,E) \mid A \sim \delta_{b\vee e}\}$ // H-CondSample
  observe 1 from Bern(if A then 0.99 else 0.01); // with $f(b,e) \coloneqq$ if $b \vee e$ then 0.99 else 0.1
$\left\{(b,e) \xleftarrow{\mathsf{Bern}(0.01)\otimes\mathsf{Bern}(0.1)} (B,E) \;\middle|\; A \sim f \cdot \delta_{b\vee e}\right\}$ // H-CondScore
$\left\{(b,e) \xleftarrow{\mathsf{Bern}(0.01)\otimes\mathsf{Bern}(0.1)} (B,E) \;\middle|\; \mathsf{L}\,(f(b,e))\right\}$ // H-Cons
$\{(B,E) \sim f \cdot (\mathsf{Bern}(0.01) \otimes \mathsf{Bern}(0.1))\}$ // H-Cons with E-Bayes
$\{E \sim \mathsf{Bern}(^{0.099}/_{0.11682}) * \mathsf{NormConst}\}$ // H-Cons
$\{\mathbb{E}[E] = ^{0.099}/_{0.11682} * \mathsf{NormConst}\}$ // H-Frame, H-Cons
  return E
$\{\mathbb{E}[E] = ^{0.099}/_{0.11682} * \mathsf{NormConst}\}$ // H-Return
$\{\Pr[E=1] \approx 0.847 * \mathsf{NormConst}\}$ // H-Cons

Fig. 13. BurglarAlarm (above); BaSL proof sketch for deriving the posterior probability of earthquake (below)

'frame' Y onto X to obtain $X \sim \mathsf{Bern}(1/2) * Y \sim \mathsf{Bern}(1/2)$. As we show shortly in §4, the (bi)entailment $X \sim \mu * Y \sim \nu \dashv\vdash (X,Y) \sim \mu \otimes \nu$ holds and we thus use H-Cons to obtain $(X,Y) \sim \mathsf{Bern}(1/2) \otimes \mathsf{Bern}(1/2)$.

We next condition on (X,Y) and sample Z using the conditional sampling axiom H-CondSample to show Z has a conditional distribution $\delta_{x\vee y}$, (jointly) conditioning on the random variable $(X,Y)=(x,y)$ for $(x,y) \in \{0,1\}^2$. Upon subsequently observing $Z=1$, we apply the conditional observation axiom H-CondObserve to prove the updated distribution 'ignores' the case when $X=0$ and $Y=0$. Let us consider the updated likelihood of all four cases of $(x,y) \in \{0,1\}^2$: when $x=y=0$, the likelihood is 0, otherwise it is 1. That is, the likelihood proposition $\mathsf{L}(\text{if } x=y=0 \text{ then } 0 \text{ else } 1)$ holds, or equivalently, $\mathsf{L}(1-[x=y=0])$ holds. Using the *internal Bayes' theorem* (formulated later in Theorem 4.11), we thus know the updated (X,Y) has distribution $(X,Y) \sim (1-[x=y=0]) \cdot \mathsf{Bern}(1/2)^2$. We compute the probability of (X,Y) being $(0,1)$, $(1,0)$ or $(1,1)$ is $1/3$ in

$\{\top_1\}$
  let X = sample(Bern(1/2)) in
$\{X \sim \mathsf{Bern}(1/2)\}$
  let Y = sample(Bern(1/2)) in
$\{X \sim \mathsf{Bern}(1/2) * Y \sim \mathsf{Bern}(1/2)\}$
$\{(X,Y) \sim \mathsf{Bern}(1/2) \otimes \mathsf{Bern}(1/2)\}$
  let Z = return $X \vee Y$ in
$\{(x,y) \xleftarrow{\mathsf{Bern}(1/2)\otimes\mathsf{Bern}(1/2)} (X,Y) \mid Z \sim \delta_{x\vee y}\}$
  observe$(Z=1)$;
$\{(x,y) \xleftarrow{\mathsf{Bern}(1/2)\otimes\mathsf{Bern}(1/2)} (X,Y) \mid Z \sim \ell_{=1} \cdot \delta_{x\vee y}\}$
$\{(x,y) \xleftarrow{\mathsf{Bern}(1/2)\otimes\mathsf{Bern}(1/2)} (X,Y) \mid \mathsf{L}(1-[x=y=0])\}$
  return (X,Y)
$\{(X,Y) \sim (1-[x=y=0]) \cdot \mathsf{Bern}^2(1/2)\}$
$\{(X,Y) \sim \mathsf{Unif}\{(0,1),(1,0),(1,1)\} * \mathsf{NormConst}\}$
$\{\mathbb{E}[XY] = 1/3 \wedge \mathbb{E}[X] = \mathbb{E}[Y] = 2/3 * \mathsf{NormConst}\}$
$\{\mathsf{Cov}[X,Y] < 0 * \mathsf{NormConst}\}$

Fig. 15. A BaSL proof sketch of Collider showing that return values (X,Y) are negatively correlated

$\{\top_1\}$
  let X = **sample**(Exp(1)) **in**
$\{X \sim \mathsf{Exp}(1)\}$ // H-Sample
  score(e^X)
$\{X \sim \exp \cdot \mathsf{Exp}(1)\}$ // H-Score
$\{X \sim \mathsf{Leb}_{[0,\infty)}\}$ // H-Cons

Fig. 16. Lebesgue

$\{X \sim \mathsf{Leb}_{[0,\infty)}\}$
  observe X **from** μ
$\{X \sim \mathsf{pdf}_\mu \cdot \mathsf{Leb}_{[0,\infty)}\}$ // H-Score
$\{X \sim \mu\}$ // H-Cons

Fig. 17. Conditioning of improper Lebesgue prior

each case, and thus calculate the covariance $\mathrm{Cov}[X, Y] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = 1/3 - 2/3 \cdot 2/3 = -1/9$. Using H-Cons, we then formally prove $\mathrm{Cov}[X, Y] < 0$, stating that X and Y are negatively correlated.

3.5 Modelling Improper Prior: Correctness of the Semantic Lebesgue Measure

We now consider a common technique in Bayesian statistical modelling known as *improper priors*. An improper prior is a distribution with an infinite normalising constant. One canonical such example [Narayanan et al. 2016; Staton 2020] is the *Lebesgue measure*, $\mathsf{Leb}_{[0,\infty)}$, which assigns length $b - a$ to any interval (a, b) with $0 \le a < b$. As shown by Staton [2020], a Bppl program can simulate the Lebesgue measure as the computation **let** X = **sample**(Exp(1)) **in score**(e^X) since the *measure* (as opposed to probability) of $X \in (a, b)$ is $\int_{(a,b)} e^x e^{-x} \mathrm{d}x = b - a$. In fact, X has an *improper prior distribution* – the normalising constant is currently $\int_0^\infty e^x \cdot e^{-x}\, \mathrm{d}x = \int_0^\infty 1\, \mathrm{d}x = \infty$.

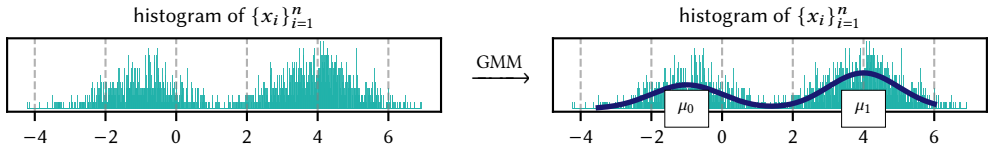
BaSL is the *first* logic that can reason about improper priors because its semantic domain include *σ-finite measure spaces*. We present a BaSL proof sketch of the computation in Figure 16, proving that X is distributed according to the Lebesgue measure: $X \sim \mathsf{Leb}_{[0,\infty)}$. In fact, since $X \sim \mathsf{Leb}_{[0,\infty)}$, we know that for any measure μ with a density with respect to $\mathsf{Leb}_{[0,\infty)}$ (e.g. $\mu = \mathsf{Unif}(0, 1)$), observing X to have distribution μ results in $X \sim \mu$. We can derive this in BaSL as shown in Figure 17.

3.6 Representing the Posterior of a Bayesian Clustering Algorithm

For our next example, we use BaSL to calculate the posterior of a clustering algorithm that implements the Gaussian mixture model (GMM). The problem is as follows: suppose we have a dataset $\{x_i \in \mathbb{R}\}_{i=1}^n$ and we want to cluster them into two groups. The model works by assuming there are two normally-distributed latent variables $\mu_0, \mu_1 \in \mathbb{R}$ that represent the mean of the two groups and another latent variable $\pi \in [0, 1]$ representing the proportion of samples belonging to the two groups. For each x_i, we sample a Bernoulli random variable N with success probability π and we assume x_i is a sample drawn from $\mathcal{N}(\mu_N, 1)$ by performing soft conditioning. The diagram at the top of Figure 18 illustrates the main idea of GMM: given a dataset $\{x_i\}_{i=1}^n$, we infer the mean of clusters μ_0 and μ_1 and fit the data via the mixture of two Gaussian distributions.

Unlike the previous examples, GMM does not have a closed-form solution – we must rely on inference algorithms such as MCMC to approximate the posterior distribution of (μ, π). However, with BaSL, we can symbolically represent the formula for the posterior distribution. Similar to Lilac, we extend Bppl with a syntactic construct that encodes bounded loop over a literal list $[x_1, ..., x_n]$ by defining **for** X **in** $[x_1, ..., x_n]$ **do** $M \coloneqq M[x_1/X]; ...; M[x_n/X]$. By applying the sequencing rule H-Let inductively, we obtain the rule H-BoundedFor, which allows us to express GMM and represent the posterior of (μ, π) in Figure 18. Even though the posterior distribution cannot be explicitly simplified, we can still represent it as the postcondition $(\mu, \pi) \sim \mathbb{P}_n$.

$$\text{H-BoundedFor}\quad \frac{\text{for } i = 1, ..., n,\ \{P_{i-1}\}\ M[x_i/X]\ \{P_i\}}{\{P_0\}\ \textbf{for}\ X\ \textbf{in}\ [x_1, ..., x_n]\ \textbf{do}\ M\ \{P_n\}}$$



$\{\top_1\}$
  let $\pi = \mathbf{sample}(\mathsf{Unif}(0,1))$ **in** // sample the proportion of each cluster
$\{\pi \sim \mathsf{Unif}(0,1)\}$ // H-Sample, H-Frame
  let $\mu = \mathbf{sample}(\mathcal{N}(0,10) \otimes \mathcal{N}(0,10))$ **in** // sample from two normal distributions
$\{\mu \sim \mathcal{N}^2(0,10) * \pi \sim \mathsf{Unif}(0,10)\}$ // H-Sample
  for X **in** $[x_1, ..., x_n]$ **do** // let $\mathbb{P}_0 \coloneqq \mathcal{N}^2(0,1) \otimes \mathsf{Unif}(0,1)$
    $\{(\mu,\pi) \sim \mathbb{P}_i\}$ // H-Cons
      let $I = \mathbf{sample}(\mathsf{Bern}(\pi))$ **in** // let $f_i(m,p) \coloneqq (1-p)\mathcal{N}(x_i \mid m_0, 1) + p\mathcal{N}(x_i \mid m_1, 1)$
    $\{(m,p) \overset{\mathbb{P}_i}{\longleftarrow} (\mu,\pi) \mid I \sim \mathsf{Bern}(p)\}$ // H-CondSample
      observe X **from** $\mathsf{Normal}(\mu_I, 1)$ // let $\mathbb{P}_{i+1} \coloneqq f_i \cdot \mathbb{P}_i$
    $\{(m,p) \overset{\mathbb{P}_i}{\longleftarrow} (\mu,\pi) \mid I \sim f_i \cdot \mathsf{Bern}(p)\}$ // H-CondScore
    $\{(\mu,\pi) \sim \mathbb{P}_i\}$ // H-Cons, E-Bayes
$\{(\mu,\pi) \sim \mathbb{P}_n\}$ // H-BoundedFor

Fig. 18. An illustration Gaussian mixture model (above); an implementation of the Gaussian mixture model and a BaSL proof sketch deriving its posterior (below)

3.7 Rewriting Probabilistic Programs

As we briefly discussed in §1, we can use BaSL to justify *rewriting* probabilistic program by proving that they satisfy equivalent specifications (Hoare triples). Let us revisit the example in §1 (p. 4); as sown in Figure 19, we can combine the techniques introduced thus far (in §3.2 to §3.6) to show that the two programs satisfy equivalent specifications. The program on the left contains normally-distributed and beta-distributed random variables X and Y, respectively, along with conditioning constructs on two lists of observed data $\{x_i\}_{i=1}^N$ and $\{c_i\}_{i=1}^N$. Note that two kinds of conjugacy exist within the program: the normal-normal conjugate and the beta-Bernoulli conjugate. As such, we can simplify the program on the left by *pre-computing* the posterior and sampling from them directly, as in the program on the right. However, how do we prove that the simplification is sound, i.e. the two programs are equivalent (up to a normalising constant)? We can do this in BaSL by proving the *same specification* (Hoare triple) for both programs, as shown in Figure 19.

Specifically, using H-Norm-Conj-Norm and H-Beta-Conj-Bern, we establish a loop invariant and reason about how the distributions of X and Y change at every iteration I. We then apply the Hoare triple for bounded for-loops H-BoundedFor. As such, we can show that the random variables in both programs have the same distribution (up to a normalising constant NC, which can be safely ignored since inference algorithms return the same samples, regardless of the normalising constant), and therefore we can justify rewriting the program on the left to that on the right.

Through numerous examples showcasing the novel and hitherto-unsupported features of BaSL, we have demonstrated the expressivity of BaSL, and we believe this can serve as a logical foundation for static analysis/symbolic execution tools for probabilistic programs.

$\{\top_1\}$
  let $X = \mathbf{sample}(\mathcal{N}(0,1))$ **in**
$\{X \sim \mathcal{N}(0,1)\}$ // H-Sample
  let $Y = \mathbf{sample}(\mathsf{Beta}(m,n))$ **in**
$\{X \sim \mathcal{N}(0,1) * Y \sim \mathsf{Beta}(m,n)\}$ // H-Sample
$\{X \sim \mathcal{N}(0,1) * Y \sim \mathsf{Beta}(m,n) * \mathsf{NC}\}$ // H-Cons
// define $a_i \coloneqq \sum_{j=1}^{i} [c_j = 1]$, $b_i \coloneqq \sum_{j=1}^{i} [c_j = 0]$
// define $c_i \coloneqq \sum_{j=1}^{i} \frac{x_j}{i}$
  for I **in** $[1, ..., N]$ **do**
$\left\{\begin{array}{l} X \sim \mathcal{N}(c_i, 1) * \\ Y \sim \mathsf{Beta}(m + a_i, n + b_i) * \mathsf{NC} \end{array}\right\}$ // loop invariant
    observe x_I **from** $\mathcal{N}(M, 1)$;
$\left\{\begin{array}{l} X \sim \mathcal{N}(c_{i+1}, 1) * \mathsf{NC} * \\ Y \sim \mathsf{Beta}(m + a_i, n + b_i) \end{array}\right\}$ // H-Norm-Conj-Norm // H-Cons
    observe c_I **from** $\mathsf{Bern}(W)$
$\left\{\begin{array}{l} X \sim \mathcal{N}(c_{i+1}, 1) * \mathsf{NC} * \\ Y \sim \mathsf{Beta}(m + a_{i+1}, n + b_{i+1}) \end{array}\right\}$ // H-Beta-Conj-Bern
$\left\{\begin{array}{l} X \sim \mathcal{N}(c_N, 1) * \\ Y \sim \mathsf{Beta}(m + a_N, n + b_N) * \mathsf{NC} \end{array}\right\}$ // H-BoundedFor

$\{\top_1\}$
  let $X = \mathbf{sample}(\mathcal{N}(c_N, 1))$ **in**
$\{X \sim \mathcal{N}(c_N, 1)\}$ // H-Sample
  let $Y = \mathbf{sample}(\mathsf{Beta}(n + a_N, m + b_N))$ **in**
$\left\{\begin{array}{l} X \sim \mathcal{N}(c_N, 1) * \\ Y \sim \mathsf{Beta}(m + a_N, n + b_N) \end{array}\right\}$ // H-Sample
  return (X, Y)
$\left\{\begin{array}{l} X \sim \mathcal{N}(c_N, 1) * \\ Y \sim \mathsf{Beta}(m + a_N, n + b_N) \end{array}\right\}$ // H-Ret

Fig. 19. Using BaSL to justify that the program on the left can be *soundly rewritten* to that on the right by proving that they both satisfy the *same BaSL specification* (BaSL triple).

4 The Semantics of BaSL

Now that we have demonstrated how BaSL can function as a logical framework for proving properties in statistical models, we explain its resource-theoretic semantics in §4.4. But before this, we review the Kripke resource model of the Lilac separation logic [Li et al. 2023] in §4.1, which the model of BaSL is based on, then we motivate the need for a generalised model of randomness for Bayesian updating in §4.2 and prove that it is indeed a resource model in §4.3.

4.1 Background: A Resource Monoid for Randomisation

In separation logic, *computational resources* such as heaps are modelled by *partial commutative monoids* $(\mathcal{M}, \bullet, \mathbf{1})$ [Calcagno et al. 2007]. The set $\mathcal{M}$ represents states of the resource and the partial function $(\bullet) : \mathcal{M} \times \mathcal{M} \rightharpoonup \mathcal{M}$ combines two states $m_1, m_2 \in \mathcal{M}$ if they are *compatible* (e.g. m_1 and m_2 describe different parts of the resource). For example, the heap is modelled by $\mathcal{M}_{\text{heap}} \coloneqq \text{Loc} \rightharpoonup_{\text{fin}} \text{Val}$, where Loc is the set of memory addresses and Val is the set of values. For instance, $\{42 \mapsto \text{“a”}\} \in \mathcal{M}_{\text{heap}}$ represents a heap where address 42 stores value “a”. Moreover, given two heaps m_1, m_2, $(\bullet)$ combines them if they contain separate addresses. For example, $m_1 \coloneqq \{21 \mapsto 2025, 42 \mapsto \text{“a”}\}$ and $m_2 \coloneqq \{52 \mapsto 123\}$ can be combined to $m_1 \bullet m_2 = \{21 \mapsto 2025, 42 \mapsto \text{“a”}, 52 \mapsto 123\}$ since the addresses in m_1 (21 and 42) do not overlap with 52 in m_2. There is also an identity element $\mathbf{1} \in \mathcal{M}$ that represents the empty resource. For heaps, $\mathbf{1}$ is defined to be the empty heap $\{\}$.

Applying the same intuition to probability, one reasonable model of computational resource is that of a *random number generator*. The Lilac separation logic defined a partial commutative monoid that models a random generator by encoding the following two properties:

(1) the *distribution* of the generated numbers, i.e. a random number generator should have information about the distribution of the numbers generated; and

(2) the *usage* of the generator, which has information regarding whether the i-th number ω_i has been generated.

To model (1) and (2), Lilac uses a *measure space*. In fact, the distribution and usage property can be modelled via the *measures* and *σ-algebras*, respectively. For readers unfamiliar with these concepts, a *measurable space* is a pair $(\Omega, \mathcal{F})$, where Ω is a set and $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ is a set of subsets such that $\emptyset, \Omega \in \mathcal{F}$ and $\mathcal{F}$ is closed under complements and countable unions. We call Ω the *sample space* and $\mathcal{F}$ a *σ-algebra* of Ω. A *measure* of $(\Omega, \mathcal{F})$ is a function $\mu : \mathcal{F} \to [0, \infty]$ satisfying $\mu(\emptyset) = 0$ and $\mu(\biguplus_{i \in \mathbb{N}} U_i) = \sum_{i \in \mathbb{N}} \mu(U_i)$. The triple $(\Omega, \mathcal{F}, \mu)$ forms a *measure space*. If $\mu(\Omega) = 1$, we additionally call $(\Omega, \mathcal{F}, \mu)$ a *probability space*. For example, to model a fair die, we set $\Omega \coloneqq \{1, 2, 3, 4, 5, 6\}$, $\mathcal{F} \coloneqq \mathcal{P}(\Omega)$, and $\mu(U) \coloneqq \sum_{i=1}^{6} \mathbf{1}_U(i) \cdot 1/6$, where $\mathbf{1}_U(i)$ is 1 when $i \in U$ and 0 when $i \notin U$. We can then compute that the probability of the die having a value greater than four is $\mu(\{5, 6\}) = 1/3$.

We now consider a simplified model of Lilac's random number generator – we define our sample space to be $\Omega \coloneqq \{\mathsf{T}, \mathsf{F}\}^2$ (as opposed to $[0, 1]^{\mathbb{N}}$ in the full model of Lilac), which, intuitively, can be thought of as the computer having access to a source of randomness that can generate two independent booleans. The states of the random generator is then modelled by the following set:

$$\mathcal{M} \coloneqq \{(\mathcal{F}, \mu) \mid (\Omega, \mathcal{F}, \mu) \text{ is a probability space}\}$$

To explain how $\mathcal{M}$ models random generators, we construct the empty generator $\mathbf{1} \coloneqq (\mathcal{F}_\mathbf{1}, \mu_\mathbf{1})$ – an element of $\mathcal{M}$ that represents a state where nothing has been generated. This means:

(1) We *do not* know the probability of the first boolean being F and second boolean being T. Hence, $\{(\mathsf{F}, \mathsf{T})\} \notin \mathcal{F}_\mathbf{1}$ and we cannot apply $\mu_\mathbf{1} : \mathcal{F}_\mathbf{1} \to [0, \infty]$ to $\{(\mathsf{F}, \mathsf{T})\}$ to get the probability. Similarly, $\{(a, b)\} \notin \mathcal{F}_\mathbf{1}$ for $a, b \in \{\mathsf{T}, \mathsf{F}\}$.
(2) We *do not* know the probability of the first boolean being T, which is equivalent to saying the first boolean being T and the second boolean being T *or* F. Hence, $\{(\mathsf{T}, \mathsf{T}), (\mathsf{T}, \mathsf{F})\} \notin \mathcal{F}_\mathbf{1}$. Similarly, $\{(a, b), (c, d)\} \notin \mathcal{F}_\mathbf{1}$ for $a, b, c, d \in \{\mathsf{T}, \mathsf{F}\}$.
(3) Even though the first number has not been generated, we know that the first and/or second boolean will either be T or F. Hence, $\{(\mathsf{T}, \mathsf{F}), (\mathsf{T}, \mathsf{T}), (\mathsf{F}, \mathsf{T}), (\mathsf{F}, \mathsf{F})\} = \Omega \in \mathcal{F}_\mathbf{1}$.

For $\mathcal{F}_\mathbf{1}$ to be a σ-algebra, we need $\emptyset \in \mathcal{F}_\mathbf{1}$; hence, $\mathcal{F}_\mathbf{1} = \{\emptyset, \Omega\}$. For the probability measure $\mu_\mathbf{1} : \mathcal{F}_\mathbf{1} \to [0, \infty]$, we know that the first and second boolean have a 100% probability of being T or F. Hence, $\mu_\mathbf{1}(\{(\mathsf{T}, \mathsf{F}), (\mathsf{T}, \mathsf{T}), (\mathsf{F}, \mathsf{T}), (\mathsf{F}, \mathsf{F})\}) = 1$. See Figure 20 for an illustration.

Next, we construct $m_2 \coloneqq (\mathcal{F}_2, \mu_2) \in \mathcal{M}$ where the first boolean has a 42% probability of being T and 58% probability of being F, and only the first boolean has been generated. This means:

(1) We know the probability of the first boolean being T is 42%, which is equivalent to saying the first boolean being T and the second boolean being T or F is 42%. Hence, $\{(\mathsf{T}, \mathsf{F}), (\mathsf{T}, \mathsf{T})\} \in \mathcal{F}_2$ and $\mu_2(\{(\mathsf{T}, \mathsf{F}), (\mathsf{T}, \mathsf{T})\}) \coloneqq 0.42$. Similarly, $\{(\mathsf{F}, \mathsf{F}), (\mathsf{F}, \mathsf{T})\} \in \mathcal{F}_2$ and $\mu_2(\{(\mathsf{F}, \mathsf{F}), (\mathsf{F}, \mathsf{T})\}) \coloneqq 0.58$.
(2) We *do not* know the probability of the second boolean being F, which is equivalent to saying the second boolean being F and the first boolean being T or F. Hence, $\{(\mathsf{T}, \mathsf{F}), (\mathsf{F}, \mathsf{F})\} \notin \mathcal{F}_2$. Similarly, the singletons $\{(a, b)\}$ with $a, b \in \{\mathsf{T}, \mathsf{F}\}$ and $\{(\mathsf{T}, \mathsf{T}), (\mathsf{F}, \mathsf{T})\}$ should not be in $\mathcal{F}_2$ since they contain information about the second boolean.

Since $\{(\mathsf{T}, \mathsf{T}), (\mathsf{F}, \mathsf{T})\}$ and $\{(\mathsf{F}, \mathsf{F}), (\mathsf{F}, \mathsf{T})\}$ are in $\mathcal{F}_2$, their union Ω is also in $\mathcal{F}_2$ and $\mu_2(\Omega) = 1$, which makes sense since the probability of either events happening is 42% + 58% = 100%.

Similarly, we can construct $m_3 = (\mathcal{F}_3, \mu_3) \in \mathcal{M}$ where only the second boolean has been generated with 30% probability of being T and 70% probability of being F. Now, since m_2 only has information about the first boolean and m_3 only has information about the second boolean, they can be combined to form $m_2 \bullet m_3 = (\mathcal{F}_{23}, \mu_{23})$ such that we can refer to e.g. the probability of the first boolean being T and the second boolean being F, i.e. $\{(\mathsf{T}, \mathsf{F})\} \in \mathcal{F}_{23}$ and $\mu_{23}(\{\mathsf{T}, \mathsf{F}\}) \coloneqq$

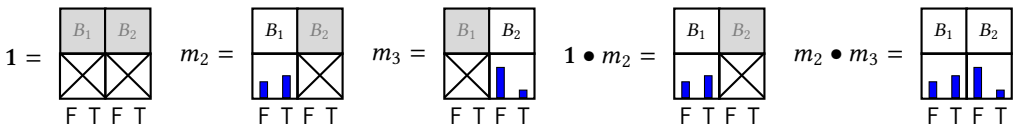


Fig. 20. Illustrations of random number generators

$\mu_2(\{(\mathsf{T},\mathsf{F}),(\mathsf{T},\mathsf{T})\}) \cdot \mu_3(\{(\mathsf{F},\mathsf{F}),(\mathsf{F},\mathsf{T})\}) = 42\% \cdot 70\% = 29.4\%$ (see Figure 20). As shown by Li et al. [2023], $(\mathcal{M}, \bullet, \mathbf{1})$ forms a partial commutative monoid (PCM).

4.2 The Need for an Extended Resource Model of Randomness

We now motivate the semantics of BASL by considering a problem we would like to solve; namely, how a random generator, such as the ones presented above, can be *updated*. In the heap model of separation logic (see first paragraph of §4.1), an update on address l with value v is modelled as an *update function* $(-)[l \mapsto v] : \mathcal{M}_{\mathsf{heap}} \to \mathcal{M}_{\mathsf{heap}}$ defined by $m[l \mapsto v] \coloneqq m \cup \{l \mapsto v\}$. We develop an analogous operation on random number generators, for which we must model the **observe/score** construct. But first, let us establish the motivation: why would we want to update a distribution? And why is it such a notoriously hard problem? The answer is twofold: we need to handle both *unnormalised measures* and the update's effect on *dependent random variables*.

Unnormalised Measures. Consider an experiment where we flip two fair coins B_1 and B_2 (modelled as booleans). The resulting random generator can be visualised as m_{before} in Figure 21. Suppose that, based on data, we know B_2 must be T; we can then update our belief about B_2 via a *likelihood function*, which is a function of type $\ell : \Omega \to [0,\infty)$, defined by $\ell(b_1,\mathsf{T}) \coloneqq 1$ and $\ell(b_1,\mathsf{F}) \coloneqq 0$. For any likelihood function ℓ and $m = (\mathcal{P}(\Omega),\mu) \in \mathcal{M}$, there is a natural update operation $(\cdot)$ such that $(\ell \cdot \mu) : \mathcal{P}(\Omega) \to [0,\infty]$ is a measure on $(\Omega, \mathcal{P}(\Omega))$ defined as follows:

$$(\ell \cdot \mu)(E) \coloneqq \sum_{(b_1,b_2)\in E} \ell(b_1,b_2) \cdot \mu(\{(b_1,b_2)\})$$

This update scales the measure μ according to ℓ. Let us write $(\ell \cdot m) \coloneqq (\mathcal{F}, \ell \cdot \mu)$ when $m = (\mathcal{F},\mu) \in \mathcal{M}$ as a shorthand. Then the update $m_{\text{after}} \coloneqq \ell \cdot m_{\text{before}}$ reflects our belief: the probability that $B_2 = \mathsf{F}$ is now zero since $\ell(b_1,\mathsf{F}) = 0$ and $m_{\text{after}}(\{B_2 = \mathsf{F}\}) = m_{\text{after}}(\{(\mathsf{T},\mathsf{F}),(\mathsf{F},\mathsf{F})\}) = 0$. However, there is a problem: m_{after} is no longer a probability measure because it does not add up to 100%:

$$\mu_{\text{after}}(\Omega) = \mu_{\text{after}}(\{B_2 = \mathsf{T}\}) + \mu_{\text{after}}(\{B_2 = \mathsf{F}\}) = 0 + 1/2 = 1/2$$

This means the current distribution is *unnormalised*, with the *normalising constant* being $1/1/2 = 2$. This also means $m_{\text{after}} \notin \mathcal{M}$ as $\mathcal{M}$ only includes probability spaces. To solve this problem, we extend $\mathcal{M}$ to include non-probability measures as well.

Dependent Random Variables. A far more challenging problem is to handle dependency between random variables. Consider an experiment where we toss a fair coin B_1, and depending on the result of B_1, we obtain B_2 by sampling from two different distributions:

$$B_2 \sim \begin{cases} \text{toss a biased coin that is always } \mathsf{T} & \text{if } B_1 = \mathsf{T} \\ \text{toss a fair coin} & \text{if } B_1 = \mathsf{F} \end{cases}$$

Assuming that $B_2 = \mathsf{T}$, what is the probability that $B_1 = \mathsf{T}$?

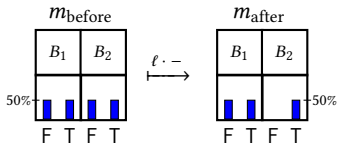


Fig. 21. Updating via likelihood function

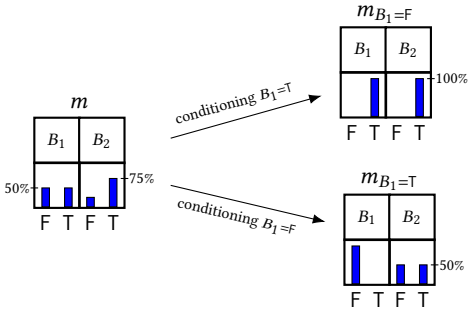
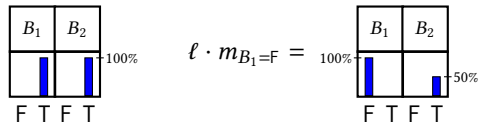


Fig. 22. A dependent random generator

Notice that B_1 and B_2 are not probabilistically independent: knowing B_1 is T means B_2 is T 100% of the time, and knowing B_1 is F means B_2 is T 50% of the time (which means overall, B_2 is T 75% of the time). Suppose $m \in \mathcal{M}$ represents this distribution; we represent the dependency between B_1 and B_2 in m pictorially in Figure 22. To see the problem, note that when we mutate the distribution of B_2 via ℓ, the distribution of B_1 changes as well: indeed, knowing $B_2 = \mathsf{T}$ makes $B_1 = \mathsf{T}$ more likely, as $B_2 = \mathsf{T}$ is more likely to be caused by flipping the biased coin that is always T, rather than the fair coin. To demonstrate how updating B_2 causes the update of B_1, we perform a 'case analysis' and apply $(\ell \cdot -)$ to the conditioned space $m_{B_1=\mathsf{T}}$ and $m_{B_1=\mathsf{F}}$:

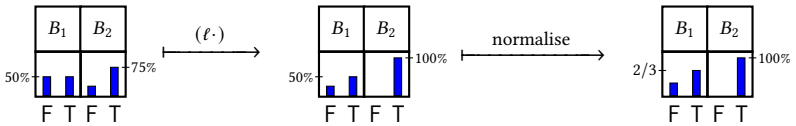
$$\ell \cdot m_{B_1=\mathsf{T}} = \quad \ell \cdot m_{B_1=\mathsf{F}} =$$



Since we now have information regarding the original distribution of B_1 (the *prior* distribution) as well as the likelihood, we can apply Bayes' theorem. Specifically, for all $b \in \{\mathsf{T}, \mathsf{F}\}$:

$$\overbrace{(\ell \cdot m)(\{B_1 = b\})}^{\text{unnormalised posterior}} = \overbrace{\mu(\{B_1 = b\})}^{\text{prior}} \cdot \overbrace{(\ell \cdot m_{B_1=b})(\{B_2 = \mathsf{T}\})}^{\text{likelihood}} = \begin{cases} 1/2 & \text{if } b = \mathsf{T} \\ 1/4 & \text{if } b = \mathsf{F} \end{cases}$$

This means $(\ell \cdot m)(\{B_1 = \mathsf{T}\}) = 1/2$ and $(\ell \cdot m)(\{B_1 = \mathsf{F}\}) = 1/4$ and $\ell \cdot m$ is an unnormalised measure ($1/2 + 1/4 \neq 1$). Normalising $\ell \cdot m$ yields the desired distribution – assuming $B_2 = \mathsf{T}$, B_1 has $1/3$ probability of being F, and $2/3$ probability of being T, as shown below:



To model these features in BASL, we develop a novel resource model for randomisation as follows:

(1) To handle unnormalised measures, we extend LILAC's partial commutative monoid by allowing non-probability measure spaces and impose several *finiteness* restrictions (Definition 4.1) and show that the resulting structure remains a partial commutative monoid (Theorem 4.3), i.e. a model of separation logic. In fact (as in LILAC), it forms an even richer structure known as a *Kripke resource monoid* (Corollary 4.5).

(2) Given an unnormalised random number generator $m \in \mathcal{M}$ and a random variable X, if the underlying space of X is a *standard Borel space* (described later in Figure 23), then the family of generators $m_{X=x}$ conditioning on $X = x$ (formally, the *disintegration* of m over X) exists (Theorem 4.7). This allows us to perform conditional reasoning: in order to reason about the dependencies between random variables, we need to reason about conditioned spaces.
(3) Random variables arising from random generators in BASL can be *updated* via a logical version of Bayes' theorem (Theorem 4.11). This result relies on BASL's *partially affine* structure (Proposition 4.9) and a theorem in disintegration theory (Lemma 4.10).

With the above theorems and guarantees, BASL admits a standard resource-theoretic semantics via a construction known as a *Kripke resource model* [Galmiche et al. 2005], which we formally develop in the rest of this section.

4.3 The Kripke Resource Model of BASL

Recall from §4.1 that our sample space was set to the booleans $\{\mathsf{T}, \mathsf{F}\}$. However, in BASL (as in LILAC) we fix the Hilbert cube $(\Omega, \Sigma_\Omega) \coloneqq ([0,1]^{\mathbb{N}}, \mathcal{B}[0,1]^{\mathbb{N}})$ as our underlying sample space, which can be thought of as the random number generator having the ability to independently generate a stream of numbers between 0 and 1. We next define the Kripke resource monoid of BASL, which intuitively comprises the unnormalised random number generators described in §4.2.

Definition 4.1. Let $\mathcal{F}$ be a σ-algebra of Ω and $\mu : \mathcal{F} \to [0, \infty]$ a measure. The pair $(\mathcal{F}, \mu)$ is a *random generator* if the following conditions hold:

(1) μ is a σ-finite measure: there exists a countable sequence $\{U_i \in \mathcal{F}\}_{i \in \mathbb{N}}$ such that $\{U_i\}_{i \in \mathbb{N}}$ covers Ω (i.e. $\bigcup_{i \in \mathbb{N}} U_i = \Omega$) and $\mu(U_i) < \infty$, for all $i \in \mathbb{N}$.
(2) μ has non-zero total measure: $\mu(\Omega) > 0$.
(3) $\mathcal{F}$ is a sub-σ-algebra of Σ_Ω: $\mathcal{F} \subseteq \Sigma_\Omega$.
(4) $\mathcal{F}$ is countably generated: there exists a countable set of subsets $\{F_i \subseteq \Omega\}_{i \in \mathbb{N}}$ such that $\mathcal{F}$ is the least σ-algebra containing $\{F_i\}_{i \in \mathbb{N}}$.
(5) $\mathcal{F}$ has a *finite footprint*: there exists an $n \in \mathbb{N}$ such that: $\forall F \in \mathcal{F}.\ \exists F' \subseteq [0,1]^n.\ F = F' \times \Omega$.

We write $\mathcal{M}$ for the set of random generators of the Hilbert cube (Ω, Σ_Ω). Condition (1) describes the space of interest – we are not only interested in finite measures, but also measures with infinite normalising constants so that we can model *improper priors*, as demonstrated in §3.5. Condition (2) is needed so that $\mathcal{M}$ forms a PCM, and it affects the way we interpret Hoare triples (§4.3). Conditions (3) and (4) are needed so that we can consider *disintegrations* of μ with respect to its random variables (Theorem 4.7). Condition (5) is needed because we want to ensure there is enough space to generate new random variables (as in LILAC [Li et al. 2023, §2.5]).

Definition 4.2 ([Li et al. 2023, Definition 2.2]). Let $(\mathcal{F}, \mu), (\mathcal{G}, \nu) \in \mathcal{M}$. Then $(\mathcal{H}, \rho)$ is the *independent combination* of $(\mathcal{F}, \mu)$ and $(\mathcal{G}, \nu)$ if $\mathcal{H}$ is the smallest σ-algebra containing $\mathcal{F}$ and $\mathcal{G}$, and for all $F \in \mathcal{F}, G \in \mathcal{G}$, $\rho(F \cap G) = \mu(F) \cdot \nu(G)$.

THEOREM 4.3 (PCM). *Let m be the independent combination of $m_1, m_2 \in \mathcal{M}$. Then $m \in \mathcal{M}$ and it is unique. Let $(\bullet) : \mathcal{M} \times \mathcal{M} \rightharpoonup \mathcal{M}$, mapping $m_1, m_2 \in \mathcal{M}$ to their independent combination if it exists, and $\mathbf{1}$ for the trivial probability space over Ω. Then $(\mathcal{M}, \bullet, \mathbf{1})$ is a partial commutative monoid.*

PROOF NOTES. The proof of uniqueness, similar to LILAC, relies on the uniqueness of measures theorem, but for σ-finite measures instead. Identity and commutativity of $(\bullet)$ follow from properties of σ-algebras. For associativity, suppose $m_1, m_2, m_3 \in \mathcal{M}$ and $m_{(12)3}$ is defined; we define $m_{23} = (\mathcal{F}_{23}, \mu_{23})$ by choosing a set $V \in \mathcal{F}_1$ satisfying $0 < \mu_1(V) < \infty$ and define $\mu_{23}(U) \coloneqq \frac{\mu_{(12)3}(V \cap U)}{\mu_1(V)}$. We then construct a λ-system $\Lambda \subseteq \mathcal{F}_{23}$ of $\mathcal{F}_{23}$-measurable sets such that $F_{23} \in \Lambda$ satisfies the property

$\mu_{(12)3}(F_1 \cap F_{23}) = \mu_1(F_1)\mu_{23}(F_{23})$ whenever $F_1 \in \mathcal{F}_1$, $\mu_1(F_1) < \infty$ and $\mu_{23}(F_{23}) < \infty$. Finally, we apply the π-λ theorem and show that all measurable sets in $\mathcal{F}_{23}$ satisfy the property and therefore establish associativity (see Ho et al. [2025, Theorem C.7] in the technical appendix for the full proof).

In a logic of bunched implications (which includes separation logics), a *Kripke resource monoid* (KRM) is the basis for providing a satisfiability relation by defining a *Kripke resource model* [Galmiche et al. 2005]. We now show that our PCM can be extended to a KRM.

Definition 4.4 ([Barthe et al. 2019, Definition 1]). A *(partial) Kripke resource monoid* is a tuple $(\mathcal{M}, \bullet, \mathbf{1}, \sqsubseteq)$ such that $(\mathcal{M}, \bullet, \mathbf{1})$ is a partial commutative monoid and $(\sqsubseteq) \subseteq \mathcal{M} \times \mathcal{M}$ is a preorder such that $(\bullet)$ is bifunctorial over $(\sqsubseteq)$; i.e. for all $m, n, m', n' \in \mathcal{M}$, if $m \sqsubseteq n$, $m' \sqsubseteq n'$ and $m \bullet m'$, $n \bullet n'$ are defined, then $m \bullet m' \sqsubseteq n \bullet n'$.

Corollary 4.5 (KRM). *Let $(\sqsubseteq) \subseteq \mathcal{M} \times \mathcal{M}$ be a preorder defined by $(\mathcal{F}_1, \mu_1) \sqsubseteq (\mathcal{F}_2, \mu_2)$ if $\mathcal{F}_1 \subseteq \mathcal{F}_2$ and $\mu_2|_{\mathcal{F}_1} = \mu_1$ (Li et al. [2023, Theorem 2.4]). Then $(\mathcal{M}, \bullet, \mathbf{1}, \sqsubseteq)$ is a Kripke resource monoid.*

A desirable property of $\mathcal{M}$ is that it is closed under conditioning of random variables. That is, suppose $(\mathcal{F}, \mu) \in \mathcal{M}$, $X : \Omega \to \mathcal{A}$ is a measurable function and π is a distribution of X generated by mutating the likelihood of its current distribution; then the space $(\mathcal{F}, \mu_x^+|_{\mathcal{F}})$ conditioning on $X = x$ (almost) always exists (Theorem 4.7). To prove this, we apply the following *Rokhlin-Simmons disintegration theorem* [Simmons 2012], which is a variant of the disintegration theorem that holds for σ-finite measures in standard Borel spaces.

Lemma 4.6 (disintegration). *Let $\mu : \Sigma_\Omega \to [0, \infty]$ be a σ-finite Borel measure, $X : \Omega \to \mathcal{A}$ a $(\Sigma_\Omega, \Sigma_A)$-measurable function and $\pi : \Sigma_A \to [0, \infty]$ a σ-finite measure that dominates the pushforward $X_*\mu$. Then there exists a π-almost-surely-unique (X, π)-disintegration $\{\mu_x\}_{x \in A}$.*

Theorem 4.7 (conditional). *Let $(\mathcal{F}, \mu) \in \mathcal{M}, X : \Omega \to \mathcal{A}$ be a $(\Sigma_\Omega, \Sigma_A)$-measurable function and $\pi : \Sigma_A \to [0, \infty]$ a σ-finite measure that dominates $X_*\mu$. Then there exists a measure $\mu^+ : \Sigma_\Omega \to [0, \infty]$ satisfying $\mu^+|_{\mathcal{F}} = \mu$. Further, the (X, π)-disintegration $\{\mu_x^+\}_{x \in A}$ of μ^+ exists and $(\mathcal{F}, \mu_x^+|_{\mathcal{F}}) \in \mathcal{M}$ for π-almost-all $x \in A$.*

Proof. Readers familiar with disintegration will note that this is a disintegration theorem in disguise. However, there are several non-trivialities. For existence of a Borel measure, since $\mu : \mathcal{F} \to [0, \infty]$ is assumed to be a countably-generated σ-finite measure on a sub-Borel σ-algebra $\mathcal{F}$, the extension μ^+ exists following from the result of Fremlin [2011, Proposition 433K]. We then apply the Rokhlin-Simmons disintegration theorem (Lemma 4.6) and obtain the conditional measure $\{\mu_x^+\}_{x \in A}$ and restrict them to the σ-algebra $\mathcal{F}$. □

4.4 Semantics of BaSL Assertions

With the Kripke resource monoid defined, we now formulate a *satisfiability relation* for BaSL assertions. In particular, we give semantics to *well-typed assertions*, where an assertion P is well-typed under context $\Gamma; \Delta$ if $\Gamma; \Delta \vdash P$, as defined in Figure 23. For instance, the assertion $P \coloneqq (X \sim \mathsf{Unif}(0, a) \Rightarrow \mathbb{E}[X] = a/2)$ is well-typed under the context $\Gamma \coloneqq \Gamma', a : \mathbb{R}$ and $\Delta \coloneqq \Delta', X : (\mathbb{R}, \mathcal{B}(\mathbb{R}))$. Notice that the map $(\gamma, a) \mapsto \mathsf{Unif}(0, a)$ is a function $[\![\Gamma]\!] \to \mathcal{G}(\mathbb{R}, \mathcal{B}(\mathbb{R}))$, which means we have $\Gamma \vdash_{\mathsf{meas}} \mathsf{Unif}(0, a) : (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ by T-ProbMeas. Then, notice that X is a name of Δ, which means $\Gamma; \Delta \vdash X \sim \mathsf{Unif}(0, 1)$ by T-Dist. Similarly, $\Gamma; \Delta \vdash \mathbb{E}[X] = x$ by T-Expectation. Combining both assertions with T-BinModality yields the correct conclusion $\Gamma; \Delta \vdash P$.

We define the semantics of BaSL assertions through the satisfiability relation in Figure 24. We write $(\gamma, D, m) \vDash P$ to denote that P holds in state m under the deterministic context γ and the list

Deterministic context: $\Gamma = [x_1 : A_1, ..., x_n : A_n]$
Probabilistic context: $\Delta = [X_1 : \mathcal{A}_1, ..., X_n : \mathcal{A}_n]$

$$\textsc{T-RandExpr}\ \frac{E \in [\![\Gamma]\!] \to \mathbf{Meas}([\![\Delta]\!], \mathcal{A})}{\Gamma; \Delta \vdash_{\mathsf{re}} E : \mathcal{A}} \qquad \textsc{T-DetExpr}\ \frac{e \in [\![\Gamma]\!] \to A}{\Gamma \vdash_{\mathsf{de}} e : A} \qquad \textsc{T-Meas}\ \frac{\pi \in [\![\Gamma]\!] \to M_\sigma(\mathcal{A})}{\Gamma \vdash_{\mathsf{meas}} \pi : \mathcal{A}}$$

$$\textsc{T-True}\ \frac{}{\Gamma; \Delta \vdash \top} \qquad \textsc{T-False}\ \frac{}{\Gamma; \Delta \vdash \bot}$$

$$\textsc{T-BinModality}\ \frac{\Gamma; \Delta \vdash P \qquad \Gamma; \Delta \vdash Q}{\Gamma; \Delta \vdash P \odot Q} \qquad \textsc{T-DetQuant}\ \frac{\Gamma, x : A; \Delta \vdash P}{\Gamma; \Delta \vdash \boldsymbol{Q} x : A.P} \qquad \textsc{T-RandQuant}\ \frac{\Gamma; \Delta, X : \mathcal{A} \vdash P}{\Gamma; \Delta \vdash \boldsymbol{Q}_{\mathsf{rv}} X : \mathcal{A}.P}$$

$$\textsc{T-Expectation}\ \frac{\Gamma; \Delta \vdash_{\mathsf{re}} E : (\mathbb{R}, \mathcal{B}\mathbb{R}) \qquad \Gamma; \Delta \vdash_{\mathsf{de}} e : \mathbb{R}}{\Gamma; \Delta \vdash \mathbb{E}[E] = e}$$

$$\textsc{T-Dist}\ \frac{\Gamma; \Delta \vdash_{\mathsf{re}} E : \mathcal{A} \qquad \Gamma \vdash_{\mathsf{meas}} \pi : \mathcal{A}}{\Gamma; \Delta \vdash E \sim \pi} \qquad \textsc{T-Ownership}\ \frac{\Gamma; \Delta \vdash_{\mathsf{re}} E : \mathcal{A}}{\Gamma; \Delta \vdash \mathsf{own}\, E}$$

$$\textsc{T-Conditioning}\ \frac{\Gamma; \Delta \vdash_{\mathsf{re}} E : \mathcal{A} \qquad \Gamma \vdash_{\mathsf{meas}} \pi : \mathcal{A} \qquad \Gamma, x : A; \Delta \vdash P}{\Gamma; \Delta \vdash x \overset{\pi}{\leftarrow} E \mid P}$$

$$\textsc{T-Likelihood}\ \frac{\Gamma \vdash_{\mathsf{de}} e : [0, \infty)}{\Gamma; \Delta \vdash \mathsf{L}(e)} \qquad \textsc{T-Hoare}\ \frac{\Gamma; \Delta \vdash P \qquad \Gamma; \Delta \vdash_{\mathsf{prog}} M : \mathcal{A} \qquad \Gamma; \Delta, X : \mathcal{A} \vdash Q}{\Gamma; \Delta \vdash \{P\}\, M\, \{X : \mathcal{A}.Q\}}$$

where A ranges over sets, $\mathcal{A}$ ranges over standard Borel spaces, $\odot \in \{\wedge, \vee, \Rightarrow, *, -\!\!*\}$, $\boldsymbol{Q} \in \{\forall, \exists\}$ and $M_\sigma(\mathcal{A})$ is the set of σ-finite measures over $\mathcal{A}$.

Fig. 23. Typing judgements for BaSL assertions

of random variables D. The semantics of propositional, first-order and separation logic connectives and quantifiers ($\wedge$, $\vee$, $\Rightarrow$, $*$, $-\!\!*$, $\forall$, $\exists$) is standard. In order for BaSL to be sound, the 'custom' probabilistic propositions, $\mathsf{own}\, E$, $\mathsf{L}(e)$, $\mathbb{E}[E] = e$, $x \overset{\pi}{\leftarrow} E \mid P$ and $\{P\}\, M\, \{X.Q\}$, must satisfy *Kripke monotonicity* as follows.

Proposition 4.8 (Kripke monotonicity). *Let $m \sqsubseteq m'$, $\Gamma; \Delta \vdash P$, $\gamma \in [\![\Gamma]\!]$ and $D \in \mathsf{RV}[\![\Delta]\!]$. Then $(\gamma, D, m) \vDash P$ implies $(\gamma, D, m') \vDash P$.*

We next explain the intuitive meaning of the custom probabilistic assertions in BaSL. The *ownership* $\mathsf{own}\, E$ and *distribution* $E \sim \pi$ assertions are expressed via measurability of random variables. Specifically, $(\gamma, D, m) \vDash \mathsf{own}\, E$ holds iff $E_{\gamma,D} \coloneqq E(\gamma) \circ D$ (the expression E applied to (γ, D)) is an $\mathcal{F}$-measurable function. Similarly, $(\gamma, D, m) \vDash E \sim \pi$ holds iff $E_{\gamma,D}$ is an $\mathcal{F}$-measurable function and the pushforward of E with respect to μ is π ($\pi(\gamma) = ((E_{\gamma,D})_* \mu)$). The semantics of the expected value assertion $\mathbb{E}[E] = e$ follows its usual interpretation in statistics: a random expression has expected value e_γ if $E_{\gamma,D}$ integrates (with respect to the probability measure μ) to e_γ.

The remaining three assertions, namely the *likelihood proposition* $\mathsf{L}(e)$, the *conditioning modality* $x \overset{\pi}{\leftarrow} E \mid P$ and the *Hoare triple* $\{P\}\, M\, \{X.Q\}$ have non-trivial semantics, as we describe below.

Likelihood $\mathsf{L}(e)$. Recall from §2 that $\mathsf{L}(e)$ asserts that the current state has *likelihood* e. Indeed, a state $(\mathcal{F}, \mu) \in \mathcal{M}$ is more *likely* if the total measure $\mu(\Omega)$ is higher. Hence, $(\mathcal{F}, \mu)$ has likelihood $e(\gamma)$ when $\mu(\Omega) = e(\gamma)$. In fact, the proposition NormConst (mentioned in §2) is simply defined as having a non-zero likelihood k:

$$\mathsf{NormConst} \coloneqq \exists k : (0, \infty).\mathsf{L}(k)$$

$(\gamma, D, m) \vDash \top$	always	
$(\gamma, D, m) \vDash \bot$	never	
$(\gamma, D, m) \vDash P \wedge Q$	$\overset{\text{def}}{\iff}$	$(\gamma, D, m) \vDash P$ and $(\gamma, D, m) \vDash Q$
$(\gamma, D, m) \vDash P \vee Q$	$\overset{\text{def}}{\iff}$	$(\gamma, D, m) \vDash P$ or $(\gamma, D, m) \vDash Q$
$(\gamma, D, m) \vDash P \Rightarrow Q$	$\overset{\text{def}}{\iff}$	for all $m' \sqsupseteq m$, $(\gamma, D, m) \vDash P$ implies $(\gamma, D, m') \vDash Q$
$(\gamma, D, m) \vDash P * Q$	$\overset{\text{def}}{\iff}$	there exists $m_1 \bullet m_2 \sqsubseteq m$ such that $(\gamma, D, m_1) \vDash P$, $(\gamma, D, m_2) \vDash Q$
$(\gamma, D, m) \vDash P \mathrel{-\!\!*} Q$	$\overset{\text{def}}{\iff}$	if $m' \bullet m$ is defined then $(\gamma, D, m') \vDash P$ implies $(\gamma, D, m' \bullet m) \vDash Q$
$(\gamma, D, m) \vDash \forall x : A.P$	$\overset{\text{def}}{\iff}$	for all $x \in A$, $((\gamma, x), D, m) \vDash P$
$(\gamma, D, m) \vDash \exists x : A.P$	$\overset{\text{def}}{\iff}$	for some $x \in A$, $((\gamma, x), D, m) \vDash P$
$(\gamma, D, m) \vDash \forall_{\text{rv}} X : \mathcal{A}.P$	$\overset{\text{def}}{\iff}$	for all $X \in \mathsf{RV}(\mathcal{A})$, $(\gamma, (D, X), m) \vDash P$
$(\gamma, D, m) \vDash \exists_{\text{rv}} X : \mathcal{A}.P$	$\overset{\text{def}}{\iff}$	for some $X \in \mathsf{RV}(\mathcal{A})$, $(\gamma, (D, X), m) \vDash P$
$(\gamma, D, m) \vDash \mathsf{own}\, E$	$\overset{\text{def}}{\iff}$	$E(\gamma) \circ D$ is $\mathcal{F}$-measurable
$(\gamma, D, m) \vDash E \sim \pi$	$\overset{\text{def}}{\iff}$	$E(\gamma) \circ D$ is $\mathcal{F}$-measurable and $\pi(\gamma) = (E(\gamma) \circ D)_* \mu$
$(\gamma, D, m) \vDash \mathbb{E}[E] = e$	$\overset{\text{def}}{\iff}$	$(\gamma, D, (\mathcal{F}, \mu)) \vDash \mathsf{L}(1)$ and $\int_\Omega E(\gamma) \circ D \, \mathrm{d}\mu = e(\gamma)$
$(\gamma, D, m) \vDash \mathsf{L}(e)$	$\overset{\text{def}}{\iff}$	$\mu(\Omega) = e(\gamma)$
$(\gamma, D, (\mathcal{F}, \mu)) \vDash x \overset{\pi}{\leftarrow} E \mid P$	$\overset{\text{def}}{\iff}$	$(\gamma, D, (\mathcal{F}, \mu)) \vDash \mathsf{own}\, E$ and $X_* \mu$ is absolutely continuous with respect to π and for all measures $\mu^+ : \Sigma_\Omega \to [0, \infty]$ satisfying $\mu^+|_{\mathcal{F}} = \mu$ and all disintegrations $\{\mu^+_x\}_{x \in \mathcal{A}}$ of μ^+ along (X, π) and for π-almost-all $x \in A$, $((\gamma, x), D, (\mathcal{F}, \mu^+_x|_{\mathcal{F}})) \vDash P$ where $X \coloneqq E(\gamma) \circ D$
$(\gamma, D, m) \vDash \{P\}\, M\, \{X : \mathcal{A}.Q\}$	$\overset{\text{def}}{\iff}$	for all $m_{\text{pre}} \in \mathcal{M}$ with $(\gamma, D, m_{\text{pre}}) \vDash P$, $m_{\text{fr}} \in \mathcal{M}$ with $(\mathcal{F}_0, \mu_0) \coloneqq m_{\text{pre}} \bullet m_{\text{fr}}$ defined, measures $\mu^+_0 : \Sigma_\Omega \to [0, \infty]$ with $\mu^+_0|_{\mathcal{F}_0} = \mu_0$, if $\mu^+_0(\{\omega \in \Omega \mid [\![M_\gamma]\!](D(\omega), \mathcal{A})\}) > 0$, then there exists (1) $X \in \mathsf{RV}(\mathcal{A})$, (2) $m_{\text{post}} \in \mathcal{M}$ with $(\mathcal{F}_1, \mu_1) \coloneqq m_{\text{post}} \bullet m_{\text{fr}}$ defined, and (3) measure $\mu^+_1 : \Sigma_\Omega \to [0, \infty]$ with $\mu^+_1|_{\mathcal{F}_1} = \mu_1$ s.t. $(\gamma, (D, X), m_{\text{post}}) \vDash Q$, and for all $f : \Omega \to \mathcal{B}$ and $U \in \Sigma_{\mathcal{B}} \otimes \Sigma_{\mathcal{A}}$, $\int_\Omega [\![M_\gamma]\!](D(\omega), \{x \in A \mid (f(\omega), x) \in U\})\, \mu^+_0(\mathrm{d}\omega) = \mu^+_1(\{\omega \in \Omega \mid (f(\omega), X(\omega)) \in U\})$

Fig. 24. Semantics of BaSL assertions

Note that $\mathsf{L}(1)$ constitutes the multiplicative unit (unit of $*$) in BaSL. Specifically, let $(\gamma, D, (\mathcal{F}, \mu)) \vDash X \sim \mathbb{P}$ for a probability measure $\mathbb{P}$; then $(\mathcal{F}, \mu)$ satisfies $\mathsf{L}(1)$ as the probability measure is by definition normalised. Recall that in a Kripke resource model, the multiplicative unit is a proposition I satisfying the following for any KRM $(\mathcal{M}, \bullet, \mathbf{1}, \sqsubseteq)$ [Galmiche et al. 2005, Definition 2.5]:

$$\text{for all } m \in \mathcal{M},\ m \vDash I \iff e \sqsubseteq m$$

Unfolding $\sqsubseteq$ in our KRM, we know that if I is the multiplicative unit and $(\gamma, D, (\mathcal{F}, \mu)) \vDash I$, then $\mu(\Omega) = 1$. Hence, $\mathsf{L}(1)$ is the unit and we define $\top_1 \coloneqq \mathsf{L}(1)$. Recall that a proposition P *entails* Q (written $P \vdash Q$) if $(\gamma, D, m) \vDash P$ implies $(\gamma, D, m) \vDash Q$, for all (γ, D, m). With $\top_1$ being the multiplicative unit, we then know the bi-entailment $P * \top_1 \dashv\vdash P$ holds.

BaSL is partially affine. The fact that $\top_1$ is the multiplicative unit has significant implications on BaSL's reasoning rules. In particular, this means we cannot *forget* about 'unnormalised propositions'. Recall that a separation logic is *affine* if the entailment $P * Q \vdash P \wedge Q$ holds [Galmiche et al. 2005]. For example, the Iris separation logic is affine [Jung et al. 2018], and we can e.g. forget about

pointers via the entailments $(x \hookrightarrow v * y \hookrightarrow v') \vdash (x \hookrightarrow v \wedge y \hookrightarrow v') \vdash x \hookrightarrow v$, i.e. given two addresses x and y, weakening $*$ to $\wedge$ allows us to forget information about y. On the other hand, a separation logic is *linear* (or *boolean*) when the multiplicative unit is only satisfied by the empty element $\mathbf{1} \in \mathcal{M}$, i.e. when $m \vDash I$ implies $m = \mathbf{1}$ [Galmiche et al. 2005]. Consequently, the entailment $P * Q \vdash P \wedge Q$ does not always hold, and we cannot forget information. In fact, this is the original approach to separation logic taken by O'Hearn and Pym [1999].

BaSL is neither affine nor linear – the entailment $P * Q \vdash P \wedge Q$ holds in certain restricted cases. Indeed, BaSL is *partially affine*, a category of logics first described by Charguéraud [2020], whereby only a class of (not necessarily all) assertions are designated as *affine*, namely those that can be 'dropped'. In BaSL, we define an assertion to be affine, written $\mathsf{affine}(P)$, as follows:

$$\mathsf{affine}(P) \overset{\text{def}}{\iff} \text{for all } \gamma, D, \mathcal{F}, \mu,\ (\gamma, D, (\mathcal{F}, \mu)) \vDash P \text{ implies } \mu(\Omega) = 1$$

Intuitively, this means only normalised assertions (e.g. $X \sim \mathbb{P}$ for a probability measure $\mathbb{P}$) are affine and assertions with unnormalised components (e.g. $\mathsf{L}(e)$ when $e \neq 1$, or $\mathsf{NormConst}$) are not affine. Affine assertions enjoy the property that they can be dropped, as stated in Proposition 4.9 below.

Proposition 4.9 (affine assertions). *The following entailments hold:*

$$\frac{\mathsf{affine}(Q)}{P * Q \vdash P}\ E\text{-}*\text{-}\textsc{Weak}_1 \qquad \frac{\mathsf{affine}(P)}{P * Q \vdash Q}\ E\text{-}*\text{-}\textsc{Weak}_2 \qquad \frac{\mathsf{affine}(P) \qquad \mathsf{affine}(Q)}{P * Q \vdash P \wedge Q}\ E\text{-}*\text{-}\textsc{Weak}$$

Conditioning $x \overset{\pi}{\leftarrow} E \mid P$. As explained in §2, the conditioning assertion $x \overset{\pi}{\leftarrow} E \mid P$ lets us use the assertion P to describe the behaviour of a state μ assuming $E = x$ and $(\gamma, D, \mathcal{F}, \mu) \vDash E \sim \pi$. Its semantics is as follows. To show that P holds conditionally for almost every x, we require that $(\gamma, D, \mathcal{F}, \mu_x^+) \vDash P$ holds for almost all x, where μ^+ is a Borel measure extending μ and μ_x^+ is the measure μ^+ conditioned on $E = x$. The Borel measure extension condition is a technical condition to ensure the existence of the conditioned space $\{\mu_x^+\}_{x \in A}$, which are a family of σ-finite measures that exist by Theorem 4.7. We can now see how the conditioning modality interacts with the likelihood assertion $\mathsf{L}(e)$ by revisiting BayesCoin. Suppose $X \sim \mathsf{Unif}(0,1)$ and we assert the likelihood of $X = x$ to be x as in the example of Figure 4; then the state satisfies the proposition $x \overset{\mathsf{Unif}(0,1)}{\longleftarrow} X \mid \mathsf{L}(x)$. The resulting distribution is proportional to the $\mathsf{Beta}(2,1)$ distribution because Bayes' theorem states that $\Pr[X \in U \mid \mathit{Flip}_1 = 1] = \int_0^1 \mathbf{1}_U(x) \cdot 2x \, \mathrm{d}x = \mathsf{Beta}(U \mid 2, 1)$. Now, in order to derive the desired postcondition ($X \sim \mathsf{Beta}(2,1)$ with a normalising constant), we need to internalise a notion of Bayes' theorem within BaSL.

The Internal Bayes' Theorem. In disintegration theory, the following theorem states that the Radon-Nikodym derivative of the two absolutely-continuous measures $X_*\mu \ll \pi$ is almost surely equal to the total measure of a π-disintegration (Lemma 4.10). Using this result, we can internalise Bayes' theorem as a BaSL bi-entailment (Theorem 4.11).

Lemma 4.10 ([Chang and Pollard 1997, Theorem 2]). *Let $\{\mu_x\}_{x \in A}$ be an (X, π)-disintegration of μ. Then π dominates the pushforward $X_*\mu$ and for π-almost-all $x \in A$, $\frac{\mathrm{d}X_*\mu}{\mathrm{d}\pi}(x) = \mu_x(\Omega)$.*

Theorem 4.11 (internal Bayes' theorem). *For any Borel measurable function $f : \mathcal{A} \to [0, \infty)$ and random expression $\Gamma; \Delta \vdash_{\mathsf{re}} E : \mathcal{A}$, the following bi-entailment holds:*

$$\overbrace{x \overset{\pi}{\leftarrow} E}^{\text{prior}} \mid \overbrace{\mathsf{L}(f(x))}^{\text{likelihood}} \dashv\vdash \overbrace{E \sim f \cdot \pi}^{\text{posterior}}$$

Proof. Let $m = (\mathcal{F}, \mu) \in \mathcal{M}$, (γ, D, m) be a configuration and $X(\omega) \coloneqq E(\gamma)(D(\omega))$. To prove the $\vdash$ direction, we assume $(\gamma, D, (\mathcal{F}, \mu)) \vDash x \overset{\pi}{\leftarrow} E \mid \mathsf{L}(f(x))$. Then for any Borel extension μ^+ and

(X, π)-disintegration $\{\mu_x^+\}_{x\in A}$ we know $(\gamma, D, (\mathcal{F}, \mu_x^+|_{\mathcal{F}})) \vDash \mathsf{L}(f(x))$ holds for π-almost-all $x \in A$. This implies $\mu_x^+(\Omega) = f(x)$ holds for π-almost-all $x \in A$, which implies for all $F \in \Sigma_A$:

$$X_*\mu(F) = X_*\mu^+(F) = \int_A \mu_x^+(X^{-1}(F))\,\pi(\mathrm{d}x) = \int_F \mu_x^+(\Omega)\,\pi(\mathrm{d}x) = \int_F f\,\mathrm{d}\pi = (f\cdot\pi)(F).$$

The first equality follows from our assumption that μ^+ is a Borel extension of μ. The second equality is a disintegration axiom; the third holds because $\mathbf{1}_F(x)\mu_x^+(\Omega) = X_*\mu_x^+(F)$; and the last holds because for almost all x, we have $(\gamma, D, (\mathcal{F}, \mu_x^+|_{\mathcal{F}})) \vDash \mathsf{L}(f(x))$. Hence, we know $(\gamma, D, (\mathcal{F}, \mu)) \vDash E \sim f\cdot\pi$.

To prove the $\dashv$ direction, we assume $(\gamma, D, (\mathcal{F}, \mu)) \vDash E \sim f\cdot\pi$ holds; then $(\gamma, D, (\mathcal{F}, \mu)) \vDash \mathsf{own}\, E$ by definition of $\vDash$. Next, for any Borel extension μ^+ and (X, π)-disintegration $\{\mu_x^+\}_{x\in A}$, we know $(\gamma, D, (\mathcal{F}, \mu_x^+|_{\mathcal{F}})) \vDash \mathsf{L}(f(x))$ holds for π-almost-all $x \in A$ because for any $F \in \Sigma_A$:

$$\int_F \mu_x^+(\Omega)\,\pi(\mathrm{d}x) = \int_F \frac{\mathrm{d}X_*\mu}{\mathrm{d}\pi}\,\mathrm{d}\pi = \int_F \frac{\mathrm{d}(f\cdot\pi)}{\mathrm{d}\pi}\,\mathrm{d}\pi = \int_F f\,\mathrm{d}\pi.$$

The first equality follows from Lemma 4.10; the second follows from our assumption. Note that $X_*\mu = f\cdot\pi$ is absolutely continuous with respect to π as $(f\cdot\pi)(F) = 0$ for any π-null-set F. □

Hoare Triple $\{P\}\ M\ \{X.Q\}$. The interpretation of a Hoare triple in BaSL is similar to that of Lilac, but with one key difference: a BaSL Hoare triple denotes *partial correctness*. A *partial correctness triple* $\{P\}\ M\ \{Q\}$ states that if a state satisfies P and executing M from that state *terminates*, then the resulting state satisfies Q. One reason of assuming termination is due to its undecidability. Similarly, determining whether a probabilistic program has a zero normalising constant is undecidable Staton [2020, §2.2.4] – this is not a relevant concern for existing probabilistic logics (including Lilac) as they do not support Bayesian updating. As such, BaSL triples denote partial correctness. Specifically, $\{P\}\ M\ \{X.Q\}$ holds iff starting from a state $m_{\mathrm{pre}} \coloneqq (\mathcal{F}_{\mathrm{pre}}, \mu_{\mathrm{pre}})$ satisfying the precondition P and an arbitrary frame m_{fr} (such that $m_{\mathrm{pre}} \bullet m_{\mathrm{fr}}$ is defined), if '$[\![M]\!]$ normalises to a non-zero constant from $m_{\mathrm{pre}} \bullet m_{\mathrm{fr}}$', then there must exist a random variable X and a postcondition state $m_{\mathrm{post}} \coloneqq (\mathcal{F}_{\mathrm{post}}, \mu_{\mathrm{post}})$ that satisfies Q and is compatible with the frame m_{fr} such that $m_{\mathrm{post}} \bullet m_{\mathrm{fr}}$ is also defined and 'executing M from m_{pre} results in m_{post}'.

To express that '$[\![M]\!]$ normalises to a non-zero constant from $m_{\mathrm{pre}} \bullet m_{\mathrm{fr}}$', we require that $\mu_0^+(\{\omega \in \Omega \mid [\![M_\gamma]\!](D(\omega), \mathcal{A})\}) > 0$, where μ_0^+ is a Borel extension of $m_{\mathrm{pre}} \bullet m_{\mathrm{fr}}$. This means there exists a measurable set of Ω with non-zero-μ_0^+ measure such that executing $[\![M_\gamma]\!]$ with random variables D generated from the seed ω leads to a non-zero normalising constant. Similar to Lilac, we quantify over arbitrary extensions of random variables D_{ext} to prove a substitution lemma (see Lemma D.4).

To express 'executing M from the state m_{pre} results in m_{post}', we require that the following hold for any measurable set U:

$$\int_\Omega [\![M_\gamma]\!](D(\omega), \{x \mid (D_{\mathrm{ext}}(\omega), D(\omega), x) \in U\})\,\mu_0^+(\mathrm{d}\omega) = \mu_1^+\{\omega \mid (D_{\mathrm{ext}}(\omega), D(\omega), X(\omega)) \in U\}.$$

where μ_0^+ is a Borel measure that extends $m_{\mathrm{pre}} \bullet m_{\mathrm{fr}}$, and μ_1^+ is a Borel measure that extends $\mu_{\mathrm{post}} \bullet m_{\mathrm{fr}}$. Comparing the BaSL semantics to the standard denotational semantics of Bppl in the category of s-finite kernels, the behaviour of the Hoare triple can be characterised by Proposition 4.12. Intuitively, suppose μ_0^+ is a state that satisfies precondition P and μ_1^+ is a state that satisfies the postcondition Q. Then the resulting measure of $[\![M_\gamma]\!]$ with random variables D along the random source μ_0^+ is equal to the 'output' random variable X with the random source μ_1^+, as stated below.

Proposition 4.12. *Let* $(\gamma, D, \mu) \vDash P$, $\mathcal{MA}$ *be the space of all measures of* $\mathcal{A}$, *and* $[\![-]\!] : \mathbf{Syn} \to \mathbf{sfKrn}$ *be the semantic functor defined in Ho et al. [2025, §B]. Suppose* $\Delta \vdash M : \tau$ *and* $\{P\}\ M\{X.Q\}$

holds with respect to μ_0^+ and μ_1^+. Then the following diagram commutes:

$$\begin{array}{ccccc} \mathcal{M}\Omega & \xrightarrow{D_*} & \mathcal{M}[\![\Delta]\!] & \xrightarrow{[\![M_\gamma]\!]_*} & \mathcal{M}[\![\tau]\!] \\ {\scriptstyle \mu_0^+}\uparrow & & & & \| \\ \mathbf{1} & \xrightarrow[\mu_1^+]{} & \mathcal{M}\Omega & \xrightarrow[X_*]{} & \mathcal{M}[\![\tau]\!] \end{array}$$

We finally show that the BaSL proof system in Figure 9 is sound, with the full proof given in Ho et al. [2025, §E]. We write $\vDash \{P\}\, M\, \{X.Q\}$ to denote that $(\gamma, D, m) \vDash \{P\}\, M\, \{X.Q\}$ holds for all (γ, D, m).

Theorem 4.13 (soundness). *The BaSL proof system is* sound*: for all P, M, Q, if $\vdash \{P\}\, M\{X.Q\}$ is derivable using the rules in Figure 9, then $\vDash \{P\}\, M\, \{X.Q\}$ holds.*

5 Conclusions, Related and Future Work

We developed BaSL by extending probabilistic separation logic to verify Bayesian programs/statistical models. To this end, we devised a semantic model rich enough to encode probabilistic programming concepts such as conditional distributions, unnormalised distributions, Bayesian updating and improper priors. We then demonstrated the utility of BaSL by proving properties such as correlation, expected values and posterior distributions in various statistical models.

Related Work: Semantics of BPPLs. The semantics of randomised languages is well-established: starting from the seminal work of Kozen [1981] on linear operator semantics, there are numerous works on semantic domains for probability and non-determinism, e.g. by Jones and Plotkin [1989]. These led to the study of Bayesian inference from a programming language theory perspective [Gordon et al. 2014; van de Meent et al. 2021] with research on their operational [Borgström et al. 2016] and denotational semantics [Dahlqvist and Kozen 2019; Huot et al. 2023]. Based on the theory of concrete sheaves [Matache et al. 2022], Heunen et al. [2017]; Vákár et al. [2019] developed a category known as (ω-)quasi-Borel space (**Qbs**) for reasoning about higher-order, recursive Bayesian probabilistic programs and proved the existence of a strong monad of s-finite measures for a monadic semantics of higher-order BPPLs. The BaSL proof system is underpinned by the category of s-finite kernels developed by Staton [2017].

Related Work: Probabilistic Separation Logics. Separation logic (SL) developed a *modular* theory for reasoning about computational resources [O'Hearn and Pym 1999] for pointer-manipulating programs. This led to the development of abstract models for SL [Calcagno et al. 2007; Galmiche et al. 2005; Jung et al. 2018]. Barthe et al. gave a probabilistic interpretation of SL in Psl (probabilistic SL) and proved the correctness of algorithms such as the one-time pad cipher.

One of the overarching themes of probabilistic separation logics is the investigation of *conditional probability*. Bao et al. [2021a] developed DIBI based on bunched implications (BI) by extending Psl with the ';' connective for describing conditional independence of random variables, while Bao et al. [2021b] developed Lina for reasoning about negative dependence of random variables. Li et al. [2023] proposed Lilac, a measure-theoretic interpretation of probabilistic SL that supports desirable features such as continuous distributions and 'mathematical' random variables. Lilac handles conditional independence by introducing the *conditioning modality*, which was later adopted by BlueBell [Bao et al. 2025] (a relational probabilistic SL) and psOL [Zilberstein et al. 2024] (a concurrent probabilistic program logic). Recently, Li et al. [2024] developed a categorical model of Lilac by drawing an analogy between random sampling and fresh name generations in the theory of nominal sets. BaSL generalises the model of Lilac and brings a novel perspective of conditional

Table 2. Features of probabilistic separation/bunched logics

	PSL	DIBI	LINA	LILAC	PSOL	BLUEBELL	BASL
Discrete distribution	✓	✓	✓	✓	✓	✓	✓
Probabilistic independence	✓	✓	✓	✓	✓	✓	✓
Continuous distribution				✓			✓
Negative dependence			✓				
Conditional independence		✓		✓	✓	✓	✓
Conditioning modality				✓	✓	✓	✓
Concurrency					✓		
Relational reasoning						✓	
Bayesian reasoning							✓

reasoning by supporting Bayesian updating, hence giving an axiomatic semantics of probabilistic programming via SL. We summarise the features of probabilistic separation/BI logics in Table 2.

Related Work: Program Logic for Bayesian Conditioning. Based on quasi-Borel spaces, Sato et al. [2019] derived a family of program logics known as PPV. Their semantic insight is that the logical assertions in the category **Qbs** corresponds to fibrations, and used a technique known as *categorical* $\top\top$*-lifting* [Katsumata et al. 2018] to give semantics to predicates and derive logics for a condtioning construct known as **query**. By contrast, our model is more closely inspired by measure theory, and we use a resource monoid of a well-behaved subset of σ-finite measure spaces over $[0,1]^{\mathbb{N}}$ to give semantics to assertions. An interesting direction of future work is to determine whether our definition of Hoare triples is an instance of categorical $\top\top$-lifting. Practically, while PPV can handle higher-order functions, our approach, apart from a first-class handling of probabilistic independence, has the following advantages: (1) BASL has a general modality for expressing conditional distributions: Sato et al. [2019] develop a program logic on top of quasi-Borel spaces (**Qbs**), while the probability theory of **Qbs** is very promising, it is still under active development. For example, consider the (quasi-Borel) space of measurable real functions $\mathbb{R}^{\mathbb{R}}$, it is currently an open question whether conditioning makes sense for $\mathbb{R}^{\mathbb{R}}$. Our logic uses standard measure spaces, for which the probability theory is well-established. This allows us to derive a general conditioning modality compatible with Bayesian update. As a consequence, our logic can, via a first-class modality, express conditional probability distributions, which is useful for expressing properties such as conditional expectation/independence. (2) Also, random variables (in the sense of measurable functions from the sample space) are first-class obejcts in BASL, as opposed to variable names in PPV. An immediate implication is that programs in PPV need to be written in a specific form to retain information about its distribution via the **query** construct.

Future Work. We will consider three avenues of future research. First, we will mechanise BASL and its soundness proof in a theorem prover such as Rocq. Second, we will extend BASL to support more sophisticated language features including (1) mutable states; (2) recursion; and (3) higher-order functions. These extensions will allow us to express programs in a class of statistical models known as *Bayesian nonparametric models* [Mak et al. 2021; Orbanz and Teh 2017], namely statistical models with an unbounded number of random variables [van de Meent et al. 2021]. Finally, we will apply BASL to develop program simplification/symbolic execution tools for probabilistic programs.

Acknowledgments

We thank the POPL 2026 reviewers and the members of the Veritas Lab and the Functional Programming Group at Imperial for their constructive feedback. We thank John Li for helpful discussions on the technical aspects of the Lilac separation logic. We thank Alberto Croquevielle for his insights on the measure-theoretic components of the proofs. Azalea Raad is supported by a UKRI fellowship MR/V024299/1, by the EPSRC grant EP/X037029/1, and by VeTSS.

References

Sheldon Axler. 2019. *Measure, Integration & Real Analysis*. Springer International Publishing. doi:10.1007/978-3-030-33143-6

Jialu Bao, Simon Docherty, Justin Hsu, and Alexandra Silva. 2021a. A Bunched Logic for Conditional Independence. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–14. doi:10.1109/LICS52264.2021.9470712

Jialu Bao, Emanuele D'Osualdo, and Azadeh Farzan. 2025. Bluebell: An Alliance of Relational Lifting and Independence for Probabilistic Reasoning. *Proc. ACM Program. Lang.* 9, POPL, Article 58 (Jan. 2025), 31 pages. doi:10.1145/3704894

Jialu Bao, Marco Gaboardi, Justin Hsu, and Joseph Tassarotti. 2021b. A Separation Logic for Negative Dependence. *CoRR* abs/2111.14917 (2021). arXiv:2111.14917 https://arxiv.org/abs/2111.14917

Gilles Barthe, Justin Hsu, and Kevin Liao. 2019. A Probabilistic Separation Logic. *CoRR* abs/1907.10708 (2019). arXiv:1907.10708 http://arxiv.org/abs/1907.10708

Patrick Billingsley. 1995. *Probability and measure* (3. ed ed.). Wiley, New York [u.a.].

Johannes Borgström, Ugo Dal Lago, Andrew Gordon, and Marcin Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. *SIGPLAN Not.* 51, 9 (Sept. 2016), 33–46. doi:10.1145/3022670.2951942

Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. 2011. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC. doi:10.1201/b10905

Cristiano Calcagno, Peter O'Hearn, and Hongseok Yang. 2007. Local Action and Abstract Separation Logic. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*. 366–378. doi:10.1109/LICS.2007.30

Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software* 76, 1 (2017), 1–32. doi:10.18637/jss.v076.i01

Joseph Chang and David Pollard. 1997. Conditioning as disintegration. *Statistica Neerlandica* 51, 3 (1997), 287–317. doi:10.1111/1467-9574.00056

Arthur Charguéraud. 2020. Separation logic for sequential programs (functional pearl). *Proc. ACM Program. Lang.* 4, ICFP, Article 116 (Aug. 2020), 34 pages. doi:10.1145/3408998

Ulices Santa Cruz and Yasser Shoukry. 2022. NNLander-VeriF: A Neural Network Formal Verification Framework for Vision-Based Autonomous Aircraft Landing. arXiv:2203.15841 [cs.LG] https://arxiv.org/abs/2203.15841

Marco Cusumano-Towner, Feras Saad, Alexander Lew, and Vikash Mansinghka. 2019. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*. 221–236. doi:10.1145/3314221.3314642

Fredrik Dahlqvist and Dexter Kozen. 2019. Semantics of higher-order probabilistic programs with conditioning. *Proc. ACM Program. Lang.* 4, POPL, Article 57 (Dec. 2019), 29 pages. doi:10.1145/3371125

David Fremlin. 2011. *Measure Theory: Topological Measure Spaces. Volume 4*. Torres Fremlin.

Didier Galmiche, Daniel Méry, and David Pym. 2005. The semantics of BI and resource tableaux. *Mathematical Structures in Computer Science* 15, 6 (2005), 1033–1088. doi:10.1017/S0960129505004858

Stuart Geman and Donald Geman. 1984. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6, 6 (1984), 721–741. doi:10.1109/TPAMI.1984.4767596

Andrew Gordon, Thomas Henzinger, Aditya Nori, and Sriram Rajamani. 2014. Probabilistic Programming. In *Proceedings of the on Future of Software Engineering*. ACM, 167–181. doi:10.1145/2593882.2593900

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A convenient category for higher-order probability theory *(LICS '17)*. IEEE Press, Article 77, 12 pages. doi:doi/10.5555/3329995.3330072

Shing Hin Ho, Nicolas Wu, and Azalea Raad. 2025. Bayesian Separation Logic. arXiv:2507.15530 [cs.PL] https://arxiv.org/abs/2507.15530

Mathieu Huot, Alexander Lew, Vikash Mansinghka, and Sam Staton. 2023. ωPAP Spaces: Reasoning Denotationally About Higher-Order, Recursive Probabilistic and Differentiable Programs. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–14. doi:10.1109/LICS56636.2023.10175739

Claire Jones and Gordon Plotkin. 1989. A probabilistic powerdomain of evaluations. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*. IEEE Press, 186–195. doi:10.1109/LICS.1989.39173

Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming* 28 (2018), e20. doi:10.1017/S0956796818000151

Shin-ya Katsumata, Tetsuya Sato, and Tarmo Uustalu. 2018. Codensity Lifting of Monads and its Dual. *Logical Methods in Computer Science* Volume 14, Issue 4, Article 6 (Oct 2018). doi:10.23638/LMCS-14(4:6)2018

Dexter Kozen. 1981. Semantics of probabilistic programs. *J. Comput. System Sci.* 22, 3 (1981), 328–350. doi:10.1016/0022-0000(81)90036-2

Peter M. Lee. 2012. *Bayesian Statistics: An Introduction* (4th ed.). Wiley Publishing.

John 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

John Li, Jon Aytac, Philip Johnson-Freyd, Amal Ahmed, and Steven Holtzen. 2024. A Nominal Approach to Probabilistic Separation Logic. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '24)*. Article 55, 14 pages. doi:10.1145/3661814.3662135

Carol Mak, Fabian Zaiser, and Luke Ong. 2021. Nonparametric Hamiltonian Monte Carlo. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 7336–7347. https://proceedings.mlr.press/v139/mak21a.html

Cristina Matache, Sean Moss, and Sam Staton. 2022. Concrete categories and higher-order recursion: With applications including probability, differentiability, and full abstraction *(LICS '22)*. Article 57, 14 pages. doi:10.1145/3531130.3533370

Richard McElreath. 2020. *Statistical Rethinking: A Bayesian Course with Examples in R and STAN* (2nd ed.). Chapman and Hall/CRC. doi:10.1201/9780429029608

Natalia Muehlemann, Tianjian Zhou, Rajat Mukherjee, Munshi Imran Hossain, Satrajit Roychoudhury, and Estelle Russek-Cohen. 2023. A Tutorial on Modern Bayesian Methods in Clinical Trials. *Therapeutic Innovation & Regulatory Science* 57, 3 (2023), 402–416. doi:10.1007/s43441-023-00515-3

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic inference by program transformation in Hakaru (system description). In *International Symposium on Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*. Springer, 62–79. doi:10.1007/978-3-319-29604-3_5

Peter O'Hearn and David Pym. 1999. The Logic of Bunched Implications. *Bulletin of Symbolic Logic* 5, 2 (1999), 215–244. doi:10.2307/421090

Peter Orbanz and Yee Teh. 2017. *Bayesian Nonparametric Models*. 107–116. doi:10.1007/978-1-4899-7687-1_928

John Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS '02)*. IEEE Computer Society, USA, 55–74. doi:10.1109/LICS.2002.1029817

Tetsuya Sato, Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Justin Hsu. 2019. Formal verification of higher-order probabilistic programs: reasoning about approximation, convergence, Bayesian inference, and optimization. *Proc. ACM Program. Lang.* 3, POPL, Article 38 (Jan. 2019), 30 pages. doi:10.1145/3290351

David Simmons. 2012. Conditional measures and conditional expectation; Rohlin's Disintegration Theorem. *Discrete and Continuous Dynamical Systems* 32 (07 2012). doi:10.3934/dcds.2012.32.2565

Natalia Slusarz, Ekaterina Komendantskaya, Matthew L. Daggitt, and Robert Stewart. 2022. Differentiable Logics for Neural Network Training and Verification. arXiv:2207.06741 [cs.AI] https://arxiv.org/abs/2207.06741

Sam Staton. 2017. Commutative Semantics for Probabilistic Programming. In *Programming Languages and Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 855–879. doi:10.1007/978-3-662-54434-1_32

Sam Staton. 2020. Probabilistic Programs as Measures. In *Foundations of Probabilistic Programming*. Cambridge University Press, 43–74. doi:10.1017/9781108770750.003

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. doi:10.1145/3290377

David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. 2016. Design and Implementation of Probabilistic Programming Language Anglican. *CoRR* abs/1608.05263 (2016). arXiv:1608.05263 http://arxiv.org/abs/1608.05263

Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming. *Proc. ACM Program. Lang.* 3, POPL, Article 36 (Jan. 2019), 29 pages. doi:10.1145/3290349

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2021. An Introduction to Probabilistic Programming. arXiv:1809.10756 [stat.ML] https://arxiv.org/abs/1809.10756

Matthijs Vákár and Luke Ong. 2018. On S-Finite Measures and Kernels. arXiv:1810.01837 [math.PR] https://arxiv.org/abs/1810.01837

Noam Zilberstein, Alexandra Silva, and Joseph Tassarotti. 2024. Probabilistic Concurrent Reasoning in Outcome Logic: Independence, Conditioning, and Invariants. arXiv:2411.11662 [cs.LO] https://arxiv.org/abs/2411.11662

Received 2025-07-10; accepted 2025-11-06