First Steps Towards Probabilistic Iris: A Separation Logic with Independence, Conditioning, and Dynamic Heap Allocation

Janine Lohse
MPI-SWS
Germany
jlohse@mpi-sws.org

Tim Rohde
MPI-SWS and
Saarland University
Germany
timrohde@mpi-sws.org

Jimmy Xin
MPI-SWS
Germany
jimmyxin@mpi-sws.org

Niklas Mück
MPI-SWS
Germany
mueck@mpi-sws.org

Iona Kuhn
CISPA Helmholtz Center
for Information Security
Germany
iona.kuhn@cispa.de

Derek Dreyer
MPI-SWS
Germany
dreyer@mpi-sws.org

Deepak Garg
MPI-SWS
Germany
dg@mpi-sws.org

Emanuele D'Osualdo
University of Konstanz
Germany
emanuele.dosualdo@uni-konstanz.de

Abstract

There has recently been exciting progress in the realm of *probabilistic separation logics*. An important subclass of these—including PSL, Lilac, Bluebell, and pcOL—are *general-purpose probabilistic logics* (or GPLs, for short), meaning that they provide primitive Hoare-style assertions about probability distributions on the program state, along with powerful modularity principles like *independence* and *conditioning*. However, none of these logics support reasoning about dynamically allocated memory (*i.e.,* pointers into a heap), let alone the more sophisticated resource algebra-based ghost state of modern separation logics like Iris. We argue that this is due to a fundamental obstacle: since the shape of memory (and identity of memory locations) may differ under different random outcomes, it is unclear how pointer ownership can be harmonized with probabilistic independence and conditioning. Furthermore, none of the existing GPLs that support both independence and conditioning have, to our knowledge, been mechanized in a proof assistant.

In this paper, we take a substantial first step towards a marriage of GPLs and modern separation logics like Iris, in the form of **Amaryllis**. Amaryllis is the first GPL to support independence and conditional reasoning while also handling dynamic memory allocation. To overcome the aforementioned obstacle, we propose a new *indexed valuation*-style model of probabilistic assertions, whereby ownership of a standard Iris-style resource (*e.g.,* heaps) can be promoted to ownership at the level of distributions in a generic fashion. We then show how to adapt the central Iris notions of *frame-preserving update*, *authoritative resource algebras*, and the *weakest precondition modality* to be sound for probabilistic reasoning and validate dynamic allocation. Finally, we have mechanized all our results in the Rocq proof assistant and developed an Iris-based proof mode for conducting proofs within Amaryllis.

1 Introduction

The past few years have seen a flurry of work on program logics for reasoning modularly about probabilistic programs. Roughly speaking, these probabilistic logics can be divided into two types, which may be understood as "property-specific" vs. "general-purpose". *Property-specific probabilistic logics* (or PPLs) [1, 22, 13, 11, 12, 18] encode various specific probabilistic properties of interest (*e.g.,* error bounds, expected time complexity, etc.) and then derive rules for reasoning about them within the framework of standard Hoare or separation logics [26, 25] (*e.g.,* Iris [17, 16]). *General-purpose probabilistic logics* (or GPLs) [7, 9, 20, 5, 32] instead provide native support for program predicates describing *distributions* over program states, as well as logical connectives describing *independence* and *conditioning* of distributions, neither of which are directly expressible in the property-specific logics.

Thus far, the PPLs have exhibited the advantage that, since they are built from (in some sense) more standard parts (*e.g.,* defining some appropriate form of ghost state and applying the standard Iris methodology), they are easier to construct and soundly apply in the context of more fully-featured programming languages (*e.g.,* with higher-order state). In contrast, the GPLs have the potential advantage of being more general and expressive, supporting a wider range of modular probabilistic reasoning principles in one logic.

However—notably—none of the existing GPLs support reasoning about *dynamically allocated memory* (*i.e.,* pointers into a heap), let alone the more sophisticated resource algebra-based ghost state of modern separation logics like Iris. This may come as a surprise, given that dynamically allocated pointers were the *raison d'être* of the original version of separation logic [25]. But there is good reason for it: multiple, fundamental technical challenges must be overcome in order to harmonize GPLs' support for independence and conditioning—implemented using a non-standard model of separating conjunction—with the *resource*-based (*e.g.,* heap) model of separation in traditional separation logics.

As a long-term goal, we aim to explore a synthesis of PPLs and GPLs in the form of Probabilistic Iris: an adaptation of the Iris framework that would support rich programming language features and modern separation logic features *alongside* primitive assertions about probability distributions. In this paper, we take a substantial first step towards that goal with **Amaryllis**, the first GPL to support reasoning about Iris-style resources, including dynamically allocated memory (heaps). Though we use heaps as a motivation, Amaryllis's approach is not actually specific to heap resources at

Conference'17, Washington, DC, USA
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

all: the logic is parametric in a (non-step-indexed) Iris-style resource algebra describing stateful, non-probabilistic resources, and shows how to generically lift such a resource algebra to a model of probabilistic assertions supporting independence, conditioning, and modular reasoning about state. This generic lifting requires a fundamental rethink of various aspects of existing GPL models. To build added confidence in the resulting constructions, we have fully mechanized Amaryllis in the proof assistant (Rocq/Iris).

Contributions. We make the following contributions:

(1) We observe that existing GPL models—which interpret assertions as predicates on distributions over program states—run into a major road block when we incorporate dynamic memory allocation because they do not account for the possibility that the shape of memory (and identity of memory locations) may differ under different random outcomes.
(2) We resolve this problem by proposing an *indexed-valuation*-style model for GPLs in which probabilistic resources comprise two parts: (1) a probability distribution on *outcomes* (*i.e.,* results of coin flips), and (2) a mapping from outcomes to separation logic resources (*e.g.*, heaps). In this model, separating conjunction is defined *per random outcome*, rather than requiring disjointness of heaps across all outcomes as prior models do. Consequently, it allows different random outcomes to result in differently shaped heaps, enabling a clean accounting of dynamic heap allocation.
(3) On top of this probabilistic resource model, we build a program logic, Amaryllis, which supports key reasoning principles of prior GPLs—notably, independence and conditional reasoning—while also supporting dynamic allocation.
(4) Amaryllis is formalized in the Iris framework. Towards that end, we show for the first time how to adapt central concepts in Iris—such as "frame-preserving update" and "the authoritative resource algebra", which are used in turn to derive a weakest-precondition modality—so that they remain sound under probabilistic (and conditional) reasoning.
(5) We demonstrate the effectiveness of Amaryllis on a range of illustrative examples.
(6) We have mechanized all results of this paper in the Rocq proof assistant, and also developed an Iris-based proof mode for conveniently conducting proofs within Amaryllis. As advised by the program chairs, we have made the (anonymized) proof repository available online [3].

Non-goals, limitations, and future work. In order to focus attention on the core problem of handling dynamic heap allocation and Iris-style resources, we place several restrictions on the probabilistic setting and the types of programs we consider. Specifically, we restrict attention to terminating programs (as Lilac [20] and Bluebell [5] do) and discrete probability distributions (as Bluebell [5] and pcOL [32] do). We also do not attempt to incorporate Iris's mechanisms for step-indexing [4, 2], higher-order ghost state [15], and concurrency/invariants [17]. Lastly, for simplicity, we only consider distributions with finite support. In future work, we expect that it should be possible to generalize Amaryllis to handle countably infinite support and almost-sure termination of unbounded loops, but that some of Amaryllis's proof rules for conditional reasoning will not apply to continuous distributions. Despite present restrictions, the mechanization of Amaryllis required a significant effort, including approximately 50K lines of Rocq proof.

2 Key Ideas

In this section we give an overview of the main contributions of Amaryllis.

2.1 State-of-the-Art Reasoning in GPLs

Amaryllis's aim is to provide a modular and expressive program logic for stateful probabilistic languages, able to prove properties of the state distribution induced by such programs. Example of properties are: bounds on the probability of certain events, or on expected values, the exact distribution of a certain variable, the (conditional) probabilistic independence of two variables.

The main judgments of general-purpose probabilistic logics (GPLs) are Hoare triples of the form $\{P\}\ e\ \{V.\, Q(V)\}$. Here, e is a probabilistic expression, the semantics of which can be seen as a function taking a distribution over state and returning a joint distribution of state and return value. The *precondition* P is an assertion constraining the input distribution, while the *postcondition* Q constrains the output distribution. A *random* T *variable* (for some type T) is a function[1] from the outcomes of a distribution to T. The postcondition is parametric on V, which is a random value variable representing the return value. As an example, $\{X \sim \mathsf{Ber}_{1/2}\}\ e\ \{V.\, V \sim \mathsf{Ber}_{1/2}\}$ states that the program e returns a value distributed as a fair coin if X is initially distributed as a fair coin.

Amaryllis builds on previous work, starting from the seminal PSL [9] and further developed in Lilac [20], Bluebell [5], and pcOL [32]. The success of these logics lies in modular reasoning, which is enabled by two crucial ideas: (1) *probabilistic independence* as separation, and (2) *conditioning* as a modality.

Independence as separation. PSL introduced the idea that separating conjunction $P * Q$ can model probabilistic independence in a logic of assertions over distributions. For example, an assertion $X \sim \mathsf{Ber}_{1/2} * Y \sim \mathsf{Ber}_{1/2}$ does not just state that both X and Y are distributed as fair coins, but also that they are *independent*: knowing the outcome of one does not reveal any information about the outcome of the other. More precisely, the probability of seeing both v in X and w in Y is the product of the two constituent probabilities.

Internalizing independence as separation introduces a very useful principle of *locality*, expressed as the Frame rule of Separation Logic:

$$\textsc{Frame}\quad \frac{\{P\}\ e\ \{V.\, Q(V)\}}{\{P * R\}\ e\ \{V.\, Q(V) * R\}}$$

Suppose, for instance, that we have $\{\mathsf{True}\}\ e\ \{X.\, X \sim \mathsf{Ber}_{1/2}\}$; then by Frame we can prove $\{Y \sim \mathsf{Ber}_{1/2}\}\ e\ \{X.\, X \sim \mathsf{Ber}_{1/2} * Y \sim \mathsf{Ber}_{1/2}\}$ without having to verify e again from the larger precondition. Since generating/consuming independent variables is commonplace in probabilistic programs, framing turns out to be extremely valuable.

[1] Random variables are usually required to be measurable in some ambient probability space. To avoid confusion we will explicitly state, when relevant, if we assume them to be measurable, and in which space they are.

Conditioning as a modality. However, framing alone is not enough to achieve modularity in the probabilistic setting. Imagine we are provided with the triple

$$\forall b \in \mathbb{B}.\{\lceil \ell \mapsto b \rceil\}\ f(\ell)\ \{\lceil \ell \mapsto \neg b \rceil\} \tag{1}$$

where $\lceil \cdot \rceil$ states that the predicate inside is true with probability 1. Here the contents of location ℓ are assumed to be the deterministic value b. Intuitively, we should be able to prove that if the contents of ℓ are distributed as a fair coin, written $\ell \rightsquigarrow \mathrm{Ber}_{1/2}$, then

$$\{\ell \rightsquigarrow \mathrm{Ber}_{1/2}\}\ f(\ell)\ \{\ell \rightsquigarrow \mathrm{Ber}_{1/2}\}. \tag{2}$$

holds. But the frame rule alone is not sufficient to derive (2) from (1)—indeed, this derivation is not possible in the original PSL. Intuitively, we need to merge the two instances of (1) (for the two choices of b) into a single one by blending them with half probability each. To represent such "blending" at the level of assertions, one can define an operator $P \oplus_q Q$ which, roughly speaking, should hold on a distribution if it can be seen as the sum weighted by $q \in [0,1]$ and $1-q$ of two distributions which satisfy P and Q respectively. With this addition we can formulate a second principle of locality—*outcome locality*—in the form of the following rule:

$$\frac{\{P_1\}\ e\ \{Q_1\} \qquad \{P_2\}\ e\ \{Q_2\}}{\{P_1 \oplus_q P_2\}\ e\ \{Q_1 \oplus_q Q_2\}}\ \oplus\text{-LIFT}$$

In our example, one can establish that $\ell \rightsquigarrow \mathrm{Ber}_{1/2} \equiv \lceil \ell \mapsto 0 \rceil \oplus_{1/2} \lceil \ell \mapsto 1 \rceil$ and (2) would then follow from $\oplus$-LIFT.

This kind of reasoning is an instance of a more general pattern: *reasoning under conditioning*. Often in probabilistic proofs it is necessary to perform case analysis on some random variable X (like the value at ℓ in the example). This formally corresponds to proving something about a distribution μ, by considering the family of *conditional distributions* $\mu|_{X=v}$ (*i.e.,* the restriction of μ to the case where X is the deterministic value v), one for each possible value v.

The Lilac logic [20] proposed to represent conditioning using a *conditioning modality*, which can be used as a generalization of $\oplus_q$. In Amaryllis we directly adopt this idea, building on later variants of the modality defined for discrete distributions in [5, 32]. Intuitively, Amaryllis's conditioning modality $\mathbf{C}_{x \leftarrow \mu} P(x)$ holds if there is a random variable X distributed as μ such that for every $v \in \mathrm{supp}(\mu)$, $P(v)$ holds on the distribution conditioned on $X = v$; for example, $\lceil \ell \mapsto 0 \rceil \oplus_q \lceil \ell \mapsto 1 \rceil$ can be expressed as $\mathbf{C}_{b \leftarrow \mathrm{Ber}_q} \lceil \ell \mapsto b \rceil$. The modality $\mathbf{C}_{x \leftarrow \mu} P(x)$ leaves implicit (*i.e.,* existentially quantified) which variable X is being conditioned; to make X apparent, it is sufficient to write $P(x)$ such that $\forall x.\, P(x) \vdash \lceil X = x \rceil$.

The growing body of work on probabilistic separation logics provides convincing evidence that these principles are key for making reasoning scale. Yet, existing logics either do not support mutable state (Lilac) or only support a finite variable store (PSL, Bluebell, pcOL). This is not by accident: as we now explain, dynamic heap allocation poses serious challenges for the techniques developed so far.

2.2 The Challenge of Dynamic Allocation

The core of the issue with previous models and dynamic allocation is the difficulty of harmonizing in a single logic *independence* and *disjointness* of state.

Lilac uses separation exclusively for *independence*, with the consequence of only supporting immutable state. This is because frames can record too much information, which needs to be preserved for Frame to hold. For example $X \sim \mathrm{Ber}_{1/2} * \lceil X \in \{0,1\} \rceil$ is satisfiable, so owning $X \sim \mathrm{Ber}_{1/2}$ alone is not enough to be able to mutate X.

In logics that support mutation, one needs to also embed in the model information about disjointness of locations. In all the logics with mutation (except for Bluebell which we address later), the independence and disjointness are conflated in the definition of separating conjunction: a distribution of states μ satisfies $P * Q$ just if there are two *disjoint* sets of locations L_1, L_2 such that for some μ_1 distribution of memories of L_1 satisfying P and some μ_2 distribution of memories of L_2 satisfying Q we have $\mu = \mu_1 \otimes \mu_2$ (here $\otimes$ is the product of distributions).

This definition does not work for dynamic allocation. To illustrate why that is, we assume a deterministic allocator, to show that the core issue arises equally in this simpler setting. In the technical development, Amaryllis only assumes a non probabilistic but nondeterministic allocator, obtaining an even stronger result.

What makes dealing with allocation hard is the necessary dependency of the allocator on the current heap (as it needs, at the very least, to know which locations are available). Since the size and shape of the heap may depend on the previous random choices, the specific locations picked by the allocator will also depend on previous random choices.

To see the issue consider the following program:

$$\begin{aligned} \mathit{dfl} \triangleq\ & \texttt{let \_ = (if flip(1/2) then ref 0) in} \\ & \texttt{(ref flip(1/2), ref flip(1/2))} \end{aligned} \tag{3}$$

Depending on an initial coin flip, the heap may grow by one location before the two returned locations are allocated. Ideally, we would like to prove a postcondition that asserts that we end up with two disjoint heap locations storing the results of two independent coins:

$$\{\mathsf{True}\}\ \mathit{dfl}\ \{(X,Y).\, X \rightsquigarrow \mathrm{Ber}_{1/2} * Y \rightsquigarrow \mathrm{Ber}_{1/2}\} \tag{4}$$

Unfortunately, this triple does not hold (semantically) in the model sketched above. The reason is that it is impossible to partition the locations into L_1, L_2 so that X only gets locations from L_1 and Y from L_2, across all random outcomes. For instance, starting from an empty heap, if the first flip is 0 the two returned locations would be $(\texttt{0x0}, \texttt{0x1})$, but would be $(\texttt{0x1}, \texttt{0x2})$ if the first flip was 1. Therefore, although in any particular execution the two locations would be different, when considering the whole distribution at once, the possible locations associated with X and the ones with Y overlap in $\texttt{0x1}$.

Bluebell proposed a fractional-permission-based way of decoupling ownership of locations and independence; their solution, however, is specific to static variable stores, and does not obviously generalize to heaps with dynamic allocation. Moreover, the Bluebell logic only supports the $\oplus$-LIFT rule with heavy side conditions which disable the use of frame around it, hindering modularity. (See Section 6 for details.)

2.3 A Model of Per-Outcome Disjointness

We solve the problem from the previous section by defining disjointness of heap resources *per random outcome*, rather than insisting on disjointness of heaps across all outcomes.

To realize this idea, we propose a new model of probabilistic separation logic which is parametric on a standard resource algebra. Specifically, given a standard resource algebra M (*e.g.,* heaps) formalizing the desired notion of memory ownership (*e.g.,* disjoint union of heaps), we show how to build a resource algebra representing *distributions* of elements of M so that can make assertions about distributions of random variables, probabilistic conditional independence, and ownership of memory.

Separation on the standard resources M is represented as a partial commutative binary operator $\cdot$. For the case of heaps, the resources are finite maps h from locations to values, and $h_1 \cdot h_2$ is defined as disjoint union $h_1 \uplus h_2$ if the two maps have disjoint domains, otherwise the composition is undefined. The crucial question is: what is the correct way of lifting the operation $\cdot$ to an operation $\hat{\cdot}$ on *distributions* of resources $\mathbb{D}(M)$? A natural choice is to make $\mu_1 \mathbin{\hat{\cdot}} \mu_2$ only defined if the composition $r_1 \cdot r_2$ is defined for *every* $r_1 \in \mathrm{supp}(\mu_1)$ and $r_2 \in \mathrm{supp}(\mu_2)$ (*i.e.,* for every pair of resources that have non-zero probability). This choice gives rise to the model of PSL and pcol.

Given our diagnosis of the problem with allocation, the issue with the above $\hat{\cdot}$ is that it considers pairs of resources that do not live in the same branch of the random choices made by the program. The domain $\mathbb{D}(M)$ does not however record the choices explicitly, making it impossible to fix the situation.

In Amaryllis, we therefore move to a different representation of distributions of resources, which is a variant of so-called *indexed valuations* [28], constructed as follows. We assume an arbitrary countable set of *random choice identifiers* $\iota \in \mathit{Rid}$. We then fix the outcome space to $\Omega = \mathit{Rid} \to \mathbb{B}$ of maps from random choice identifiers to booleans:[2] these abstractly represent the coin tosses performed by the program so far. At a high level, we define a probabilistic resource to be a pair $(\mathcal{P}, R)$ where $\mathcal{P}$ is a probability space over outcomes Ω, and $R \colon \Omega \to M$ is a *random resource variable* (which is *not* required to be measurable in $\mathcal{P}$). Thanks to the fact that we have an abstract representation of the outcomes of random choices as $\rho \in \Omega$, all our resources are now indexed in R by the specific random branch they belong to. This allows us to define $(\mathcal{P}_1, R_1) \mathbin{\hat{\cdot}} (\mathcal{P}_2, R_2)$ as the pair $\left(\mathcal{P}_1 \circledast \mathcal{P}_2, (\lambda\rho.\, R_1(\rho) \cdot R_2(\rho))\right)$, that is: the probability spaces are composed using the independent product $\circledast$ of Lilac, which encodes probabilistic independence, while the stateful resources (*e.g.,* heaps) are composed *outcome-wise*.

In the Amaryllis model, we can give the desired meaning to the triple in (4). The assertion $X \rightsquigarrow \mathrm{Ber}_{1/2}$ is given a meaningful semantics by letting $X \colon \Omega \to \mathit{Loc}$ be a *random location variable*. Given some $\mu \in \mathbb{D}(\mathit{Val})$, the assertion $X \rightsquigarrow \mu$ holds on $(\mathcal{P}, R)$ if the function $\lambda\rho.\, R(\rho)(X(\rho))$ is measurable in $\mathcal{P}$ and is distributed as μ in $\mathcal{P}$. In particular, $X(\rho)$ needs to be a location owned by $R(\rho)$, for every ρ not deemed impossible in $\mathcal{P}$. Then the triple in (4) expresses exactly that (i) outcome-wise, the two functions X and Y are different locations; and (ii) they point at values distributed as independent fair coins.

[2] Although Ω is uncountable, the probability spaces we consider only contain information about a finite set $R \subseteq \mathit{Rid}$. We make this precise in Section 3.

2.4 Probabilistic Specifications in Amaryllis

Now that we have an appropriate model of distributions of heaps, we want to build a program logic. Consider a possible axiom for allocation:

$$\textsc{hoare-alloc-v}\qquad \{\mathsf{True}\}\ \mathtt{ref}\, v\ \{L.\, L \mapsto v\}$$

The rule uses Amaryllis's "points-to" assertion, which is defined in general as follows. Given $L \colon \Omega \to \mathit{Loc}$ and $V \colon \Omega \to \mathit{Val}$, the assertion $L \mapsto V$ holds on a resource $(\mathcal{P}, R)$ just if $L(\rho) \in \mathrm{dom}(R(\rho))$ and $R(\rho)(L(\rho)) = V(\rho)$ for all $\rho \in \mathrm{supp}(\mathcal{P})$. That is, the assertion $L \mapsto V$ implies per-outcome ownership of the *heap resource* $L(\rho) \mapsto V(\rho)$, but crucially it *does not* imply ownership of the *distribution* of either L or V. This is because (i) in the model, R is not required to be measurable in $\mathcal{P}$, and (ii) the assertion only constrains R, not $\mathcal{P}$.

Notice in particular how if $L \mapsto V$ were to own the distribution of L, it would be disastrous. As explained in Section 2.2, L is potentially dependent on (*i.e.,* correlated with) every past random choice. Hence, if $L \mapsto V$ asserted something about the distribution of L, the postcondition of $\mathtt{ref}(v)$ would not be independent of the program context, and the specification of $\mathtt{ref}(v)$ would not be frame-preserving. By confining $L \mapsto V$ to describing heap and not distribution ownership, we ensure that the specification enjoys the frame rule. The assertion $L \rightsquigarrow \mu$ which we used in Section 2.1 can now be defined as shorthand for $\exists V.\, L \mapsto V * V \sim \mu$.

The HOARE-ALLOC-V triple is valid in Amaryllis, but how could we reuse it in a context where the initial value stored at the allocated location is not a deterministic value v but rather a *random* value variable V? To allow for this, Amaryllis makes a natural generalization: our triples are in fact not over single expressions $e \in \mathit{Expr}$, but over random expression variables $E \colon \Omega \to \mathit{Expr}$. This allows us to directly prove:

$$\textsc{hoare-alloc}\qquad \{\mathsf{True}\}\ \mathtt{ref}\, V\ \{L.\, L \mapsto V\}$$

where $\mathtt{ref}\, V$ stands for $(\lambda\rho.\, \mathtt{ref}\, V(\rho))$. HOARE-ALLOC subsumes HOARE-ALLOC-V by choosing $V = \lambda\_.\, v$, and precisely describes the effect of the command without requiring ownership of the distribution of V; in the case where we do initially own $V \sim \mu$, by Frame we can also get $\{V \sim \mu\}\ \mathtt{ref}\, V\ \{L.\, L \mapsto V * V \sim \mu\}$.

Using our triples, we are able to provide a very general version of Iris's "bind" rule—a functional version of the Hoare Logic rule of sequential composition:

$$\textsc{hoare-bind}\qquad \frac{\{P\}\ E\ \{W.\, P'(W)\} \qquad \forall W.\{P'(W)\}\ K[W]\ \{V.\, Q(V)\}}{\{P\}\ K[E]\ \{V.\, Q(V)\}}$$

Here $K[\,\cdot\,]$ is an evaluation context, adapted to work on random expression variables. Using HOARE-BIND we can introduce random variables for intermediate results; for example, to handle $\mathtt{ref}\,(\mathtt{flip}(q))$ we can first bind on the flip and then apply the rules for allocation as sketched before:

$$\frac{\{\mathsf{true}\}\ \mathtt{flip}(q)\ \{V.\, V \sim \mathrm{Ber}_q\} \qquad \{V \sim \mathrm{Ber}_q\}\ \mathtt{ref}\, V\ \{L.\, L \mapsto V * V \sim \mathrm{Ber}_q\}}{\{\mathsf{true}\}\ \mathtt{ref}\,(\mathtt{flip}(q))\ \{L.\, L \mapsto V * V \sim \mathrm{Ber}_q\}}$$

2.5 Probabilistic Frame-Preserving Updates

We have already discussed the importance of the ⊕-LIFT rule for supporting outcome-local reasoning. Changing the model as we do in Amaryllis, however, presents a serious challenge: it is unclear how and why the ⊕-LIFT rule should hold. In fact, PCOL supports ⊕-LIFT by relying on the simplifying effect of conflating disjointness and independence; BLUEBELL only supports the rule by disabling further framing via a heavy side-condition.

In Amaryllis, ⊕-LIFT is in fact a special case of the C-LIFT rule:

C-LIFT

$$\frac{\forall x \in \mathrm{supp}(\mu).\ \{P(x)\}\ E\ \{V.\ Q(x)(V)\}}{\left\{\mathbf{C}_{x\leftarrow\mu}.P(x)\right\}\ E\ \left\{V.\ \mathbf{C}_{x\leftarrow\mu}.Q(x)(V)\right\}}$$

We show that C-LIFT is valid in Amaryllis, by strengthening the notion of frame-preservation embedded in the meaning of triples to a new notion we call *probabilistic frame preservation*. We achieve the result in a modular fashion, following the Iris methodology, which prescribes triples and their rules to be derived from the combination of more primitive modalities and their rules: triples are described in terms of a *weakest precondition* (WP) modality $\mathsf{wp}\ e\ \{\Phi\}$ which, in turn, is built around the *update* modality $\Rrightarrow P$. Below, we explain these components, and how we modified them in Amaryllis.

Intuitively, Iris's WP modality $\mathsf{wp}\ e\ \{\Phi\}$ holds on a resource a if every execution of e from a results in a return value v and resource b which satisfies the postcondition $\Phi(v)$. In addition, $\mathsf{wp}\ e\ \{\Phi\}$ insists that every possible frame of a is preserved by e. Using this modality, a triple $\{P\}\ e\ \{\Phi\}$ can be encoded as $P \vdash \mathsf{wp}\ e\ \{\Phi\}$.

A crucial aspect of WP is that it transforms the current resource into a new resource that satisfies the postcondition, while preserving all frames (thus validating FRAME). This property is represented by the valid entailment $P * \mathsf{wp}\ e\ \{\Phi\} \vdash \mathsf{wp}\ e\ \{v.\ P * \Phi(v)\}$. In Iris, WP is itself a derived formula, which uses the so-called *update modality* $\Rrightarrow P$ which, roughly speaking, holds on a resource a if P holds on some resource b such that every frame of a is also a valid frame of b (*i.e.,* we can "update" a to b so that P holds without "breaking" any frame of a). The frame property of WP is thus a reflection of the frame property of updates $P * \Rrightarrow Q \vdash \Rrightarrow (P * Q)$.

In Amaryllis, we similarly derive the definition of WP from an update modality. In order for triples to validate C-LIFT, the WP needs to satisfy the C-WP-SWAP rule:

C-WP-SWAP

$$\mathbf{C}_{x\leftarrow\mu}\ \mathsf{wp}\ E\ \{V.\ Q(x)(V)\} \vdash \mathsf{wp}\ E\ \left\{V.\ \mathbf{C}_{x\leftarrow\mu}\ Q(x)(V)\right\}$$

The rule says that WP and conditioning commute; consider for example the case where $\mu = \mathsf{Ber}_q$. The rule states that, if E produces a distribution satisfying $Q(b)$ (for $b \in \{0,1\}$) from the current distribution conditioned on $X = b$ for some (implicit) X, then E, run from the unconditional distribution, produces a distribution that satisfies $Q(0) \oplus_q Q(1)$.

In the same way the frame property of WP follows from the one on update, the proof of C-WP-SWAP boils down to proving a similar rule for update:

C-UPD-SWAP

$$\mathbf{C}_{x\leftarrow\mu} \Rrightarrow P(x) \vdash \Rrightarrow \mathbf{C}_{x\leftarrow\mu}\ P(x)$$

Unfortunately, if we only require the update to be frame-preserving, as is done in standard Iris, C-UPD-SWAP would not hold. The reason is that there are frame-preserving updates which do not commute with conditioning. In particular, conditioning on some random variable X (in some $\mathcal{P}$) is only possible if X is measurable in $\mathcal{P}$. Consequently, all updates performed under branches of this conditioning must at least preserve the measurability of X or else the conditioning ceases to be meaningful, and yet some frame-preserving updates do not!

Consider for example updating some $\mathcal{P}_0$ to $\mathcal{P}_1$ where X is measurable in $\mathcal{P}_0$ but not in $\mathcal{P}_1$. "Forgetful" updates like this are always frame-preserving, because if every event that is measurable locally is independent of every event that the frame can measure, then if we make fewer events measurable locally, the frame would remain independent of the local resource. The update, however, *does not preserve conditioning*: conditioning on X in some $\mathcal{P}$ is only possible if X is measurable in $\mathcal{P}$, and so all updates performed in branches of this conditioning must at least preserve the measurability of X or else the conditioning ceases to be meaningful. To see why, observe the case where we start from some $\mathcal{P}$ where X is measurable, and distributed as a fair coin. If we condition on X we would have $\mathcal{P} = ½ \cdot \mathcal{P}|_{X=0} + ½ \cdot \mathcal{P}|_{X=1}$. But if in the first branch of the conditioning we update $\mathcal{P}|_{X=0}$ to $\mathcal{P}'_0$ by making X not measurable, then X would no longer be measurable in $\mathcal{P}' = ½ \cdot \mathcal{P}'_0 + ½ \cdot \mathcal{P}|_{X=1}$, causing the conditioning to become senseless.

This is already a problem without considering dynamic allocation: for instance, in PCOL triples only insist on frame preservation, so the logic was forced to impose side conditions that exclude updates like the one above in crucial rules for conditioning.

In Amaryllis, we propose a new definition of the update modality, which imposes stronger constraints on the allowed updates such that both the frame property of updates and C-UPD-SWAP are validated. The idea is to require updates to also preserve the validity of any possible conditioning they might be part of. We call such updates *probabilistic frame-preserving* (or PFP). For this new notion, we show that:

(1) Any standard frame-preserving update on elements of M (the underlying standard resources) can be lifted to a PFP update on the random resource variable component of our model. As special cases, we get that mutation and dynamic allocation of heap cells are PFP.
(2) Taking a new sample from a distribution is a PFP update to the probability space component of the model.

2.6 Adapting Iris's "Authoritative" Resource Algebra to the Probabilistic Setting

To fully derive WP within the logic on top of update, standard Iris makes essential use of a construction called the "authoritative resource algebra" $\mathsf{Auth}(M)$. In essence, $\mathsf{Auth}(M)$ consists of two kinds of elements, built on top of the elements of an underlying resource algebra M (*e.g.,* heaps): (1) the *authority* $\bullet g$, which gives exclusive ownership of a resource $g \in M$ representing the "global" view of the state, and (2) the *fragment* $\circ a$, which gives ownership of a resource $a \in M$ representing local ownership of a piece of the state. Fragments compose: $\circ a_1 \cdot \circ a_2 = \circ(a_1 \cdot_M a_2)$. Exclusivity of the authority is represented by the fact that $\bullet g_1 \cdot \bullet g_2$ is undefined,

and thus in any valid separation of resources there is at most one authority. Crucially, $\bullet g \cdot \circ a$ is only valid if a is a "sub-resource" of g (a concept specified in the definition of M).

In Amaryllis we build our model with an authoritative construction on top of the resource algebra we sketched in Section 2.3. Intuitively, fragments $\circ(\mathcal{P}, R)$ support the meaning of the user-facing assertions like $L \mapsto V$ which indicate local resources. The authority[3] $\bullet(\mathcal{P}, R)$ is used in the encoding of WP to constrain the local fragments to be sub-resources of the global resource $(\mathcal{P}, R)$.

The model we defined using the authoritative construction validates all the desired rules, except for one (first proposed in [5]) that represents a key property of conditioning:

C-FRAME-BB
$$P * \mathbf{C}_{v \leftarrow \mu}\, Q(v) \vdash \mathbf{C}_{v \leftarrow \mu}(P * Q(v))$$

The rule is involved in virtually every conditional proof. For example, to condition on X in $Y \sim \mu' * X \sim \mu$, one first applies $X \sim \mu \vdash \mathbf{C}_{x \leftarrow \mu}\lceil X = x \rceil$ to obtain $Y \sim \mu' * \mathbf{C}_{x \leftarrow \mu}\lceil X = x \rceil$, and then C-FRAME-BB to push the independent resource $Y \sim \mu'$ under the conditioning modality, obtaining $\mathbf{C}_{x \leftarrow \mu}(\lceil X = x \rceil * Y \sim \mu')$. This makes Y available to each branch of the conditioning when we subsequently apply the rule C-LIFT.

In Amaryllis there is a counterexample to the validity of C-FRAME-BB: suppose we start from $\mathsf{Own}(\bullet(\mathcal{P}, R)) * X \sim \mathsf{Ber}_{1/2}$; then we know that the global distribution $\mathcal{P}$ will see X having a fair coin distribution. If we condition on X, obtaining $\mathsf{Own}(\bullet(\mathcal{P}, R)) * \mathbf{C}_{x \leftarrow \mathsf{Ber}_{1/2}}\lceil X = x \rceil$, an application of C-FRAME-BB would bring us to $\mathbf{C}_{x \leftarrow \mathsf{Ber}_{1/2}}\, \mathsf{Own}(\bullet(\mathcal{P}, R)) * \lceil X = x \rceil$, which is inconsistent, since in every branch of the conditioning we would be simultaneously claiming that X is deterministic (by $\lceil X = x \rceil$) and distributed as a fair coin (by the information in $\mathcal{P}$).

In Amaryllis, we resolve this issue by reinterpreting the meaning of the authority: it no longer represents the global resource in absolute terms, but only relative to the current conditional distribution. Specifically, we define the authoritative algebra so that conditioning validates two new variations of C-FRAME-BB:

C-FRAME
$$\boxtimes P * \mathbf{C}_{v \leftarrow \mu}\, Q(v) \vdash \mathbf{C}_{v \leftarrow \mu}(\boxtimes P * Q(v))$$

C-AUTH
$$\frac{\forall v.\, Q(v) \vdash \lceil X = v \rceil}{\mathsf{Own}(\bullet g) * \mathbf{C}_{v \leftarrow \mu}\, Q(v) \vdash \mathbf{C}_{v \leftarrow \mu}(\mathsf{Own}(\bullet(g|_{X=v})) * Q(v))}$$

The rule C-FRAME uses a new modality $\boxtimes P$ to guard the resource to be brought inside a conditioning: it is the identity on fragments ($\mathsf{Own}(\circ a) \dashv\vdash \boxtimes\mathsf{Own}(\circ a)$), but cannot wrap the authority ($\boxtimes\mathsf{Own}(\bullet g) \vdash \mathsf{False}$). The rule C-AUTH allows for the sound transfer of global information across a conditioning, by performing the relevant conditioning on the global state. The premise of the rule serves to identify the random variable X being conditioned (which is needed in order to condition the global resource g on X). The two rules together cover all the user-facing uses of C-FRAME-BB (which would only involve fragments), as well as the new manipulations on the authority required when proving properties of WP.

Equipped with this machinery, Amaryllis supports the encoding of a rather standard-looking definition of WP within the base probabilistic separation logic, which can be manipulated abstractly using our new rules of conditioning and update.

[3] In the technical development an authoritative construction is applied both to the probabilistic resources and the underlying standard resources. The points we make in this section apply equally to the variant of Section 3.

3 The Amaryllis Model

This section describes the model of Amaryllis. We start by recapping prior concepts (Preliminaries) and then describe the specifics of our new model.

3.1 Preliminaries

A set of subsets of Ω is a *σ-algebra* if it contains Ω and is closed under countable unions and complements. The set of all σ-algebras over Ω is denoted $\mathbb{A}(\Omega)$. Given $\mathcal{F} \in \mathbb{A}(\Omega)$, an element of $\mathcal{F}$ is called *event* and Ω is called *outcome space*. A distribution $\mu \in \mathbb{D}(\mathcal{F})$ over some $\mathcal{F} \in \mathbb{A}(\Omega)$ is an additive function $\mu : \mathcal{F} \to [0, 1]$ with $\mu(\Omega) = 1$. The *support* of $\mu \in \mathbb{D}(\mathcal{F})$ is defined as $\mathrm{supp}\,\mu = \Omega \setminus \{\rho \in \Omega \mid \exists X.\ \rho \in X \wedge \mu(X) = 0\}$. The *Dirac distribution* at $v \in \Omega$ is defined as $\delta_v(v) = 1$ and $\delta_v(w) = 0$ for all $w \neq v$.

A *probability space* $\mathcal{P}$ is a tuple $\mathcal{P} = (\Omega, \mathcal{F}, \mu)$ where $\mathcal{F} \in \mathbb{A}(\Omega)$ and μ is a distribution over $\mathcal{F}$. When Ω is clear from the context, we often omit it. The probability space $\mathcal{P} = (\mathcal{F}, \mu)$ is *complete* if for all events $X \in \mathcal{F}$ with $\mu(X) = 0$ and all $\rho \in X$, it holds that $\{\rho\} \in \mathcal{F}$. Given $\mathcal{P}_1 = (\mathcal{F}_1, \mu_1), \mathcal{P}_2 = (\mathcal{F}_2, \mu_2)$ with $\mathcal{F}_1 \in \mathbb{A}(\Omega_1)$ and $\mathcal{F}_2 \in \mathbb{A}(\Omega_2)$, their product $\mathcal{P}_1 \otimes \mathcal{P}_2$ is the probability space over $\Omega_1 \times \Omega_2$ with σ-algebra $\sigma(\{X_1 \times X_2 \mid X_1 \in \mathcal{F}_1, X_2 \in \mathcal{F}_2\})$, and distribution $(\mu_1 \otimes \mu_2)(X_1 \times X_2) = \mu_1(X_1) \cdot \mu_2(X_2)$.

The *independent product* [20] $\mathcal{P}_1 \circledast \mathcal{P}_2$ of two probability spaces $\mathcal{P}_1 = (\mathcal{F}_1, \mu_1)$ and $\mathcal{P}_2 = (\mathcal{F}_2, \mu_2)$ over the same outcome space is defined as $(\sigma(\mathcal{F}_1 \cup \mathcal{F}_2), \mu)$, where $\mu(X_1 \cap X_2) = \mu_1(X_1) \cdot \mu_2(X_2)$ for any $X_1 \in \mathcal{F}_1$ and $X_2 \in \mathcal{F}_2$, and $\sigma(\mathcal{F}_1 \cup \mathcal{F}_2)$ is the smallest σ-algebra containing all elements of $\mathcal{F}_1 \cup \mathcal{F}_2$. The independent product is unique if it exists [20]. For a probability space $\mathcal{P} = (\mathcal{F}, \mu)$ and a set $X \in \mathcal{F}$, the *conditioning* $\mathcal{P}|_X$ is defined as $(\mathcal{F}, \mu')$, where $\mu'(Y) = \mu(Y \cap X)/\mu(X)$ for all $Y \in \mathcal{F}$.

If $\mathcal{F} \in \mathbb{A}(\Omega_1)$ and $F : \Omega_1 \to \Omega_2$ then F is *measurable* if $\forall X \subseteq \Omega_2.\, F^{-1}(X) \in \mathcal{F}$. Such a measurable function is also called a *random variable*. A random variable F is *distributed* as distribution μ if $\forall i \in \mathrm{supp}(\mu).\, \mathcal{P}(F^{-1}(i)) = \mu(i)$.

For with $\mathcal{F}_1 \in \mathbb{A}(\Omega)$, a restriction of $\mu \in \mathbb{D}(\mathcal{F}_1)$ to $\mathcal{F}_2 \in \mathbb{A}(\Omega)$ ($\mathcal{F}_2 \subseteq \mathcal{F}_1$) is the distribution $\mu|_{\mathcal{F}_2}$ with $\mu|_{\mathcal{F}_2}(X) = \mu(X)$ for all events in $\mathcal{F}_2$. The *extension order* relates $(\mathcal{F}_1, \mu_1) \sqsubseteq (\mathcal{F}_2, \mu_2)$ iff $\mathcal{F}_1 \subseteq \mathcal{F}_2$ and $\mu_1 = \mu_2|_{\mathcal{F}_1}$.

An *ordered resource algebra* (ORA) is a tuple $(M, \preccurlyeq, \mathcal{V}, \cdot, |\_|, \varepsilon)$ where M is a set called the *carrier type*, the *resource order* $\preccurlyeq$ is a pre-order over M, $\mathcal{V} : M \to \mathit{Prop}$ is the *validity predicate*, the *resource composition* $\cdot$ is a partial commutative and associative binary operation on M, the *core* $|\_|$ is a unary operation on M and $\varepsilon \in M$ is the *unit*. The unit should satisfy $\mathcal{V}(\varepsilon)$ and $\varepsilon \cdot a = a$ for any $a \in M$. Further, for all $a, b, c \in M$, $\mathcal{V}(a \cdot b) \Rightarrow \mathcal{V}(a)$, $a \preccurlyeq b \Rightarrow \mathcal{V}(b) \Rightarrow \mathcal{V}(a)$ and $a \preccurlyeq b \Rightarrow a \cdot c \preccurlyeq b \cdot c$ must hold.

3.2 Our Resource Algebra

The core of our model is an ORA defined on probability spaces over the outcome space $\Omega = Rid \to \mathbb{B}$. The probability spaces $\mathcal{P}$ we work with can be factorized as $\mathcal{P} = \mathcal{P}_R \otimes \mathbb{1}_{(Rid \setminus R) \to \mathbb{B}}$ for a finite $R \subseteq Rid$, where $\mathbb{1}_\Omega \in \mathbb{P}(\Omega)$ is the σ-algebra $\{\Omega, \emptyset\}$ equipped with the trivial distribution. In our Rocq implementation, we use a concrete representation of such spaces that keeps track of R explicitly. For simplicity, we omit the details of this tracking here and present our definitions directly on probability spaces over $Rid \to \mathbb{B}$. The details of our representation can be found in Appendix B.

3.2.1 Our ORA on Probability Spaces. Our ORA's definition is shown below. Crucially, the ORA's resource composition is the independent product of probability spaces, and its persistent core is the *sure* subspace (probability 0 or 1) of the probability space.

$$\begin{aligned}
&\mathsf{PSp} && \triangleq \{\mathcal{P} \mid \mathcal{P} \text{ is a complete PSp. over } Rid \to \mathbb{B}\} \cup \{\lightning\} \\
&\mathcal{V}(a) && \triangleq a \neq \lightning \\
&\mathcal{P}_1 \cdot \mathcal{P}_2 && \triangleq \mathcal{P}_1 \circledast \mathcal{P}_2 \\
&|(\mathcal{F}, \mu)| && \triangleq (\{X \in \mathcal{F} \mid \mu(X) = 0 \vee \mu(X) = 1\}, \mu) \\
&\varepsilon && \triangleq \mathbb{1}_{Rid \to \mathbb{B}} \\
&\mathcal{P} \preccurlyeq \mathcal{P}' && \triangleq \mathcal{P} \sqsubseteq \mathcal{P}' \qquad \text{(in particular, } \lightning \not\preccurlyeq \mathcal{P} \text{ if } \mathcal{P} \neq \lightning\text{)}
\end{aligned}$$

3.2.2 Random Resource Variables and Authoritative Component. Next, we present the ORA $\mathsf{PSpAuth}_M$, a slightly simplified version of our full model. The actual full model also contains a *namespace* component that tracks a set of globally free randomness IDs: this is used to ease the proof that allocation of new RIDs is a probabilistic frame-preserving update, but omitted here for simplicity. The details are in Appendix B.

Given an Iris resource algebra M, an element of $\mathsf{PSpAuth}_M$ is a triple of a probability space, an M-valued random variable (called a *random resource variable*) and an authoritative component. For the random resource variable component, all resource algebra connectives of M are lifted pointwise. The authoritative component behaves like Iris' authoritative RAs: It is an upper bound on the probability space (in the extension order), and it is *exclusive* — two different authorities cannot be composed. Following Iris terminology, the probability space component is also called a *fragment* of the authoritative part.

For clarity, we highlight expressions related to the random resource variable component and the authoritative component in orange and blue, respectively.

$$\begin{aligned}
&\mathsf{PSpAuth}_M \triangleq \mathsf{PSp} \times RV_M \times \mathsf{Option}(\mathsf{PSp}) \\
&(\mathcal{P}, R, a) \cdot (\mathcal{P}', R', a') \triangleq (\mathcal{P} \cdot \mathcal{P}', \lambda\rho.\ R(\rho) \cdot R'(\rho), a \odot a') \\
&\mathcal{V}(\mathcal{P}, R, a) \triangleq \mathcal{V}(\mathcal{P}) \wedge \forall \rho, \mathcal{V}(R(\rho)) \wedge \\
&\qquad\qquad (a = \mathcal{P}_a \Rightarrow \mathcal{V}(\mathcal{P}_a) \wedge \mathcal{P} \preccurlyeq \mathcal{P}_a) \\
&|(\mathcal{P}, R, a)| \triangleq (|\mathcal{P}|, \lambda\rho.\ |R(\rho)|, \bot) \\
&\varepsilon \triangleq (\varepsilon, \lambda.\ \varepsilon, \bot) \\
&(\mathcal{P}, R, a) \preccurlyeq (\mathcal{P}', R', a') \triangleq \mathcal{P} \preccurlyeq \mathcal{P}' \wedge \forall \rho.\ R(\rho) \preccurlyeq R'(\rho) \wedge \\
&\qquad\qquad (a \neq \bot \Rightarrow a = a')
\end{aligned}$$

where $a \odot \bot = a = \bot \odot a$, $a \odot b = \lightning$, and $RV_M := (Rid \to \mathbb{B}) \to M$.

For projecting out the three components of the resource algebra, we define $\pi_{\mathsf{frag}}(\mathcal{P}, R, a) = \mathcal{P}$, $\pi_{RV}(\mathcal{P}, R, a) = R$, $\pi_{\mathsf{auth}}(\mathcal{P}, R, a) = a$.

3.3 Weighted Sum and Conditioning

To define the conditioning modality, we find it useful to first define a *weighted sum* operation $wsum : \forall A.\, \mathbb{D}(A) \to (A \to \mathsf{PSpAuth}_M) \to \mathsf{PSpAuth}_M$ that combines the resources of exclusive probabilistic branches into a single resource. We first define the weighted sum on probability spaces, and then extend it to the full ORA.

3.3.1 Weighed Sum on Probability Spaces. We define the weighted sum on probability spaces as follows.

$$wsum_{\mathsf{PSp}}(\mu, \kappa) \triangleq \begin{cases} \mathbf{bind}\left(\mu, \lambda v.\ \kappa(v)|_{\bigcap_{v \in \mathrm{supp}\,\mu} \pi_{\mathcal{F}}(\kappa(v))}\right) & \text{if has\_rv}(\mu, \kappa) \\ \lightning & \text{otherwise} \end{cases}$$

where

$$\text{has\_rv}(\mu, \kappa) := \forall v, w \in \mathrm{supp}\,\mu.\ \mathrm{supp}(\kappa(v)) \cap \mathrm{supp}(\kappa(w)) = \emptyset$$

The weighted sum is defined as the standard dependent product of μ and κ (the **bind** operation of the Giry monad) but only if the supports of κ's branches are disjoint, as codified in the condition has_rv(μ, κ). This is equivalent to requiring that μ be the distribution of *some* random variable, and that κ be an exclusive union of branches defined by the possible values of that random variable. If this is the case, we define the σ-algebra of the weighted sum to be the set of events that are measurable in all branches.

3.3.2 Weighted Sum on our ORA. We extend the definition of weighted sum to our ORA. To take the weighted sum of random resource variables, consider a specific outcome ρ. Since the supports of κ's branches must be pairwise disjoint (see above), there is at most one branch v such that ρ has nonzero probability in that branch. If such a v exists, define $R(\rho) = \pi_{RV}(\kappa(v)(\rho))$. If ρ is not possible in any branch, we define the weighted sum to be ε at ρ.[4].

Formally, define get_rv$(\mu, \kappa)(\rho)$ as the unique $v \in \mathrm{supp}(\mu)$ such that $\rho \in \mathrm{supp}\,\pi_{\mathsf{frag}}(\kappa(v))$ if such a v exists. Then,

$$\begin{aligned}
&wsum(\mu, \kappa) \triangleq (\mathcal{P}, R, a) \\
&\mathcal{P} \triangleq wsum_{\mathsf{PSp}}(\mu, \lambda v.\, \pi_{\mathsf{frag}}(\kappa\, v)) \\
&R \triangleq \lambda\rho.\, \text{if } \rho \in \mathrm{supp}(\mathcal{P}) \text{ then } (\pi_{RV}(\kappa(\text{get_rv}(\mu, \kappa)))\ \rho) \text{ else } \varepsilon
\end{aligned}$$

Note that the random resource variable of any branch is almost surely equal to R conditional on being in that branch.

We require that the authoritative part either exist in all branches of κ or none of them. If it exists in all of them, we define the resulting authority as the $wsum_{\mathsf{PSp}}$ of the individual authoritative parts.

$$\begin{aligned}
&a \triangleq \bot && \text{if } \forall v.\ \pi_{\mathsf{auth}}(\kappa(v)) = \bot \\
&a \triangleq wsum_{\mathsf{PSp}}(\mu, \kappa') && \text{if } \forall v.\ \pi_{\mathsf{auth}}(\kappa(v)) = \kappa'(v)
\end{aligned}$$

3.3.3 Conditioning. Next, we define conditioning.

$$(\mathbf{C}_{v \leftarrow \mu}\, \Phi(v))(x) \triangleq \exists \kappa.\ wsum(\mu, \kappa) \preccurlyeq x \wedge \forall v \in \mathrm{supp}\,\mu.\ \Phi(v)(\kappa(v))$$

This definition is equivalent to the following axiomatic characterization of conditioning (here, $\bot|_E \triangleq \bot$ for all E):

$$\begin{aligned}
(\mathbf{C}_{v \leftarrow \mu}\, \Phi(v))(\mathcal{P}, R, a) \Leftrightarrow\ &\exists X.\ X \text{ is distributed as } \mu \text{ in } \mathcal{P} \wedge \\
&\forall v \in \mathrm{supp}\,\mu.\ \Phi(v)(\mathcal{P}|_{X=v},\ a|_{X=v},\ R)
\end{aligned}$$

[4] Our logic ignores probability 0 branches, so it might seem that ε could be replaced by an arbitrary element of our ORA. However, for technical reasons, we must ensure that those branches are valid. ε is always valid.

3.4 Probabilistic Frame-Preserving Updates

Next, we define PFP updates, which generalize Iris updates. Like Iris updates, PFP updates are upwards closed with respect to the resource order and preserve frames. Additionally, we require PFP updates to *preserve conditioning*, i.e., the validity of weighted sums. Formally, a PFP update from resources x to x' preserves conditioning if

$$\forall \mu, \kappa, v.\ \mathcal{V}(wsum(\mu, \kappa[v \mapsto x])) \Rightarrow \mathcal{V}(wsum(\mu, \kappa[v \mapsto x']))$$

Next, we consider upwards-closedness of updates. Our weighted sum is not monotone: $(\forall v.\ \kappa(v) \preccurlyeq \kappa'(v)) \not\Rightarrow wsum(\mu, \kappa) \preccurlyeq wsum(\mu, \kappa')$, since κ might not be induced by a random variable even if κ' is, whence $wsum(\mu, \kappa)$ would be $\lightning$. As a result, upwards-closedness of updates needs to be encoded explicitly. To do that, we define an operator $\circ$ that characterizes the resource order: $x \preccurlyeq y \Leftrightarrow \exists z. x \circ y \equiv z$. The concrete definition of $\circ$ is omitted here and can be found in Appendix B and in our Rocq development [3].

We can now define *probabilistic contexts*, K_M, comprising the operators $\circ, wsum$ and the resource composition $(\cdot)$, and define context application, $K_M[\![x]\!]$.

$$K_M \triangleq \cdot \mid \mathsf{frame}(cx, K_M) \mid \mathsf{cond}(\mu, \kappa, v, K_M) \mid \mathsf{ext}(cx, K_M)$$

$$\begin{aligned} \cdot[\![x]\!] &\triangleq x \\ \mathsf{frame}(cx, K_M)[\![x]\!] &\triangleq cx \cdot K_M[\![x]\!] \\ \mathsf{cond}(\mu, \kappa, v, K_M)[\![x]\!] &\triangleq wsum(\mu, \kappa[v \mapsto K_M[\![x]\!]]) \\ \mathsf{ext}(cx, K_M)[\![x]\!] &\triangleq cx \circ K_M[\![x]\!] \end{aligned}$$

Equipped with these definitions, we define *probabilistic frame-preserving (PFP) updates*, as those resource updates that preserve validity in all contexts:

$$\Rightarrow\!\!\!\!\vDash P \triangleq \lambda a.\ \forall K_M.\ \mathcal{V}(K_M[\![a]\!]) \Rightarrow \exists a'.\ \mathcal{V}(K_M[\![a']\!]) \wedge P(a')$$

We have shown that allocating a randomness ID is a probabilistic-frame preserving update, and that frame-preserving updates on the random resource variable (including non-deterministic updates) can be lifted to PFP updates:

Theorem 3.1 (Lifting frame-preserving updates).

$$\begin{aligned} &\left(\forall \rho\ cx.\ \mathcal{V}(R(\rho) \cdot cx) \Rightarrow \exists x'.\ \mathcal{V}(x' \cdot cx) \wedge \Phi(\rho)(x')\right) \Rightarrow \\ &\forall K_M.\ \mathcal{V}(K_M[\![(\varepsilon, R, \bot)]\!]) \Rightarrow \exists R'.\ \mathcal{V}(K_M[\![(\varepsilon, R', \bot)]\!]) \wedge \\ &\qquad\qquad \forall \rho.\ \Phi(\rho)(R'(\rho)) \end{aligned}$$

This theorem says that if a random resource variable R can be updated in the Iris frame-preserving sense to a resource that satisfies the predicate Φ in all probabilistic branches, then the probabilistic resource $(\varepsilon, R, \bot)$ can be PFP updated (in the sense defined above) to satisfy Φ in all branches. Intuitively, the theorem holds in our model because all operations on random resource variables are defined per branch (per outcome). A weighted sum has no effect on non-zero probability branches of the random resource variable, and ignores zero probability branches. Thus, an update in a branch of the random resource variable, whether probability 0 or not, will always preserve the validity of arbitrary weighted sums. See our Rocq development [3] for the formal proof.

4 The Amaryllis Logic

4.1 Propositions and Connectives

Our logic's propositions pProp are predicates over $\mathrm{PSpAuth}_M$ that are upwards-closed with respect to the resource order and, additionally, proper with respect to almost-sure equality of the random resource variable. To define them formally, we first define a slight variation of our resource order, the *almost-sure resource order* that ignores probability 0 branches:

$$\begin{aligned} &(\mathcal{P}, R, a) \preccurlyeq_{as} (\mathcal{P}', R, a') \triangleq \mathcal{P} \preccurlyeq \mathcal{P}' \wedge \\ &\qquad (a \neq \bot \Rightarrow a \equiv a') \wedge \forall \rho \in \mathrm{supp}(\mathcal{P}) \Rightarrow R(\rho) \preccurlyeq R'(\rho) \\ &\mathsf{pProp} := \{\Phi \colon \mathrm{PSpAuth}_M \to \mathsf{Prop} \mid \forall x, x'.\ x \preccurlyeq_{as} x' \wedge \Phi(x') \Rightarrow \Phi(x)\} \end{aligned}$$

The conditioning $(\mathrm{C}_{v \leftarrow \mu} \Phi(v))(x))$ and update $(\Rightarrow\!\!\!\!\vDash P)$ modalities were already defined in Section 3. $\mathsf{Own}(x)$, $\boxtimes$, $X \sim \mu$ and sure propositions $(\lceil \Phi \rceil)$ can be defined as follows:

$$\begin{aligned} \mathsf{Own}(x')(x) &\triangleq x' \preccurlyeq_{as} x \\ (\boxtimes \Phi)(\mathcal{P}, R, a) &\triangleq \Phi(\mathcal{P}, R, \varepsilon) \\ (X \sim \mu)(\mathcal{P}, R, a) &\triangleq X \text{ is distributed as } \mu \text{ in } \mathcal{P} \\ \lceil \Phi \rceil &\triangleq \Phi \sim \delta_{\mathsf{true}} \end{aligned}$$

The remaining primitive connectives $*, -\!\!*, \forall, \exists, \wedge, \vee, \Rightarrow$, pure propositions and the Iris's persistency modality $\square$ for duplicable assertions can all be defined in the standard way. As examples, we show the definitions of $*, -\!\!*$ and $\square$.

$$\begin{aligned} (P * Q)(x) &\triangleq \exists x_1, x_2.\ x_1 \cdot x_2 \preccurlyeq x \wedge P(x_1) \wedge Q(x_2) \\ (P -\!\!* Q)(x) &\triangleq \forall x'.\ \mathcal{V}(x \cdot x') \wedge P(x') \Rightarrow Q(x \cdot x') \\ (\square P)(x) &\triangleq P(|x|) \end{aligned}$$

It can be shown that all connectives preserve the almost-sure upwards-closedness of pProp.

In the following, we instantiate M with $\mathsf{Auth}(gmap(Loc \to Val))$, the resource algebra that is used in standard Iris to model (deterministic) points-to assertions. Our pointsto-assertion can then be defined as follows.

$$L \mapsto V \triangleq \mathsf{Own}(\varepsilon, (\lambda \rho.\ \circ[L(\rho) \mapsto V(\rho)], \bot))$$

Selected primitive proof rules are shown in Figure 1. The rules are straightforward to read, thanks to the setup of our model.

4.2 Language and Semantics

We consider the language pHeapLang, a variant of Iris's HeapLang language, extended with real numbers r and a sampling primitive $\mathtt{flip}(e)$ that samples a boolean value from the Bernoulli distribution with bias e.

$$\begin{aligned} Val \ni v &::= z \in \mathbb{Z} \mid r \in \mathbb{R} \mid b \in \mathbb{B} \mid \ell \in Loc \mid \mathtt{rec}\ \mathrm{f}\ x = e \mid (v_1, v_2) \\ Expr \ni e &::= v \mid x \mid e_1\ e_2 \mid \mathtt{ref}\ e \mid \mathtt{get}\ e \mid \mathtt{set}\ e\ e \mid e_1 + e_2 \mid e_1 \le e_2 \mid \ldots \\ &\quad \mid (e_1, e_2) \mid \mathtt{fst}\ e \mid \mathtt{snd}\ e \mid \mathtt{if}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e \mid \mathtt{flip}(e) \end{aligned}$$

We use notation $\lambda x.\ e \triangleq \mathtt{rec}\ \_x = e$, and $\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \triangleq (\lambda x.\ e_2)e_1$.

pHeapLang comes equipped with a small step semantics:

$$\rightsquigarrow \colon (Expr \times State) \times (Expr \times State \times Expr \times State \times \mathbb{R})$$

DISTR-SUBST
$X \sim \mu * \lceil X = Y \rceil \vdash Y \sim \mu$

DISTR-DIRAC
$\vdash v \sim \delta(v)$

OWN-SUBST
$\mathsf{Own}(\varepsilon, X) * \lceil X = Y \rceil \vdash \mathsf{Own}(\varepsilon, Y)$

SURE-PERSISTENT
$\lceil X = Y \rceil \vdash \Box \lceil X = Y \rceil$

C-SKOLEM
$\mathbf{C}_{v \leftarrow \mu}\, \exists a.\, \Phi(a, v) \dashv\vdash \exists f.\, \mathbf{C}_{v \leftarrow \mu}\, \Phi(f(v), v)$

C-BIND
$\mathbf{C}_{v \leftarrow \mu}\, X \sim \kappa(v) \vdash X \sim \mathbf{bind}(\mu, \kappa)$

C-FUSE
$\mathbf{C}_{v \leftarrow \mu}\, \mathbf{C}_{w \leftarrow \kappa(v)}\, \Phi(v, w) \dashv\vdash \mathbf{C}_{(v,w) \leftarrow \mu \prec \kappa}\, \Phi(v, w)$

C-RV
$\mathbf{C}_{v \leftarrow \mu}\, \Phi(v) \vdash \exists X.\, \mathbf{C}_{v \leftarrow \mu} \lceil X = v \rceil * \Phi(v)$

C-INTRO
$X \sim \mu \dashv\vdash \mathbf{C}_{v \leftarrow \mu} \lceil X = v \rceil$

C-DIRAC
$\mathbf{C}_{v \leftarrow \delta(i)}\, \Phi(v) \dashv\vdash \Phi(i)$

C-CONS
$$\frac{\forall v \in \mathrm{supp}(\mu).\ \Phi_1(v) \vdash \Phi_2(v)}{\mathbf{C}_{v \leftarrow \mu}\, \Phi_1(v) \vdash \mathbf{C}_{v \leftarrow \mu}\, \Phi_2(v)}$$

POINTSTO-STR-CONVEX
$\mathsf{StrConvex}(L \mapsto V)$

DISTR-STR-CONVEX
$\mathsf{StrConvex}(X \sim \mu)$

DISTR-FRAMEABLE
$X \sim \mu \vdash \boxtimes X \sim \mu$

PURE-FRAMEABLE
$\Phi \text{ pure} \Rightarrow \Phi \vdash \boxtimes \Phi$

SEP-FRAMABLE
$\boxtimes P * \boxtimes Q \dashv\vdash \boxtimes (P * Q)$

$\mu \prec \kappa \triangleq \lambda(v, w).\ \mu(v)\kappa(v, w)$

$\mathsf{StrConvex}(\Psi) \triangleq \mathbf{C}_{v \leftarrow \mu}\, \Phi(v) * \Psi \vdash \Psi * \mathbf{C}_{v \leftarrow \mu}\, \Phi(v)$

Figure 1: Selected proof rules of Amaryllis

where $e, \sigma \rightsquigarrow e_1, \sigma_1, e_2, \sigma_2, p$ can be read as: e, σ steps to e_1, σ_1 with probability p, and to e_2, σ_2 with probability $1 - p$. For instance, we have that $\mathtt{flip}(p), \sigma \rightsquigarrow \mathtt{true}, \sigma, \mathtt{false}, \sigma, p$. All standard non-probabilistic steps $e, \sigma \rightsquigarrow_{np} e', \sigma'$ are lifted to probabilistic steps $e, \sigma, \rightsquigarrow e', \sigma', e_2, \sigma_2, 1$ for all e_2, σ_2. For example, $\mathtt{ref}\, v, \sigma \rightsquigarrow (), \sigma[l \mapsto v], e_2, \sigma_2, 1$ for all $l \notin \mathrm{dom}\, \sigma, e_2, \sigma_2$. We use this representation for non-probabilistic steps to avoid having to represent them differently from probabilistic steps.

We lift this semantics to a collecting big-step semantics on indexed valuations:

$$\Downarrow : (\mathsf{PSpD} \times RV_{Expr} \times RV_{State}) \times (\mathsf{PSpD} \times RV_{Expr} \times RV_{State})$$

Intuitively, for each step $\rightsquigarrow$ that is executed in some branch, a new randomness ID is allocated and the random variables are updated accordingly. For example, consider the program $\mathtt{let}\ x = \mathtt{flip}(1/2)\ \mathtt{in}\ \mathtt{ref}\,(\mathtt{flip}(1/2) + x)$, executed in an initially empty state and probability space. The first flip will create an RID ι_1 and leave us with the random expression variable $\boldsymbol{\lambda}\rho.\, \mathtt{ref}\,(\mathtt{flip}(1/2) + \rho(\iota_1))$. The second flip creates two new randomness IDs, ι_2 and ι_3, one for each of the two outcomes of the first flip, leaving us with the random expression variable $\boldsymbol{\lambda}\rho.$ if $\rho(\iota_1)$ then $\rho(\iota_2)$ else $\rho(\iota_3) + 1$. The last step, which is the allocation of a reference, will result in the random expression variable $\boldsymbol{\lambda}.\, ()$ and the random *state* variable $\boldsymbol{\lambda}\rho.$ if $\rho(\iota_1)$ then $\rho(\iota_2)$ else $\rho(\iota_3) + 1$. Note that this is not the only possible reduction as the RIDs and the RV values for probability 0 branches are chosen nondeterministically.

The formal definition of our language's operational semantics is in Appendix C.

4.3 The Weakest Precondition Modality

We define the *weakest precondition modality* $\mathsf{wp} : RV_{Expr} \to (RV_{Val} \to \mathsf{pProp})$ using the primitive connectives.

$$\mathsf{wp}\ E\ \{v.\, \Phi(v)\} \triangleq \forall \mathcal{P}, \Sigma.\ E, \Sigma \text{ measurable in } \mathcal{P} \mathrel{-\!\!*} S(\mathcal{P}, \Sigma) \mathrel{-\!\!*} \forall V, \mathcal{P}', \Sigma'.\ \langle \mathcal{P}, E, \Sigma \rangle \Downarrow \langle \mathcal{P}', V, \Sigma' \rangle \mathrel{-\!\!*} \Rrightarrow S(\mathcal{P}', \Sigma') * \Phi(V)$$

where the *state interpretation* S is defined as

$$S(\mathcal{P}, \Sigma) := \mathsf{Own}(|\mathcal{P}|, (\lambda\rho.\, \bullet \Sigma(\rho)), \mathcal{P})$$

and the postcondition Φ is required to be proper with respect to almost-sure equality: $\lceil V = V' \rceil * \Phi(V) \vdash \Phi(V')$.

In the weakest precondition, we first quantify over all possible current 'probabilistic states': A pair $(\mathcal{P}, \Sigma)$ is a *possible probabilistic state* if (1) both E and Σ are measurable in $\mathcal{P}$, and (2) $\mathcal{P}$ and Σ are both consistent with the 'local information', i.e. is is not contradictory to assume that $\mathcal{P}$ and Σ represent the ground truth. Statement (2) is equivalent to assuming the state interpretation.

Next, we quantify over the all random value variables and probabilistic states that the current random expression variable might evaluate to. For each option, we require that we can update our local information such that (1) our local information is again consistent with the new probabilistic state $\mathcal{P}', \Sigma'$, and (2) the postcondition holds on the output random value variable.

Hoare triples are defined on top of the weakest precondition modality: $\{P\}\ E\ \{Q\} \triangleq P \vdash \mathsf{wp}\ E\ \{Q\}$.

Proof rules for the weakest precondition can be found in Figure 2. The rules should be intuitive, and those for non-probabilistic constructs are analogous to standard proof rules in Iris, except that they now apply to random expression variables rather than deterministic expressions.

The following adequacy theorem, our main result, states that our WP definition is sound.

Theorem 4.1 (Adequacy). *If* $\langle \varepsilon, (\boldsymbol{\lambda}\_.\, e), (\boldsymbol{\lambda}\_.\, \sigma) \rangle \Downarrow \langle \mathcal{P}, V, \Sigma \rangle$ *and* $\vdash \mathsf{wp}\ (\boldsymbol{\lambda}\_.\, e)\ \{V.\, V \sim \mu\}$*, then* V *is distributed as* μ *in* $\mathcal{P}$*.*

5 Illustrative Examples

We present three illustrative examples of verification in Amaryllis.

5.1 Deriving Uniform Sampling from Bernoulli Sampling

Our first example illustrates how our rule C-WP-SWAP enables reasoning about conditional program expressions modularly. We verify the following function u, which implements a uniform sampler over the integers $\{0, 1, \ldots, n-1\}$ using our Bernoulli sampler, flip.

$$\mathtt{rec}\ \mathrm{u}\, n \triangleq \mathtt{if}\ n = 1\ \mathtt{then}\ 0\ \mathtt{else}\ \mathtt{if}\ \mathtt{flip}(\tfrac{1}{n})\ \mathtt{then}\ (n-1)\ \mathtt{else}\ \mathrm{u}(n-1)$$

WP-ALLOC
$\mathsf{wp}\ \mathtt{ref}\ V\ \{V'.\ \exists L.\ \lceil V' = L\rceil * L \mapsto V\}$

WP-SAMPLE
$0 < p < 1 \vdash \mathsf{wp}\ \mathtt{flip}(p)\ \{V.\ V \sim \mathsf{Ber}_p\}$

WP-STORE
$L \mapsto V \vdash \mathsf{wp}\ \mathtt{set}\, L\, V'\ \{L \mapsto V'\}$

WP-LOAD
$L \mapsto V \vdash \mathsf{wp}\ \mathtt{get}\, L\ \{V'.\ L \mapsto V * \lceil V = V'\rceil\}$

WP-PURE
$$\frac{E \rightsquigarrow_{pure} E'}{\mathsf{wp}\ E'\ \{\Phi\} \vdash \mathsf{wp}\ E\ \{\Phi\}}$$

WP-AS-EQ
$\lceil E_1 = E_2\rceil \twoheadrightarrow \mathsf{wp}\ E_1\ \{\Phi\} \twoheadrightarrow \mathsf{wp}\ E_2\ \{\Phi\}$

WP-VALUE
$\models\!\!\Rightarrow \Phi(V) \vdash \mathsf{wp}\ V\ \{\Phi\}$

WP-MONO
$\mathsf{wp}\ E\ \{\Phi\} \twoheadrightarrow (\forall V.\ \Phi(V) \twoheadrightarrow \models\!\!\Rightarrow \Psi(V)) \twoheadrightarrow \mathsf{wp}\ E\ \{\Psi\}$

WP-BIND
$\mathsf{wp}\ E\ \{V.\ \mathsf{wp}\ K[V]\ \{\Phi\}\} \vdash \mathsf{wp}\ K[E]\ \{\Phi\}$

Figure 2: Selected proof rules about weakest preconditions

We show $\{n \in \mathbb{N}_{>0}\}\ \mathrm{u}\, n\ \{V.\ V \sim \mathsf{Unif}_n\}$ by induction on n. The base case, $n = 1$, follows directly from WP-PURE, WP-VALUE and DISTR-DIRAC. For the induction step, we first apply WP-BIND and WP-SAMPLE to obtain the proof goal:

$$V \sim \mathsf{Ber}_{1/n} \vdash \mathsf{wp}\ \mathtt{if}\ V\ \mathtt{then}\ (n-1)\ \mathtt{else}\ \mathrm{u}(n-1)\ \{V'.\ V' \sim \mathsf{Unif}_n\}$$

Next, we would like to condition on V but to do that, we have to transform the left and right sides of the entailment to forms that have the conditioning connective $\mathbf{C}_{v \leftarrow \mathsf{Ber}_{1/n}}$ at the top-level so that we can apply C-CONS. On the left side, we apply C-INTRO to obtain $\mathbf{C}_{v \leftarrow \mathsf{Ber}_{1/n}} \lceil V = v\rceil$. On the right side, we first apply WP-MONO and C-BIND to reduce the postcondition $V'.\ V' \sim \mathsf{Unif}_n$ to $\mathbf{C}_{v \leftarrow \mathsf{Ber}_{1/n}}\, V' \sim (\text{if } v \text{ then } \delta_{n-1} \text{ else } \mathsf{Unif}_{n-1})$ and obtaining a revised goal:

$$\mathbf{C}_{v \leftarrow \mathsf{Ber}_{1/n}} \lceil V = v\rceil \vdash \mathsf{wp}\ \mathtt{if}\ V\ \mathtt{then}\ (n-1)\ \mathtt{else}\ \mathrm{u}(n-1)\ \left\{\mathbf{C}_{v \leftarrow \mathsf{Ber}_{1/n}}\, V' \sim (\text{if } v \text{ then } \delta_n \text{ else } \mathsf{Unif}_{n-1})\right\}$$

This brings us to the key step in the proof: We apply C-WP-SWAP to rewrite the right to a form with $\mathbf{C}_{v \leftarrow \mathsf{Ber}_{1/n}}$ at the top level.

$$\mathbf{C}_{v \leftarrow \mathsf{Ber}_{1/n}} \lceil V = v\rceil \vdash \mathbf{C}_{v \leftarrow \mathsf{Ber}_{1/n}} \mathsf{wp}\ \mathtt{if}\ V\ \mathtt{then}\ (n-1)\ \mathtt{else}\ \mathrm{u}(n-1)\ \{V' \sim (\text{if } v \text{ then } \delta_n \text{ else } \mathsf{Unif}_{n-1})\}$$

Next, we apply C-CONS and use WP-AS-EQ to rewrite with $\lceil V = v\rceil$. A case analysis on v leaves us with two subgoals corresponding to the `then` and the `else` branches: $\mathsf{wp}\ (n-1)\ \{V'.\ V' \sim \delta_{n-1}\}$ and $\mathsf{wp}\ \mathrm{u}(n-1)\ \{V'.\ V' \sim \mathsf{Unif}_{n-1}\}$. These subgoals follow from C-DIRAC and the induction hypothesis, respectively. □

5.2 Markov Blankets

Our next example is a Markov blanket: We sample three values x_1, x_2, x_3 with x_2, x_3 being dependent on x_1, x_2 respectively, and show that x_1 and x_3 are independent *conditional on* x_2 (x_2 blankets or covers x_1's influence on x_3; hence, the term Markov blanket). The example, shown below, further illustrates reasoning with conditioning, and is an adaptation of a similar example included in the BLUEBELL paper [5]. The triple we obtain is stronger than BLUEBELL's in that it fully supports further framing around chain, enabling reuse.

$$\mathsf{chain}(f,g,h) \triangleq \mathtt{let}\ x_1 = f()\ \mathtt{in}\ \mathtt{let}\ x_2 = g(x_1)\ \mathtt{in}\ \mathtt{let}\ x_3 = h(x_2)\ \mathtt{in}\ (x_1, x_2, x_3)$$

We formalize the specification "x_1 and x_3 are independent conditional on x_2" as follows:

$$\begin{aligned}&\{\mathsf{True}\}\ \boldsymbol{\lambda}.\, f()\ \{V.V \sim \mu_1\} \Rightarrow\\ &(\forall x.\ \{\mathsf{True}\}\ \boldsymbol{\lambda}.\, g(x)\ \{V.V \sim \kappa_2(x)\}) \Rightarrow\\ &(\forall y.\ \{\mathsf{True}\}\ \boldsymbol{\lambda}.\, h(y)\ \{V.V \sim \kappa_3(y)\}) \Rightarrow\\ &\quad \{\mathsf{True}\}\ \mathsf{chain}(f,g,h)\ \{V.\ \exists V_1, V_2, V_3, \lceil V = (V_1, V_2, V_3)\rceil *\\ &\qquad \exists \mu.\ \mathbf{C}_{v \leftarrow \mu}(\lceil V_2 = v\rceil * \exists \mu'.\ V_1 \sim \mu' * V_3 \sim \kappa_3(v))\}\end{aligned}$$

The first step of the proof is replacing the postcondition of chain with a stronger, easier to use postcondition by proving the following entailment. The proof of this entailment is nontrivial, but it is the same as in Bluebell's original example, so we defer it to the appendix.

$$\begin{aligned}&\mathbf{C}_{v_1 \leftarrow \mu_1}\left(\lceil V_1 = v_1\rceil * \mathbf{C}_{v_2 \leftarrow \kappa_2(v_1)}(\lceil V_2 = v_2\rceil * V_3 \sim \kappa_3(v_3))\right)\\ &\vdash \exists \mu.\ \mathbf{C}_{v \leftarrow \mu}(\lceil V_2 = v\rceil * \exists \mu'.\ V_1 \sim \mu' * V_3 \sim \kappa_3(v))\end{aligned}$$

Using WP-MONO, our proof goal becomes

$$\begin{aligned}&\{\mathsf{True}\}\ \mathsf{chain}(f,g,h)\ \{V.\ \exists V_1, V_2, V_3, \lceil V = (V_1, V_2, V_3)\rceil *\\ &\qquad \mathbf{C}_{v_1 \leftarrow \mu_1}\left(\lceil V_1 = v_1\rceil * \mathbf{C}_{v_2 \leftarrow \kappa_2(v_1)}(\lceil V_2 = v_2\rceil * V_3 \sim \kappa_3(v_3))\right)\}\end{aligned}$$

To prove this, we first apply f's Hoare triple using WP-BIND to obtain a random variable $V_1 \sim \mu_1$ that is stored in x_1. Next, to use g's Hoare triple, we must condition on V_1 after applying C-WP-SWAP to the goal. However, we cannot immediately apply C-WP-SWAP since the conditioning $\mathbf{C}_{v_1 \leftarrow \mu_1}(\ldots)$ in the goal is under an existential quantifier. We commute the conditioning and existential quantification use a *derived* rule.

C-EXISTS-RV
$$\frac{\mathbf{C}_{v \leftarrow \mu}\, \exists V.\ \Phi(v)(V) \qquad \forall v, V_1, V_2.\lceil V_1 = V_2\rceil * \Phi(v)(V_1) \vdash \Phi(v)(V_2)}{\exists V.\ \mathbf{C}_{v \leftarrow \mu}\, \Phi(v)(V)}$$

C-EXISTS-RV is derived as follows. First, apply C-SKOLEM to the first premise to obtain a function f satisfying $\mathbf{C}_{v \leftarrow \mu}\, \Phi(v)(f(v))$. Next, use C-RV to obtain a random variable $X \sim \mu$ that the conditioning is actually on. Define $V := \boldsymbol{\lambda}\rho.\, f(X(\rho))\rho$ and apply C-CONS to get the goal $\lceil X = v\rceil * \Phi(v)(f(v)) \vdash \Phi(v)(\boldsymbol{\lambda}\rho.\ f(X(\rho)))$, which follows from the rule's second premise.

Using C-EXISTS-RV, we can continue the proof of the example: after moving the existentials below the conditioning, we can apply C-WP-SWAP to condition on V_1, and WP-BIND to use the Hoare triple

for g. To complete the proof, we repeat this pattern once more for h. □

This example cannot be verified easily in existing GPLs for different reasons. Bluebell, due to its c-wp-swap rule which must be used to verify the three function calls, requires ownership of every variable in the precondition of the triple, thus prohibiting framing of resources around the final chain triple, hindering its reuse. All other GPLs conflate probabilistic independence and disjointness and, therefore, they cannot express assertions like chain's postcondition, which mentions all three variables V_1, V_2, V_3 on both sides of a separating conjunction.

5.3 Sorted Linked Lists with Random Elements

Our next example is a standard pointer-based sorted linked list, with three operations: init (new empty list), ins (insert a value) and del (remove a value and return it) operations. The list is an interesting example for us because it leverages dynamic allocation, which prior GPLs do not support. Additionally, *sorted* lists interact with randomness in challenging ways. If a sampled value is inserted into a sorted list, the position at which it ends up depends on what samples drawn for other elements of the list. Hence, the *structure* of the list is itself randomized. Despite this entanglement between data and list structure, we are able to prove a strong specification: If each inserted value's distribution is independent of the list's state at the time of insertion, then any value removed from the list is independent of the remaining list's state at the time of removal.

We start with an implementation of the sorted list's three operations in our language, which should be unsurprising.

$$\begin{aligned}
\text{init} &\triangleq \lambda x.\, \texttt{ref}\,() \\
\text{ins} &\triangleq \lambda x\,\ell.\ (\texttt{rec loop}\,\ell' = \texttt{let}\,v = \texttt{get}\,\ell'\,\texttt{in} \\
&\quad \texttt{if}\,v = () \vee x \le \texttt{fst}\,v\,\texttt{then ref}\,(x,\ell')\,\texttt{else} \\
&\quad \texttt{set}\,\ell'\,(\texttt{fst}\,v,\ \texttt{loop}(\texttt{snd}\,v));;\ell')\ell \\
\text{del} &\triangleq \lambda x\,\ell.\ (\texttt{rec loop}\,\ell' = \texttt{let}\,v = \texttt{get}\,\ell'\,\texttt{in} \\
&\quad \texttt{if}\,v = ()\,\texttt{then}\,\ell'\,\texttt{else} \\
&\quad \texttt{if}\,x = \texttt{fst}\,v\,\texttt{then snd}\,v\,\texttt{else} \\
&\quad \texttt{set}\,\ell'\,(\texttt{fst}\,v, \texttt{loop}(\texttt{snd}\,v));;\ell')\ell
\end{aligned}$$

To verify our implementation, we will define a representation predicate $\mathsf{psll} : RV_{Loc} \times 2^{RV_{\mathbb{Z}} \times \mathbb{D}(\mathbb{Z})} \to \mathsf{pProp}$ for randomized sorted linked lists. Informally, $\mathsf{psll}(L, ds)$ holds iff ds is a set of the form $\{(V_1, \mu_1), \ldots, (V_n, \mu_n)\}$, L is a random list containing the random elements $V_1, \ldots, V_n$ in sorted order, and each V_i is distributed as μ_i. (Technically, L, V_is are random variables.) Equipped with this definition, we prove the following specification for the list operations in Amaryllis:

$$\begin{aligned}
&\{\mathsf{True}\}\ \text{init}\ ()\ \{\exists L.\ \lceil V = L\rceil * \mathsf{psll}(L, \emptyset)\} \\
&\{\mathsf{psll}(L, ds) * X \sim \mu\}\ \text{ins}\ X\ L \\
&\qquad \{V.\ \exists L'.\ \lceil V = L'\rceil * \mathsf{psll}(L', (X, \mu) \cup ds)\} \\
&\{\mathsf{psll}(L, (X, \mu) \cup ds)\}\ \text{del}\ X\ L \\
&\qquad \{V.\ \exists L'.\ \lceil V = L'\rceil * \mathsf{psll}(L', ds) * X \sim \mu\}
\end{aligned}$$

The precondition of ins requires that the inserted element X's distribution be independent of $\mathsf{psll}(L, ds)$. Dually, the postcondition of del states that the removed element's distribution is independent of the rest of the list.

We define psll in three steps. First, we define the standard list representation predicate $\mathsf{ll} : RV_{Loc} \to \mathit{list}\ \mathbb{Z} \to \mathsf{pProp}$. $\mathsf{ll}(L, zs)$ holds if the *random location* L holds the *deterministic elements* of zs in order.

$$\mathsf{ll}(L, zs) \triangleq \begin{cases} L \mapsto \lambda.\,() & \text{for } zs = [] \\ \exists L'.\ L \mapsto (\lambda\rho.\,(z, L'(\rho))) * \mathsf{ll}(L', zs) & \text{for } zs = z :: zs \end{cases}$$

Next, we define $\mathsf{sll}(L, zs) := \mathsf{ll}(L, zs) * \mathsf{sorted}(zs)$ to add that the lists be sorted. Using $\mathsf{sll}(L, zs)$ we prove standard non-probabilistic specifications for the list operations, e.g.,

$$\{\mathsf{sll}(L, zs_1 +\!\!+ z :: zs_2)\}\ \text{del}\ z\ L\ \{V.\exists L'.\ \lceil V = L'\rceil * \mathsf{sll}(L', zs_1 +\!\!+ zs_2)\}$$

Finally, we use Amaryllis's conditioning operator $\mathbf{C}$ to lift sll to the predicate $\mathsf{psll}(L, ds)$ that relates a random list L to a set ds of random elements and their distributions. The distributions must be independent.

$$\begin{aligned}
\mathsf{psll}(L, ds) \triangleq \mathbf{C}_{(v_1,\ldots,v_n) \leftarrow \pi_2(ds_1) \otimes \cdots \otimes \pi_2(ds_n)}\ \exists zs.\ (&\mathsf{sll}(L, zs) * \\
&\lceil \pi_1(ds_1) = v_1 \wedge \ldots \wedge \pi_1(ds_n) = v_n\rceil * \\
&zs \text{ is a permutation of } v_1, \ldots, v_n)
\end{aligned}$$

The proof of the probabilistic specification of init is straightforward. In verifying del, a key step is the transformation of the postcondition into a conditioning on $\mu \otimes \pi_2(ds_1) \otimes \cdots \otimes \pi_2(ds_n)$ using the strong convexity of the sll predicate. After that, an application of c-wp-swap and c-cons leads to a proof goal with deterministic values, which is established using the non-probabilistic specification of del. The proof strategy for ins is similar. For more details, refer to our Rocq development [3]. □

To further test modularity of Amaryllis, we verified a client for our list. The client iterates from 0 to i and, at each step, randomly decides whether to insert an element sampled from Unif_i or to not insert anything:

$$\begin{aligned}
&\texttt{rec}\ \text{rand_list}\ i = \texttt{if}\ i \le 0\ \texttt{then}\ \text{init}()\ \texttt{else} \\
&\quad \texttt{if}\ \texttt{flip}(1/2)\ \texttt{then}\ \text{ins}(\text{u}(i), \text{rand_list}(i-1))\ \texttt{else}\ \text{rand_list}(i-1)
\end{aligned}$$

We have shown that, conditional on the outcomes of the $\texttt{flip}(1/2)$ operations, we obtain a probabilistic sorted linked list with a length corresponding to the number of successful flips:

$$\begin{aligned}
&\{\mathsf{True}\}\ \text{rand_list}\ n\ \{V.\ \exists L.\ \lceil V = L\rceil * \\
&\qquad \mathbf{C}_{bs \leftarrow \mu_{\text{nflips}}}(\exists xs.\ \mathsf{psll}(L, xs) * \text{len}(xs) = \#\text{ones}(bs))\}
\end{aligned}$$

This postcondition also entails that the elements of the list are conditionally independent. The proof can be found in our Rocq development [3]. □

6 Related Work

We divide probabilistic logics in two main groups: the *general-purpose probabilistic logics* (GPLs) and the *property-specific probabilistic logics* (PPLs). GPLs provide native support for assertions describing properties of the whole *distribution* over program states (*e.g.,* probabilistic (conditional) independence, which is not directly expressible in PPLs). PPLs instead provide ways to reason about

specific probabilistic properties of interest (*e.g.,* error bounds, expected time complexity, etc.) within the framework of standard Hoare/separation logics (*e.g.,* Iris, WP calculi).

General-purpose probabilistic logics. Ellora [7] advocate for assertions on probability distributions in the context of Hoare-style logic. PSL [9] pioneered the idea of internalizing probabilistic independence via separating conjunction for a discrete probabilistic imperative language on a fixed set of variables. The separation of PSL insists on both disjointness of variables (*i.e.,* locations) and probabilistic independence *at the same time*, which (as explained in Section 2.2) is problematic when considering dynamic memory allocation. Reasoning under conditioning was then studied in DIBI [6], which internalized conditioning using a non-commutative connective in addition to PSL-style separation.

Lilac [20] introduced a measure-theoretic model of separation that internalizes *purely* probabilistic independence of *continuous* random variables. Lilac's programs are first-order functional programs with bounded loops, immutable state, and the ability to sample from continuous distributions. On top of the new model of separation (refined in follow-up work [21]), Lilac proposed to express reasoning under conditioning by using a new *conditioning* modality based on disintegrations.

Later, Bluebell [5] proposed a variant of Lilac's conditioning modality (in the context of *discrete* distributions) which was shown to satisfy very modular entailment rules, and could encode and mix both unary and relational concepts within a unified framework. The Bluebell logic was the first to show how mutable state could be integrated with Lilac's notion of separation without conflating disjointness and independence, but its solution is specific to finite variable stores and does not generalize to dynamic allocation. Furthermore, Bluebell illustrated the advantages of c-wp-swap, but their model only supports the rule with a heavy side-condition that disables the use of frame around it, thus inhibiting modularity.

BaSL [14] looks at the problem of supporting the Bayesian updating statement **observe** in a Lilac-like logic. Like Lilac, BaSL also lacks a general c-wp-swap rule, and remedies this by providing some specialized rules for conditional sampling. Understanding the conditions under which c-wp-swap can be validated in a model for continuous distributions remains an open problem.

Outcome Separation Logic [31] is a probabilistic logic for a language with dynamic allocation, but where separation is used exclusively for heap disjointness, not probabilistic independence.

pcOL [32] is the first logic to integrate invariants and concurrency into a GPL. pcOL supports almost-surely terminating programs with finite variable stores, sampling from discrete distributions, and adopts a Lilac/Bluebell-style conditioning modality, that also enjoys a rather general version of the c-wp-swap rule (their Split1 and Split2 rules). The proof of soundness, however, relies on the simplifying effect of adopting a PSL-style separation conflating disjointness and independence, and thus is fundamentally incompatible with dynamic allocation. Moreover, our approach based on PFP updates allows us to unify Split1 and Split2 and remove some of their side-conditions, widening their applicability.

Concerning mechanization of GPLs, there is some recent work by Yan et al. [29] that mechanizes PSL in Isabelle/HOL and applies it to security verification for oblivious algorithms. However, to our knowledge, Amaryllis is the first GPL supporting both independence and conditioning that has been mechanized in a proof assistant.

Property-specific probabilistic logics. The property-specific logics for probabilistic programs stick to traditional assertions over state, but include resources that can be used to prove certain specific properties of the induced distributions (*e.g.,* expectations, bounds on probability of events). One subcass of this category are *quantitative logics* where assertions evaluate to a quantity, like probability or expected value, instead of a binary truth value [10, 18, 19]. This idea nicely enables modular reasoning and leads to automation, but can't be applied to other probabilistic properties like independence.

Most closely related to our work are PPLs based on Iris [16], where assertions are standard separation logic assertions on state, augmented with ghost state representing some quantity related to the distribution. Eris [1] aims at proving bounds on the probability that the postcondition will be violated. A very nice feature of Eris is that the bounds themselves are treated as ghost resources, called "error credits". Tachis [13] uses the same technique to prove bounds on expected runtime. Coneris [22] extends the approach to handle concurrency.

PPLs have been also shown to work well for proving relational properties, *e.g.,* equivalence of the distribution induced by two programs. The pRHL [8] logic pioneered the use of probabilistic couplings to reduce relational properties to proving relational triples with standard assertions on state. Polaris [27] and Clutch [11], for example, prove contextual refinement in Iris, using probabilistic couplings. Approxis [12] provides a way to prove *approximate* refinements. Bluebell showed how couplings can be derived on top of conditioning in a relational GPL, which could offer an approach for a future relational extension of Amaryllis.

As a result of using standard assertions over state, the Iris-based PPLs can reuse the Iris infrastructure as is (including step-indexing and impredicative invariants), and support very rich source languages including higher-order features and sometimes concurrency. None of them internalize probabilistic independence, however, and they are specialized to proving specific properties of the program semantics. As a future direction, it would be interesting to encode these abstractions (*e.g.,* error credits) on top of the Amaryllis base logic, for example by deriving variations of our WP that encode the abstraction-specific invariants.

Nondeterminism and Outcome Logic. Integrating nondeterminism and random choices in a compositional way is a longstanding issue, given that general solutions have been proven impossible [33]. A well-known solution, proposed by Varacca and Winskel [28], shows that by representing distributions with the more detailed *indexed valuations*, which record the random choices that lead to an outcome through indexes, probabilistic choice can be combined with nondeterminism compositionally. As explained in Section 2.2, the use of an indexed valuations-style model in Amaryllis is not motivated by nondeterminism (of the allocator), but by our need to lift the underlying notion of separation of state to separation of distributions. As a byproduct of using indexed valuations, we can also combine random choices with nondeterminism in the allocator, and prove triples that are valid independently of the specific (non-probabilistic) implementation of allocation one picks.

Demonic Outcome Logic [30] explored how one can reason about nondeterminism where the choice can be made according to unknown probabilistic choices, by using the convex powerset construction [28]. As interesting future work, one could adapt the operational semantics used by Amaryllis so to match the convex sets of the demonic semantics, and prove triples which are valid even when the allocator is probabilistic. This extension might open the opportunity to encode pcOL-style reasoning for concurrency in Amaryllis.

Advanced language features. While many GPLs do not support reasoning about almost-surely terminating loops (*i.e.,* loops that terminate with probability 1), standard techniques to prove almost-sure termination (*e.g.,* [24, 23]) can be integrated in these logics, as demonstrated by the support for loops in pcOL [32]. To be able to do so in Amaryllis, we would first need to remove the finite-support restriction on the probability spaces in the model, which is the subject of future work.

As we remarked, many Iris-based PPLs can handle advanced features like impredicative invariants (and thus can reason about higher-order state and concurrency). The support for impredicativity and higher-order ghost state [15] hinges on the step-indexing technique [4, 2]. Integrating step-indexing in the model of Amaryllis is thus an important and challenging future direction as well.

References

[1] Alejandro Aguirre, Philipp G. Haselwarter, Markus de Medeiros, Kwing Hei Li, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2024. Error credits: Resourceful reasoning about error bounds for higher-order probabilistic programs. *Proc. ACM Program. Lang.*, 8, ICFP, 284–316. DOI: 10.1145/3674635.

[2] Amal Ahmed. 2004. *Semantics of types for mutable state.* Ph.D. Dissertation. Princeton University.

[3] Anonymous. 2026. Rocq development of "First Steps Towards Probabilistic Iris: A Separation Logic with Independence, Conditioning, and Dynamic Heap Allocation". (Jan. 2026). https://zenodo.org/records/18327732?preview=1&token=eyJhbGciOiJIUzUxMiJ9.eyJpZCI6IjFhNGVhZDMyLWNjMzMtNGFmOS05MzMyLTg2NGQ3NDkyNjNlMyIsImRhdGEiOnt9LCJyYW5kb20iOiJjZmJkYmZjNTI0MmQ1ZDQ4NzhjYjE3Nzg0NGE5NjViYiJ9.pzQNLX96n8aCRyB-SzfV71-1oKLAPXm-I_N4-OyrwALViOuoY7NU-kAOah7VY2N6OD066UbuqN3TdSb22nqXsg.

[4] Andrew W. Appel and David A. McAllester. 2001. An indexed model of recursive types for foundational proof-carrying code. *TOPLAS*, 23, 5, 657–683. DOI: 10.1145/504709.504712.

[5] 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, 1719–1749. DOI: 10.1145/3704894.

[6] Jialu Bao, Simon Docherty, Justin Hsu, and Alexandra Silva. 2021. A bunched logic for conditional independence. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 1–14. DOI: 10.1109/LICS52264.2021.9470712.

[7] Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. An assertion-based program logic for probabilistic programs. In *Programming Languages and Systems*. Springer International Publishing, Cham, 117–144.

[8] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanesco, and Pierre-Yves Strub. 2015. Relational reasoning via probabilistic coupling. In Springer, 387–401. DOI: 10.1007/978-3-662-48899-7_27.

[9] Gilles Barthe, Justin Hsu, and Kevin Liao. 2020. A probabilistic separation logic. *Proc. ACM Program. Lang.*, 4, POPL, 55:1–55:30. DOI: 10.1145/3371123.

[10] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Thomas Noll. 2019. Quantitative separation logic: a logic for reasoning about probabilistic pointer programs. *Proc. ACM Program. Lang.*, 3, POPL, 34:1–34:29. DOI: 10.1145/3290347.

[11] Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2024. Asynchronous probabilistic couplings in higher-order separation logic. *Proc. ACM Program. Lang.*, 8, POPL, 753–784. DOI: 10.1145/3632868.

[12] Philipp G. Haselwarter, Kwing Hei Li, Alejandro Aguirre, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2025. Approximate relational reasoning for higher-order probabilistic programs. *Proc. ACM Program. Lang.*, 9, POPL, 1196–1226. DOI: 10.1145/3704877.

[13] Philipp G. Haselwarter, Kwing Hei Li, Markus de Medeiros, Simon Oddershede Gregersen, Alejandro Aguirre, Joseph Tassarotti, and Lars Birkedal. 2024. Tachis: higher-order separation logic with credits for expected costs. *Proc. ACM Program. Lang.*, 8, OOPSLA2, 1189–1218. DOI: 10.1145/3689753.

[14] 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, (Jan. 2026), 29 pages. DOI: 10.1145/3776696.

[15] Ralf Jung, Robbert Krebbers, Lars Birkedal, and Derek Dreyer. 2016. Higher-order ghost state. In *ICFP*, 256–269. DOI: 10.1145/2951913.2951943.

[16] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *JFP*, 28, e20. DOI: 10.1017/S0956796818000151.

[17] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *POPL*, 637–650.

[18] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest precondition reasoning for expected run-times of probabilistic programs. In *ESOP* (LNCS). Vol. 9632. Springer, 364–389. DOI: 10.1007/978-3-662-49498-1_15.

[19] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest precondition reasoning for expected runtimes of randomized algorithms. *J. ACM*, 65, 5, 30:1–30:68. DOI: 10.1145/3208102.

[20] John M. Li, Amal Ahmed, and Steven Holtzen. 2023. Lilac: A modal separation logic for conditional probability. *Proc. ACM Program. Lang.*, 7, PLDI, 148–171. DOI: 10.1145/3591226.

[21] John M. 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 2024, Tallinn, Estonia, July 8-11, 2024*. Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, (Eds.) ACM, 55:1–55:14. DOI: 10.1145/3661814.3662135.

[22] Kwing Hei Li, Alejandro Aguirre, Simon Oddershede Gregersen, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2025. Modular reasoning about error bounds for concurrent probabilistic programs. *Proc. ACM Program. Lang.*, 9, ICFP, Article 245, (Aug. 2025), 30 pages. DOI: 10.1145/3747514.

[23] Rupak Majumdar and V. R. Sathiyanarayana. 2025. Sound and complete proof rules for probabilistic termination. *Proc. ACM Program. Lang.*, 9, POPL, 1871–1902. DOI: 10.1145/3704899.

[24] Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science*. Springer. ISBN: 978-0-387-40115-7. DOI: 10.1007/B138392.

[25] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. 2001. Local reasoning about programs that alter data structures. In *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings* (Lecture Notes in Computer Science). Laurent Fribourg, (Ed.) Vol. 2142. Springer, 1–19. DOI: 10.1007/3-540-44802-0_1.

[26] John C. Reynolds. 2002. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*. IEEE Computer Society, 55–74. DOI: 10.1109/LICS.2002.1029817.

[27] Joseph Tassarotti and Robert Harper. 2019. A separation logic for concurrent randomized programs. *Proc. ACM Program. Lang.*, 3, POPL, 64:1–64:30. DOI: 10.1145/3290377.

[28] Daniele Varacca and Glynn Winskel. 2006. Distributing probability over non-determinism. *Math. Struct. Comput. Sci.*, 16, 1, 87–113. DOI: 10.1017/S0960129505005074.

[29] Pengbo Yan, Toby Murray, Olga Ohrimenko, Van-Thuan Pham, and Robert Sison. 2025. Combining classical and probabilistic independence reasoning to verify the security of oblivious algorithms. In *Formal Methods*. André Platzer, Kristin Yvonne Rozier, Matteo Pradella, and Matteo Rossi, (Eds.) Springer Nature Switzerland, Cham, 188–205. ISBN: 978-3-031-71162-6.

[30] Noam Zilberstein, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. 2025. A demonic outcome logic for randomized nondeterminism. *Proc. ACM Program. Lang.*, 9, POPL, 539–568. DOI: 10.1145/3704855.

[31] Noam Zilberstein, Angelina Saliling, and Alexandra Silva. 2024. Outcome separation logic: local reasoning for correctness and incorrectness with computational effects. *Proc. ACM Program. Lang.*, 8, OOPSLA1, 276–304. DOI: 10.1145/3649821.

[32] Noam Zilberstein, Alexandra Silva, and Joseph Tassarotti. 2026. Probabilistic concurrent reasoning in outcome logic: independence, conditioning, and invariants. *Proc. ACM Program. Lang.*, 10, POPL.

[33] Maaike Zwart and Dan Marsden. 2019. No-go theorems for distributive laws. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019,*

Vancouver, BC, Canada, June 24-27, 2019. IEEE, 1–13. DOI: 10.1109/LICS.2019.8785707.

A Proof Rules

We give a list of proof rules in this section. The proof rules for conditioning can be found in figure 3, for C-frameable in figure 4 and for WP in figure 5.

A.1 Example Proofs

Here we write down the details of the derivation that we used for the Markov blanket example in Section 5.2.

We define $\mu_0 \triangleq \mu_1 \prec \kappa_2$. Following Bayes' theorem, we define $\mu_2 \triangleq \lambda v_2.\ \sum_{v_1} \mu_0(v_1, v_2)$ and $\kappa_1 v_2 \triangleq \lambda v_1.\ \frac{\mu_0(v_1,v_2)}{\mu_2(v_2)}$ such that we can invert the conditional probability: $\mu_1 \prec \kappa_2(v_1, v_2) = \mu_2 \prec \kappa_1(v_2, v_1)$. We use this equality together with C-TRANSF and derive the following:

$$
\begin{aligned}
& \mathbf{C}_{v_1 \leftarrow \mu_1} \lceil V_1 = v_1 \rceil * \mathbf{C}_{v_2 \leftarrow \kappa_2(v_1)} \lceil V_2 = v_2 \rceil * V_3 \sim \kappa_3(v_3) \\
\vdash\ & \mathbf{C}_{v_1 \leftarrow \mu_1} \mathbf{C}_{v_2 \leftarrow \kappa_2(v_1)} \lceil V_1 = v_1 \rceil * \lceil V_2 = v_2 \rceil * V_3 \sim \kappa_3(v_3) && \text{C-FRAME} \\
\vdash\ & \mathbf{C}_{(v_1,v_2) \leftarrow \mu_1 \prec \kappa_2} \lceil V_1 = v_1 \rceil * \lceil V_2 = v_2 \rceil * V_3 \sim \kappa_3(v_3) && \text{C-FUSE} \\
\vdash\ & \mathbf{C}_{(v_2,v_1) \leftarrow \mu \prec \kappa} \lceil V_1 = v_1 \rceil * \lceil V_2 = v_2 \rceil * V_3 \sim \kappa_3(v_3) && \text{C-TRANSF} \\
\vdash\ & \mathbf{C}_{v_2 \leftarrow \mu} \mathbf{C}_{v_1 \leftarrow \kappa(v_2)} \lceil V_1 = v_1 \rceil * \lceil V_2 = v_2 \rceil * V_3 \sim \kappa_3(v_3) && \text{C-FUSE} \\
\vdash\ & \mathbf{C}_{v_2 \leftarrow \mu} \lceil V_2 = v_2 \rceil * \left(\mathbf{C}_{v_1 \leftarrow \kappa(v_2)} \lceil V_1 = v_1 \rceil \right) * V_3 \sim \kappa_3(v_3) && \text{DISTR-STR-CONVEX} \\
\vdash\ & \mathbf{C}_{v_2 \leftarrow \mu} \lceil V_2 = v_2 \rceil * V_1 \sim \kappa(v_2) * V_3 \sim \kappa_3(v_3) && \text{C-INTRO} \\
\vdash\ & \exists \mu\, \kappa.\ \mathbf{C}_{v \leftarrow \mu} \lceil V_2 = v \rceil * V_1 \sim \kappa(v) * V_2 \sim \kappa_3(v) \\
\vdash\ & \exists \mu.\ \mathbf{C}_{v \leftarrow \mu} \lceil V_2 = v \rceil * \exists \mu'.\ V_1 \sim \mu' * V_2 \sim \kappa_3(v) && \text{C-CONS}
\end{aligned}
$$

B Full Model

In this section, we present the full model by presenting five layers of resource algebras that build on top of each other:

B.1 Bluebell's PSp Resource Algebra

A unary and finite version of the Bluebell's PSp_Ω resource algebra forms the basis for our probabilistic RA [5]. The resources are probability spaces, and the resource composition is the *Independent Product*.

In case the independent product does not exist, the probability spaces compose to an invalid element $\lightning$.

B.2 Dynamic Domain Extensions with PSpD

In this section, we present a finite representation of probability spaces over $\mathit{Rid} \to \mathbb{B}$. We use Bluebell's PSp_Ω but do not choose a fixed Ω, but instead dynamically extend Ω on the fly as we encounter new randomness IDs. All randomness IDs that are not mentioned in the current probability space are considered to have an unknown distribution. That way, elements of the resource algebra can still represent the infinite probability space that we are interested in, while actually being finite.

The resource algebra PSpD in this layer explicitly tracks which RIDs are mentioned in the probability space. Its type is therefore $\mathit{Rid} \times \mathrm{PSp}$. The *domain extension operation* Ext extends an element $(I, \mathcal{F}, \mu) \in \mathrm{PSpD}$ with a set of new RIDs I', without adding any information about the distribution of those RIDs. This operation can be seen as going from a resource algebra element where the RIDs in I' are implicitly unknown to an equivalent resource algebra element where they are explicitly mentioned but still unknown.

$$
\begin{aligned}
& \mathrm{Ext}((I, \mathcal{F}, \mu), I') := \\
& \quad \mathrm{Compl}(\{\{m_1 \cup m_2 \mid m_1 \in E \wedge m_2 : I' \to \mathbb{B}\} \mid E \in \mathcal{F}\}, \mu_{ext}) \\
& \mu_{ext}(E) := \mu(\{m : I \to \mathbb{B} \mid \exists m' \in E, m \preccurlyeq m'\})
\end{aligned}
$$

The function Compl calculates the completion of the probability space. We can use the domain extension operation to define an equivalence relation on this resource algebra:

$$
\begin{aligned}
& (I_1, \mathcal{F}_1, \mu_1) \equiv (I_2, \mathcal{F}_2, \mu_2) := \\
& \quad \mathrm{Ext}((I_1, \mathcal{F}_1, \mu_1), I_2 \setminus I_1) = \mathrm{Ext}((I_2, \mathcal{F}_2, \mu_2), I_1 \setminus I_2)
\end{aligned}
$$

The resource composition is defined as follows:

$$
\begin{aligned}
& (I_1, \mathcal{F}_1, \mu_1) \cdot (I_2, \mathcal{F}_2, \mu_2) := \\
& \quad (I_1 \cup I_2,\ \mathrm{Ext}((I_1, \mathcal{F}_1, \mu_1), I_2 \setminus I_1)\ \cdot_{PSp}\ \mathrm{Ext}((I_2, \mathcal{F}_2, \mu_2), I_1 \setminus I_2))
\end{aligned}
$$

The persistent core of a resource algebra element is the *sure* part of the probability space: it consists of all probability 0 and 1 events.

$$|(I, \mathcal{F}, \mu)| = (I, \{E \in \mathcal{F} \mid \mu(E) = 0 \vee \mu(E) = 1\}, \mu)$$

Weighted Sum. Next, we'll define the weighted sum on this layer. We would like the weighted sum to represent *conditioning*, thus we require that the decomposition given by the branches of the weighted sum is induced by a random variable (the random variable we are conditioning on). This is equivalent to requring the support of the branches to be pairwise disjoint.

$$
\begin{aligned}
& \mathrm{has\_rv}(\mu, \kappa) := \\
& \quad \forall \rho.\ \forall i\ j \in \mathit{supp}\ \mu.\ \mathrm{poss}(\rho, \kappa(i)) \to \mathrm{poss}(\rho, \kappa(j)) \to i = j
\end{aligned}
$$

where $\mathrm{poss}(\rho, (\mathcal{F}, \mu)) := \exists m \in \mathit{supp}(\mathcal{F}, \mu).\ \forall \iota \in \mathrm{dom}\ (m).\ m[\iota] = \rho(\iota)$.

In the weighted sum on this layer, an event is measurable if it is measurable in all branches. Its probability is the sum of the probabilities in the branches.

$$\sigma\_\mathrm{intersec}(\mu, \kappa) := \bigcap \pi_\sigma(\mathrm{Ext}(\kappa(i), \bigcup_{i \in \mathrm{supp}(\mu)} I(\kappa(i))))$$

$$
\begin{aligned}
\mathit{wsum}(\mu, \kappa) := \mathit{bind}(\mu, \lambda i.\ \mathrm{Ext}(\kappa(i), \bigcup_{i \in \mathrm{supp}(\mu)} I(\kappa(i)))_{|\sigma\_\mathrm{intersec}(\mu,\kappa)}) \\
\text{if } \mathrm{has\_rv}(\mu, \kappa) \wedge \forall i \in \mathrm{supp}(\mu).\ \mathcal{V}(\kappa(i))
\end{aligned}
$$

$$\pi_\mu(\mathit{bind}(\mu, \kappa))(X) = \sum_{v \in \mathit{supp}(\mu)} \mu(v) \cdot \pi_\mu(\kappa(v))(X)$$

Operation $\circ$. We define the operation $\circ$ on this layer using the weighted sum:

$$
\begin{aligned}
& (I_1, \mathcal{F}_1, \mu_1) \circ (I_2, \mathcal{F}_2, \mu_2) = \\
& \mathit{wsum}(\mu_1|_{\mathrm{atoms}(\mathcal{F}_1 \cap \mathcal{F}_2)},\ \lambda A.\ (F_1, \mu_1)|_A \cdot (F_2, \mu_2)|_A) \\
& \qquad \text{if } \forall E \in F_1 \cap F_2.\ \mu_1(E) = \mu_2(E)
\end{aligned}
$$

$$\vdash X \sim \mu \mathrel{-\!\!*} \lceil X = Y \rceil \mathrel{-\!\!*} Y \sim \mu \qquad \vdash (\lambda.\, v) \sim \delta(v) \qquad \mathsf{Own}(\varepsilon, X) * \lceil X = Y \rceil \vdash \mathsf{Own}(\varepsilon, Y) \qquad \lceil X = Y \rceil \vdash \Box \lceil X = Y \rceil$$

$$\mathsf{StrConvex}(\Psi) \triangleq \mathbf{C}_{v \leftarrow \mu}\, \Phi(v) * \Psi \vdash \Psi * \mathbf{C}_{v \leftarrow \mu}\, \Phi(v)$$

C-UPD-SWAP

$$\mathbf{C}_{x \leftarrow \mu} \Rrightarrow P(x) \vdash \Rrightarrow \mathbf{C}_{x \leftarrow \mu}\, P(x)$$

C-CONS

$$\frac{\forall v \in \mathrm{supp}(\mu).\ \Phi_1(v) \vdash \Phi_2(v)}{\mathbf{C}_{v \leftarrow \mu}\, \Phi_1(v) \vdash \mathbf{C}_{v \leftarrow \mu}\, \Phi_2(v)}$$

C-FUSE

$$\mathbf{C}_{v \leftarrow \mu}\, \mathbf{C}_{w \leftarrow \kappa(v)}\, \Phi(v, w) \dashv\vdash \mathbf{C}_{(v,w) \leftarrow \mu \prec \kappa}\, \Phi(v, w)$$

C-FORALL

$$\mathbf{C}_{v \leftarrow \mu}\, \forall x.\ \Phi(x, v) \vdash \forall x.\ \mathbf{C}_{v \leftarrow \mu}\, \Phi(x, v)$$

C-INTRO

$$X \sim \mu \dashv\vdash \mathbf{C}_{v \leftarrow \mu} \lceil X = v \rceil$$

C-RV

$$\mathbf{C}_{v \leftarrow \mu}\, \Phi(v) \vdash \exists X.\ \mathbf{C}_{v \leftarrow \mu} \lceil X = v \rceil * \Phi(v)$$

C-TRANSF

$$\frac{f \text{ injective} \qquad \forall v \in \mathrm{supp}(\mu_1).\ \mu_1(v) = \mu_2(f(v))}{\mathbf{C}_{v \leftarrow \mu_2}\, \Phi(v) \vdash \mathbf{C}_{v \leftarrow \mu_1}\, \Phi(f(v))}$$

C-SKOLEM

$$\mathbf{C}_{v \leftarrow \mu}\, \exists a.\ \Phi(a, v) \dashv\vdash \exists f.\ \mathbf{C}_{v \leftarrow \mu}\, \Phi(f(v), v)$$

C-BIND

$$\mathbf{C}_{v \leftarrow \mu}\, X \sim \kappa(v) \vdash X \sim \mathbf{bind}(\mu, \kappa)$$

C-FALSE

$$\frac{\Phi(v) \vdash \bot \qquad v \in \mathrm{supp}(\mu)}{\mathbf{C}_{v \leftarrow \mu}\, \Phi(v) \vdash \bot}$$

C-DIRAC

$$\mathbf{C}_{v \leftarrow \delta(i)}\, \Phi(v) \dashv\vdash \Phi(i)$$

C-AUTH

$$\frac{\forall v.\ Q(v) \vdash \lceil X = v \rceil}{\mathsf{Own}(\bullet g) * \mathbf{C}_{v \leftarrow \mu}\, Q(v) \vdash \mathbf{C}_{v \leftarrow \mu}(\mathsf{Own}(\bullet(g|_{X=v})) * Q(v))}$$

C-PURE

$$\mathrm{supp}\, \mu \subseteq E * \mathbf{C}_{v \leftarrow \mu}\, \Phi(v) \dashv\vdash \mathbf{C}_{v \leftarrow \mu}\, v \in E * \Phi(v)$$

PURE-STR-CONVEX

$$\frac{\Psi \text{ pure}}{\mathsf{StrConvex}(\Psi)}$$

POINTSTO-STR-CONVEX

$$\mathsf{StrConvex}(L \mapsto V)$$

DISTR-STR-CONVEX

$$\mathsf{StrConvex}(X \sim \mu)$$

Figure 3: Proof rules for conditioning

C-FRAMEABLE-MONO

$$\frac{P \vdash Q}{\boxtimes P \vdash \boxtimes Q}$$

$$\boxtimes P * \mathbf{C}_{v \leftarrow \mu}\, \Phi(v) \vdash \mathbf{C}_{v \leftarrow \mu}(\boxtimes P) * \Phi(v) \qquad \boxtimes P \vdash P \qquad \boxtimes \boxtimes P \dashv\vdash \boxtimes P \qquad \boxtimes \forall v.\ \Phi(v) \dashv\vdash \forall v.\ \boxtimes \Phi(v)$$

$$\boxtimes \exists v.\ \Phi(v) \dashv\vdash \exists v.\ \boxtimes \Phi(v) \qquad \blacksquare P \vdash \boxtimes P \qquad \boxtimes P * \boxtimes Q \dashv\vdash \boxtimes(P * Q)$$

C-FRAMEABLE-PERS

$$\square P \vdash \boxtimes P \qquad X \sim \mu \vdash \boxtimes(X \sim \mu) \qquad \mathsf{Own}(X) \vdash \boxtimes \mathsf{Own}(X)$$

Figure 4: Proof rules for C-frameable

B.3 Tracking undeterminate RIDs with PSpU

On this layer, we additionally track in each resource algebra element a set of RIDs that is *undeterminate*, i.e. that the probability space has no information on. Let $\mathcal{N}(a)$ be the largest probability 0 event in the probability space of a.

$$\begin{aligned}
\mathsf{PSpU} &:= \{(U, a) \mid a \in \mathsf{PSpD} \wedge \mathsf{is_undeterminate}(U, a)\} \\
(U, a) \equiv (U', a') &:= U = U' \wedge a \equiv a' \\
(U, a) \cdot (U', a') &:= (U \cap U', a \cdot a') \\
\mathcal{V}((U, a)) &:= \mathcal{V}(a) \\
(U, a) \circ (U, a') &:= (U \cap U', a \circ a') \\
wsum(\mu, \kappa) &:= \Big(\bigcap_{i \in \mathrm{supp}(\mu)} \pi_U(\kappa(i)), wsum(\mu, \pi_a(\kappa(i)))\Big) \\
|(U, a)| &:= (U, |a|)
\end{aligned}$$

$$\mathsf{is\_undeterminate}(U, a) := \forall E \in a.\ E \cap \mathcal{N}(a) = \emptyset \rightarrow \forall m_1 . \forall m_2\ m_3 : (U \cap I(a)) \rightarrow \mathbb{B}.\ m_1 \cup m_2 \in E \rightarrow m_1 \cup m_3 \in E$$

WP-BIND
$$\mathsf{wp}\ E\ \{V.\ \mathsf{wp}\ K[V]\ \{\Phi\}\} \vdash \mathsf{wp}\ K[E]\ \{\Phi\}$$

WP-LOAD
$$L \mapsto V \vdash \mathsf{wp}\ \mathtt{get}\, L\ \{X.\ L \mapsto V * \lceil X = V\rceil\}$$

WP-ALLOC
$$\mathsf{wp}\ \mathtt{ref}\ V\ \{V'.\ \exists L.\ \lceil V' = L\rceil * L \mapsto V\}$$

WP-SAMPLE
$$\frac{0 < p < 1}{\mathsf{wp}\ \mathtt{flip}(p)\ \{V.\ V \sim \mathsf{Ber}_p\}}$$

WP-STORE
$$L \mapsto V \vdash \mathsf{wp}\ \mathtt{set}\, L\, V'\ \{V''.\ \lceil V'' = ()\rceil * L \mapsto V'\}$$

WP-PURE
$$\frac{E \rightsquigarrow_{pure} E'}{\mathsf{wp}\ E'\ \{\Phi\} \vdash \mathsf{wp}\ E\ \{\Phi\}}$$

WP-AS-EQ
$$\lceil E_1 = E_2\rceil \twoheadrightarrow \mathsf{wp}\ E_1\ \{\Phi\} \twoheadrightarrow \mathsf{wp}\ E_2\ \{\Phi\}$$

WP-VALUE
$$\Rrightarrow \Phi(V) \vdash \mathsf{wp}\ V\ \{\Phi\}$$

WP-MONO
$$\mathsf{wp}\ E\ \{\Phi\} \twoheadrightarrow (\forall V.\ \Phi(V) \twoheadrightarrow \Rrightarrow \Psi(V)) \twoheadrightarrow \mathsf{wp}\ E\ \{\Psi\}$$

UPD-WP
$$\Rrightarrow \mathsf{wp}\ E\ \{\Phi\} \vdash \mathsf{wp}\ E\ \{\Phi\}$$

WP-UPD
$$\mathsf{wp}\ E\ \{V.\ \Rrightarrow \Phi(V)\} \vdash \mathsf{wp}\ E\ \{\Phi\}$$

C-WP-SWAP
$$\mathbf{C}_{v \leftarrow \mu}\, V.\ \mathsf{wp}\ E\ \{\Phi(v, V)\} \vdash \mathsf{wp}\ E\ \{V.\ \mathbf{C}_{v \leftarrow \mu}\, \Phi(v, V)\}$$

Figure 5: WP rules

B.4 The Authoritative Probability Space RA PSpU_Auth

Since we want to use our RA in the state interpretation, we need to make it authoritative. Additionally, we add a *Namespace* to the authoritative part that tracks a set of globally free randomness IDs. This will simplify allocating a new RID since we can ensure that it doesn't yet exist anywhere in the context.

$$\mathsf{PSpU\_Auth} := \mathsf{PSpU} \times \mathit{Option}(\mathit{Ex}(\mathsf{PSpD} \times N))$$

Composition, $\circ$ and equivalence are pointwise on the underlying pRAs. Validity requires the fragment to be included in the authoritative part, and the RIDs in the namespace to be globally undeterminate.

$$\begin{aligned}\mathcal{V}(a, \mathit{Some}(\mathit{ex}(b, N))) &:= \mathcal{V}(a) \wedge \mathcal{V}(b) \wedge \pi_{PSp}(a) \preccurlyeq_\circ b \wedge \\ &\quad N \subseteq \pi_U(b) \wedge \mathsf{is\_undeterminate}(N, b) \\ \mathcal{V}(a, \bot) &:= \mathcal{V}(a)\end{aligned}$$

The weighted sum is also pointwise, given that all branches are valid:

$$\begin{aligned}\mathit{wsum}(\mu, \kappa) := (&\mathit{wsum}(\mu, \pi_{\mathit{frag}}(\kappa)), \mathit{wsum}(\mu, \pi_{\mathit{auth}}(\kappa))) \\ &\text{if } \forall i \in \mathit{supp}(\mu).\ \mathcal{V}(\kappa(i))\end{aligned}$$

B.5 Adding Iris Resources as Random Variables with $\mathsf{PSp\_RV}_M$

Next, we add (non-probabilistc) Iris resources as random variables. For example, $\lambda\rho.\ L(\rho) \mapsto \rho(\iota)$ says that the location $L : \mathit{RV}_{\mathit{loc}}$ (which might be a different location based on the random choices) maps to whatever value the randomness ID ι resolves to.

The key challenge here is how to combine multiple Iris RVs in the weighted sum. We can use that we already require conditioning on a random variable.

First, we need a way to extract the random variable that is conditioned on from the branches (up to almost sure equality). Define $\mathsf{get\_rv}(\mu, \kappa)(\rho)$ as the unique $i \in \mathrm{supp}(\mu)$ such that $\mathrm{poss}(\kappa(i), \rho)$ if such an i exists. For example, for the weighted sum $\mathit{wsum}(\mu, \lambda v.\ [X = v])$, we have that $\mathsf{get\_rv}(\mu, \kappa) =_{as} X$.

This helps us in combining Iris resources across branches, for example:

$$\mathit{wsum}(\mu, \lambda v.\ ([X = v],\ \lambda\rho.\ L(\rho) \mapsto v)) = (X \sim \mu,\ \lambda\rho.\ L(\rho) \mapsto X(\rho))$$

Let M be a unital RA.

$$\begin{aligned}\mathsf{PSp\_RV}_M &:= \mathsf{PSpU\_Auth} \times (\mathit{RV}_M) \\ \mathcal{V}(a, X) &:= \mathcal{V}(a) \wedge \forall\rho.\ \mathcal{V}(X(\rho)) \\ (a, X) \equiv (a', X') &:= (a \equiv a') \wedge \forall\rho.\ X(\rho) \equiv X'(\rho) \\ (a, X) \cdot (b, Y) &:= (a \cdot b,\ \lambda\rho.\ X(\rho) \cdot Y(\rho)) \\ (a, X) \circ (b, Y) &:= (a \circ b,\ \lambda\rho.\ X(\rho) \cdot Y(\rho)) \\ \mathit{wsum}(\mu, \kappa) &:= (\mathit{wsum}(\mu, \pi_1(\kappa)), \\ &\qquad \lambda\rho.\ \text{if } \mathrm{poss}(\mathit{wsum}(\mu, \pi_{1,\mathit{frag}(\kappa)}), \rho) \\ &\qquad \text{then } \pi_2(\kappa)(\mathsf{get\_rv}(\mu, \pi_{1,\mathit{frag}}(\kappa))(\rho))\rho) \\ &\qquad \text{else } \varepsilon\end{aligned}$$

Note that in the random variable that we get from the weighted sum, ρs that are probability 0 in all branches are mapped to the unit ε. Those cases don't matter on the level of the logic and they cannot be mapped to a unique branch - we choose ε to ensure that those cases still remain valid.

C Operational Semantics

The language is given by a small step relation $\rightsquigarrow$. The proposition $e, \sigma \rightsquigarrow e_1, \sigma_1, e_2, \sigma_2, p$ states that expression e under state σ steps to e_1 and σ_1 with probability p and to e_2 and σ_2 with probability $1 - p$.

We lift this small step to a multi step $E_1, \Sigma_1, \mathcal{P}_1, N_1 \rightsquigarrow_{RV} E_2, \Sigma_2, \mathcal{P}_2, N_2$ which allows us to compose several steps. We are using PSpD as model for probability spaces to simplify the allocation of RIDs. Additionally, we track all globally free RIDs in N_1 and N_2. Because the expressions and state are lifted to $\mathit{RV}_{\mathit{Expr}}$ resp. $\mathit{RV}_{\mathit{State}}$ but the probability space only contains partial maps we relate partial maps and functions using $\prec$. We define $m \prec \rho := \forall \iota \in \mathrm{dom}\ (m).\ m[\iota] = \rho(\iota)$.

$$
\begin{aligned}
&E_1, \Sigma_1, \mathcal{P}_1, N_1 \rightsquigarrow_{RV} E_2, \Sigma_2, \mathcal{P}_2, N_2 := \\
&\exists X, \iota, e_1, \sigma_1, e_2, \sigma_2, p. \\
&\quad X \in \mathsf{atoms}(\mathcal{P}_1) \wedge \\
&\quad \iota \in N_1 \wedge \\
&\quad \mathcal{P}_2 = wsum(\pi_\mu(\mathcal{P}_1)|_{\mathsf{atoms}(\mathcal{P}_1)}, \\
&\qquad \lambda Y.\, \mathcal{P}_1|_Y \cdot \mathsf{if}\ X = Y\ \mathsf{then}\ \iota \sim \mathsf{Ber}_p\ \mathsf{else}\ \varepsilon) \wedge \\
&\quad N_2 = N_1 \setminus \{\iota\} \wedge \\
&\quad (\forall m \in X. \\
&\qquad E_1(m), \Sigma_1(m) \rightsquigarrow e_1, \sigma_1, e_2, \sigma_2, p) \wedge \\
&\quad (\forall m \in X, \rho_t, \rho_f. \\
&\qquad m[\iota \mapsto \mathtt{true}] \prec \rho_t \to m[\iota \mapsto \mathtt{false}] \prec \rho_f \to \\
&\qquad E_2(\rho_t) = e_1 \wedge \Sigma_2(\rho_t) = \sigma_1 \wedge \\
&\qquad (p \neq 1 \to E_2(\rho_f) = e_2 \wedge \Sigma_2(\rho_f) = \sigma_2)) \wedge \\
&\quad (\forall Y \in \mathsf{atoms}(\mathcal{P}_1), m \in Y, \rho. \\
&\qquad Y \neq X \to m \prec \rho \to \\
&\qquad E_1(\rho) = E_2(\rho) \wedge \Sigma_1(\rho) = \Sigma_2(\rho))
\end{aligned}
$$

An atom X of $\mathcal{P}_1$ and an $\iota \in N_1$ are nondeterministically chosen. The step will happen in branch X of $\mathcal{P}_1$ and it will allocate RID ι. The step in X goes to $e_1, \sigma_1, e_2, \sigma_2, p$ which means that we split X into two branches: one containing e_1, σ_1, having probability p and one containing e_2, σ_2 having probability $1 - p$. $\mathcal{P}_2$ is therefore constructed by leaving every branch of $\mathcal{P}_1$ unchanged except X which we split into two atoms X_1 and X_2 by composing with $\iota \sim \mathsf{Ber}_p$. The RID ι is undeterminate in each atom of $\mathcal{P}_2$ except in X_1 and X_2, where it is `true` in X_1 and `false` in X_2.

E_2 and Σ_2 should be equal to E_1 and Σ_1 for any ρ that is neither in X_1 nor X_2 because there was no step done in this branch. However, for a ρ_t in X_1 they should be valued as e_1 and σ_1 and similarly for X_2 to create two different branches with (e_1, σ_1) and (e_2, σ_2).

Assuming that E_1 and Σ_1 are measurable in $\mathcal{P}_1$ and the RIDs in $\mathcal{P}_1$ are disjoint of N_1 then this definition of $E_1, \Sigma_1, \mathcal{P}_1, N_1 \rightsquigarrow_{RV} E_2, \Sigma_2, \mathcal{P}_2, N_2$ has the following properties:

- E_2 and Σ_2 are measurable in $\mathcal{P}_2$
- We can rewrite with almost-sure equalities at the positions of $E_1, \Sigma_1, E_2, \Sigma_2$
- The RIDs in $\mathcal{P}_2$ are disjoint of N_2
- Two consecutive steps in different atoms of $\mathcal{P}_1$ can be swapped

From this we build our big step semantics:

BIG-STEP-REFL

$$\frac{\forall m \in \mathrm{supp}(\mathcal{P}), \rho.\ m \prec \rho \to E(\rho) = V(\rho) \qquad \lceil \Sigma_1 = \Sigma_2 \rceil_{\mathcal{P}}}{E, \Sigma_1, \mathcal{P}, N \Downarrow V, \Sigma_2, \mathcal{P}, N}$$

BIG-STEP-STEP

$$\frac{E_1, \Sigma_1, \mathcal{P}_1, N_1 \rightsquigarrow_{RV} E_2, \Sigma_2, \mathcal{P}_2, N_2 \quad E_2, \Sigma_2, \mathcal{P}_2, N_2 \Downarrow V, \Sigma_3, \mathcal{P}_3, N_3}{E_1, \Sigma_1, \mathcal{P}_1, N_1 \Downarrow V, \Sigma_3, \mathcal{P}_3, N_3}$$

They are collecting semantics, collecting all computations from each branch of the computation tree into one evaluation. Each branch of the tree is evaluated until it results into a value but the order in which the branches are evaluated is nondeterministic. The fact that a multi step only changes one branch, let's us easily decompose the computation tree into its subtrees. If we have given $E_1, \Sigma_1, \mathcal{P}_1, N_1 \Downarrow V, \Sigma_2, \mathcal{P}_2, N_2$ and some random variable $(X \sim \mu)_{\mathcal{P}_1}$, then we can find a big step evaluation starting from $E_1, \Sigma_1, \mathcal{P}_1|_{X=v}, N'$ for all $v \in \mathrm{supp}(\mu)$. This fact is crucial for the proof of C-WP-SWAP.