New foundations for probabilistic separation logic[*][†]

John Li[1], Amal Ahmed[2], and Steven Holtzen[3]

[1,2,3]Northeastern University

Probabilistic reasoning frequently requires decomposing a situation into probabilistically independent pieces. We present a separation logic supporting this decomposition. Inspired by an analogy with mutable state where sampling corresponds to dynamic allocation, we show how probability spaces over a fixed, ambient sample space appear to be the natural analogue of heap fragments, and present a new combining operation on them such that probability spaces behave like heaps and measurability of random variables behaves like ownership. Unlike prior work [1], the resulting program logic enjoys a frame rule identical to the ordinary one, and naturally accommodates advanced features like continuous random variables and reasoning about quantitative properties of programs.

To illustrate the analogy between probability and mutable state, consider the following program:

$$\begin{aligned}&\texttt{let } X = \texttt{flip } 1/2 \texttt{ in}\\&\texttt{let } Y = \texttt{flip } 1/2 \texttt{ in}\\&\texttt{observe } X = \texttt{T};\\&\texttt{ret } (X, Y)\end{aligned} \qquad (\textsc{flip}2)$$

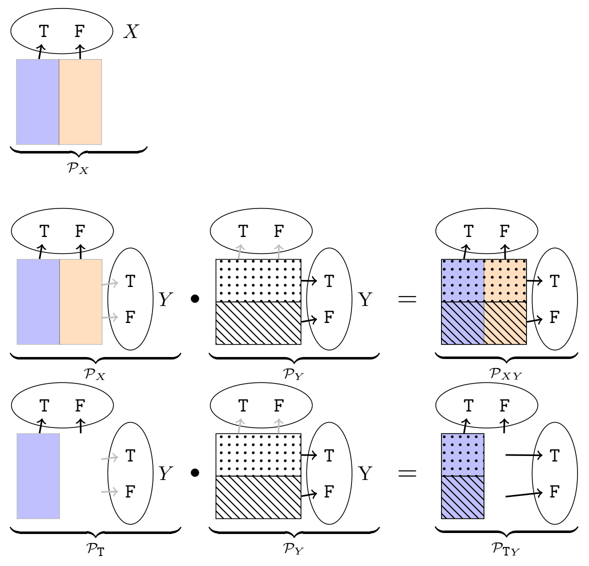
This program uses `flip` to sample the result of two uniformly distributed values in $\{\texttt{T}, \texttt{F}\}$, and `observe` to condition on the event $X = \texttt{T}$. The idea is to think of this program as carrying along a probability space $(\Omega, \mathcal{F}, \mu)$ as it executes, analogous to how programs with mutable state carry along a heap. For this example we fix the sample space Ω to be the unit square $[0, 1] \times [0, 1]$. This allows us to visualize σ-algebras $\mathcal{F}$ as partitionings of the unit square into a finite number of regions – each region denotes an event – and μ as the map that sends each event to its area. We can visualize execution of $\textsc{flip}2$ as:

$\texttt{let } X = \texttt{flip } 1/2 \texttt{ in}$

$\texttt{let } Y = \texttt{flip } 1/2 \texttt{ in}$

$\texttt{observe } X = \texttt{T};$

$\texttt{ret } (X, Y)$



*Published as an extended abstract at LAFI 2023

†Full paper at `https://john-ml.github.io/lilac-a-modal-separation-logic-for-conditional-probability.pdf`

Initially, $\mathcal{F}$ only contains the trivial events $\emptyset$ and Ω, analogous to how programs in heap-manipulating languages start with an empty heap. After the first line is executed two things change:

- Two new events are allocated, each covering half the sample space. These are the blue partition on the left and the orange partition on the right; they form the probability space labeled $\mathcal{P}_X$. This is like how `new` allocates a fresh memory cell on the heap: probability spaces correspond to heap fragments.
- The `flip` operation yields a random variable $X : \Omega \to \{\mathtt{T}, \mathtt{F}\}$ that maps blue points in Ω to `T` and orange points in Ω to `F`; this is depicted by the arrows. Concretely, $X(\omega_1, \omega_2) = \mathtt{T}$ if $\omega_1 < 1/2$ and `F` otherwise. This is like how `new` returns the location of the newly allocated heap cell: random variables correspond to locations.

The next line allocates a second probability space $\mathcal{P}_Y$, visualized by the dotted region and dashed region, and a new random variable Y that associates dotted points to `T` and dashed points to `F`; concretely, $Y(\omega_1, \omega_2) = \mathtt{T}$ if $\omega_2 > 1/2$ and `F` otherwise. In heap-manipulating languages, `new` generates a *fresh* heap cell, so that the entire heap after executing `new` is a *disjoint union* of the old heap and the newly allocated cell. Analogously, `flip` allocates a probability space *statistically independent* from the old one, so that the entire space after executing the second `flip` is what we dub an *independent combination*, written $(\bullet)$, of the old space $\mathcal{P}_X$ and the newly allocated one $\mathcal{P}_Y$. The space labeled $\mathcal{P}_{XY}$ visualizes the independent combination $\mathcal{P}_X \bullet \mathcal{P}_Y$. The events of $\mathcal{P}_{XY}$ (i.e., the four partitions in the figure) form the σ-algebra generated by the events of $\mathcal{P}_X$ and $\mathcal{P}_Y$; their probabilities, as suggested by the areas of the partitions, are products of probabilities of corresponding events in $\mathcal{P}_X$ and $\mathcal{P}_Y$.

Finally, the `observe` statement conditions on the event $X = \mathtt{T}$. This destructively updates the underlying probability space from the independent combination $\mathcal{P}_X \bullet \mathcal{P}_Y$ to the independent combination $\mathcal{P}_{\mathtt{T}} \bullet \mathcal{P}_Y$, where $\mathcal{P}_{\mathtt{T}}$ is the probability space which assigns probability 1 to the event $X = \mathtt{T}$. Since X is statistically independent from Y, this observe statement does not affect the distribution of Y in any way; this is completely analogous to how, in the heap-manipulating setting, a write to a location ℓ does not affect the values stored in locations disjoint from ℓ.

This intuition that probabilistic programs manipulate probability spaces as they execute motivates our model of separation logic. In particular, the insight that disjoint union of heaps corresponds to independent combinations of probability spaces underlies our interpretation of the standard separation logic connectives. We prove that probability spaces over a fixed, ambient sample space form a Kripke resource monoid [2] under independent combination. This gives a standard interpretation of separating conjunction, in which $\mathcal{P} \models P_1 * P_2$ iff there exists a "splitting" of $\mathcal{P}$ into an independent combination $\mathcal{P}_1 \bullet \mathcal{P}_2$ such that $\mathcal{P}_1 \models P_1$ and $\mathcal{P}_2 \models P_2$, and validates all the usual laws of separation logic. Atop this foundation, we give meaning to Hoare triples and to the following probability-specific propositions:

- $\mathsf{own}\, x$ asserts "ownership" of a random variable x but no knowledge of its distribution, like $x \mapsto -$ in ordinary separation logic. Its interpretation is given by measurability of the random variable denoted by x with respect to the underlying probability space.
- $x \sim \mu$ asserts a random variable x is distributed as μ, like $x \mapsto v$ in ordinary separation logic.
- $x = y$ is equality of random variables.

Then we validate the usual derived rules; in particular, our logic enjoys a frame rule identical to the usual one, and rules for `flip` and `observe` that closely resemble the corresponding rules for allocation and mutable update in ordinary separation logic:

$$\textsc{Frame}\ \frac{\{P\}\ e\ \{x.\, Q(x)\}}{\{F * P\}\ e\ \{x.\, F * Q(x)\}} \qquad \textsc{Flip}\ \{\mathsf{emp}\}\ \mathsf{flip\ p}\ \{x.\, x \sim \mathrm{Ber}\, p\} \qquad \textsc{Observe}\ \{\mathsf{own}\ b\}\ \mathtt{observe}\ b\ \{x.\, b = \mathtt{T}\}$$

As evidence that our interpretation of separating conjunction faithfully captures probabilistic independence, we prove that the iterated separating conjunction $\mathsf{own}\ x_1 \ * \ \ldots \ * \ \mathsf{own}\ x_n$ is semantically valid if and only if the random variables denoted by $x_1, \ldots, x_n$ are mutually independent.

Though we only mentioned programs involving discrete random variables here, our model of separation logic is defined in terms of general probability spaces, and so supports reasoning about programs that sample from continuous distributions as well.

Finally, we sketch how our logic can be further extended to support reasoning about probabilities of specific events, expectations of random variables, and conditional probability; thanks to the close correspondence between our semantic model and standard probability-theoretic objects (probability spaces and random variables), we describe how these extensions can be performed in a relatively straightforward way.

References

[1] Gilles Barthe, Justin Hsu, and Kevin Liao. A probabilistic separation logic. *Proc. ACM Program. Lang.*, 4(POPL), dec 2019.

[2] Didier Galmiche, Daniel Méry, and David Pym. The semantics of bi and resource tableaux. *Mathematical Structures in Computer Science*, 15(6):1033–1088, 2005.