Towards Symbolic Execution for Probability and Non-determinism

Jack Czenszak*
Northeastern University
Boston, MA, United States

John M. Li*
Northeastern University
Boston, MA, United States

Steven Holtzen
Northeastern University
Boston, MA, United States

Knowledge compilation is emerging as an effective tool for scaling exact inference to complex probabilistic programs [5, 15, 18]. The essence of knowledge compilation is to encode a probabilistic program as a compact data structure, such as a binary decision diagram. This inference strategy strongly resembles *symbolic execution*. In parallel work, we have used this observation to repurpose the symbolic executor Rosette [21, 22] to perform probabilistic inference. Here, we aim to develop the mathematical foundation for this adaptation of Rosette and expand its applications beyond probability and non-determinism.

The key insight is that both probabilistic choice and non-deterministic choice are (commutative) affine effects. We show that every affine effect gives rise to a denotational semantics for symbolic execution. This semantics is inspired by models of name generation [8, 16, 19, 20] and the sheaf-theoretic approach to the Giry monad [17]. Our semantics is based on a category of worlds containing *symbolic heaps*. Types then denote world-dependent sets, and programs denote world-dependent functions. We show that this semantics is sound and complete in the sense of Torlak and Bodik [22]: it captures all and only those outcomes that are reachable by computation using the original affine effect (Theorem 0.1).

Affine Effects. Probability and non-determinism, without observation and failure, are both *affine effects* [4]. Programmatically, this means that (i) two monadic let-bindings may be exchanged if doing so does not create a free variable and (ii) a monadic let-binding may be dropped if its bound variable is not referenced. Figure 1 depicts these rules for a monadic PPL with coin flips. We say that a monad is affine if it validates these two equations.[1]

Noticably, this purely monadic formulation of affine effects is presentation-invariant. For example, non-determinism is typically presented by a single primitive operation $\overline{\mathsf{amb}} : 1 \to \mathcal{P}_\#(\mathbb{B})$ [10], but the usual model of non-determinism—the non-empty, finite power set monad $\mathcal{P}_\#$ [11]—does not distinguish $\overline{\mathsf{amb}}$ from other maps $1 \to \mathcal{P}_\#(\mathbb{B})$. In symbolic execution, the presentation of an effect in terms of primitive operations is crucial: a symbolic executor represents the results of primitive operations as *symbolic variables*.

In order to work with a particular presentation of an affine effect, we import the theory of algebraic effects [12, 13].

*Both authors contributed equally to this research.

[1] In Kaddar and Staton [6, Definition 2.3], this is called the *dataflow property*.

$$\begin{pmatrix} x \leftarrow \overline{\mathsf{flip}}\ 1/7; \\ y \leftarrow \overline{\mathsf{flip}}\ 1/4; \\ \mathsf{ret}\ (x, y) \end{pmatrix} \equiv \begin{pmatrix} y \leftarrow \overline{\mathsf{flip}}\ 1/4; \\ x \leftarrow \overline{\mathsf{flip}}\ 1/7; \\ \mathsf{ret}\ (x, y) \end{pmatrix} \qquad \begin{pmatrix} x \leftarrow \overline{\mathsf{flip}}\ 1/3; \\ \mathsf{ret}\ 3 \end{pmatrix} \equiv (\mathsf{ret}\ 3)$$

Figure 1. Examples of commutative (left) and affine (right) equational laws for a monadic PPL with coin flips.

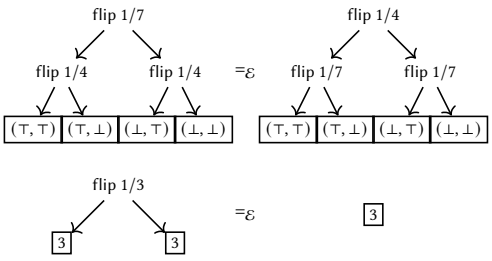

Figure 2. Examples of commutative (top) and affine (bottom) equational laws from Figure 1, as algebraic syntax trees. Each left subtree corresponds to the "true" branch of computation.

Following Bauer [2], we say an *algebraic theory* $\mathbb{T} = (\Sigma, \mathcal{E})$ consists of a *signature* $\Sigma = \{(\mathsf{op}_i, A_i)\}_{i \in I}$ and *equations* $\mathcal{E}$. Each *operation symbol* op_i in Σ has *arity* A_i. Informally, an arity is the set of possible outcomes for a particular operation symbol. For instance, the operation symbols amb and flip p—where p is a rational number between 0 and 1—both have arity $\mathbb{B}$.

Every theory $\mathbb{T}$ freely generates a monad $\mathcal{T}$ on the category of sets [2, 3, 9]. We briefly recall the construction of this monad. Given Σ and a set X, define $\mathrm{Tree}_\Sigma(X)$ to be the set of trees where (i) every leaf must be an element of X, and (ii) every node is an A_i-ary operation symbol op_i in Σ together with $|A_i|$-many subtrees. The monad $\mathcal{T}$ maps a set X to the set $\mathrm{Tree}_\Sigma(X)$ quotiented by the equations in $\mathcal{E}$. Tangibly, $\mathcal{T}(X)$ models computations that depend on the output of effectful operations and produce a value in X. We say $\mathbb{T}$ is an *affine algebraic theory* if $\mathcal{T}$ models an affine effect.

In the monadic programming language induced by $\mathcal{T}$, each operation symbol op in Σ with arity A automatically corresponds to a *generic operation* $\overline{\mathsf{op}} : 1 \to \mathcal{T}(A)$:

$$\overline{\mathsf{op}}() = [\mathsf{op}(a_1, \ldots, a_n)] \qquad a_1, \ldots, a_n \in A$$

That is, $\overline{\mathsf{op}}()$ is the equivalence class containing the syntax tree with one node op and a leaf for each $a_i \in A$. Figure 2 reiterates the examples in Figure 1 using generic operations.

Note the similarity between Figure 2 and probabilistic decision trees.

The function of a symbolic executor is to compute compact representations of these decision trees. To do this, symbolic executors require *symbolic variables*, which serve as abstract representations of the outcomes of effectful operations. As our work seeks to justify the probabilistic adaptation of Rosette, we recount symbolic variables in the context of *symbolic unions*.

Symbolic Unions. Rosette represents non-deterministic values as symbolic unions, which are dynamically generated and merged throughout a program's run time [14, 22]. When Rosette is restricted to Boolean non-determinism, new symbolic unions are introduced using the operation $\widehat{\mathsf{sym}}$, the symbolic analogue of $\overline{\mathsf{amb}}$.

To evaluate $\widehat{\mathsf{sym}}$, Rosette returns a *fresh* symbolic union

$$\widehat{\mathsf{sym}} \Downarrow [\mathbf{x} : \top, \neg\mathbf{x} : \bot]$$

for some fresh name $\mathbf{x}$ called a *symbolic variable* [22]. Intuitively, this symbolic union reads as "if the Boolean formula $\mathbf{x}$ is true, then the symbolic union has value $\top$; if the formula $\neg\mathbf{x}$ is true, the union has value $\bot$." As symbolic unions are manipulated during a program, they may include an arbitrary number of guarded concrete values, where guards are disjoint Boolean formulae over *all* generated symbolic variables [22].

In concurrent work, we have adapted this operational semantics to probabilistic choice. The symbolic probabilistic choice operation $\widehat{\mathsf{flip}}\ p$ can be evaluated similarly to $\widehat{\mathsf{sym}}$, but our operational semantics additionally requires a *weight map* ϖ to track the probability that each symbolic variable is true:

$$(\widehat{\mathsf{flip}}\ p, \varpi) \Downarrow ([\mathbf{x} : \top, \neg\mathbf{x} : \bot], \varpi[\mathbf{x} \mapsto p])$$

In this work, we seek a denotational account of this symbolic execution strategy, that additionally generalizes to other affine effects. What distinguishes symbolic execution from the usual monadic semantics for an algebraic effect is that *every primitive operation is tagged with a unique symbolic variable*. It follows that every total Boolean assignment to the symbolic variables created at run time uniquely determines a concrete value of the program's result.

Compared to $\overline{\mathsf{amb}}$ whose effect is local, the behavior of $\widehat{\mathsf{sym}}$ therefore depends on *the global count of generated symbolic variables*. For example, if k symbolic variables have been created during a program, $\widehat{\mathsf{sym}}$ will generate a fresh symbolic variable $\mathbf{x}_{k+1}$ and return $[\mathbf{x}_{k+1} : \top, \neg\mathbf{x}_{k+1} : \bot]$, which is equivalently a function $\mathbb{B}^{k+1} \to \mathbb{B}$ depending only on the $(k+1)$-th input.

Hence, the semantics of an arbitrary symbolic executor must include global state that tags each symbolic operation executed during run time. Moreover, a symbolic union over a set X must behave as a map whose domain is the product of possible outcomes for all executed symbolic operations and whose codomain is X. This finally motivates our definition of *symbolic heaps*.

$$\omega_1 = \left(\{\mathbf{x}_1, \mathbf{x}_2\}, \begin{Bmatrix} \mathbf{x}_1 \mapsto (\mathsf{flip}\ 1/7, \mathbb{B}), \\ \mathbf{x}_2 \mapsto (\mathsf{flip}\ 1/4, \mathbb{B}) \end{Bmatrix}\right) \qquad \mathcal{L}(\mathbb{B}^2)(\omega_1) = \mathbf{Set}(\mathbb{B}^2, \mathbb{B}^2)$$

$$\omega_2 = (\{\mathbf{x}_1\}, \{\mathbf{x}_1 \mapsto (\mathsf{flip}\ 1/3, \mathbb{B})\}) \qquad \mathcal{L}(\mathbb{Z})(\omega_2) = \mathbf{Set}(\mathbb{B}, \mathbb{Z})$$

(a) Symbolic Heaps **(b)** Lifting

Figure 3. Examples of symbolic probability, following examples in Figure 2.

Categorical Symbolic Execution. To store executed symbolic operations, along with their unique tags, we introduce the concept of a symbolic heap. This sets up a framework for symbolically executing any affine algebraic theory.

Let $\mathbb{T} = (\Sigma, \mathcal{E})$ be an affine theory and denote the *category of symbolic heaps* as $\mathcal{W}$. Each object of $\mathcal{W}$ is an Σ-valued heap—that is, a pair (L, h) consisting of a finite set L of names (symbolic variables) and a function $h : L \to \Sigma$ mapping each name to an operation symbol and arity.

We then *lift* a set X to a set of symbolic unions $\mathcal{L}(X)$ over X as follows: for any heap (L, h), define $\mathcal{L}(X)(L, h)$ to be the set of functions $\prod_{\{\ell \mapsto (\mathsf{op}, A)\} \in h} A \to X$. Figure 3 illustrates examples of lifting for symbolic probability.

Lifting defines a functor $\mathcal{L} : \mathbf{Set} \to [\mathcal{W}, \mathbf{Set}]$, where $[\mathcal{W}, \mathbf{Set}]$ is a functor category inhabited by categorical symbolic unions. Interestingly, it can be shown that, for every set X, the lifting $\mathcal{L}(X)$ is an atomic sheaf.

The *symbolic execution monad* $\widehat{\mathcal{T}}$ on $[\mathcal{W}, \mathbf{Set}]$ models extensions of the heap with new tagged operations, much like in models of name generation [19]. Categorically, this monad is a coend that quantifies over symbolic heaps. As in algebraic effects, each operation symbol op in Σ with arity A yields a *generic symbolic operation* $\widehat{\mathsf{op}} : \mathcal{L}(1) \to \widehat{\mathcal{T}}(\mathcal{L}(A))$, which extends the heap with $\ell \mapsto (\mathsf{op}, A)$ for a fresh name ℓ.

Now, a symbolic executor is correct if it exactly captures the reachable branches of computation [22]. Our main theorem captures this statement of correctness:

Theorem 0.1. *For any affine algebraic theory $\mathbb{T}$, there exists a natural surjection $\chi : \widehat{\mathcal{T}}\mathcal{L} \to \mathcal{L}\mathcal{T}$ that commutes with generic operations and the strong monad operations.*

Conclusion. Our categorical semantics concisely justify using symbolic execution to perform probabilistic inference. Moreover, our correctness theorem suggests that symbolic execution strategies may exist for other affine effects, such as general semi-ring programming [1, 7].

However, this correctness theorem does not nearly capture the expressivity of Rosette. Most notably, our current theorem does not include mutable state or recursion. The primary focus of future work will be to enable more sophisticated language features by generalizing our results from **Set**

to other Cartesian-closed categories, including ω**Cpo** and **Nom**.

Another major focus will be the addition of both observation and failure. In particular, we will investigate generalizations in which the "maybe" monad transformer—and monad transformers in general—are applied to $\mathcal{T}$. Moreover, we will continue to study symbolic liftings as atomic sheaves.

References

[1] Kevin Batz, Adrian Gallus, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Tobias Winkler. Weighted programming: a programming paradigm for specifying mathematical models. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA1):1–30, 2022.

[2] Andrej Bauer. What is algebraic about algebraic effects and handlers?, 2019. URL https://arxiv.org/abs/1807.05923.

[3] Kwok-Ho Cheung. *Distributive interaction of algebraic effects*. PhD thesis, University of Oxford, 2017.

[4] Tobias Fritz. A synthetic approach to markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020.

[5] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. Scaling exact inference for discrete probabilistic programs. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–31, November 2020. ISSN 2475-1421. doi: 10.1145/3428208. URL http://dx.doi.org/10.1145/3428208.

[6] Younesse Kaddar and Sam Staton. A model of stochastic memoization and name generation in probabilistic programming: categorical semantics via monads on presheaf categories. *Electronic Notes in Theoretical Informatics and Computer Science*, 3, 2023.

[7] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. An algebraic prolog for reasoning about possible worlds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 209–214, 2011.

[8] John M. Li, Jon Aytac, Philip Johnson-Freyd, Amal Ahmed, and Steven Holtzen. A nominal approach to probabilistic separation logic. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706608. doi: 10.1145/3661814.3662135. URL https://doi.org/10.1145/3661814.3662135.

[9] Fred EJ Linton. Some aspects of equational categories. In *Proceedings of the Conference on Categorical Algebra: La Jolla 1965*, pages 84–94. Springer, 1966.

[10] John McCarthy. A basis for a mathematical theory of computation. In *Studies in Logic and the Foundations of Mathematics*, volume 26, pages 33–70. Elsevier, 1959.

[11] Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.

[12] Gordon Plotkin and John Power. Semantics for algebraic operations. *Electronic Notes in Theoretical Computer Science*, 45:332–345, 2001.

[13] Gordon Plotkin and John Power. Algebraic operations and generic effects. *Applied categorical structures*, 11:69–94, 2003.

[14] Sorawee Porncharoenwase, Luke Nelson, Xi Wang, and Emina Torlak. A formal foundation for symbolic evaluation with merging. *Proc. ACM Program. Lang.*, 6(POPL), January 2022. doi: 10.1145/3498709. URL https://doi.org/10.1145/3498709.

[15] Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. Sppl: probabilistic programming with fast exact symbolic inference. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI '21. ACM, June 2021. doi: 10.1145/3453483.3454078. URL http://dx.doi.org/10.1145/3453483.3454078.

[16] Marcin Sabok, Sam Staton, Dario Stein, and Michael Wolman. Probabilistic programming semantics for name generation. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–29, 2021.

[17] Alex Simpson. Probability sheaves and the giry monad. In *7th Conference on Algebra and Coalgebra in Computer Science (CALCO 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2017.

[18] Steffen Smolka, Praveen Kumar, David M Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. Scalable verification of probabilistic networks. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 190–203, 2019.

[19] Ian Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, December 1994. URL http://www.inf.ed.ac.uk/~stark/namhof.html. Also available as Technical Report 363, University of Cambridge Computer Laboratory.

[20] Sam Staton, Hongseok Yang, Frank Wood, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16. ACM, July 2016. doi: 10.1145/2933575.2935313. URL http://dx.doi.org/10.1145/2933575.2935313.

[21] Emina Torlak and Rastislav Bodik. Growing solver-aided languages with rosette. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pages 135–152, 2013.

[22] Emina Torlak and Rastislav Bodik. A lightweight symbolic virtual machine for solver-aided host languages. *ACM SIGPLAN Notices*, 49 (6):530–541, 2014.