Universal Semantics for the Stochastic λ-Calculus

Pedro H. Azevedo de Amorim*, Dexter Kozen*, Radu Mardare†, Prakash Panangaden‡, Michael Roberts*
*Cornell University
†University of Strathclyde
‡McGill University

2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) | 978-1-6654-4895-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/LICS52264.2021.9470747

***Abstract*—We define sound and adequate denotational and operational semantics for the stochastic lambda calculus. These two semantic approaches build on previous work that used an explicit source of randomness to reason about higher-order probabilistic programs.**

I. Introduction

Probabilistic programming has enjoyed a recent resurgence of interest, buoyed by the emergence of new languages and applications in the statistical analysis of large datasets and machine learning. Recent foundational research has focused on semantic models for higher-order functional languages.

One approach that is radically different from other approaches is that of [1], which involves Boolean-valued models for the stochastic λ-calculus. Based on an original idea of Scott [2], the paper [1] succeeded in incorporating random variables in a set-theoretic model of the untyped λ-calculus. The approach was formulated in terms of a nonstandard Boolean-valued interpretation of set theory based on the idea of Boolean-valued models of ZF set theory (see [3]). Boolean-valued models were first introduced by Scott [4] as an alternative technique to Cohen forcing for obtaining independence results in set theory. The independence of the Continuum Hypothesis was obtained by introducing an arbitrarily large set of real-valued random variables. The measure algebra of a standard Borel space Ω, a complete Boolean algebra, was used as a set of generalized truth values instead of the usual two-element Boolean algebra.

Scott also observed that these ideas could be given a probabilistic interpretation. The basic intuitions were briefly laid out in [2] and the formal development carried out in [1]. The primary goal was to develop an equational theory in which equations between stochastic λ-terms have probabilistic meaning and take values in a complete Boolean algebra. The intention was to provide reasoning principles for evaluating the equality of λ-terms under various program transformations.

The language contains a binary probabilistic choice operator $\oplus$, which captures the idea that a choice is to be made between two terms based on a random process. The source of randomness is called a *tossing process*, a random variable $T : \Omega \to 2^\omega$ giving a sequence of independent fair coin flips.

The semantics presented in this paper differ from those of [1] in several key ways. The semantics of [1] use static scoping for random coins. This causes β-reduction for unrestricted terms to be unsound, as a random coin may be used in more than one probabilistic decision. It is sound only under a certain restriction, namely that all probabilistic decisions in the argument be resolved before applying the function. This is a major impediment to the development of an operational semantics for which adequacy can be proved; indeed an operational semantics is not given in [1]. In contrast, we dynamically scope random coins, allowing them to be supplied at function call time. The nonstandard Boolean-valued foundations of the [1] semantics further complicate the development of an operational semantics, as they would call for a Boolean-valued operational semantics. In this work, we present a semantics with simpler domain-theoretic semantics with standard foundations.

A more operational approach was taken in [5]. That work presented an operational semantics for the stochastic λ-calculus as an idealized version of the Church language [6], along with reasoning principles and applications to the correctness of an implementation of trace Markov chain Monte Carlo processes. That work did not define a denotational semantics, which obliged them to reason combinatorially about programs.

In this paper we modify the approaches described above to conform to each other. We amend the stochastic denotational semantics of [1] to alter the scoping discipline of random sources in a way that still permits the Boolean-valued view of [1], yet allows the formulation of big- and small-step operational rules similar to [5] without the artificial restriction mentioned above. We prove soundness and adequacy of the operational semantics with respect to the reformulated stochastic semantics of [1], solving the main problem left open in that paper.

The organization of this paper and our main contributions are as follows.

Syntax: In §II we review the syntax of the stochastic λ-calculus as presented in [1], but with one change: We use capsules to represent recursive functions instead of an explicit fixpoint constructor. A *capsule* [7] is a pair $\langle M, \sigma \rangle$, where M is a stochastic λ-term and σ is an environment, such that

- $\mathsf{FV}(M) \subseteq \mathsf{dom}\,\sigma$, and
- $\forall x \in \mathsf{dom}\,\sigma \;\; \mathsf{FV}(\sigma(x)) \subseteq \mathsf{dom}\,\sigma$.

Capsules represent a finite coalgebraic representation of a closed regular λ-coterm (an infinite λ-term). This representation obviates the need for an explicit fixpoint constructor.

Tossing Processes: In §III we undertake a comprehensive exposition of *tossing processes*, or measure-preserving transformations of the Cantor space of infinite coin sequences.

978-1-6654-4895-6/21/$31.00 ©2021 IEEE

These processes arise in the study of behavioral invariance of programs, i.e. programs that behave the same way except for coin usage. We characterize the computable and continuous processes, both partial and total, and show their relationship to prefix codes. We also identify a general class of processes called *tree processes* that we later use in §VII to characterize the relationship between the coin usage patterns of our big- and small-step operational semantics.

Computability of tossing processes: Also in §III, we show how to embed the Cantor space 2^ω in a Scott domain in a natural way, thereby laying the groundwork for our modified denotational semantics. Consider the set $2^{\leq\omega}$ of finite and infinite binary strings ordered by the prefix relation. This is an algebraic DCPO whose compact elements are the finite strings. Define $x{\uparrow} = \{y \in 2^{\leq\omega} \mid x \preceq y\}$ for $x \in 2^{\leq\omega}$, where $\preceq$ is the prefix relation. The basic Scott-open sets are $x{\uparrow}$ for $x \in 2^*$. These are well known folklore results;[1] the domain is usually known as the domain of binary streams.

The infinite streams or sequences, with the subspace topology inherited from the Scott topology, is homeomorphic to Cantor space. Lemmas 7 and 8 establish a formal relationship between these two spaces and their continuous maps. This "Scottified" Cantor space gives an explicit characterization of functions that behave continuously with respect to coin usage in the sense that halting computations depend only on finite prefixes of the coin sequence. This allows us to discuss computable and continuous tossing processes. All the tree processes are Scott-continuous.

A Simplified Stochastic Semantics: In §IV, we review the stochastic denotational semantics of [1]. That semantics is based on a semantic map

$$(\!|-|\!) : \mathsf{Exp} \to \mathsf{Env} \to \mathsf{Cont} \to \mathsf{Toss} \to \mathsf{RV}$$

where

- RV is the set of *random variables* $\Omega \to \mathsf{Val}$ from a sample space Ω taking values in a reflexive CPO Val,
- Exp is the set of *stochastic λ-terms* M,
- Env is the set of *environments* $e : \mathsf{Var} \to \mathsf{RV}$,
- Cont is the set of *continuations* $c : \mathsf{RV} \to \mathsf{RV}$, and
- Toss is the set of *tossing processes* $T : \Omega \to 2^\omega$.

Thus $(\!|M|\!)ECT : \mathsf{RV}$.

We can simplify the exposition as follows:

- Suppose we restrict continuations to be of the form $SF = \lambda f.\lambda\omega.F\omega(f\omega)$ for some $F : \Omega \to [\mathsf{Val} \to \mathsf{Val}]$, where $[\mathsf{Val} \to \mathsf{Val}]$ denotes the Scott-continuous deterministic maps.[2] Then all continuations that arise in the inductive definition of $(\!|M|\!)$ are also of this form. Formally adopting this restriction allows us to eliminate continuations altogether.
- A tossing process determines how a supplied source of randomness is used in a computation. In [1] they are of type $\Omega \to 2^\omega$, where Ω is an abstract sample space. For our purposes, there is no reason not to assume that the sample space is 2^ω with Lebesgue measure, so a *tossing process* is now any measurable map $T : 2^\omega \to 2^\omega$ such that T^{-1} preserves measure. Examples are $\mathsf{tl}(\alpha) = \alpha_1\alpha_2\alpha_3\cdots$ and $\mathsf{evens}(\alpha) = \alpha_0\alpha_2\alpha_4\cdots$. This allows a more concrete treatment as developed in §III.
- We can omit the fixpoint operator of [1] using capsules [7], as described in §III.

In the treatment of [1], general β-reduction is unsound, precluding any standard operational semantics. This is because the source of randomness used by a function in the evaluation of its body is a coin sequence packaged with the function at the site of the function's definition. Thus randomness, like environments, is statically scoped. This can lead to the reuse of coins at different locations in the program, thereby breaking linearity. For example, in the evaluation of $(\lambda x.xa(xb))(\lambda y.cy \oplus dy)$, the same coin is used twice in the resolution of two $\oplus$'s when the body of the first expression is evaluated.

To achieve adequacy with respect to an operational semantics, we modify the denotational semantics of functions to allow the random source to be supplied as a parameter at the call site.

Deterministic Denotational Semantics: In §V, we observe that in the stochastic semantics, the value of $(\!|-|\!)$ depends not on the whole tossing process T nor the environment E, which are random variables parameterized by a sample point $\omega \in \Omega$, but only on their values. Intuitively, each run of the program corresponds to one trial, which is determined by a single sample point ω. This is the same observation used to eliminate continuations. This allows us to develop an intermediate *deterministic* denotational semantics in which probabilistic choices are resolved in advance, after which the program runs deterministically, making probabilistic decisions based on a presampled infinite stack of random numbers.

The deterministic denotational semantics is built on a reflexive domain of values constructed using the Scottified Cantor space of §III. In §VI, we prove the equivalence of the stochastic denotational semantics of [1] (as modified in §IV) and the deterministic semantics of §V (Theorem 16).

Operational Rules: In §VII, we give big-step and small-step structured operational semantics in the style of [8]. The big-step rules take the form $\langle M, e\rangle \Downarrow_\alpha \langle v, f\rangle$, which means that $\langle M, e\rangle$ reduces to normal form $\langle v, f\rangle$ with coins $\alpha \in 2^\omega$. The small-step rules take the form $\langle M, e\rangle \to_x \langle N, f\rangle$, which means that $\langle M, e\rangle$ reduces to $\langle N, f\rangle$ via a computation that consumes exactly a prefix $x \in 2^*$ of the infinite coin sequence.

In Theorem 17, we prove the equivalence of the big- and small-step rules, which use their random coins in a different pattern. The relationship is characterized by a tree process as described in §III.

Soundness and Adequacy: In §VIII, we prove the soundness and adequacy of our denotational semantics with respect to our big-step operational semantics (Theorem 18). Unlike most adequacy proofs that use logical relations, this proof is a relatively straightforward inductive argument, as the deterministic

[1] https://en.wikipedia.org/wiki/Scott_domain

[2] The operation S is the familiar S-combinator from combinatory logic.

denotational semantics and the big-step operational semantics use their coins in the same pattern.

II. Syntax

Let Var be a countable set of program variables $x, y, \ldots$. Let Exp denote the set of untyped λ-terms $M, N, K, \ldots$ with the usual abstraction and application operators plus an additional binary operator $\oplus$ for probabilistic choice. Let Λ denote the set of λ-abstractions, λ-terms of the form $\lambda x.\, M$.

A. Capsules

A *capsule* is a pair $\langle M, \sigma\rangle$, where $M \in \mathsf{Exp}$ and $\sigma : \mathsf{Var} \rightharpoonup \Lambda$ is a *capsule environment*, such that

(i) $\mathsf{FV}(M) \subseteq \mathsf{dom}\,\sigma$
(ii) $\forall x \in \mathsf{dom}\,\sigma\ \ \mathsf{FV}(\sigma(x)) \subseteq \mathsf{dom}\,\sigma$.

Here $\mathsf{dom}\,\sigma$ refers to the domain of σ and $\mathsf{FV}\,M$ refers to the set of free variables of M. A capsule is *reduced* if its first component is in Λ. Reduced capsules are denoted with lowercase letters, as $\langle v, \sigma\rangle$.

A capsule is a finite coalgebraic representation of a regular closed λ-coterm (infinitary λ-term), which is an element of the final coalgebra for the signature of the λ-calculus. Capsules give a convenient representation of recursive functions without the need of fixpoint combinators.

Capsules are considered equivalent modulo α-conversion, including α-conversion of the variables used in σ. In terms of nominal sets with the variables as atoms, the support of a capsule is $\varnothing$. Capsules are also considered equivalent modulo garbage collection in the sense that we can assume without loss of generality that $\mathsf{dom}\, e$ is a minimal set of variables satisfying (i) and (ii).

The capsule β-reduction rule is

$$\langle (\lambda x.\, M)\, v, \sigma\rangle \to \langle M[y/x], \sigma[v/y]\rangle \quad (y \text{ fresh})$$

applied in a call-by-value evaluation order. This mechanism captures static scoping without closures, heaps, or stacks [7]. Here we are using the notation $[-/-]$ for both substitution (as in $M[y/x]$) and rebinding (as in $\sigma[v/y]$).

Capsules were introduced in [7]. For the stochastic λ-calculus, we augment the system with the new syntactic construct $M \oplus N$ for probabilistic choice.

III. Tossing Processes

The *Cantor space* 2^ω is the space of infinite bitstreams. It is the topological power of ω copies of the two-element discrete space $2 = \{0, 1\}$. Elements of 2^ω are denoted $\alpha, \beta, \ldots$. The topology is generated by basic open sets $I_x = \{\alpha \in 2^\omega \mid x \prec \alpha\}$, where $x \in 2^*$ and $\prec$ denotes the strict prefix relation. The sets I_x are called *intervals*. The topology is also generated by the standard metric $d(\alpha, \beta) = 2^{-n}$, where n is the length of the longest common prefix of α and β, or 0 if $\alpha = \beta$.

The Borel sets $\mathcal{B}$ of the Cantor space are the smallest σ-algebra containing the open sets. The uniform (Lebesgue) measure $\Pr$ on $(2^\omega, \mathcal{B})$ is generated by its values on intervals: $\Pr(\{\alpha \mid x \prec \alpha\}) = 2^{-|x|}$. The Lebesgue measurable sets are the smallest σ-algebra containing the Borel sets and all subsets of null sets. The set of null sets is denoted $\mathcal{N}$.

A *tossing process* is any measurable map $T : 2^\omega \to 2^\omega$ such that T^{-1} preserves measure; that is, for all $A \in \mathcal{B}$, $\Pr(T^{-1}(A)) = \Pr(A)$. Given an infinite bitstream $\alpha = \alpha_0\alpha_1\alpha_2\cdots$, we can define the examples $\mathsf{tl}(\alpha) = \alpha_1\alpha_2\alpha_3\cdots$ and $\mathsf{evens}(\alpha) = \alpha_0\alpha_2\alpha_4\cdots$. A tossing process determines how a supplied source of randomness is used in a computation.

Lemma 1. *T is a tossing process iff for all $x \in 2^*$,*

$$\Pr(\{\alpha \mid x \prec T(\alpha)\}) = 2^{-|x|}$$

Proof. We have $x \prec T(\alpha)$ iff $\alpha \in T^{-1}(\{\gamma \mid x \prec \gamma\})$, therefore

$$\begin{aligned} &T \text{ is a tossing process} \\ &\Leftrightarrow \forall x \in 2^*\ \ \Pr(T^{-1}(\{\gamma \mid x \prec \gamma\})) = \Pr(\{\gamma \mid x \prec \gamma\}) \\ &\Leftrightarrow \forall x \in 2^*\ \ \Pr(\{\alpha \mid x \prec T(\alpha)\}) = 2^{-|x|}. \end{aligned}$$

□

A. Computable and Continuous Processes

For a function $f : 2^\omega \to 2^\omega$ to be computable, it must be possible to emit each digit of the output stream after reading only finitely many digits of the input stream. For example, one can emit the nth digit of $\mathsf{tl}\,\alpha$ after reading $n+1$ digits of α, and one can emit the nth digit of $\mathsf{evens}\,\alpha$ after reading the first $2n-1$ digits of α.

Lemma 2. *All computable tossing processes $T : 2^\omega \to 2^\omega$ are continuous. All continuous functions $2^\omega \to 2^\omega$ are uniformly continuous with respect to the standard metric.*

Proof. To be computable, it must be the case that any finite prefix $x \prec T\alpha$ of the output is determined by some finite prefix of the input α. This implies that $x \prec T\beta$ for any β that agrees with α on a sufficiently long prefix; in other words, $\{\beta \mid y \prec \beta\} \subseteq T^{-1}(\{\gamma \mid x \prec \gamma\})$ for some $y \prec \alpha$. Thus $T^{-1}(\{\gamma \mid x \prec \gamma\})$ is open. As x was arbitrary, T is continuous.

It is a standard result that any continuous function on a compact metric space is uniformly continuous. □

For example, tl is Lipschitz with constant 2: $d(\mathsf{tl}\,\alpha, \mathsf{tl}\,\beta) \le 2d(\alpha, \beta)$. The maps evens and odds are not Lipschitz, but they are Hölder of order $1/2$; that is, both maps satisfy $d(T\alpha, T\beta) \le \sqrt{d(\alpha, \beta)}$.

There is a subtle distinction between "reading" and "consuming" a digit. The latter refers to using the digit to make a probabilistic choice. One can read digits without consuming them; they can be saved to make probabilistic choices later, at which point they are consumed. It is important for independence that digits not be consumed more than once.

Uniform continuity fails if we allow tossing processes to be partial. A *partial tossing process* is a measure-preserving partial measurable function $T : 2^\omega \rightharpoonup 2^\omega$. Such a function is necessarily almost everywhere defined, since $\mathsf{dom}\, T = T^{-1}(2^\omega)$, which must have measure 1.

Lemma 3. *All computable partial tossing processes $T : 2^\omega \to 2^\omega$ are continuous. There is a computable partial tossing process that is continuous but not uniformly continuous.*

Proof. Computable partial tossing processes $T : 2^\omega \to 2^\omega$ are continuous for the same reason that total ones are.

For the second statement, define T coinductively as follows:

$$T(0^*10\alpha) = 0\,T(\alpha) \qquad T(0^*11\alpha) = 1\,T(\alpha).$$

The domain of definition of T is $(0^*1)^\omega$, the measure-1 set of streams containing infinitely many 1's. It is continuous, since if α and β share a prefix with at least $2n$ 1's, then $T\alpha$ and $T\beta$ share a prefix of length n. It is not uniformly continuous, as there is no bound on the number of input digits that need to be read before producing the next output digit. □

The T of the previous lemma is undefined on the nullset 2^*0^ω. One can define T arbitrarily on this set, but Lemma 2 says that the resulting total tossing process cannot be continuous. This does not rule out the possibility that every total tossing process might be equivalent modulo $\mathcal{N}$ to some continuous partial tossing process. However, this too is false.

Lemma 4. *There is a tossing process that is not equivalent modulo $\mathcal{N}$ to any continuous partial tossing process.*

Proof. The proof uses [9, Exercises 7 and 8, p. 59]. A complete proof can be found in the Appendix. □

Lemma 5. *All continuous tossing processes, partial or total, are surjective.*

Proof. For any $\beta \in 2^\omega$, $T^{-1}(\{\beta\}) = \bigcap_n T^{-1}(\{\alpha \mid \alpha_n = \beta_n\})$. This is the intersection of a collection of closed sets with the finite intersection property in a compact space, therefore it is nonempty. □

Every tossing process T is equivalent modulo $\mathcal{N}$ to a partial T' that is "almost continuous" in the sense that all ${T'}^{-1}(\{\alpha \mid x \prec \alpha\})$ are Δ^0_2, that is, both G_δ and F_σ. One can obtain T' from T by deleting countably many nullsets $G_x \setminus F_x$, where G_x and F_x are G_δ and F_σ, respectively, such that $\Pr(F_x) = \Pr(G_x)$ and $F_x \subseteq T^{-1}(\{\alpha \mid x \prec \alpha\}) \subseteq G_x$. The sets F_x and G_x exist by Lebesgue measurability.

The following theorem gives a characterization of the continuous partial and total tossing processes $T : 2^\omega \to 2^\omega$. A *binary prefix code* is a nonempty set of prefix-incomparable finite-length binary strings. A binary prefix code P is *exhaustive* if all $\alpha \in 2^\omega$ have a prefix in P. An exhaustive prefix code is necessarily finite by compactness.

If P and Q are two binary prefix codes, write $P \preceq Q$ if every element of Q is an extension of some element of P; that is, for every $y \in Q$, there exists $x \in P$ such that $x \preceq y$.

A *coding function* is a map $x \mapsto P_x$, where P_x is a prefix code, such that

- $P_\varepsilon = \{\varepsilon\}$;
- if x and y are prefix-incomparable, then $P_x \cap P_y = \varnothing$;
- if $x \preceq y$, then $P_x \preceq P_y$.

In addition, $x \mapsto P_x$ is said to be *exhaustive* provided

- if P is an exhaustive prefix code, then so is $\bigcup_{x \in P} P_x$.

Theorem 6. *For every continuous partial tossing process T, there is a unique coding function $x \mapsto P_x$ such that*

(i) *$x \prec T\alpha$ iff $y \prec \alpha$ for some $y \in P_x$; in other words,*

$$T^{-1}(I_x) = \{\alpha \mid x \prec T\alpha\} = \bigcup_{y \in P_x} I_y;$$

(ii) *P_x is $\preceq$-minimal among prefix codes satisfying* (i)*;*

(iii) $\Pr(\bigcup_{y \in P_x} I_y) = 2^{-|x|}$.

If T is total, then $x \mapsto P_x$ is exhaustive. Moreover, every coding function of this form gives rise to a continuous partial or total tossing process.

Proof. A proof can be found in the Appendix. □

B. Tree Processes

Let $t : 2^* \to \omega$ be a labeled tree with no repetition of labels along any path; that is, if $x, y \in 2^*$ with $x \prec y$, then $t(x) \neq t(y)$. Each such tree gives rise to a continuous tossing process $T : 2^\omega \to 2^\omega$ as follows. Given $\alpha = \alpha_0\alpha_1\alpha_2\cdots \in 2^\omega$, let $T(\alpha) = \beta_0\beta_1\beta_2\cdots$, where $\beta_n = \alpha_{t(\beta_0\beta_1\cdots\beta_{n-1})}$. Thus the bit of the input sequence α that is tested in the nth step can depend on the outcomes of previous tests as determined by t. The restriction "no repetition of labels along any path" ensures that no coin is used more than once.

Every such T is measurable and measure-preserving, thus a tossing process:

$$T^{-1}(\{\gamma \mid \beta_0\cdots\beta_{n-1} \prec \gamma\}) = \{\alpha \mid \beta_0\cdots\beta_{n-1} \prec T(\alpha)\}$$
$$= \{\alpha \mid \bigwedge_{i=0}^{n-1} T(\alpha)_i = \beta_i\} = \bigcap_{i=0}^{n-1} \{\alpha \mid \alpha_{t(\beta_0\cdots\beta_{i-1})} = \beta_i\}$$

$$\Pr(T^{-1}(\{\gamma \mid \beta_0\cdots\beta_{n-1} \prec \gamma\}))$$
$$= \Pr(\bigcap_{i=0}^{n-1} \{\alpha \mid \alpha_{t(\beta_0\cdots\beta_{i-1})} = \beta_i\})$$
$$= \prod_{i=0}^{n-1} \Pr(\{\alpha \mid \alpha_{t(\beta_0\cdots\beta_{i-1})} = \beta_i\}) = 2^{-n}.$$

Such processes are called *tree processes*.

Tree processes are uniformly continuous in the standard metric: $T(\alpha)$ and $T(\beta)$ agree on their length-n prefixes provided α and β agree on their length-m prefixes, where m is the supremum of the labels on all nodes of depth n or less in the tree t.

C. Scottifying the Cantor Space

We can embed the Cantor space 2^ω in a Scott domain in a natural way. Consider the set $2^{\leq\omega}$ of finite and infinite binary strings ordered by the prefix relation. This is an algebraic CPO whose compact elements are the finite strings. Define $x{\uparrow} = \{y \in 2^{\leq\omega} \mid x \prec y\}$ for $x \in 2^{\leq\omega}$. The basic Scott-open sets are $x{\uparrow}$ for $x \in 2^*$.

Lemma 7.

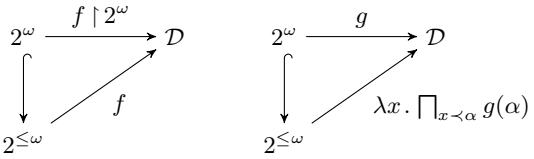
(i) *If B is a Scott-open set of $2^{\leq\omega}$, then $B \cap 2^\omega$ is a Cantor-open set of 2^ω.*
(ii) *If A is a Cantor-open set of 2^ω, then $\{x \in 2^{\leq\omega} \mid x\uparrow \subseteq A\}$ is a Scott-open set of $2^{\leq\omega}$, and is largest Scott-open set B such that $B \cap 2^\omega = A$.*

Thus the Cantor space 2^ω is a subspace of the Scott space $2^{\leq\omega}$. The "Scottified" Cantor space gives an explicit characterization of functions that behave continuously with respect to coin usage in the sense that computations depend only on finite prefixes of the coin sequence.

Let $\mathcal{D}$ be a continuous ω-CPO ordered by $\sqsubseteq$ with a meet operation $\bigsqcap$. Let $\prec$ be the proper prefix relation on strings.

Lemma 8.

(i) *If $f : 2^{\leq\omega} \to \mathcal{D}$ is Scott-continuous, then $f \restriction 2^\omega : 2^\omega \to \mathcal{D}$ is Cantor-continuous.*
(ii) *If $g : 2^\omega \to \mathcal{D}$ is Cantor-continuous, then g extends to a Scott-continuous map $\lambda x . \bigsqcap_{x \prec \alpha} g(\alpha) : 2^{\leq\omega} \to \mathcal{D}$.*



Proof. A proof can be found in the Appendix. □

IV. Stochastic Semantics

We review briefly the stochastic semantics from [1]. This semantics was based on a map

$$(\!|-|\!) : \mathsf{Exp} \to \mathsf{Env} \to \mathsf{Cont} \to \mathsf{Toss} \to \mathsf{RV}$$

where

- RV is the set of *random variables* $\Omega \to \mathsf{Val}$ from a sample space Ω taking values in a reflexive CPO Val,
- Exp is the set of *stochastic λ-terms* M,
- Env is the set of *environments* $E : \mathsf{Var} \to \mathsf{RV}$,
- Cont is the set of *continuations* $C : \mathsf{RV} \to \mathsf{RV}$, and
- Toss is the set of *tossing processes* $T : \Omega \to 2^\omega$.

Thus $(\!|M|\!)ECT : \mathsf{RV}$. The Boolean-valued semantics interpreted properties in the Boolean algebra of measurable sets of Ω.

We can simplify the definition of [1] with a few observations.

(i) In [1], the map $(\!|-|\!)$ is parameterized by continuations $C : (\Omega \to \mathsf{Val}) \to (\Omega \to \mathsf{Val})$. Suppose we restrict continuations to be of the form $SF = \lambda f . \lambda\omega . F\omega(f\omega)$ for some $F : \Omega \to [\mathsf{Val} \to \mathsf{Val}]$, where $[\mathsf{Val} \to \mathsf{Val}]$ denotes the Scott-continuous deterministic maps.[3] Then all continuations that arise in the inductive definition of $(\!|-|\!)$ are also of this form. Formally adopting this restriction allows us to eliminate continuations altogether, thereby simplifying the presentation. This also makes sense at an intuitive level: A single trial is a single evaluation of the program and depends only on one sample from Ω.
(ii) Tossing processes in [1] are of type $\Omega \to 2^\omega$, where Ω is an abstract sample space. A large part of the development of [1] was concerned with invariance properties of measure-preserving transformations of Ω. For our purposes, there is no reason not to take the sample space to be 2^ω with the standard Lebesgue measure. Thus tossing processes become measure-preserving maps $T : 2^\omega \to 2^\omega$. This allows a more concrete treatment. A comprehensive characterization of such processes is given in §III.
(iii) The definition of [1] included a fixpoint operator. Our use of capsules allows us to eliminate this operator without loss of expressiveness.

In addition to these simplifications, we introduce a more radical change that will admit a full-fledged operational semantics, namely the dynamic scoping of the random source.

The type of the semantic map is now

$$(\!|-|\!) : \mathsf{Exp} \to \mathsf{Env} \to \mathsf{Toss} \to \mathsf{RV}.$$

The values RV do not form a reflexive CPO, however they are built out of a reflexive CPO, as explained below in §VI-A.

$$\mathsf{Fun} : \mathsf{RV} \to [\mathsf{Toss} \to \mathsf{RV} \to \mathsf{RV}]$$
$$\mathsf{Lam} : [\mathsf{Toss} \to \mathsf{RV} \to \mathsf{RV}] \to \mathsf{RV}$$

We will define these functions explicitly below in §VI-A.

Definition 9.

(i) $(\!|x|\!)ET = E(x)$
(ii) $(\!|MN|\!)ET = \mathsf{Fun}((\!|M|\!)E(\pi_0^3 \circ T))(\pi_1^3 \circ T)((\!|N|\!)E(\pi_2^3 \circ T))$
(iii) $(\!|\lambda x . M|\!)ET = \mathsf{Lam}(\lambda Tv . (\!|M|\!)E[v/x]T)$
(iv) $(\!|M \oplus N|\!)ET = \lambda\omega . \mathsf{hd}(T\omega) \,?\, (\!|M|\!)E(\mathsf{tl} \circ T)\omega : (\!|N|\!)E(\mathsf{tl} \circ T)\omega$

where clause (ii) uses the notation $\pi_i^3(\alpha)$ (α_i when evident from the context) to refer to the subsequence of α consisting of bits whose indices are $i \bmod 3$; thus $\pi_1^3(\alpha_0\alpha_1\alpha_2\cdots) = \alpha_1\alpha_4\alpha_7\cdots$, and clause (iv) uses the ternary predicate

$$b \,?\, s : t = \begin{cases} s, & \text{if } b = 1, \\ t, & \text{if } b = 0. \end{cases} \tag{1}$$

In $(\!|M|\!)E$, we assume that $\mathsf{FV}(M) \subseteq \mathsf{dom}\, E$.

V. Deterministic Semantics

The observation of §IV that allowed continuations to be eliminated can be carried further. All components in the definition of $(\!|-|\!)$ are parameterized by sample points $\omega \in \Omega$, but as observed, there is no resampling in the course of a single trial; it is the same ω. The function $(\!|-|\!)$ does not really depend on the whole tossing process T or the whole environment E, which are random variables, but only on their values. This observation allows us to develop an intermediate *deterministic* denotational semantics in which all probabilistic choices are

[3] The operation S is the familiar S-combinator from combinatory logic.

resolved in advance. The program runs deterministically, resolving probabilistic choices by consulting a preselected stack of random bits. In this section we introduce this semantics and develop some of its basic properties. Later, in §VI, we will prove that it is equivalent to the stochastic semantics of [1] as modified in §IV (Theorem 16).

A. A Domain of Values

Barendregt [10, §5] presents several constructions of reflexive CPOs that can serve as denotational models of the untyped λ-calculus. One concrete such model, due to Engeler [11], [12], is a reflexive ω-algebraic CPO $\mathcal{P}(Q)$ ordered by inclusion, where Q is a certain countable set. The basic Scott-open sets are $a{\uparrow} = \{b \mid a \subseteq b\}$, where a is a finite subset of Q. A function $\mathcal{P}(Q) \to \mathcal{P}(Q)$ is *continuous* if it is continuous in this topology; equivalently, if $fb = \bigcup_{c \in \mathcal{P}_{\text{fin}}(b)} fc$.

In this section we present a version of the Engeler model modified to include a random source as an argument to continuous functions using the Scottified Cantor space of §III-C. Define

$$Q_0 = \{\varnothing\} \quad Q_{n+1} = Q_n \uplus (2^* \times \mathcal{P}_{\text{fin}}(Q_n) \times Q_n) \quad Q = \bigcup_n Q_n$$

and let $\mathsf{Val} = \mathcal{P}(Q)$, ordered by inclusion. The basic Scott-open sets of Val are $a{\uparrow} = \{b \mid a \subseteq b\}$, where $a \in \mathcal{P}_{\text{fin}}(Q)$. A function $\mathsf{Val} \to \mathsf{Val}$ is *continuous* if it is continuous in this topology.

A function $f : 2^\omega \to \mathsf{Val} \to \mathsf{Val}$ is *continuous* if it is continuous in both variables with respect to the Scott topology on Val and the Cantor topology on 2^ω. The continuous functions of this type are denoted $[2^\omega \to \mathsf{Val} \to \mathsf{Val}]$. Intuitively, f is continuous if its value on $\alpha \in 2^\omega$ and $b \in \mathsf{Val}$ depends only on finite prefixes of α and finite subsets of b.

Lemma 10. *Let $f : [2^\omega \to \mathsf{Val} \to \mathsf{Val}]$. Then*

$$f\alpha b = \bigcup_{c \in \mathcal{P}_{\text{fin}}(b)} \bigcup_{x \prec \alpha} \bigcap_{\beta \in I_x} f\beta c.$$

Proof. Let $c \in \mathsf{Val}$. By the continuity of f in its first argument, $\lambda\alpha\,.\,f\alpha c : 2^\omega \to \mathsf{Val}$ is continuous, therefore for any basic open set $a{\uparrow}$,

$$\alpha \in (\lambda\alpha\,.\,f\alpha c)^{-1}(a{\uparrow}) \Leftrightarrow \exists x \prec \alpha\ I_x \subseteq (\lambda\alpha\,.\,f\alpha c)^{-1}(a{\uparrow}).$$

Then

$$\begin{aligned} a \subseteq f\alpha c &\Leftrightarrow f\alpha c \in a{\uparrow} \\ &\Leftrightarrow \alpha \in (\lambda\alpha\,.\,f\alpha c)^{-1}(a{\uparrow}) \\ &\Leftrightarrow \exists x \prec \alpha\ I_x \subseteq (\lambda\alpha\,.\,f\alpha c)^{-1}(a{\uparrow}) \\ &\Leftrightarrow \exists x \prec \alpha\ \forall \beta \in I_x\ \beta \in (\lambda\alpha\,.\,f\alpha c)^{-1}(a{\uparrow}) \\ &\Leftrightarrow \exists x \prec \alpha\ \forall \beta \in I_x\ a \subseteq f\beta c \\ &\Leftrightarrow a \subseteq \bigcup_{x \prec \alpha} \bigcap_{\beta \in I_x} f\beta c. \end{aligned}$$

As a was arbitrary, for any $b \in \mathsf{Val}$,

$$f\alpha b = \bigcup_{c \in \mathcal{P}_{\text{fin}}(b)} f\alpha c = \bigcup_{c \in \mathcal{P}_{\text{fin}}(b)} \bigcup_{x \prec \alpha} \bigcap_{\beta \in I_x} f\beta c.$$

□

To obtain a reflexive domain, we need to construct continuous maps

$$\begin{aligned} \mathsf{fun} &: \mathsf{Val} \to [2^\omega \to \mathsf{Val} \to \mathsf{Val}] \\ \mathsf{lam} &: [2^\omega \to \mathsf{Val} \to \mathsf{Val}] \to \mathsf{Val} \end{aligned}$$

such that $\mathsf{fun} \circ \mathsf{lam} = \mathsf{id}$.

$$\begin{aligned} \mathsf{fun}\, a &= \lambda\beta v\,.\,\{q \in Q \mid \exists x \prec \alpha\ \exists b \in \mathcal{P}_{\text{fin}}(v)\ (x, b, q) \in a\} \\ &= \bigcup_{x \prec \alpha} \bigcup_{c \in \mathcal{P}_{\text{fin}}(b)} \{q \mid (x, c, q) \in a\} \qquad (2) \\ \mathsf{lam}\, f &= \{(x, c, q) \in 2^* \times \mathcal{P}_{\text{fin}}(Q) \times Q \mid \forall \beta \in I_x\ q \in f\beta c\} \\ &\quad \cup \{\varnothing\}. \qquad (3) \end{aligned}$$

Then

$$\begin{aligned} \mathsf{fun}(\mathsf{lam}\, f)\alpha b &= \bigcup_{x \prec \alpha} \bigcup_{c \in \mathcal{P}_{\text{fin}}(b)} \{q \mid (x, c, q) \in \mathsf{lam}\, f\} \\ &= \bigcup_{x \prec \alpha} \bigcup_{c \in \mathcal{P}_{\text{fin}}(b)} \{q \mid \forall \beta \in I_x\ q \in f\beta c\} \\ &= \bigcup_{c \in \mathcal{P}_{\text{fin}}(b)} \bigcup_{x \prec \alpha} \bigcap_{\beta \in I_x} f\beta c = f\alpha b. \end{aligned}$$

Also, note that since $\bot = \varnothing$ in Val, $\mathsf{fun}\ \bot = \lambda\beta v\,.\,\bot$, but $\mathsf{lam}\ \bot = \{\varnothing\} \neq \bot$. This is important for call-by-value, as we must distinguish Ω from $\lambda x\,.\,\Omega$ for our adequacy result of §VIII.

B. The Semantic Function

For partial functions $f : D \rightharpoonup E$, define $\mathsf{dom}\, f = \{x \in D \mid f(x) \text{ is defined}\}$. Equivalently, for functions $f : D \to E_\bot$, define $\mathsf{dom}\, f = \{x \in D \mid f(x) \neq \bot\}$. Let $\mathsf{FV}(M)$ denote the free variables of M.

The type of our deterministic semantic function is

$$(\!|-|\!) : \mathsf{Exp} \to \mathsf{Env}' \to 2^\omega \to \mathsf{Val},$$

where $\mathsf{Env}' = \mathsf{Var} \to \mathsf{Val}$ is the set of (deterministic) environments.

Definition 11.

(i) $(\!|x|\!)e\alpha = e(x)$
(ii) $(\!|MN|\!)e\alpha = \mathsf{fun}((\!|M|\!)e(\pi_0^3(\alpha)))(\pi_1^3(\alpha))((\!|N|\!)e(\pi_2^3(\alpha)))$
(iii) $(\!|\lambda x\,.\,M|\!)e\alpha = \mathsf{lam}(\lambda\beta v\,.\,(\!|M|\!)e[v/x]\beta)$
(iv) $(\!|M \oplus N|\!)e\alpha = \mathsf{hd}\,\alpha\,?\,(\!|M|\!)e(\mathsf{tl}\,\alpha) : (\!|N|\!)e(\mathsf{tl}\,\alpha)$

where clause (iv) uses the ternary predicate (1) *and* fun *and* lam *are defined in* (2) *and* (3)*, respectively. In $(\!|M|\!)e$, we assume that $\mathsf{FV}(M) \subseteq \mathsf{dom}\, e$. In (iii), we interpret the metaexpression $\lambda\beta v\,.\,(\!|M|\!)e[v/x]\beta$ as strict; thus*

$$(\lambda\beta v\,.\,(\!|M|\!)e[v/x]\beta)\,\alpha\,\bot = \bot.$$

Note that this is completely deterministic. Probabilistic choices are resolved by consulting a preselected stack of random bits α.

In the clause for MN, instead of evens and odds as in [1], we divide the coins into three streams for use in, respectively,

the evaluation of M, the evaluation of N, and the application of the value of M to the value of N.

This definition is well founded, but the resulting metaexpression is a λ-term that must be evaluated in the metasystem, and that evaluation may not terminate. We define the value to be $\bot$ when that happens. For example, consider $(\!(\Omega)\!)e\alpha$, where $\Omega = (\lambda x\,.\,xx)(\lambda x\,.\,xx)$. Define

$$\begin{aligned} u &\triangleq (\!(\lambda x\,.\,xx)\!)e\alpha \\ &= \mathsf{lam}\,(\lambda\beta v\,.\,(\!(xx)\!)e[v/x]\beta) \\ &= \mathsf{lam}\,(\lambda\beta v\,.\,\mathsf{fun}\,((\!(x)\!)e[v/x]\beta_0)\,\beta_1\,((\!(x)\!)e[v/x]\beta_2)) \\ &= \mathsf{lam}\,(\lambda\beta v\,.\,\mathsf{fun}\;v\,\beta_1\,v) \neq \bot. \end{aligned}$$

Note that this value is independent of α, due to the fact that coins are dynamically scoped. Then

$$\begin{aligned} (\!(\Omega)\!)e\alpha &= (\!((\lambda x\,.\,xx)(\lambda x\,.\,xx))\!)e\alpha \\ &= \mathsf{fun}((\!(\lambda x\,.\,xx)\!)e\alpha_0)\,\alpha_1\,((\!(\lambda x\,.\,xx)\!)e\alpha_2) \\ &= \mathsf{fun}\,u\,\alpha_1\,u \\ &= \mathsf{fun}(\mathsf{lam}(\lambda\beta v\,.\,\mathsf{fun}\;v\,\beta_1\,v))\,\alpha_1\,u \\ &= (\lambda\beta v\,.\,\mathsf{fun}\;v\,\beta_1\,v)\,\alpha_1\,u \\ &= \mathsf{fun}\;u\,\alpha_{11}\,u \\ &= \cdots \end{aligned}$$

Lemma 12. *For $e_1, e_2 : \mathsf{Var} \to \mathsf{Val}$, if $e_1 \sqsubseteq e_2$ and $\mathsf{FV}(M) \subseteq \mathsf{dom}\,e_1$, then $(\!(M)\!)e_1 \sqsubseteq (\!(M)\!)e_2$.*

Proof. Note that if $e_1 \sqsubseteq e_2$, then $\mathsf{dom}\,e_1 \subseteq \mathsf{dom}\,e_2$ and $e_1(x) \sqsubseteq e_2(x)$ for all $x \in \mathsf{dom}\,e_1$. The proof is a straightforward induction on the structure of M. Suppose $e_1 \sqsubseteq e_2$.

$$(\!(x)\!)e_1\alpha = e_1(x) \sqsubseteq e_2(x) = (\!(x)\!)e_2\alpha,\;\; x \in \mathsf{dom}\,e_1.$$

$$\begin{aligned} (\!(MN)\!)e_1\alpha &= \mathsf{fun}((\!(M)\!)e_1(\pi_0^3(\alpha)))(\pi_1^3(\alpha))((\!(N)\!)e_1(\pi_2^3(\alpha))) \\ &\sqsubseteq \mathsf{fun}((\!(M)\!)e_2(\pi_0^3(\alpha)))(\pi_1^3(\alpha))((\!(N)\!)e_2(\pi_2^3(\alpha))) \\ &= (\!(MN)\!)e_2\alpha. \end{aligned}$$

$$\begin{aligned} (\!(\lambda x\,.\,M)\!)e_1\alpha &= \mathsf{lam}(\lambda\beta v\,.\,(\!(M)\!)e_1[v/x]\beta) \\ &\sqsubseteq \mathsf{lam}(\lambda\beta v\,.\,(\!(M)\!)e_2[v/x]\beta) \\ &= (\!(\lambda x\,.\,M)\!)e_2\alpha. \end{aligned}$$

$$\begin{aligned} (\!(M \oplus N)\!)e_1\alpha &= \mathsf{hd}\,\alpha\,?\,(\!(M)\!)e_1(\mathsf{tl}\,\alpha) : (\!(N)\!)e_1(\mathsf{tl}\,\alpha) \\ &\sqsubseteq \mathsf{hd}\,\alpha\,?\,(\!(M)\!)e_2(\mathsf{tl}\,\alpha) : (\!(N)\!)e_2(\mathsf{tl}\,\alpha) \\ &= (\!(M \oplus N)\!)e_2\alpha. \end{aligned}$$

□

To extend $(\!(-)\!)$ to capsules, we combine a semantic environment $\mathsf{Var} \to \mathsf{Val}$ as used in Definition 11 and a capsule environment $\mathsf{Var} \to \Lambda_\bot$ in a single mixed environment $\sigma : \mathsf{Var} \to \mathsf{Val} + \Lambda_\bot$[4]. From this we can obtain a new semantic environment $\sigma^* : \mathsf{Var} \to \mathsf{Val}$ as follows. Consider the map

$$\begin{aligned} &P_\sigma : (\mathsf{Var} \to \mathsf{Val}) \to (\mathsf{Var} \to \mathsf{Val}) \\ &P_\sigma(\ell)(x) = \begin{cases} \sigma(x), & \sigma(x) \in \mathsf{Val} \\ (\!(\sigma(x))\!)\ell\alpha, & \sigma(x) \in \Lambda \\ \bot, & \sigma(x) = \bot \end{cases} \end{aligned}$$

where in the second case we use Definition 11(iii). The α there can be any element of 2^ω, as $(\!(\lambda x\,.\,M)\!)\ell : 2^\omega \to \mathsf{Val}$ is a constant function. The third case is already included in the first, so henceforth we omit explicit mention of it. Note that $\mathsf{dom}\,P_\sigma(\ell) = \mathsf{dom}\,\sigma$.

Lemma 13. *$P_\sigma(\ell)$ is monotone in both σ and ℓ.*

Proof. Since $\Lambda_\bot$ is a flat domain, if $\sigma_1 \sqsubseteq \sigma_2$ and $\sigma_1(x) \in \Lambda$, then $\sigma_1(x) = \sigma_2(x)$. It follows that for all $\ell : \mathsf{Var} \to \mathsf{Val}$ and $x \in \mathsf{Var}$,

$$\begin{aligned} P_{\sigma_1}(\ell)(x) &= \begin{cases} \sigma_1(x), & \sigma_1(x) \in \mathsf{Val} \\ (\!(\sigma_1(x))\!)\ell\alpha, & \sigma_1(x) \in \Lambda \end{cases} \\ &\sqsubseteq \begin{cases} \sigma_2(x), & \sigma_2(x) \in \mathsf{Val} \\ (\!(\sigma_1(x))\!)\ell\alpha, & \sigma_2(x) \in \Lambda \end{cases} \\ &= \begin{cases} \sigma_2(x), & \sigma_2(x) \in \mathsf{Val} \\ (\!(\sigma_2(x))\!)\ell\alpha, & \sigma_2(x) \in \Lambda \end{cases} \\ &= P_{\sigma_2}(\ell)(x). \end{aligned} \tag{4}$$

If $\ell_1 \sqsubseteq \ell_2$, then by Lemma 12, $(\!(\sigma(x))\!)\ell_1\alpha \sqsubseteq (\!(\sigma(x))\!)\ell_2\alpha$ for $\sigma(x) \in \Lambda$, therefore $P_\sigma(\ell_1)(x) \sqsubseteq P_\sigma(\ell_2)(x)$. □

By the Knaster-Tarski theorem, P_σ has a least fixpoint

$$\sigma^*(x) = \begin{cases} \sigma(x), & \sigma(x) \in \mathsf{Val} \\ (\!(\sigma(x))\!)\sigma^*\alpha, & \sigma(x) \in \Lambda \end{cases} \tag{5}$$

and we define $(\!(M)\!)\sigma\alpha = (\!(M)\!)\sigma^*\alpha$, where the right-hand side is by Definition 11. With this formalism, we have

$$(\!(\mathsf{rec}\,f\,.\,\lambda x\,.\,M)\!)\sigma = (\!(f)\!)\sigma[\lambda x\,.\,M/f].$$

Lemma 14. *If $\sigma_1, \sigma_2 : \mathsf{Var} \to \mathsf{Val} + \Lambda_\bot$ are mixed environments with $\sigma_1 \sqsubseteq \sigma_2$, then $\sigma_1^* \sqsubseteq \sigma_2^*$. In addition, if $\sigma_1(x) = \sigma_2(x)$ for all $x \in \mathsf{dom}\,\sigma_1$, then $\sigma_1^*(x) = \sigma_2^*(x)$ for all $x \in \mathsf{dom}\,\sigma_1^* = \mathsf{dom}\,\sigma_1$.*

Proof. From (5) and the fact that $(\!(\lambda x\,.\,M)\!)e\alpha \neq \bot$ we have that $\mathsf{dom}\,\sigma_i = \mathsf{dom}\,\sigma_i^*$. By Lemma 13, we have $P_{\sigma_1}(\sigma_2^*) \sqsubseteq P_{\sigma_2}(\sigma_2^*) = \sigma_2^*$, so σ_2^* is a prefixpoint of P_{σ_1}. Since σ_1^* is the least prefixpoint, $\sigma_1^* \sqsubseteq \sigma_2^*$.

In addition, if σ_1 and σ_2 agree on $\mathsf{dom}\,\sigma_1$, then for $x \in \mathsf{dom}\,\sigma_1$, equality holds in (4), thus $P_{\sigma_1}(\ell)(x) = P_{\sigma_2}(\ell)(x)$. As this is true for all ℓ, we have

$$\sigma_1^*(x) = \sup_\alpha P_{\sigma_1}^\alpha(\bot)(x) = \sup_\alpha P_{\sigma_2}^\alpha(\bot)(x) = \sigma_2^*(x). \quad \square$$

[4]In the coproduct, the $\bot$'s of the two domains are coalesced.

Lemma 15. *Let $\sigma : \mathsf{Var} \to \mathsf{Val} + \Lambda_\perp$ be a mixed environment. If $v \in \Lambda$, $y \notin \mathsf{FV}(v)$, and $y \notin \operatorname{dom}\sigma$, then $\sigma[v/y]^* = \sigma^*[(\!(v)\!)\sigma^*\alpha/y]$.*

Proof.

$$\begin{aligned}
&\sigma[v/y]^*(y)\\
&= P_{\sigma[v/y]}(\sigma[v/y]^*)(y)\\
&= \begin{cases}\sigma[v/y](y), & \sigma[v/y](y) \in \mathsf{Val}\\ (\!(\sigma[v/y](y))\!)\sigma[v/y]^*\alpha, & \sigma[v/y](y) \in \Lambda\end{cases}\\
&= (\!(v)\!)\sigma[v/y]^*\alpha \qquad (6)\\
&= (\!(v)\!)\sigma^*\alpha \qquad (7)\\
&= \sigma^*[(\!(v)\!)\sigma^*\alpha/y](y),
\end{aligned}$$

where the inference (6) is because $\sigma[v/y](y) = v$ and $v \in \Lambda$ and the inference (7) is by Lemma 14. For $x \neq y$, $x \in \operatorname{dom}\sigma$,

$$\begin{aligned}
&\sigma[v/y]^*(x)\\
&= P_{\sigma[v/y]}(\sigma[v/y]^*)(x)\\
&= \begin{cases}\sigma[v/y](x), & \sigma[v/y](x) \in \mathsf{Val}\\ (\!(\sigma[v/y](x))\!)\sigma[v/y]^*\alpha, & \sigma[v/y](x) \in \Lambda\end{cases}\\
&= \begin{cases}\sigma(x), & \sigma(x) \in \mathsf{Val}\\ (\!(\sigma(x))\!)\sigma[v/y]^*\alpha, & \sigma(x) \in \Lambda\end{cases} \qquad (8)\\
&= \begin{cases}\sigma(x), & \sigma(x) \in \mathsf{Val}\\ (\!(\sigma(x))\!)\sigma^*\alpha, & \sigma(x) \in \Lambda\end{cases} \qquad (9)\\
&= P_\sigma(\sigma^*)(x)\\
&= \sigma^*(x)\\
&= \sigma^*[(\!(v)\!)\sigma^*\alpha/y](x),
\end{aligned}$$

where the inference (8) is because $\sigma[v/y](x) = \sigma(y)$ and the inference (9) is by Lemma 14. □

VI. Relating the Stochastic and Deterministic Semantics

In Definition 9, we have reworked the semantic function of [1] to be of type

$$(\!|-|\!) : \mathsf{Exp} \to \mathsf{Env} \to \mathsf{Toss} \to \mathsf{RV},$$

and in Definition 11 we have given a deterministic semantics of type

$$(\!(-)\!) : \mathsf{Exp} \to \mathsf{Env}' \to 2^\omega \to \mathsf{Val},$$

where

- RV are the *random variables* $\Omega \to \mathsf{Val}$ from a sample space Ω taking values in a reflexive CPO Val,
- Exp are the *stochastic λ-terms* M,
- Env are the (*stochastic*) *environments* $E : \mathsf{Var} \to \mathsf{RV}$,
- Env′ are the (*deterministic*) *environments* $e : \mathsf{Var} \to \mathsf{Val}$,
- Toss are the *tossing processes* $T : \Omega \to 2^\omega$.

In this section we establish the formal relationship between these two semantics (Theorem 16). The idea is that in the stochastic semantics, although all data are parameterized by a sample point $\omega \in \Omega$, it is actually the same ω throughout a single run of the program. All independence requirements are satisfied by the way randomness is allocated to the different tasks. For example, in the clause for $(\!|MN|\!)$, the coin sequence is broken into three disjoint sequences to use in three distinct tasks (evaluation of M, evaluation of N, and application of M to N). This is equivalent to three independent tossing processes. In the clause for $\oplus$, we resolve the probabilistic choice using the head coin, but then throw it away and continue with the tail of the coin sequence, so the head coin is not reused. Because of these considerations, linearity is maintained.

A. Fun *and* Lam

We first show how to define Fun and Lam with the desired properties from fun and lam. Recall that

$$\mathsf{Val} \underset{\mathsf{lam}}{\overset{\mathsf{fun}}{\rightleftarrows}} [2^\omega \to \mathsf{Val} \to \mathsf{Val}]$$

and we need

$$\Omega \to \mathsf{Val} \underset{\mathsf{Lam}}{\overset{\mathsf{Fun}}{\rightleftarrows}} [(\Omega \to 2^\omega) \to (\Omega \to \mathsf{Val}) \to (\Omega \to \mathsf{Val})].$$

We define Fun and Lam in two steps:

$$\mathsf{Fun} = \mathsf{Fun}_2 \circ \mathsf{Fun}_1 \qquad \mathsf{Lam} = \mathsf{Lam}_1 \circ \mathsf{Lam}_2,$$

where

$$\Omega \to \mathsf{Val} \underset{\mathsf{Lam}_1}{\overset{\mathsf{Fun}_1}{\rightleftarrows}} [\Omega \to (2^\omega \to \mathsf{Val} \to \mathsf{Val})]$$

$$\underset{\mathsf{Lam}_2}{\overset{\mathsf{Fun}_2}{\rightleftarrows}} [(\Omega \to 2^\omega) \to (\Omega \to \mathsf{Val}) \to (\Omega \to \mathsf{Val})]$$

Fun_1 and Lam_1 are just the covariant hom-functor in Set applied to fun and lam, respectively.

$$\begin{aligned}
\mathsf{Fun}_1 &= \mathsf{Set}(\Omega, \mathsf{fun}) = \mathsf{fun} \circ -\\
\mathsf{Lam}_1 &= \mathsf{Set}(\Omega, \mathsf{lam}) = \mathsf{lam} \circ -
\end{aligned}$$

Then

$$\begin{aligned}
(\mathsf{Fun}_1 \circ \mathsf{Lam}_1)f &= ((\mathsf{fun} \circ -) \circ (\mathsf{lam} \circ -))f\\
&= (\mathsf{fun} \circ -)(\mathsf{lam} \circ f) = \mathsf{fun} \circ \mathsf{lam} \circ f = f,
\end{aligned}$$

so $\mathsf{Fun}_1 \circ \mathsf{Lam}_1 = \mathsf{id}$. Also,

$$\begin{aligned}
\mathsf{Lam}_1(\lambda\omega.f) &= (\mathsf{lam} \circ -)(\lambda\omega.f)\\
&= \mathsf{lam} \circ \lambda\omega.f = \lambda\omega.(\mathsf{lam} \circ \lambda\omega.f)\omega\\
&= \lambda\omega.\,\mathsf{lam}((\lambda\omega.f)\omega) = \lambda\omega.\,\mathsf{lam}(f). \qquad (10)
\end{aligned}$$

We define

$$\mathsf{Fun}_2 = \lambda fg.S(Sfg),$$

where $S = \lambda gh\omega.g\omega(h\omega)$ is the familiar S-combinator from combinatory logic. Then

$$\mathsf{Fun}_2 fgh\omega = S(Sfg)h\omega = Sfg\omega(h\omega) = (f\omega)(g\omega)(h\omega). \qquad (11)$$

The function Fun_2 is injective:

$$\begin{aligned}&\mathsf{Fun}_2 f_1 = \mathsf{Fun}_2 f_2\\ &\Rightarrow \lambda g\,.\,S(Sf_1g) = \lambda g\,.\,S(Sf_2g)\\ &\Rightarrow \forall g\forall h\forall\omega\ (f_1\omega)(g\omega)(h\omega) = (f_2\omega)(g\omega)(h\omega)\\ &\Rightarrow \forall y\forall z\forall\omega\ f_1\omega yz = f_2\omega yz\\ &\Rightarrow f_1 = f_2.\end{aligned}$$

We define Lam_2 to be the inverse of Fun_2 on the image of Fun_2. Thus

$$\mathsf{Lam}_2(\lambda ghw\,.\,(fw)(gw)(hw)) = \mathsf{Lam}_2(\mathsf{Fun}_2 f) = f. \quad (12)$$

Also,

$$\begin{aligned}&\mathsf{Fun}_2(\mathsf{Lam}_2(\lambda ghw\,.\,(fw)(gw)(hw)))\\ &= \mathsf{Fun}_2 f = \lambda ghw\,.\,(fw)(gw)(hw),\end{aligned}$$

so $\mathsf{Fun}_2 \circ \mathsf{Lam}_2 = \mathsf{id}$ on its domain. Then

$$\begin{aligned}\mathsf{Fun}\circ\mathsf{Lam} &= \mathsf{Fun}_2\circ\mathsf{Fun}_1\circ\mathsf{Lam}_1\circ\mathsf{Lam}_2\\ &= \mathsf{Fun}_2\circ\mathsf{Lam}_2 = \mathsf{id}.\end{aligned}$$

Moreover, using (10), (11), and (12),

$$\begin{aligned}&\mathsf{Fun}\,fghw = \mathsf{Fun}_2(\mathsf{Fun}_1 f)ghw\\ &= \mathsf{Fun}_2(\mathsf{fun}\circ f)ghw\\ &= (\mathsf{fun}\circ f)w(gw)(hw) = \mathsf{fun}(fw)(gw)(hw) \quad (13)\\ &\mathsf{Lam}(\lambda ghw\,.\,f(gw)(hw))\\ &= \mathsf{Lam}_1(\mathsf{Lam}_2(\lambda ghw\,.\,(\lambda w\,.\,f)w(gw)(hw)))\\ &= \mathsf{Lam}_1(\lambda w\,.\,f) = \lambda w\,.\,\mathsf{lam}(f). \quad (14)\end{aligned}$$

Note that the domain $\Omega \to \mathsf{Val}$ is not reflexive with respect to $[(\Omega \to 2^\omega) \to (\Omega \to \mathsf{Val}) \to (\Omega \to \mathsf{Val})]$ under Fun and Lam, but only with respect to $[\Omega \to (2^\omega \to \mathsf{Val} \to \mathsf{Val})]$ (and its image in $[(\Omega \to 2^\omega) \to (\Omega \to \mathsf{Val}) \to (\Omega \to \mathsf{Val})]$ under Fun_2). However this is all we need for Theorem 16.

B. Relating the Deterministic and Stochastic Semantics

The following theorem gives the formal relationship between the stochastic and deterministic denotational semantics.

Theorem 16. $\lambda w\,.\,(\!|M|\!)e(Tw) = (\!|M|\!)(\lambda xw\,.\,ex)T$.

Proof. The proof follows by case analysis. A complete proof can be found in the appendix. □

VII. Operational Semantics

In this section we give big- and small-step operational rules in the style of [8] and prove their equivalence. The two styles use their coins in different patterns and the relationship must be formally specified. This is done using the tree processes of §III-B.

A. Big-step rules

The notation $\langle M, e\rangle \Downarrow_\alpha \langle v, f\rangle$ means that $\langle M, e\rangle$ reduces to $\langle v, f\rangle$ under the big-step rules below with coins $\alpha \in 2^\omega$.

$$\langle x, e\rangle \Downarrow_\alpha \langle e(x), e\rangle$$

$$\frac{\langle M, e\rangle \Downarrow_\alpha \langle v, f\rangle}{\langle M \oplus N, e\rangle \Downarrow_{0\alpha} \langle v, f\rangle} \qquad \frac{\langle N, e\rangle \Downarrow_\alpha \langle v, f\rangle}{\langle M \oplus N, e\rangle \Downarrow_{1\alpha} \langle v, f\rangle}$$

$$\frac{\left\{\begin{array}{c}\langle M, e\rangle \Downarrow_{\pi_0^3(\alpha)} \langle \lambda x\,.\,K, e_0\rangle\\ \langle N, e_0\rangle \Downarrow_{\pi_1^3(\alpha)} \langle u, e_1\rangle\\ \langle K[y/x], e_1[u/y]\rangle \Downarrow_{\pi_2^3(\alpha)} \langle v, f\rangle\end{array}\right\}}{\langle MN, e\rangle \Downarrow_\alpha \langle v, f\rangle}$$

where in the third premise of the last rule, y is a fresh variable.

B. Small-step rules

The notation $\langle M, e\rangle \to_x \langle N, f\rangle$ means that $\langle M, e\rangle$ reduces to $\langle N, f\rangle$ under the small-step rules below via a computation that consumes exactly coins $x \in 2^*$ in order from left to right. The notation $\langle M, e\rangle \to_\alpha \langle N, f\rangle$ means that $\langle M, e\rangle \to_x \langle N, f\rangle$ for some $x \prec \alpha$, where $\alpha \in 2^\omega$.

$$\frac{\langle M, e\rangle \to_x \langle M', f\rangle}{\langle MN, e\rangle \to_x \langle M'N, f\rangle} \qquad \frac{\langle N, e\rangle \to_x \langle N', f\rangle}{\langle vN, e\rangle \to_x \langle vN', f\rangle}$$

$$\langle M, e\rangle \to_\varepsilon \langle M, e\rangle \qquad \langle x, e\rangle \to_\varepsilon \langle e(x), e\rangle$$

$$\langle (\lambda x\,.\,M)v, e\rangle \to_\varepsilon \langle M[y/x], e[v/y]\rangle \quad (y \text{ fresh})$$

$$\langle M \oplus N, e\rangle \to_0 \langle M, e\rangle \qquad \langle M \oplus N, e\rangle \to_1 \langle N, e\rangle$$

$$\frac{\langle M, e\rangle \to_x \langle N, f\rangle \quad \langle N, f\rangle \to_y \langle K, g\rangle}{\langle M, e\rangle \to_{xy} \langle K, g\rangle}$$

$$\frac{\langle M, e\rangle \to_x \langle N, f\rangle \quad x \prec \alpha}{\langle M, e\rangle \to_\alpha \langle N, f\rangle}$$

C. Relation of Big- and Small-Step Semantics

The big- and small-step operational semantics use their coins in different patterns, and we need a way to characterize how they relate. The big-step rule for application breaks its coin sequence up into three independent coin sequences to evaluate the function, to evaluate the argument, and to apply the function, respectively; whereas the small-step rules just use their coins sequentially.

The relationship is characterized by a *tree process* as described in §III. The construction is given in the proof of the following theorem.

Theorem 17. *For all $\langle M, e\rangle$ there exists a tree process $T_{\langle M,e\rangle}$ such that for all α, v, f,*

$$\langle M, e\rangle \Downarrow_\alpha \langle v, f\rangle \Leftrightarrow \langle M, e\rangle \to_{T_{\langle M,e\rangle}(\alpha)} \langle v, f\rangle.$$

Proof. The rules for the big-step semantics define proof trees by which one concludes that an instance of the big-step relation holds. We proceed by induction on the structure of these proof trees. The base case corresponds to reading a variable from the environment: $\langle x, e\rangle \Downarrow_\alpha \langle e(x), e\rangle$. This case is immediate since we can just take the tree process to be the

one that defines the identity function; note that the environment stores only values so there is no further reduction.

For the case

$$\frac{\langle M, e\rangle \Downarrow_\alpha \langle v, f\rangle}{\langle M \oplus N, e\rangle \Downarrow_{0\alpha} \langle v, f\rangle}$$

we have, by induction, a tree process $T_{\langle M,e\rangle}$, call it T' for short, such that

$$\langle M, e\rangle \to_{T'(\alpha)} \langle v, f\rangle$$

and analogously for the other branch of the choice. We can define the tree $t_{\langle(,M\rangle\oplus N)e}$ by

$$t_{\langle M\oplus N,e\rangle}(\alpha) = = \begin{cases} t_{\langle M,e\rangle}(\alpha') \text{ if } \alpha = 0\alpha' \\ t_{\langle N,e\rangle}(\alpha') \text{ if } \alpha = 1\alpha' \end{cases}$$

It is a routine calculation to verify the result in this case.

For the case $\langle MN, e\rangle$, take the tree

$$t_{\langle MN,e\rangle}(w) = \begin{cases} 3 \cdot t_{\langle K[y/x],e_1[u/y]\rangle}(z) + 2, \\ \quad \text{if } w = xyz \wedge \langle M, e\rangle \to_x \\ \quad \langle \lambda x\,.\,K, e_0\rangle \\ \quad \text{and } \langle N, e_0\rangle \to_y \langle u, e_1\rangle, \\ 3 \cdot t_{\langle N,e_0\rangle}(y) + 1, \\ \quad \text{if } w = xy \wedge \langle M, e\rangle \to_x \langle \lambda x\,.\,K, e_0\rangle \\ \quad \text{and } \langle N, e_0\rangle \to_y \text{NV}, \\ 3 \cdot t_{\langle M,e\rangle}(w), \text{ if } \langle M, e\rangle \to_w \text{NV}, \end{cases}$$

where NV means "some capsule that is not reduced," and let $T_{\langle MN,e\rangle}$ be the associated tossing process. Then $\langle MN, e\rangle \Downarrow_\alpha \langle v, f\rangle$ occurs iff there exist K, u, e_0, e_1, and y fresh such that

$$\langle M, e\rangle \Downarrow_{\pi_0^3(\alpha)} \langle \lambda x\,.\,K, e_0\rangle \qquad \langle N, e_0\rangle \Downarrow_{\pi_1^3(\alpha)} \langle u, e_1\rangle$$
$$\langle K[y/x], e_1[u/y]\rangle \Downarrow_{\pi_2^3(\alpha)} \langle v, f\rangle$$

By the induction hypothesis, this occurs iff there exist x, y, z such that

$$\begin{aligned} &\langle M, e\rangle \to_x \langle \lambda x\,.\,K, e_0\rangle && x \prec T_{\langle M,e\rangle}(\pi_0^3(\alpha)) \\ &\langle N, e_0\rangle \to_y \langle u, e_1\rangle && y \prec T_{\langle N,e_0\rangle}(\pi_1^3(\alpha)) \\ &\langle K[y/x], e_1[u/y]\rangle \to_z \langle v, f\rangle && z \prec T_{\langle K[y/x],e_1[u/y]\rangle}(\pi_2^3(\alpha)) \end{aligned}$$

By construction of $t_{\langle MN,e\rangle}(\alpha)$, $xyz \prec T_{\langle MN,e\rangle}(\alpha)$, so this occurs iff

$$\langle MN, e\rangle \to_x \langle (\lambda x\,.\,K)N, e_0\rangle \to_y \langle (\lambda x\,.\,K)u, e_1\rangle$$
$$\to_\varepsilon \langle K[y/x], e_1[u/y]\rangle \to_z \langle v, f\rangle,$$

which occurs iff $\langle MN, e\rangle \to_{T_{\langle MN,e\rangle}(\alpha)} \langle v, f\rangle$. □

VIII. Soundness and Adequacy

The following theorem asserts the soundness and adequacy of our denotational semantics with respect to our big-step operational semantics.

Theorem 18.

(i) *If* $\langle M, \sigma\rangle \Downarrow_\alpha \langle \lambda x\,.\,N, \tau\rangle$, *then for any* γ, $(\!(M)\!)\sigma^*\alpha = (\!(\lambda x\,.\,N)\!)\tau^*\gamma = \mathsf{lam}\,(\lambda\beta v\,.\,(\!(N)\!)\tau^*[v/x]\beta)$.

(ii) *If* $(\!(M)\!)\sigma^*\alpha = \mathsf{lam}\,f$ *for some* $f : [2^\omega \to \mathsf{Val} \to \mathsf{Val}]$, *then* $\langle M, \sigma\rangle \Downarrow_\alpha$.

Proof. (i) The proof is by induction on the derivation of $\langle M, \sigma\rangle \Downarrow_\alpha \langle v, \tau\rangle$. Let us do the easy cases first. For variables, we have $\langle x, \sigma\rangle \Downarrow_\alpha \langle \sigma(x), \sigma\rangle$ and

$$(\!(x)\!)\sigma^*\alpha = \sigma^*(x) = P_\sigma(\sigma^*)(x) = (\!(\sigma(x))\!)\sigma^*\beta.$$

For abstractions, we have $\langle \lambda x\,.\,M, \sigma\rangle \Downarrow_\alpha \langle \lambda x\,.\,M, \sigma\rangle$ and

$$(\!(\lambda x\,.\,M)\!)\sigma^*\alpha = (\!(\lambda x\,.\,M)\!)\sigma^*\beta$$

for any β, since the semantics of abstractions does not depend on α.

For choice, suppose $\langle M \oplus N, \sigma\rangle \Downarrow_\alpha \langle v, \tau\rangle$. If $\mathsf{hd}\,\alpha = 0$, then $\langle M, \sigma\rangle \Downarrow_{\mathsf{tl}\,\alpha} \langle v, \tau\rangle$. By the induction hypothesis, $(\!(M)\!)\sigma^*(\mathsf{tl}\,\alpha) = (\!(v)\!)\tau^*\beta$, so $(\!(M \oplus N)\!)\sigma^*(0\,\mathsf{tl}\,\alpha) = (\!(v)\!)\tau^*\beta$. By a similar argument, if $\mathsf{hd}\,\alpha = 1$, then $(\!(M \oplus N)\!)\sigma^*(1\,\mathsf{tl}\,\alpha) = (\!(v)\!)\tau^*\beta$. Thus in either case, $(\!(M \oplus N)\!)\sigma^*\alpha = (\!(v)\!)\tau^*\beta$.

The most involved case is application. Suppose $\langle MN, \sigma\rangle \Downarrow_\alpha \langle v, \tau\rangle$. Then for some $\lambda x\,.\,K$, σ_0, u, and σ_1 such that $\sigma \sqsubseteq \sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \tau$,

$$\langle M, \sigma\rangle \Downarrow_{\pi_0^3(\alpha)} \langle \lambda x\,.\,K, \sigma_0\rangle \qquad \langle N, \sigma_0\rangle \Downarrow_{\pi_2^3(\alpha)} \langle u, \sigma_1\rangle$$
$$\langle K[y/x], \sigma_1[u/y]\rangle \Downarrow_{\pi_1^3(\alpha)} \langle v, \tau\rangle$$

where $y \in \mathsf{Var}$ is fresh. By the induction hypothesis,

$$(\!(M)\!)\sigma^*(\pi_0^3(\alpha)) = (\!(\lambda x\,.\,K)\!)\sigma_0^*\beta \tag{15}$$
$$(\!(N)\!)\sigma_0^*(\pi_2^3(\alpha)) = (\!(u)\!)\sigma_1^*\beta \tag{16}$$
$$(\!(K[y/x])\!)\sigma_1[u/y]^*(\pi_1^3(\alpha)) = (\!(v)\!)\tau^*\beta. \tag{17}$$

Then

$$\begin{aligned} &\mathsf{fun}((\!(M)\!)\sigma^*(\pi_0^3(\alpha))) \\ &= \mathsf{fun}((\!(\lambda x\,.\,K)\!)\sigma_0^*\gamma) && \text{by (15)} \\ &= \mathsf{fun}((\!(\lambda y\,.\,K[y/x])\!)\sigma_0^*\gamma),\ y \text{ fresh} && \text{by } \alpha\text{-conversion} \\ &= \mathsf{fun}(\mathsf{lam}(\lambda\beta v\,.\,(\!(K[y/x])\!)\sigma_0^*[v/y]\beta)) \\ &= \lambda\beta v\,.\,(\!(K[y/x])\!)\sigma_0^*[v/y]\beta. && (18) \end{aligned}$$

By (16) and Lemma 14, since σ_0 is an extension of σ,

$$(\!(N)\!)\sigma^*(\pi_2^3(\alpha)) = (\!(N)\!)\sigma_0^*(\pi_2^3(\alpha)) = (\!(u)\!)\sigma_1^*\beta. \tag{19}$$

By Lemma 15, since y is fresh,

$$\sigma_1[u/y]^* = \sigma_1^*[(\!(u)\!)\sigma_1^*\gamma/y]. \tag{20}$$

Using (17)–(20) and Lemma 14,

$$\begin{aligned} &(\!(MN)\!)\sigma^*\alpha \\ &= \mathsf{fun}((\!(M)\!)\sigma^*(\pi_0^3(\alpha)))(\pi_1^3(\alpha))((\!(N)\!)\sigma^*(\pi_2^3(\alpha))) \\ &= (\lambda\beta v\,.\,(\!(K[y/x])\!)\sigma_0[v/y]\beta)(\pi_1^3(\alpha))((\!(u)\!)\sigma_1^*\gamma) \\ &= (\!(K[y/x])\!)\sigma_0^*[(\!(u)\!)\sigma_1^*\gamma/y](\pi_1^3(\alpha)) \\ &= (\!(K[y/x])\!)\sigma_1^*[(\!(u)\!)\sigma_1^*\gamma/y](\pi_1^3(\alpha)) \\ &= (\!(K[y/x])\!)\sigma_1[u/y]^*(\pi_1^3(\alpha)) \\ &= (\!(v)\!)\tau^*\beta. \end{aligned}$$

(ii) For variables, we have $(\!(x)\!)\sigma^*\alpha = \sigma^*(x) = \mathsf{lam}\, f$ for some $f : [2^\omega \to \mathsf{Val} \to \mathsf{Val}]$. Since $\mathsf{dom}\,\sigma = \mathsf{dom}\,\sigma^*$, we must have $\sigma(x) = \lambda x.K$ for some $\lambda x.K \in \Lambda$. By definition of σ^*, $\sigma^*(x) = (\!(\lambda x.K)\!)\sigma^*\alpha = \mathsf{lam}(\lambda\beta v.(\!(K)\!)\sigma^*[v/x]\beta)$. As lam is injective, $f = \lambda\beta v.(\!(K)\!)\sigma^*[v/x]\beta$, and $\langle x, \sigma\rangle \Downarrow_\alpha \langle \lambda y.K, \sigma\rangle$.

For λ-abstractions, $(\!(\lambda x.M)\!)\sigma^*\alpha = \mathsf{lam}(\lambda\beta v.(\!(M)\!)\sigma^*[v/x]\beta)$ and $\langle \lambda x.M, \sigma\rangle \Downarrow_\alpha \langle \lambda x.M, \sigma\rangle$.

For choice, we have $(\!(M \oplus N)\!)\sigma^*\alpha = \mathsf{hd}\,\alpha\,?\,(\!(M)\!)\sigma^*(\mathsf{tl}\,\alpha) : (\!(N)\!)\sigma^*(\mathsf{tl}\,\alpha) = \mathsf{lam}\, f$. Either $\mathsf{hd}\,\alpha = 1$, in which case $(\!(M)\!)\sigma^*(\mathsf{tl}\,\alpha) = \mathsf{lam}\, f$ and $\langle M, \sigma\rangle \Downarrow_{\mathsf{tl}\,\alpha}$ by the induction hypothesis, or $\mathsf{hd}\,\alpha = 0$, in which case $(\!(N)\!)\sigma^*(\mathsf{tl}\,\alpha) = \mathsf{lam}\, f$ and $\langle N, \sigma\rangle \Downarrow_{\mathsf{tl}\,\alpha}$ by the induction hypothesis. In either case, $\langle M \oplus N, \sigma\rangle \Downarrow_\alpha$ by the big-step rule for choice.

Finally, for applications, suppose $(\!(MN)\!)\sigma^*\alpha = \mathsf{lam}\, f$. We have

$$(\!(MN)\!)\sigma^*\alpha = \mathsf{fun}\,((\!(M)\!)\sigma^*\alpha_0)\,\alpha_1\,((\!(N)\!)\sigma^*\alpha_2)$$

If $(\!(M)\!)\sigma^*\alpha_0 = \bot$, then

$$\begin{aligned}(\!(MN)\!)\sigma^*\alpha &= \mathsf{fun}\,((\!(M)\!)\sigma^*\alpha_0)\,\alpha_1\,((\!(N)\!)\sigma^*\alpha_2)\\ &= \mathsf{fun}\,\bot\,\alpha_1\,((\!(N)\!)\sigma^*\alpha_2)\\ &= (\lambda\beta v.\bot)\,\alpha_1\,((\!(N)\!)\sigma^*\alpha_2)\\ &= \bot,\end{aligned}$$

contradicting our assumption. Similarly, if $(\!(M)\!)\sigma^*\alpha_0 = \mathsf{lam}\, g$ but $(\!(N)\!)\sigma^*\alpha_2 = \bot$, then

$$\begin{aligned}(\!(MN)\!)\sigma^*\alpha &= \mathsf{fun}\,((\!(M)\!)\sigma^*\alpha_0)\,\alpha_1\,((\!(N)\!)\sigma^*\alpha_2)\\ &= \mathsf{fun}(\mathsf{lam}\, g)\,\alpha_1\,\bot\\ &= g\,\alpha_1\,\bot\\ &= \bot,\end{aligned}$$

again contradicting our assumption. So we can assume that $(\!(M)\!)\sigma^*\alpha_0 = \mathsf{lam}\, g$ and $(\!(N)\!)\sigma^*\alpha_2 \neq \bot$. By the induction hypothesis and Lemma 15,

$$\begin{aligned}&\langle M, \sigma\rangle \Downarrow_{\alpha_0} \langle \lambda x.K, \sigma_0\rangle \qquad \langle N, \sigma_0\rangle \Downarrow_{\alpha_2} \langle u, \sigma_1\rangle\\ &g = \lambda\beta v.(\!(K)\!)\sigma_0^*[v/x]\beta \qquad (\!(N)\!)\sigma_0^*\alpha_2 = (\!(u)\!)\sigma_1^*(-).\end{aligned}$$

$$\begin{aligned}(\!(MN)\!)\sigma^*\alpha &= \mathsf{fun}\,((\!(M)\!)\sigma^*\alpha_0)\,\alpha_1\,((\!(N)\!)\sigma_0^*\alpha_2)\\ &= \mathsf{fun}(\mathsf{lam}\, g)\,\alpha_1\,((\!(u)\!)\sigma_1^*(-))\\ &= g\,\alpha_1\,((\!(u)\!)\sigma_1^*(-))\\ &= (\lambda\beta v.(\!(K)\!)\sigma_0^*[v/x]\beta)\,\alpha_1\,((\!(u)\!)\sigma_1^*(-))\\ &= (\!(K)\!)\sigma_0^*[(\!(u)\!)\sigma_1^*(-)/x]\alpha_1\\ &= (\!(K[y/x])\!)\sigma_0^*[(\!(u)\!)\sigma_1^*(-)/y]\alpha_1\\ &= (\!(K[y/x])\!)\sigma_1^*[(\!(u)\!)\sigma_1^*(-)/y]\alpha_1\\ &= (\!(K[y/x])\!)\sigma_1[u/y]^*\alpha_1,\end{aligned}$$

and by the induction hypothesis, $\langle K[y/x], \sigma_1[u/y]\rangle \Downarrow_{\alpha_1}$. By the big-step rule for application, $\langle MN, \sigma\rangle \Downarrow_\alpha$. □

Corollary 19. *For every capsule* $\langle M, \sigma\rangle$

$$\{\alpha \in 2^\omega \mid \langle M, \sigma\rangle \Uparrow_\alpha\} = \{\alpha \in 2^\omega \mid (\!(M)\!)\sigma\alpha = \bot\}$$

IX. Related Work and Concluding Remarks

In probability theory, stochastic processes are modeled as random variables (measurable functions) defined on a probability space, which is viewed as the source of randomness. It is natural to think of probabilistic programming in a similar vein, and in many of the related approaches one sees a programming formalism augmented by a source of randomness.

The idea of modeling probabilistic programs with a stream of random data in the λ-calculus has been used in [5] as well. In that work, the authors define big-step and small-step operational semantics for an idealized version of the probabilistic language Church. Their operational semantics, like ours, is a binary relation parameterized by a source of randomness. Although their language can handle continuous distributions and soft conditioning, they have not given a denotational semantics. Our approach could accommodate continuous distributions simply by changing the source of randomness to $\mathbb{R}$. We might interpret $\mathbb{R}$ either as the usual real numbers or as its constructive version as in Real PCF [13]. To accommodate soft conditioning, we could adopt the solution proposed in [14] of adding a write-only state cell to store the weight of the execution trace, which can be done by slightly changing our domain equation. These extensions would complicate our semantics and its presentation, so we leave them for future work.

In a similar vein, the category of Quasi-Borel Spaces ($\mathbf{Qbs}$) defined in [15] also assumes that probability comes from an ambient source of randomness, which they model as a set of random variables of type $\mathbb{R} \to A$ satisfying certain properties. Furthermore, they show that in $\mathbf{Qbs}$ there is a Giry-like monad that uses the set of random variables in its definition. In [16], to accommodate arbitrary recursion, they equip every Quasi-Borel space with a complete partial order and require the set of random variables to be closed with respect to directed suprema. It would be interesting to better understand how our requirement of continuity of coin usage relates to their construction.

There has also been alternative operational semantics for languages similar to ours. In [17], an operational semantics is defined in terms of Markov kernels over the values. Since the focus of that work is on syntactic methods to reason about contextual equivalence, a denotational semantics is not defined. However, by our adequacy and soundness theorems, we can also use our semantics to reason about contextual equivalence. Furthermore, since the set $\{\alpha \in 2^\omega \mid \langle M, \sigma\rangle \Downarrow_\alpha\}$ is measurable, one can prove by induction on reduction sequences that the semantics of [17] and ours are equivalent.

An alternative domain theoretical tool that has been used to interpret randomness is the probabilistic powerdomain construction. Recently the Jung-Tix problem [18] has been solved [19], showing that it is possible to define a commutative probabilistic monad in a cartesian closed category of continuous domains. We tackle the problem from a different perspective. We leave for future work to understand the connections between the probabilistic powerdomain and our func-

tor $MX = 2^{\leq\omega} \to X$. It is worth noting that this functor is the Reader monad from functional programming. Unfortunately, if we were to use the same monad multiplication from the Reader monad—i.e. $\mu_X(t) = \lambda\alpha.t\,\alpha\,\alpha$—we would reuse the same source of randomness twice, breaking linearity of usage of random data and probabilistic independence. Furthermore, for any other natural transformation $M^2 \Rightarrow M$ that preserves such linearity conditions, the associativity monad law holds only up to a measure-preserving function. As an example, suppose that we chose the natural transformation $\mu_X(t) = \lambda\alpha : 2^\omega.t\,\alpha_0\,\alpha_1$ as our monad multiplication. In this case the associativity law becomes:

$$\lambda t : M^3(X)\,\alpha : 2^\omega.\,t\,\alpha_{00}\,\alpha_{10}\,\alpha_1 = \lambda t : M^3(X)\,\alpha : 2^\omega.\,t\,\alpha_0\,\alpha_{10}\,\alpha_{11}$$

Obviously the equation above does not hold.

As a final example of a related formalism, we mention probabilistic coherence spaces [20], [21], which use the decomposition of the usual function space into a linear function space and an exponential comonad. In [21], a fully abstract semantics is given for a probabilistic extension to PCF. They model higher-order probability by using a generalization of transition matrices. Cones of measures have also been used to construct a model of higher-order probabilistic computation [22]. It is a fascinating question to understand precisely the relationship between all these formalisms for higher-order probabilistic computation.

To conclude, while other approaches to denotational semantics for higher-order probabilistic computation have been taken, no such construction is the obviously "correct" one. To clarify this matter, the connections between different approaches would need to be well understood. But this is a hard open problem and requires in-depth understanding of the various possible approaches and how they relate, and the field is not there yet. Even though the approaches mentioned above are interesting, we do not see that they have any compelling argument suggesting that they are the only "right" semantics for probabilistic higher-order computation. In this paper we contributed to the area by focusing on the Boolean-valued semantics of [1] and modified it to accommodate a call-by-value operational semantics which we proved it sound and adequate with respect to the modified denotational semantics, solving the main open problem from that work.

Acknowledgments

Thanks to Giorgio Bacci, Fredrik Dahlqvist, Robert Furber and Arthur Azevedo de Amorim. Special thanks to Dana Scott for many inspiring conversations. Thanks to the Bellairs Research Institute of McGill University for providing a wonderful research environment.

This material is based upon work supported by a grant from the National Science Foundation under grants No. AitF-1637532, No. SaTC-1717581, and No. CCF-2008083. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Panangaden is funded by NSERC (Canada).

References

[1] G. Bacci, R. Furber, D. Kozen, R. Mardare, P. Panangaden, and D. Scott, "Boolean-valued semantics for the stochastic λ-calculus," in *Logic in Computer Science (2018)*, 2018.

[2] D. S. Scott, "Stochastic λ-calculi," *Journal of Applied Logic*, vol. 12, no. 3, pp. 369–376, 2014.

[3] J. L. Bell, *Set theory: Boolean-valued models and independence proofs*. Oxford University Press, 2011, vol. 47.

[4] D. Scott, "A proof of the independence of the continuum hypothesis," *Mathematical systems theory*, vol. 1, no. 2, pp. 89–111, 1967.

[5] J. Borgström, U. Dal Lago, A. D. Gordon, and M. Szymczak, "A lambda-calculus foundation for universal probabilistic programming," in *International Conference on Functional Programming (ICFP)*, 2016.

[6] N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," *arXiv preprint arXiv:1206.3255*, 2012.

[7] J.-B. Jeannin and D. Kozen, "Computing with capsules," in *International Workshop on Descriptional Complexity of Formal Systems*. Springer, 2012, pp. 1–19.

[8] G. D. Plotkin, *A structural approach to operational semantics*. Aarhus university, 1981.

[9] R. Walter, "Real and complex analysis," 1974.

[10] H. P. Barendregt, *The lambda calculus*. North-Holland Amsterdam, 1984, vol. 3.

[11] E. Engeler, "Algebras and combinators," *Algebra universalis*, vol. 13, no. 1, pp. 389–392, 1981.

[12] G. Longo, "Set-theoretical models of λ-calculus: theories, expansions, isomorphisms," *Annals of pure and applied logic*, vol. 24, no. 2, pp. 153–188, 1983.

[13] M. H. Escardó, "Pcf extended with real numbers," *Theoretical Computer Science*, vol. 162, no. 1, pp. 79–115, 1996.

[14] S. Staton, F. Wood, H. Yang, C. Heunen, and O. Kammar, "Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints," in *Logic in Computer Science (LICS)*, 2016.

[15] C. Heunen, O. Kammar, S. Staton, and H. Yang, "A convenient category for higher-order probability theory," in *Logic in Computer Science (LICS)*, 2017.

[16] M. Vákár, O. Kammar, and S. Staton, "A domain theory for statistical probabilistic programming," in *Principles of Programming Languages (POPL)*, 2019.

[17] V. Vignudelli, "Behavioral equivalences for higher-order languages with probabilities," Ph.D. dissertation, Università di Bologna, 2017.

[18] A. Jung and R. Tix, "The troublesome probabilistic powerdomain," *Electronic Notes in Theoretical Computer Science*, vol. 13, pp. 70–91, 1998.

[19] X. Jia, B. Lindenhovius, M. Mislove, and V. Zamdzhiev, "Commutative monads for probabilistic programming languages," in *Logic in Computer Science (LICS)*, 2021.

[20] V. Danos and T. Ehrhard, "Probabilistic coherence spaces as a model of higher-order probabilistic computation," *Information and Computation*, vol. 209, no. 6, pp. 966–991, 2011.

[21] T. Ehrhard, C. Tasson, and M. Pagani, "Probabilistic coherence spaces are fully abstract for probabilistic pcf," in *Principles of Programming Languages (POPL)*, 2014.

[22] T. Ehrhard, M. Pagani, and C. Tasson, "Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming," in *Principles of Programming Languages(POPL)*, 2017.