Stochastic λ-Calculi

Dana S. Scott

University Professor, Emeritus
Carnegie Mellon University
Visiting Scholar in Mathematics
University of California, Berkeley

September, 2013

(A report on work in progress.)
**Church's \(\lambda\)-Calculus**

**Definition.** \(\lambda\)-calculus — as a formal theory — has rules for the *explicit definition* of functions *via* equational axioms:

- **\(\alpha\)-conversion**
  \[
  \lambda x.[...x...] = \lambda y.[...y...]
  \]

- **\(\beta\)-conversion**
  \[
  (\lambda x.[...x...])(T) = [...T...]
  \]

- **\(\eta\)-conversion**
  \[
  \lambda x.F(x) = F
  \]

The basic syntax has one binary operation of *application* and one variable-binding operator of *abstraction*. These are the "logical" notions of the theory, but we can add *other constants* for special operators.

Note that third axiom will be dropped in favor of a theory employing properties of a partial ordering.
Definition. The **enumeration operator model** is given by these definitions on **sets** of integers:

1. **Pairing**: \((n, m) = 2^n (2m+1)\).
2. **Sequence numbers**: \(\langle \rangle = 0\) and \(\langle n_0, n_1, ..., n_{k-1}, n_k \rangle = (\langle n_0, n_1, ..., n_{k-1} \rangle, n_k)\).
3. **Sets**: \(\text{set}(0) = \emptyset\) and \(\text{set}((n, m)) = \text{set}(n) \cup \{m\}\).
4. **Kleene star**: \(X^* = \{n \mid \text{set}(n) \subseteq X\}\), for sets \(X \subseteq \mathbb{N}\).

**Application**

\[ F(X) = \{ m \mid \exists n \in X^*. (n, m) \in F \} \]

**Abstraction**

\[ \lambda X. [...X...] = \{0 \} \cup \{(n, m) \mid m \in [...\text{set}(n)...]\} \]

**NOTE:** This model could easily have been defined in 1957, and it satisfies the rules of \(\alpha, \beta\)-conversion (but not \(\eta\)).

(Some historical comments can be found at the end of these notes.)
(1) The powerset $\mathcal{P}(\mathbb{N}) = \{ x \mid x \subseteq \mathbb{N} \}$ is a topological space with the sets $\mathcal{U}_n = \{ x \mid n \in x^* \}$ as a basis for the topology.

(2) Functions $\Phi: \mathcal{P}(\mathbb{N})^n \to \mathcal{P}(\mathbb{N})$ are continuous iff, for all integers, $m \in \Phi(\{x_0, x_1, \ldots, x_{n-1}\})$ iff there are $k_i \in x_i^*$ for all $i < n$, such that $m \in \Phi(\text{set}(k_0), \ldots, \text{set}(k_{n-1}))$.

(3) The application operation $F(x)$ is continuous as a function of two variables.

(4) If $\Phi(x_0, x_1, \ldots, x_{n-1})$ is continuous, then the abstraction $\lambda x_0. \Phi(x_0, x_1, \ldots, x_{n-1})$ is continuous in all of the remaining variables.

(5) If $\Phi(x)$ is continuous, then $\lambda x. \Phi(x)$ is the largest set $F$ such that for all sets $T$, we have $F(T) = \Phi(T)$.

(6) And, note, therefore, that generally $F \subseteq \lambda x. F(x)$.
Some Lambda Properties

For all sets of integers \( F \) and \( G \) we have:

\[
\lambda X. F(X) \subseteq \lambda X. G(X) \iff \forall X. F(X) \subseteq G(X),
\]

\[
\lambda X. (F(X) \cap G(X)) = \lambda X. F(X) \cap \lambda X. G(X),
\]

and

\[
\lambda X. (F(X) \cup G(X)) = \lambda X. F(X) \cup \lambda X. G(X).
\]

**Definition.** A continuous operator \( \Phi(X_0, X_1, \ldots, X_{n-1}) \) is *computable* iff in the model this set is RE:

\[
F = \lambda X_0 \lambda X_1 \ldots \lambda X_{n-1}. \Phi(X_0, X_1, \ldots, X_{n-1}).
\]

**Theorems.**

- All pure \( \lambda \)-terms define *computable* operators.
- If \( \Phi(X) \) is continuous and we let \( \nabla = \lambda X. \Phi(X(X)) \), then \( P = \nabla (\nabla) \) is the *least fixed point* of \( \Phi \).
- The least fixed point of a *computable* operator is always computable.

\[
\text{Succ}(X) = \{n+1 \mid n \in X\}, \quad \text{Pred}(X) = \{n \mid n+1 \in X\}, \quad \text{and}
\]

\[
\text{Test}(Z)(X)(Y) = \{n \in X \mid 0 \in Z\} \cup \{m \in Y \mid \exists k . k+1 \in Z\},
\]

with \( \lambda \)-calculus, suffice for defining all RE sets.
**Gödel Numbering**

**Lemma.** There is a computable $V = \lambda x. V(x)$ where

(i) $V(\{0\}) = \lambda y. \lambda x. y$,
(ii) $V(\{1\}) = \lambda z. \lambda y. \lambda x. z(x)(y(x))$,
(iii) $V(\{2\}) = \text{Test}$,
(iv) $V(\{3\}) = \text{Succ}$,
(v) $V(\{4\}) = \text{Pred}$, and
(vi) $V(\{4 + (n, m)\}) = V(\{n\}) V(\{m\})$.

**Theorem.** Every *recursively enumerable set* is of the form $V(\{n\})$.

**Definition.** Modify the definition of $V$ via *finite approximations*:

(i) $V_k(\{n\}) = V(\{n\}) \cap \{i \mid i < k\}$ for $n < 5$, and
(ii) $V_k(\{4 + (n,m)\}) = V_k(\{n\}) V_k(\{m\})$.

**Theorem.** Each $V_k(\{n\}) \subseteq V_{k+1}(\{n\})$ is *finite*,
the predicate $j \in V_k(\{n\})$ is *recursive*, and we have:

$V(\{n\}) = \bigcup_{k < \infty} V_k(\{n\})$.

**Theorem.** The sets $\mathcal{L}_0$ and $\mathcal{L}_1$ are *recursively enumerable*,
*disjoint*, and *recursively inseparable*:

$\mathcal{L}_0 = \{n \mid \exists j \ [0 \in V_j(\{n\})(\{n\}) \land 1 \notin V_j(\{n\})(\{n\})]\}$
$\mathcal{L}_1 = \{n \mid \exists k \ [1 \in V_k(\{n\})(\{n\}) \land 0 \notin V_k(\{n\})(\{n\})]\}$
How to Randomize?

Definition. By a **random variable** we mean a function

$$X : [0, 1] \to \mathcal{P}(\mathbb{N}),$$

where, for $n \in \mathbb{N}$, the set

$$\{ t \in [0, 1] \mid n \in X(t) \}$$

is always **Lebesgue measurable**.

Definition. For **random variables** $X, Y : [0, 1] \to \mathcal{P}(\mathbb{N})$,

$$[X \subseteq Y] = \{ t \in [0, 1] \mid \forall n \in X(t). n \in Y(t) \}/\text{Null}.$$

Theorem. The random variables over $\mathcal{P}(\mathbb{N})$ form

a **Boolean-valued model** for the $\lambda$-calculus —

expanding the two-valued model $\mathcal{P}(\mathbb{N})$.

- This last definition is the beginning of putting

  a **Boolean-valued Logic** on random variables

  using the complete Boolean algebra of

  measurable sets **modulo** sets of measure zero.

**NOTE:** This new model gives us a programming language with randomized parameters.
Randomized Coin Tossing

**Definition.** A *coin flip* is a random variable \( F : [0,1] \rightarrow \{\{0\}, \{1\}\} \),

It is *fair* iff \( \mu[F = \{0\}] = \frac{1}{2} \).

**Definition.** *Pairing functions* for sets in \( P(\mathbb{N}) \) can be defined by these enumeration operators:

\[
\text{Pair}(X)(Y) = \{2n \mid n \in X\} \cup \{2m+1 \mid m \in Y\}
\]

\[
\text{Fst}(Z) = \{n \mid 2n \in Z\} \text{ and } \text{Snd}(Z) = \{m \mid 2m + 1 \in Z\}.
\]

**Definition.** A *tossing process* is a random variable \( T \) where \( \text{Fst}(T) \) is a fair coin flip and where \( \text{Snd}(T) \) is another tossing — with the successive flippings all being *mutually independent*.

The problem with using a coin-tossing process \( T \) is that once \( \text{Fst}(T) \) has been looked at, then that toss should be discarded, and only the coins from \( \text{Snd}(T) \) should be used in the future.
A Prototype Algorithm Language

Perhaps a solution is always to evaluate programs in the order in which expressions are written. Let's try a very sparse language.

\[ V_i \] — a variable

\[ M(N) \] — an application

\[ \lambda V_i. M \] — an abstraction

\[ M \oplus N \] — a stochastic choice

Let \[ V_i = M \] in \[ N \] — a direct valuation

The idea here is that the text \[ M \] is evaluated in an environment giving the values of free variables. Then the result is passed on to a continuation. In case a random choice is needed, the tossing process is called.

We will try to employ a continuation semantics where the denotation of a program uses the \[ \lambda \]-calculus formulation:

\[ \langle M \rangle (\text{env})(\text{cont})(\text{toss}) \]
The Semantical Equations

- $\langle V_i \rangle (E)(C)(T) =$
  $C(E(\{i\}))(T)$

- $\langle M(N) \rangle (E)(C)(T) =$
  $\langle M \rangle (E)(\lambda x. \langle N \rangle (E)(\lambda y. C(X(Y))))(T)$

- $\langle \lambda V_i . M \rangle (E)(C)(T) =$
  $C(\lambda x. \langle M \rangle (E[X/{i}]))(T)$

- $\langle M \oplus N \rangle (E)(C)(T) =$
  $\text{Test} (\text{Fst}(T)) (\langle M \rangle (E))(\langle N \rangle (E))(C)(\text{Snd}(T))$

- $\langle \text{Let } V_i = M \text{ in } N \rangle (E)(C)(T) =$
  $\langle N \rangle (E[\langle M \rangle (E)/\{i\}]/(C))(T)$

**Running a (closed) program means evaluating:**

$\langle M \rangle (\emptyset)(\lambda x. \lambda y. x)(T)$

The semantics and model as presented here, however, are only sketches. Examples of randomized algorithms need to be worked out, as well as good methods of proving probabilistic properties of programs.
Some Background References

There are many approaches to modeling $\lambda$-calculus, and expositions and historical references can be found in Cardone-Hindley [2009]. In 1972 Plotkin wrote an AI report at the University of Edinburgh entitled "A set-theoretical definition of application" which remained unpublished until it was incorporated into the more extensive paper Plotkin [1993], which discusses many kinds of models. Scott developed his model based on the powerset of the integers subsequently, but he only later realized it was basically the same as Plotkin's model. See Scott [1976] for further details where he called the idea The Graph Model.


Much earlier, enumeration reducibility was introduced by Rogers in lecture notes and mentioned by Friedberg-Rogers [1959] as a way of defining a positive reducibility between sets. Enumeration degrees are discussed at length in Rogers [1967]. There is now a vast literature on the subject. Enumeration operators are also studied in Rogers [1967] as well. Earlier, Myhill-Shepherdson [1955] defined functionals on partial functions with similar properties. Neither team saw that their operators possessed an algebra that would model $\lambda$-calculus, however.


More Background References

Some historical remarks on the notion of partial equivalence relations (PERs) as an interpretation of types are given by Bruce et al. [1990], where we learn that they were introduced by Myhill and Shepherdson [1955] for types of first-order functions, and then extended to simple types by Kreisel [1959]. Scott took the use of partial equivalence relations from the work of Kreisel and collaborators.


Two papers about introducing random features in λ-calculus are Deliguoro-Piperno [1995] and Dal Lago-Zorzia [2012]. Both of those articles have many historical references. Thanks to Thomas F. Icard III for pointing out these two references in connection with work on his Stanford Ph.D. thesis.


There is a very large literature on probabilistic powerdomains, and many technical details, as well as background and historical references can be found in the recent papers of Michael Mislove (see his WWW site). Connections between random variables and domains of distributions are also explained in these papers.
