Stochastic Lambda-Calculus

Dana S. Scott, FBA, FNAS

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

Pidgin Curry?

Combinatory logic is an abstract science dealing with objects called combinators. What their objects are need not be specified; the important thing is how they act upon each other.

One is free to-choose for one's "combinators" **anything one likes** (for example, computer programs). Well, I have chosen **birds** for my combinators — motivated, no doubt, by the memory of the late **Professor Haskell Curry**, who was both a great combinatory logician and an avid bird-watcher.

The main reason I chose combinatory logic for the central theme of this book was not for its practical applications, of which there are many, but for its great entertainment value. Here is a field considered highly technical, yet perfectly available to the general public; it is chock-full of material from which one can cull excellent recreational puzzles, and at the same time it ties up with fundamental issues in modem logic.

What could be better for a puzzle book? (Preface, p. x.)

Raymond M. Smullyan. ***To Mock a Mockingbird and Other Logic Puzzles***
Alfred A. Knopf, 1985, x + 256 pp.

Some Other Quotations

There is, however, one feature that I would like to suggest should be incorporated in the machines, and that is a ***random element***.

– Alan Turing, ***Intelligent Machinery, A Heretical Theory***

83. What is the difference between a Turing machine and the modern computer? It's the same as that between Hillary's ascent of Everest and the establishment of a Hilton hotel on its peak.

60. Dana Scott is the Church of the Lattice-Way Saints.

30. Simplicity does not precede complexity, but follows it.

– Alan Perlis, ***Epigrams on Programming***

Church's λ-Calculus

Definition. λ-calculus — as a formal theory — has rules for the ***explicit definition*** of functions *via* well known equational axioms:

α-conversion

$$\lambda X.[...X...] = \lambda Y.[...Y...]$$

β-conversion

$$(\lambda X.[...X...])(T) = [...T...]$$

η-conversion

$$\lambda X.F(X) = F$$

NOTE: The third axiom will be dropped in favor of a theory employing properties of a partial ordering.

The Enumeration Operator Model

Definitions. (1) ***Pairing:*** $(n,m) = 2^n(2m+1)$.

(2) ***Sequence numbers:*** $\langle\rangle = 0$ and

$\langle n_0, n_1, \ldots, n_{k-1}, n_k\rangle = (\langle n_0, n_1, \ldots, n_{k-1}\rangle, n_k)$.

(3) ***Sets:*** $\mathbf{set}(0) = \varnothing$ and $\mathbf{set}((n,m)) = \mathbf{set}(n) \cup \{m\}$.

(4) ***Kleene star:*** $X^* = \{n \mid \mathbf{set}(n) \subseteq X\}$, for sets $X \subseteq \mathbb{N}$.

Definition. The ***model*** is given by these definitions on the ***powerset*** of the set integers, $\mathcal{P}(\mathbb{N})$:

Application:

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

Abstraction:

$$\lambda X.[\ldots X\ldots] =$$

$$\{0\} \cup \{ (n,m) \mid m \in [\ldots \mathbf{set}(n) \ldots] \}$$

What is the Secret?

(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 $\Phi : \mathcal{P}(\mathbb{N})^n \rightarrow \mathcal{P}(\mathbb{N})$ are ***continuous*** iff, for all $m \in \mathbb{N}$, we have $m \in \Phi(X_0, X_1, \ldots, X_{n-1})$ iff there are $k_i \in X_i^*$ for each of the $i<n$, such that $m \in \Phi(\mathbf{set}(k_0), \mathbf{set}(k_1), \ldots, \mathbf{set}(k_{n-1}))$.

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

(4) If the function $\Phi(X_0, X_1, \ldots, X_{n-1})$ is continuous, then the abstraction term $\lambda X_0 . \Phi(X_0, X_1, \ldots, X_{n-1})$ 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 $F(T) = \Phi(T)$. And, therefore, generally $F \subseteq \lambda X . F(X)$.

NOTE: This model could easily have been defined in 1957!!
It clearly satisfies the rules of α, β-conversion (but not η).

This Lecture is Dedicated to the Memories of

John R. Myhill
Born: 11 August 1923, Birmingham, UK
Died: 15 February 1987, Buffalo, NY

John Shepherdson
Born: 7 June 1926, Huddersfield, UK
Died: 8 January 2015, Bristol, UK

Hartley Rogers, Jr.
Born: 6 July, 1926, Buffalo, NY
Died: 17 July, 2015, Waltham, MA

- John Myhill and John C. Shepherdson, *Effective operations on partial recursive functions*, **Zeitschrift für Mathematische Logik und Grundlagen der Mathematik**, vol. 1 (1955), pp. 310-317.

- Richard M. Friedberg and Hartley Rogers Jr., *Reducibility and completeness for sets of integers,* **Mathematical Logic Quarterly**, vol. 5 (1959), pp. 117-125. Some earlier results are presented in an abstract in **The Journal of Symbolic Logic**, vol. 22 (1957), p. 107.

- Hartley Rogers, Jr., **Theory of Recursive Functions and Effective Computability**, McGraw-Hill, 1967, xix + 482 pp.

Some Lambda Properties

Theorem. For all sets of integers F and G we have:

$$\lambda X.F(X) \subseteq \lambda X.G(X) \text{ iff } \forall X.F(X) \subseteq G(X),$$

$$\lambda X.(F(X) \cap G(X)) = \lambda X.F(X) \cap \lambda X.G(X),$$

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**:

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

How to do Recursion?

Three Basic Theorems.

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

A Principal Theorem. **These computable operators:**

$$\textbf{Succ}(X)=\{n+1 \mid n \in X\},$$

$$\textbf{Pred}(X)=\{n \mid n+1 \in X\}, \textbf{ and}$$

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

together with λ-calculus, suffice for defining all RE sets.

Gödel Numbering

Theorem. There is a computable **V** = λ`X`. **V**`(X)` where

(i) **V**`({0})` = λ`Y.`λ`X.Y`,

(ii) **V**`({1})` = λ`Z.`λ`Y.`λ`X.Z(X)(Y(X))`,

(iii) **V**`({2})` = **Test**,

(iv) **V**`({3})` = **Succ**,

(v) **V**`({4})` = **Pred**, and

(vi) **V**`({4+(n,m)})` = **V**`({n})(`**V**`({m}))`.

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

NOTE: The operator V is the analogue of the Universal Turing Machine.

Inseparable Sets?

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

(i) $\mathbf{V}_{k}(\{n\}) = \mathbf{V}(\{n\}) \cap \{i \mid i < k\}$ for $n < 5$, and

(ii) $\mathbf{V}_{k}(\{4+(n,m)\}) = \mathbf{V}_{k}(\{n\})(\mathbf{V}_{k}(\{m\}))$.

Theorem. Each $\mathbf{V}_{k}(\{n\}) \subseteq \mathbf{V}_{k+1}(\{n\})$ is ***finite***,

the predicate $j \in \mathbf{V}_{k}(\{n\})$ is ***recursive***,

and we have:

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

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

$$\mathcal{L}_0 = \{n \mid \exists\, j\, [0 \in \mathbf{V}_{j}(\{n\})(\{n\}) \wedge 1 \notin \mathbf{V}_{j}(\{n\})(\{n\})]\}$$

$$\mathcal{L}_1 = \{n \mid \exists\, k\, [1 \in \mathbf{V}_{k}(\{n\})(\{n\}) \wedge 0 \notin \mathbf{V}_{k}(\{n\})(\{n\})]\}$$

How to Randomize?

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

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

where, for $n \in \mathbb{N}$, the set $\{ t \in [0,1] \mid n \in \mathbf{X}(t) \}$ is always ***Lebesgue measurable***.

Theorem. The random variables over $\mathcal{P}(\mathbb{N})$ are closed under (pointwise) application and form a model for the $\boldsymbol{\lambda}$-calculus — expanding the original model.

This idea 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. This new model gives us a programming language with randomized parameters.

Randomized Coin Tossing

Definition. A ***coin flip*** is a random variable

$$\mathbf{F}: [0,1] \longrightarrow \{\{0\},\{1\}\},$$

It is ***fair*** iff $\mu[\![\, \mathbf{F} = \{0\} \,]\!] = 1/2$.

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

$$\mathbf{Pair}(X)(Y)=\{2n \mid n \in X\} \cup \{2m+1 \mid m \in Y\}$$

$\mathbf{Fst}(Z)=\{n \mid 2n \in Z\}$ and $\mathbf{Snd}(Z)=\{m \mid 2m+1 \in Z\}$.

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

The problem with using a coin-tossing process $\mathbf{T}$ in an algorithm is that once $\mathrm{Fst}(\mathbf{T})$ has been looked at, then that toss should be discarded, and only the new coins from $\mathrm{Snd}(\mathbf{T})$ 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.

$\mathtt{V_i}$ — a ***variable***

$\mathtt{M(N)}$ — an ***application***

$\lambda \mathtt{V_i.M}$ — an ***abstraction***

$\mathtt{M \oplus N}$ — a ***stochastic choice***

Let $\mathtt{V_i = M}$ **in** $\mathtt{N}$ — 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 λ-calculus formulation:

$$\langle\!| \mathbf{M} |\!\rangle(\mathbf{env})(\mathbf{cont})(\mathbf{toss})$$

The Semantical Equations

- $\llbracket V_i \rrbracket(E)(C)(T) = C(E(\{i\}))(T)$
- $\llbracket M(N) \rrbracket(E)(C)(T) = \llbracket M \rrbracket(E)(\lambda X.\llbracket N \rrbracket(E)(\lambda Y.C(X(Y))))(T)$
- $\llbracket \lambda V_i.M \rrbracket(E)(C)(T) = C(\lambda X.\llbracket M \rrbracket(E[X/\{i\}]))(T)$
- $\llbracket M \oplus N \rrbracket(E)(C)(T) = \mathbf{Test}(\mathbf{Fst}(T))(\llbracket M \rrbracket(E))(\llbracket N \rrbracket(E))(C)(\mathbf{Snd}(T))$
- $\llbracket \mathbf{Let}\ V_i = M\ \mathbf{in}\ N \rrbracket(E)(C)(T) = \llbracket N \rrbracket(E[\llbracket M \rrbracket(E)/\{i\}])(C)(T)$

Running a (closed) program means evaluating:

$$\llbracket M \rrbracket(\varnothing)(\lambda X.\lambda Y.X)(T)$$

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.

An Absoluteness Theorem

Theorem. If a closed program has a ***non-random*** value, then the value is the same for all tossing processes.

Proof Idea: **Working within Boolean-valued logic over the measure algebra of Lebsegue sets modulo sets of measure zero, all tossing processes are the same up to a measure-preserving automorphism of the measure algebra.**

A PLEA FOR HELP !

Let's find some good
applications for this model
with random variables!