Stochastic λ-Calculi

Dana S. Scott

University Professor, Emeritus
Carnegie Mellon University

Visiting Scholar in Mathematics
University of California, Berkeley

BLAST 2013
Chapman University

5–9 August 2013

Church's λ-Calculus

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

α-*conversion*

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

β-*conversion*

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

η-*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.

The Graph Model

Definition. Define: The pairing $(n,m) = 2^n(2m+1)$.

Define: Sequence numbers $\langle\rangle = 0$ and

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

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

Define: $X* = \{ n \mid$ **set**$(n) \subseteq X \}$ for sets X of integers.

Definition. The *enumeration operator model* is given by these definitions on *sets* of integers:

Application

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

Abstraction

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

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

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

(Some historical comments can be found at the end of these notes.)

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 $\Phi : \mathcal{P}(\mathbb{N})^n \to \mathcal{P}(\mathbb{N})$ 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 $m \in \Phi(\mathsf{set}(k_0), \ldots, \mathsf{set}(k_{n-1}))$.

(3) The application operation $\mathsf{F(X)}$ 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 $\mathsf{F(T)} = \Phi(\mathsf{T})$.

(6) And, note, therefore, that generally $\mathsf{F} \subseteq \lambda X . \mathsf{F(X)}$.

Some Lattice 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:

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

- All pure λ-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 Φ.

- The least fixed point of a *computable* operator is computable.

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

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

with λ-calculus suffices for defining all RE sets.

How to Randomize?

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

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

where, for $n \in \mathbb{N}$, the set $\{\texttt{t} \in \texttt{[0,1]} \mid n \in \mathbf{X}\texttt{(t)}\}$

is always *Lebesgue measurable*.

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

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

Theorem. The random variables over $\mathcal{P}(\mathbb{N})$ form a *Boolean-valued model* for the *λ*-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

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

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:

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

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

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 $\mathbf{T}$ is that once $\mathbf{Fst}(\mathbf{T})$ has been looked at, then that toss should be discarded, and only the coins from $\mathbf{Snd}(\mathbf{T})$ 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.

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 λ-calculus formulation:

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

The Semantical Equations

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

  $C(E(\{i\}))(T)$

- $\langle M(N) \rangle (E)(C)(T) =$

  $\langle M \rangle (E)(\lambda X. \langle N \rangle (E)(\lambda Y.C(X(Y))))(T)$

- $\langle \lambda V_i.M \rangle (E)(C)(T) =$

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

- $\langle M \oplus N \rangle (E)(C)(T) =$

  Test(**Fst**$(T))(\langle M \rangle (E))(\langle N \rangle (E))(C)($**Snd**$(T))$

- $\langle$ **Let** $V_i = M$ **in** $N \rangle (E)(C)(T) =$

  $\langle N \rangle (E[\langle M \rangle (E)/\{i\}])(C)(T)$

Running a (closed) program means evaluating:

$$\langle M \rangle (\varnothing)(\lambda X.\lambda Y.X)(\mathbf{T})$$

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.

Simulating Automata

Definition. Let § be a suitable RE set where

$$\S(F)(\{0\}) = \lambda x.x \text{ and}$$

$$\S(F)(\{(n,m)\}) = F(\{m\}) \circ \S(F)(\{n\}).$$

Theorem. Let $\Sigma \in \mathcal{P}(\mathbb{N})$ be *finite*, then the

regular languages contained in Σ^* are exactly the

sets of the form $\{\, \sigma \in \Sigma^* \mid 0 \in \S(A)(\{\sigma\})(Q) \,\}$,

where $A, Q \in \mathcal{P}(\mathbb{N})$ are *finite*.

Theorem. Let $\Sigma, Q \in \mathcal{P}(\mathbb{N})$ be *finite*, let **A** be a

finite random variable, and let $\varepsilon \in [0,1]$. Then

the **probabilistic languages** contained in Σ^* are

among the sets of the form

$$\{\, \sigma \in \Sigma^* \mid \mu[\![\, 0 \in \S(\mathbf{A})(\{\sigma\})(Q) \,]\!] > \varepsilon \,\}.$$

- **More analysis is needed as to which random automata define interesting languages.**

The Fuzzy Powerset Model

Definition. Let $\mathcal{I} = [0,1]^{\mathbb{N}}$ be the infinite-dimensional cube.

Definition. Let $E^{(n)}$ enumerate the *rational* vectors in $\mathcal{I}$ with only *finitely many non-zero coordinates*.

Definition. For $X \in \mathcal{I}$, define $E^{(n)} \ll X$ to mean that for all $i \in \mathbb{N}$ with $E^{(n)}_i > 0$, we have $E^{(n)}_i < X_i$.

Theorem. For $X \in \mathcal{I}$, we have $X = \sup\{ E^{(n)} \mid E^{(n)} \ll X \}$.

Definition. The *fuzzy powerset model* is given by these definitions on *infinite-dimensional vectors*:

Application

$$F(X)_m = \sup\{ F_{(n,m)} \mid E^{(n)} \ll X \}$$

Abstraction

$$(\lambda X.[\ldots X \ldots])_0 \quad = 1$$

$$(\lambda X.[\ldots X \ldots])_{(n,m)} = [\ldots E^{(n)} \ldots]_m$$

This model satisfies α, β-conversion (but not η). And random elements can be added just as with the $\mathcal{P}(\mathbb{N})$ model.

What is a Type?

Definition. Recall that for X,Y ∈ ℙ we write

Pair(X)(Y)=(X,Y)={2n│n ∈ X}∪{2m+1│m ∈ Y},

Fst(Z)={n│2n ∈ Z}, and

Snd(Z)={m│2m+1 ∈ Z}.

So we regard ℙ = ℙ × ℙ, and for $\mathcal{A}$ ⊆ ℙ we write

X $\mathcal{A}$ Y iff (X,Y) ∈ $\mathcal{A}$.

Definition. By a ***type*** over ℙ we understand
a ***partial equivalence relation*** $\mathcal{A}$ ⊆ ℙ where,
for all X,Y,Z ∈ ℙ, we have
X $\mathcal{A}$ Y implies Y $\mathcal{A}$ X, and
X $\mathcal{A}$ Y and Y $\mathcal{A}$ Z imply X $\mathcal{A}$ Z.
Additionally we write X:$\mathcal{A}$ iff X $\mathcal{A}$ X.

Note: It is better NOT to pass to equivalence classes
and the quotient spaces. But we can THINK
in those terms if we like.

Definition. For subspaces 𝕏 ⊆ ℙ write

[𝕏] = {(X,X)│ X ∈ 𝕏 },

so that we may regard ***subspaces*** as ***types***.

The Category of Types

Definition. The ***product*** of types $\mathcal{A}, \mathcal{B} \subseteq \mathbb{P}$ is defined as that relation where $X(\mathcal{A} \times \mathcal{B})Y$ iff **Fst**$(X) \mathcal{A}$ **Fst**(Y) and **Snd**$(X) \mathcal{B}$ **Snd**(Y).

Theorem. The product of two types is again a type, and we have
$X:(\mathcal{A} \times \mathcal{B})$ iff **Fst**$(X):\mathcal{A}$ and **Snd**$(X):\mathcal{B}$

Definition. The ***exponentiation*** of types $\mathcal{A}, \mathcal{B} \subseteq \mathbb{P}$ is that relation where $F(\mathcal{A} \to \mathcal{B})G$ iff
$\forall X,Y.\ X \mathcal{A} Y$ implies $F(X) \mathcal{B} G(Y)$.

Theorem. The exponentiation (= function space) of two types is again a type, and we have
if $F:\mathcal{A} \to \mathcal{B}$ then $\forall\ X.\ X:\mathcal{A}$ implies $F(X):\mathcal{B}$.

Note: Types do form a category — expanding the topological category of subspaces — but we wish to prove much, much more.

Isomorphism of Types

Definition. The **sum** of types $\mathcal{A}, \mathcal{B} \subseteq \mathbb{P}$ is defined as that relation where $X(\mathcal{A} + \mathcal{B})Y$ iff either $\exists X_0, Y_0 [X_0 \mathcal{A} Y_0 \ \& \ X = (0, X_0) \ \& \ Y = (0, Y_0)]$

or $\exists X_1, Y_1 [X_1 \mathcal{B} Y_1 \ \& \ X = (1, X_1) \ \& \ Y = (1, Y_1)]$.

Theorem. The sum of two types is again a type, and we have
$X : (\mathcal{A} + \mathcal{B})$ iff either **Fst**$(X) = 0$ & **Snd**$(X) : \mathcal{A}$
or **Fst**$(X) = 1$ & **Snd**$(X) : \mathcal{B}$.

Definition. Two types $\mathcal{A}, \mathcal{B} \subseteq \mathbb{P}$ are **isomorphic**, in symbols $\mathcal{A} \cong \mathcal{B}$, provided there are

$F : \mathcal{A} \to \mathcal{B}$ and $G : \mathcal{B} \to \mathcal{A}$ where

$\forall X : \mathcal{A}. \ X \ \mathcal{A} \ G(F(X))$ and $\forall Y : \mathcal{B}. \ Y \ \mathcal{B} \ F(G(Y))$.

Theorem. If types $\mathcal{A}_0 \cong \mathcal{B}_0$ and $\mathcal{A}_1 \cong \mathcal{B}_1$, then

$(\mathcal{A}_0 \times \mathcal{A}_1) \cong (\mathcal{B}_0 \times \mathcal{B}_1)$, and

$(\mathcal{A}_0 + \mathcal{A}_1) \cong (\mathcal{B}_0 + \mathcal{B}_1)$, and

$(\mathcal{A}_0 \to \mathcal{A}_1) \cong (\mathcal{B}_0 \to \mathcal{B}_1)$.

Some Categorical Properties

Definition. Let $\mathcal{T}$ be the class of all types on the powerset space $\mathbb{P}$.

Theorem. Isomorphism is is an *equivalence relation* on $\mathcal{T}$, and for all $\mathcal{A}, \mathcal{B}, \mathcal{C} \in \mathcal{T}$,

$(\mathcal{A} \times \mathcal{B}) \cong (\mathcal{B} \times \mathcal{A})$, and $(\mathcal{A} + \mathcal{B}) \cong (\mathcal{B} + \mathcal{A})$, and

$((\mathcal{A} \times \mathcal{B}) \times \mathcal{C}) \cong (\mathcal{A} \times (\mathcal{B} \times \mathcal{C}))$, and

$((\mathcal{A} + \mathcal{B}) + \mathcal{C}) \cong (\mathcal{A} + (\mathcal{B} + \mathcal{C}))$, and

$(\mathcal{A} \times (\mathcal{B} + \mathcal{C})) \cong (\mathcal{A} \times \mathcal{B}) + (\mathcal{A} \times \mathcal{C})$, and

$((\mathcal{A} \times \mathcal{B}) \to \mathcal{C}) \cong (\mathcal{A} \to (\mathcal{B} \to \mathcal{C}))$, and

$(\mathcal{A} \to (\mathcal{B} \times \mathcal{C}) \cong (\mathcal{A} \to \mathcal{B}) \times (\mathcal{A} \to \mathcal{C}))$, and

$((\mathcal{A} + \mathcal{B}) \to \mathcal{C}) \cong (\mathcal{A} \to \mathcal{C}) \times (\mathcal{B} \to \mathcal{C})$.

Theorem. The types on the powerset space $\mathbb{P}$ form a *bi-cartesian closed category*, and the isomorphism classes of types satisfy all the usual laws of *addition*, *multiplication*, and *exponentiation*.

Dependent Types

Definition. Given $\mathcal{A} \in \mathcal{T}$, an ***$\mathcal{A}$-indexed family of types*** is a function $\mathcal{B} : \mathbb{P} \to \mathcal{T}$, such that

$$\forall X_0, X_1. \; X_0 \, \mathcal{A} \, X_1 \text{ implies } \mathcal{B}(X_0) = \mathcal{B}(X_1).$$

Definition. The ***dependent product*** of an ***$\mathcal{A}$-indexed family of types***, $\mathcal{B}$, is defined as that relation such that

$$F_0(\textstyle\prod X : \mathcal{A}. \mathcal{B}(X)) F_1 \text{ iff}$$

$$\forall X_0, X_1. \; X_0 \, \mathcal{A} \, X_1 \text{ implies } F_0(X_0) \, \mathcal{B}(X_0) \, F_1(X_1).$$

Definition. The ***dependent sum*** of an ***$\mathcal{A}$-indexed family of types***, $\mathcal{B}$, is defined as that relation such that

$$Z_0(\textstyle\sum X : \mathcal{A}. \mathcal{B}(X)) Z_1 \text{ iff}$$

$$\exists X_0, Y_0, X_1, Y_1 [\, X_0 \mathcal{A} X_1 \; \& \; Y_0 \mathcal{B}(X_0) Y_1 \; \& \;$$

$$Z_0 = (X_0, Y_0) \; \& \; Z_1 = (X_1, Y_1) \,]$$

Theorem. The dependent products and dependent sums of indexed families of types are again types.

Systems of Dependent Types

Definition. We say that $\mathcal{A},\mathcal{B},\mathcal{C},\mathcal{D}$ form *a system of dependent types* iff

- $\forall X_0,X_1.[X_0 \,\mathcal{A}\, X_1 \Rightarrow \mathcal{B}(X_0) = \mathcal{B}(X_1)]$, and

- $\forall X_0,X_1,Y_0,Y_1.[X_0 \,\mathcal{A}\, X_1 \,\&\, Y_0 \,\mathcal{B}(X_0)\, Y_1 \Rightarrow$

$\mathcal{C}(X_0,Y_0) = \mathcal{C}(X_1,Y_1)]$, and

- $\forall X_0,X_1,Y_0,Y_1,Z_0,Z_1.[X_0 \,\mathcal{A}\, X_1 \,\&\, Y_0 \,\mathcal{B}(X_0)\, Y_1 \,\&$

$Z_0 \,\mathcal{C}(X_0,Y_0)\, Z_1 \Rightarrow \mathcal{D}(X_0,Y_0,Z_0) = \mathcal{D}(X_1,Y_1,Z_1)]$,

provided that $\mathcal{A} \in \mathcal{T}$, and $\mathcal{B},\mathcal{C},\mathcal{D}$ are functions on $\mathbb{P}$ to $\mathcal{T}$ of the indicated number of arguments.

Note: Clearly the definition can be extended to systems of any number of terms.

Theorem. Under the above assumptions on $\mathcal{A},\mathcal{B},\mathcal{C},\mathcal{D}$, we always have

$$\prod X:\mathcal{A}.\sum Y:\mathcal{B}(X).\prod Z:\mathcal{C}(X,Y).\,\mathcal{D}(X,Y,Z) \in \mathcal{T}.$$

Asserting Propositions

Definition. Every type $\mathcal{P} \in \mathcal{T}$ can be regarded as a **proposition** where **asserting** (or **proving** $\mathcal{P}$) means finding **evidence** E:$\mathcal{P}$.

Note: Under this interpretation of logic, asserting $(\mathcal{P} \times \mathcal{Q})$ means asserting a conjunction, asserting $(\mathcal{P} + \mathcal{Q})$ means asserting a disjunction, asserting $(\mathcal{P} \to \mathcal{Q})$ means asserting an implication, asserting $(\prod X:\mathcal{A}.\mathcal{P}(X))$ means asserting a universal quantification, and asserting $(\sum X:\mathcal{A}.\mathcal{B}(X))$ means asserting an existential quantification.

Definition. For $\mathcal{A} \in \mathcal{T}$ the **identity type** on $\mathcal{A}$ is defined as that relation such that Z$(X \equiv_\mathcal{A} Y)$W iff Z $\mathcal{A}$ X $\mathcal{A}$ Y $\mathcal{A}$ W.

Example: Given F:$(\mathcal{A} \to (\mathcal{A} \to \mathcal{A}))$, then asserting $\prod X:\mathcal{A}.\prod Y:\mathcal{A}.\prod Z:\mathcal{A}.$ F(X)(F(Y)(Z)) $\equiv_\mathcal{A}$ F(F(X)(Y))(Z) is the same as asserting that F is an associative operation.

Some Background References

There are many approaches to modeling λ-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.

• F. Cardone and J.R. Hindley. Lambda-Calculus and Combinators in the 20th Century. In: Volume 5, pp. 723-818, of Handbook of the History of Logic, Dov M. Gabbay and John Woods eds., North-Holland/Elsevier Science, 2009.

• Gordon D. Plotkin. Set-theoretical and other elementary models of the λ-calculus. Theoretical Computer Science, vol. 121 (1993), pp. 351-409.

• Dana S. Scott. Data types as lattices. SIAM Journal on Computing, vol. 5 (1976), pp. 522-587.

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 λ-calculus, however.

• 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 of the results of this paper are presented in abstract, 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.

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.

• K. Bruce, A. A. Meyer, and J. C. Mitchell. The semantics of second-order lambda calculus. In G. Huet, editor. Logical Foundations of Functional Programming, pp. 273–284. Addison-Wesley, 1990.

• G. Kreisel. Interpretation of analysis by means of constructive functionals of finite type. In A. Heyting, editor, Constructivity in Mathematics, pp. 101–128. North-Holland Co., Amsterdam, 1959.

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.

• U. Deliguoro and A. Piperno. Nondeterministic Extensions of Untyped λ-Calculus. Information and Computation, vol. 122 (1995), pp. 149–177.

• Ugo Dal Lago and Margherita Zorzia. Probabilistic operational semantics for the lambda calculus. RAIRO - Theoretical Informatics and Applications, vol. 46 (2012), pp. 413-450.

The author is very much indebted to Thomas F. Icard III for pointing out these last two references in connection with work on his Stanford Ph.D. thesis.