λ-Calculus: Then & Now

Dana S. Scott
University Professor Emeritus
Carnegie Mellon University
Visiting Scholar
University of California, Berkeley
dana.scott@cs.cmu.edu

TURING CENTENNIAL CELEBRATION
Princeton University, May 10-12, 2012

ACM TURING CENTENARY CELEBRATION
San Francisco, June 15-16, 2012

UC BERKELEY LOGIC COLLOQUIUM
Berkeley, August 24, 2012

Notes derived from the slides presented at the conferences.
A brief amount of text has been added for continuity.
The author would be happy to hear reactions and suggestions.
Version of 17 November 2013
Symbols of Princeton

Traditional          From the Graduate Alumni
                    (to encourage ecology)

The \( \lambda \)-calculus was begun
at Princeton, and the purpose of this
report is to show how it has been
recycled every decade after the 1930s
in new and useful ways.

**WARNING:** We cannot give here a complete history of Mathematical Logic and related areas. The present report may even have too much detail. But it is hoped readers might be encouraged to look further.
A Quick Look Back to Beginnings

1870s
Begriffsschrift                      Frege (1879)

1880s
What are numbers?                   Dedekind (1888)
Number-theoretic axioms             Peano (1889)

1890s
Vorlesungen über die Algebra der Logik Schröder (1890–1905)
Grundgesetze der Arithmetik         Frege (1893-1903)
Formulario Mathematico              Peano (1895-1901)
Grundlagen der Geometrie            Hilbert (1899)

1900s
Diophantine problem                 Hilbert (1900)
Russell's Paradox                   Russell (1901)
Principles of Mathematics           Russell (1903)
Richard's Paradox                   Richard (1905)
Theory of Types                     Russell (1908)

1910s
Principia Mathematica               Whitehead-Russell (1910-12-13)
Calculus of relatives               Löwenheim (1915)

WW I

1920s
Löwenheim-Skolem Theorem            Skolem (1920)
Propositional calculus completeness  Post (1921)
Monadic predicate calculus decidable Behmann (1922)
Abstract proof rules                Hertz (1922)
Primitive recursive arithmetic      Skolem (1923)
Combinators                        Schönfinkel (1924)
Function-based set theory           von Neumann (1925)
"Conceptual" undecidability         Finsler (1926)
Epsilon operator                   Hilbert-Bernays (1927)
Combinators (again)                 Curry (1927)
Ackermann function                 Ackermann (1928)
Entscheidungsproblem               Hilbert-Ackermann (1928)
Abriss der Logistik & simple type theory Carnap (1929)

It was very reasonable for Hilbert and Ackermann to emphasize
the Decision Problem, as special cases had been solved.
Church vs. Turing

Alonzo Church
Born: 14 June 1903 in Washington, D.C., USA.
Died: 11 Aug 1995 in Hudson, Ohio, USA.
Ph.D.: Princeton University, 1927, USA.

Alan Turing
Died: 7 June 1954, Wilmslow, Cheshire, UK.
Ph.D.: Princeton University, 1938, USA.


The work of Church and Turing in 1936 was done independently.
Three Pioneers

Haskell Brooks Curry

Born: 12 Sept 1900 in Millis, MA, USA.
Died: 1 Sept 1982 in State College, PA, USA.
Ph.D.: Göttingen Universität, 1930, Germany.
Thesis: Grundlagen der kombinatorischen Logik

Stephen Cole Kleene

Born: 5 Jan 1909 in Hartford, CN, USA.
Died: 25 Jan 1994 in Madison, WI, USA.
Ph.D.: Princeton University, 1934, USA.
Thesis: A Theory of Positive Integers in Formal Logic

J. Barkley Rosser

Born: 6 Dec 1907 in Jacksonville, FL, USA.
Died: 5 Sept 1989 in Madison, WI, USA.
Ph.D.: Princeton University, 1934, USA.
Thesis: A Mathematical Logic without Variables

It seems, sadly, that Alan Turing never had a chance to meet these people or Kurt Gödel.
## 1930s

<table>
<thead>
<tr>
<th>Topic</th>
<th>Author/Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combinatory logic</td>
<td>Curry (1930-32)</td>
</tr>
<tr>
<td>Herbrand's Theorem</td>
<td>Herbrand (1930)</td>
</tr>
<tr>
<td>Completeness proof</td>
<td>Gödel (1930)</td>
</tr>
<tr>
<td>Partial consistency proof</td>
<td>Herbrand (1931)</td>
</tr>
<tr>
<td>Incompleteness</td>
<td>Gödel (1931)</td>
</tr>
<tr>
<td>Untyped $\lambda$-calculus</td>
<td>Church (1932-33-41)</td>
</tr>
<tr>
<td>Studies of primitive recursion</td>
<td>Péter (1932-36)</td>
</tr>
<tr>
<td>Non-standard models</td>
<td>Skolem (1933)</td>
</tr>
<tr>
<td>Functionality in Combinatory Logic</td>
<td>Curry (1934)</td>
</tr>
<tr>
<td>Grundlagen der Mathematik</td>
<td>Hilbert-Bernays (1934-39)</td>
</tr>
<tr>
<td>Natural deduction</td>
<td>Gentzen (1934)</td>
</tr>
<tr>
<td>Number-theoretic consistency &amp; $\varepsilon_0$-induction</td>
<td>Gentzen (1934)</td>
</tr>
<tr>
<td>Inconsistency of Church’s System</td>
<td>Kleene-Rosser (1936)</td>
</tr>
<tr>
<td>Confluence theorem</td>
<td>Church-Rosser (1936)</td>
</tr>
<tr>
<td>Finite combinatory processes</td>
<td>Post (1936)</td>
</tr>
<tr>
<td>Turing machines</td>
<td>Turing (1936-37)</td>
</tr>
<tr>
<td>Recursive undecidability</td>
<td>Church-Turing (1936)</td>
</tr>
<tr>
<td>General recursive functions</td>
<td>Kleene (1936)</td>
</tr>
<tr>
<td>Further completeness proofs</td>
<td>Maltsev (1936)</td>
</tr>
<tr>
<td>Improving incompleteness theorems</td>
<td>Rosser (1936)</td>
</tr>
<tr>
<td>Fixed-point combinator</td>
<td>Turing (1937)</td>
</tr>
<tr>
<td>Computability and $\lambda$-definability</td>
<td>Turing (1937)</td>
</tr>
</tbody>
</table>

Starting out with Gödel and ending up with Turing, it would take a long time to comprehend and apply all the developments in this period.
What is the $\lambda$-Calculus?

The calculus gives rules for the explicit definition of functions; however, the type-free version also permits recursion and self-replication.

$\alpha$-conversion

$$\lambda x.[...x...] = \lambda y.[...y...]$$

$\beta$-conversion

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

$\eta$-conversion

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

Church’s original system (1932) also had rules for logic, but that was the system Kleene-Rosser (1936) proved inconsistent!

The names of the rules are due to Curry. The last rule fails in many interpretations, and special efforts are needed to make it valid.
Does $\lambda$-Calculus have Models?

Yes! There is a calculus for enumeration operators! First we need some simple definitions on integers and sets of integers:

$$(n,m) = 2^n(2m+1)$$

$$\text{set}(0) = \emptyset$$

$$\text{set}( (n,m) ) = \text{set}(n) \cup \{m\}$$

$$X^* = \{n \mid \text{set}(n) \subseteq X\}$$

**Application**

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

**Abstraction**

$$\lambda x. [...x...] =$$

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

Every set of integers can be used as an enumeration operator. The operator is computable if the set is r.e. Many compound contexts do define enumeration operators.
The Connection to Computability

**Church Numerals**

\[
\begin{align*}
0 &= \lambda F. \lambda X. X \\
n+1 &= \lambda F. \lambda X. F(n(F)(X)) \\
n+m &= \lambda F. \lambda X. n(F)(m(F)(X)) \\
n \times m &= \lambda F. n(m(F)) \\
m^n &= n(m) \\
n-1 &= \text{[a little harder]}
\end{align*}
\]

**Fixed-Point Combinator**

\[
\begin{align*}
\Lambda &= \lambda F. (\lambda X. F(X(X)))(\lambda X. F(X(X))) \\
\Lambda(F) &= F(\Lambda(F))
\end{align*}
\]

**Theorem.** For every partial recursive function \( g(n) \), there is a **constant \( \lambda \)-term** \( G \) such that

\[
G(n) = g(n), \text{ for all } n.
\]

**Kleene** and **Turing** independently proved this in different ways.

In the **model**, \( G \) denotes an r.e. set.
Some $\lambda$ -Definitions

\[
\begin{align*}
pair &= \lambda X. \lambda Y. \lambda F. F(X)(Y) \\
fst &= \lambda P. P(\lambda X. \lambda Y. X) \\
snd &= \lambda P. P(\lambda X. \lambda Y. Y) \\
succ &= \lambda N. \lambda F. \lambda X. F(N(F)(X)) \\
shft &= \lambda S. \lambda P. \pair(S(fst(P)))(fst(P)) \\
pred &= \lambda N. \snd(N(shft(succ)))(pair(0)(0))
\end{align*}
\]

Kleene's "trick" here is to introduce pairs as a data structure, and then apply iteration to get a sequence of pairs.

\[
\begin{align*}
test &= \lambda N. \lambda U. \lambda V. \snd(N(shft(\lambda X. X))(pair(V)(U))) \\
mult &= \lambda N. \lambda M. \lambda F. N(M(F)) \\
fact &= \lambda N. \test(N)(1)(\mult(N)(\test(\pred(N)))) \\
fact &= Y(\lambda F. \lambda N. \test(N)(1)(\mult(N)(F(\pred(N)))))
\end{align*}
\]

The factorial function must be the most overdefined function in the history of mankind!
Turing’s Only Student

Robin Oliver Gandy

Born: 23 September 1919, Peppard, Oxon., UK.
Ph.D.: Cambridge, 1953.
Supervisor: Alan Turing.
Students: 26 and 126 descendants.

Another pioneer, Gandy, later became a key contributor to the development of Recursive Function Theory.

It is interesting to note that both the teams of Myhill and Shepherdson and, later, Friedberg and Rogers defined enumeration operators without seeing they had models for the $\lambda$-calculus.
Church-Turing Thesis
accepted with the help of Kleene
after Turing explained his machines.

Effectively computable functions
of natural numbers can be identified with
those definable by:

• $\lambda$-calculus
• Herbrand-Gödel equations
• Partial-recursive schemata
• Turing-Post machine programs

If Gödel had stayed in Princeton, and
If Church and Kleene had argued better
for data structures in the $\lambda$-calculus,
Then surely Gödel would have accepted
$\lambda$-calculus as a foundation much earlier.

Note that Kleene proved the equivalence with
Herbrand-Gödel computability before Turing’s work.
Kleene’s Complaint

I myself, perhaps unduly influenced by rather chilly receptions from audiences around 1933-35 to disquisitions on \(\lambda\)-definability, chose, after \textit{general recursiveness} had appeared, to put my work in that format. I did later publish one paper 1962 on \(\lambda\)-definability in higher recursion theory.

I thought general recursiveness came the closest to \textit{traditional mathematics}. It spoke in a language familiar to mathematicians, extending the theory of \textit{special recursiveness}, which derived from formulations of Dedekind and Peano in the mainstream of mathematics.

I cannot complain about my audiences after 1935, although whether the improvement came from switching I do not know. In retrospect, I now feel it was too bad I did not keep active in \(\lambda\)-definability as well. So I am glad that interest in \(\lambda\)-definability has revived, as illustrated by Dana Scott’s 1963 communication.

Were the truth to be known, Kleene translated much of what he had done in \(\lambda\)-calculus into working with integers. Indeed, the \textit{application operation} \{e\}(n) defines a \textit{partial combinatory algebra} with many properties similar to the work of Curry and Rosser.
What is the Entscheidungsproblem?

To determine whether a formula of the first-order predicate calculus is provable or not.

Church’s Solution

Theorem. Only a finite number of axioms are needed to define a non-recursive set of integers.

R.M. Robinson’s Arithmetic

(1) $\forall x \forall y \ [ x = y \iff Sx = Sy ]$

(2) $\forall x \ [ x = 0 \iff \neg \exists y. x = Sy ]$

(3) $\forall x \forall y \ [ (x + 0) = x \ & \ (x + Sy) = S(x + y) ]$

(4) $\forall x \forall y \ [ (x \times 0) = 0 \ & \ (x \times Sy) = ((x \times y) + x) ]$

After the solution of Hilbert’s 10th Problem, the applicability of this theory became even easier.
**Turing's Solution**

**Theorem.** Only a finite number of axioms are needed to define the *Universal Turing Machine*.

**Minskyizing the UTS**

Starting with *Claude Shannon* in 1956, many people — often in competition with *Marvin Minsky* — proposed very small UTMs (but their operation requires extensive coding of patterns). But, axiomatically, they do not require as many axioms as Turing did.

**Post-Markov's Solution**

The basic idea of Post (1943) was that a logistic system is simply a set of rules specifying how to change one string of symbols (antecedent) into another string of symbols (consequent). This leads to:

**The Word Problem for Semigroups**

\[(1) \quad \forall x \forall y \left[ x \cdot 1 = x = 1 \cdot x \right]\]

\[(2) \quad \forall x \forall y \forall z \left[ x \cdot (y \cdot z) = (x \cdot y) \cdot z \right]\]

**Problem:** Determine the provability of

\[A_0 = B_0 \land A_1 = B_1 \land ... \land A_{n-1} = B_{n-1} \implies A_n = B_n.\]
Schönfinkel-Curry’s Solution

Schönfinkel in 1924 and then Curry in 1929, both at Göttingen, began the study of combinator calculators, which were quickly connected with Church’s \( \lambda \)-calculus of 1932.

From them — with hindsight — we get:

Another Undecidable Theory

\begin{align*}
(1) & \quad \forall x \forall y \ [ K(x)(y) = x ] \\
(2) & \quad \forall x \forall y \forall z \ [ S(x)(y)(z) = x(z)(y(z)) ] \\
(3) & \quad \neg \ K = S
\end{align*}

Problem: Determine the provability of \( T = 0 \).

The only problem with this theory is that you either need models or something like the Church–Rosser Theorem to know it is consistent. A weaker theory of deterministic reduction can be given a fairly short axiomatization and then be proved consistent by much simpler means.
What’s Happened Since the 1930s?

The 1940s

Simple type theory & $\lambda$-calculus
Church (1940)

Primitive recursive functionals
Gödel (1941-58)

WW II

Recursive hierarchies
Kleene (1943)

Theory of categories
Eilenberg-Mac Lane (1945)

New completeness proofs
Henkin (1949-50)

The 1950s

Computing and Intelligence
Turing (1950)

Rethinking combinators
Rosenbloom (1950)

IAS Computer (MANIAC)
von Neumann (1951)

Introduction to Metamathematics
Kleene (1952)

IBM 701
Thomas Watson, Jr. (1952)

Arithmetical predicates
Kleene (1955)

FORTRAN
Backus et al. (1956-57)

ALGOL 58
Bauer et al. (1958)

LISP
McCarthy (1958)

Combinatory Logic. Volume I.
Curry-Feys-Craig (1958)

Adjoint functors
Kan (1958)

Recursive functionals & quantifiers, I.&II.
Kleene (1959-63)

Countable functionals
Kleene-Kreisel (1959)
McCarthy, LISP, & $\lambda$-Calculus

LISP History according to McCarthy’s memory in 1978. Presented at the ACM SIGPLAN History of Programming Languages Conference, June 1-3, 1978. It was published in History of Programming Languages, edited by Richard Wexelblat, Academic Press 1981. Two quotations:

I spent the summer of 1958 at the IBM Information Research Department at the invitation of Nathaniel Rochester and chose differentiating algebraic expressions as a sample problem. It led to the following innovations beyond the FORTRAN List Processing Language:

(c) To use functions as arguments, one needs a notation for functions, and it seemed natural to use the $\lambda$-notation of Church (1941). I didn't understand the rest of his book, so I wasn't tempted to try to implement his more general mechanism for defining functions. Church used higher-order functionals instead of using conditional expressions. Conditional expressions are much more readily implemented on computers.

Logical completeness required that the notation used to express functions used as functional arguments be extended to provide for recursive functions, and the LABEL notation was invented by Nathaniel Rochester for that purpose. D. M. R. Park pointed out that LABEL was logically unnecessary since the result could be achieved using only $\lambda$ — by a construction analogous to Church's Y-operator, albeit in a more complicated way.

Other key McCarthy publications:

Recursive Functions of Symbolic Expressions and their Computation by Machine (Part I). The original paper on LISP from CACM, April 1960. Part II, which never appeared, was to have had some Lisp programs for algebraic computation.


Towards a Mathematical Science of Computation, IFIPS 1962 extends the results of the previous paper. Perhaps the first mention and use of abstract syntax.

Correctness of a Compiler for Arithmetic Expressions with James Painter. May have been the first proof of correctness of a compiler. Abstract syntax and Lisp-style recursive definitions kept the paper short.

An HTML site concerning Lisp history can be found at:

http://www8.informatik.uni-erlangen.de/html/lisp-enter.html
The 1960s

**Recursive procedures**
Dijkstra (1960)

**ALGOL 60**
Backus et al. (1960)

**Elementary formal systems**
Smullyan (1961)

**Grothendieck topologies**
M.Artin (1962)

**Higher-type \( \lambda \)-definability**
Kleene (1962)

**Grothendieck topoi**
Grothendieck et al. SGA 4 (1963-64-72)

**CPL**
Strachey, et al. (1963)

**Functorial semantics**
Lawvere (1963)

**Continuations (1)**
van Wijngaarden (1964)

**Adjoint functors & triples**
Eilenberg-Moore (1965)

**\textbf{Cartesian closed categories}**
Eilenberg-Kelly (1966)

**ISWIM & SECD machine**
Landin (1966)

**CUCH & combinator programming**
Böhm (1966)

**New foundations of recursion theory**
Platek (1966)

**Normalization Theorem**
Tait (1967)

**AUTOMATH & dependent types**
de Bruijn (1967)

**Finite-type computable functionals**
Gandy (1967)

**ALGOL 68**
van Wijngaarden (1968)

**Normal-form discrimination**
Böhm (1968)

**Category of sets**
Lawvere (1969)

**Typed domain logic**
Scott (1969-93)

**Domain-theoretic \( \lambda \)-models**
Scott (1969)

**Formulae-as-types**
Howard (1969 -1980)

**Adjointness in foundations**
Lawvere (1969)

---

**Theorem.** The category of \( T_0 \)-topological spaces and continuous functions is **not** cartesian closed.

**Theorem.** The category of \( T_0 \)-topological spaces **with** an equivalence relation and continuous functions **respecting** equivalence **is** cartesian closed.

**Cartesian closed categories give us the algebraic version of typed \( \lambda \)-calculus.**
# The 1970s

<table>
<thead>
<tr>
<th>Continuations (2)</th>
<th>Mazurkiewicz (1970)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuations (3)</td>
<td>F. Lockwood Morris (1970)</td>
</tr>
<tr>
<td>Continuations (4)</td>
<td>Wadsworth (1970)</td>
</tr>
<tr>
<td><em>Categorical logic</em></td>
<td>Joyal (1970+)</td>
</tr>
<tr>
<td><em>Elementary topoi</em></td>
<td>Lawvere-Tierney (1970)</td>
</tr>
<tr>
<td><em>Denotational semantics</em></td>
<td>Scott-Strachey (1970)</td>
</tr>
<tr>
<td><em>Coherence in closed categories</em></td>
<td>Kelly (1971)</td>
</tr>
<tr>
<td><em>Quantifiers and sheaves</em></td>
<td>Lawvere (1971)</td>
</tr>
<tr>
<td><em>Martin-Löf type theory</em></td>
<td>Martin-Löf (1971)</td>
</tr>
<tr>
<td><em>System F, Fω</em></td>
<td>Girard (1971)</td>
</tr>
<tr>
<td><em>Logic for Computable Functions</em></td>
<td>Milner (1972)</td>
</tr>
<tr>
<td><em>From sheaves to logic</em></td>
<td>Reyes (1974)</td>
</tr>
<tr>
<td><em>Polymorphic λ-calculus</em></td>
<td>Reynolds (1974)</td>
</tr>
<tr>
<td><em>Call-by-name, call-by-value</em></td>
<td>Plotkin (1975)</td>
</tr>
<tr>
<td><em>Modeling Processes</em></td>
<td>Milner (1975)</td>
</tr>
<tr>
<td><em>SASL</em></td>
<td>Turner (1975)</td>
</tr>
<tr>
<td><em>Scheme</em></td>
<td>Sussman-Steele (1975-80)</td>
</tr>
<tr>
<td><em>Functional programming &amp; FP</em></td>
<td>Backus (1977)</td>
</tr>
<tr>
<td><em>First-order categorical logic</em></td>
<td>Makkai-Reyes (1977)</td>
</tr>
<tr>
<td><em>Edinburgh LCF</em></td>
<td>Milner et al. (1978)</td>
</tr>
<tr>
<td><em>Let-polymorphic type inference</em></td>
<td>Milner (1978)</td>
</tr>
<tr>
<td><em>Intersection types</em></td>
<td>Coppo-Dezani (1978)</td>
</tr>
<tr>
<td><em>ML</em></td>
<td>Milner et al. (1979)</td>
</tr>
<tr>
<td>*-<em>Autonomous categories</em></td>
<td>Barr (1979)</td>
</tr>
<tr>
<td><em>Sheaves and logic</em></td>
<td>Fourman-Scott (1979)</td>
</tr>
</tbody>
</table>

This decade saw the importance of constructive logic, the applications to language design and semantics, and the connections to category theory become much clearer.
The 1980s

*Frege structures*  
Aczel (1980)

*HOPE*  
Burstall et al. (1980)

*The Lambda Calculus Book*  
Barendregt (1981-84)

*Structural Operational Semantics*  
Plotkin (1981)

*Effective Topos*  
Hyland (1982)

*Dependent types & modularity*  

*Locally CCC & type theory*  
Seely (1984)

*Calculus of Constructions*  
Coquand-Huet (1985)

*Bounded quantification*  
Cardelli-Wegner (1985)

*NUPRL*  
Constable et al. (1986)

*Higher-order categorical logic*  
Lambek-P.J.Scott (1986)

*Cambridge LCF*  
Paulson (1987)

*Linear logic*  
Girard et al. (1987-89)

*HOL*  
Gordon (1988)

*FORSYTHE*  
Reynolds (1988)

*Proofs and Types*  
Girard et al. (1989)

*Integrating logical & categorical types*  
Gray (1989)

*Computational λ-calculus & monads*  
Moggi (1989)

*Type theory, resource logic, and computer-assisted theorem proving finally became practical during these years.*
### The 1990s

<table>
<thead>
<tr>
<th><strong>HASKELL</strong></th>
<th>Hudak-Hughes-Peyton Jones-Wadler (1990)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Higher-type recursion theory</strong></td>
<td>Sacks (1990)</td>
</tr>
<tr>
<td><strong>STANDARD ML</strong></td>
<td>Milner, et al. (1990-97)</td>
</tr>
<tr>
<td><strong>Lazy λ-calculus</strong></td>
<td>Abramsky (1990)</td>
</tr>
<tr>
<td><strong>Higher-order subtyping</strong></td>
<td>Cardelli-Longo (1991)</td>
</tr>
<tr>
<td><strong>Categories, Types and Structure</strong></td>
<td>Asperti-Longo (1991)</td>
</tr>
<tr>
<td><strong>STANDARD ML of NJ</strong></td>
<td>MacQueen-Appel (1991-98)</td>
</tr>
<tr>
<td><strong>QUEST</strong></td>
<td>Cardelli (1991)</td>
</tr>
<tr>
<td><strong>Edinburgh LF</strong></td>
<td>Harper, et al. (1992)</td>
</tr>
<tr>
<td><strong>Pi-Calculus</strong></td>
<td>Milner-Parrow-Walker (1992)</td>
</tr>
<tr>
<td><strong>Categorical combinators</strong></td>
<td>Curien (1993)</td>
</tr>
<tr>
<td><strong>Translucent types &amp; modular</strong></td>
<td>Harper-Lillibridge (1994)</td>
</tr>
<tr>
<td><strong>Full abstraction for PCF</strong></td>
<td>Hyland-Ong/Abramsky, et al. (1995)</td>
</tr>
<tr>
<td><strong>Algebraic set theory</strong></td>
<td>Joyal-Moerdijk (1995)</td>
</tr>
<tr>
<td><strong>Object Calculus</strong></td>
<td>Abadi-Cardelli (1996)</td>
</tr>
<tr>
<td><strong>Typed intermediate languages</strong></td>
<td>Tarditi, Morrisett, et al. (1996)</td>
</tr>
<tr>
<td><strong>Proof-carrying code</strong></td>
<td>Necula-Lee (1996)</td>
</tr>
<tr>
<td><strong>Computability and totality in domains</strong></td>
<td>Berger (1997)</td>
</tr>
<tr>
<td><strong>Typed assembly language</strong></td>
<td>Morrisett, et al. (1998)</td>
</tr>
<tr>
<td><strong>Type theory via exact categories</strong></td>
<td>Birkedal, et al. (1998)</td>
</tr>
<tr>
<td><strong>Categorification</strong></td>
<td>Baez (1998)</td>
</tr>
</tbody>
</table>

Abstract ideas now found many applications in language implementation and in compiling.
## The New Millennium

<table>
<thead>
<tr>
<th>Concept</th>
<th>Authors</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicative topos</td>
<td>Moerdijk-Palmgren</td>
<td>(2000)</td>
</tr>
<tr>
<td>Differential $\lambda$-calculus</td>
<td>Ehrhard/Regnier</td>
<td>(2003)</td>
</tr>
<tr>
<td>Modular Structural Operational Semantics</td>
<td>Mosses</td>
<td>(2004)</td>
</tr>
<tr>
<td>A $\lambda$-calculus for real analysis</td>
<td>Taylor</td>
<td>(2005+)</td>
</tr>
<tr>
<td>Univalence axiom</td>
<td>Voevodsky</td>
<td>(2006+)</td>
</tr>
<tr>
<td>The safe $\lambda$-calculus</td>
<td>Ong, et al.</td>
<td>(2007)</td>
</tr>
<tr>
<td>Higher topos theory</td>
<td>Lurie</td>
<td>(2009)</td>
</tr>
<tr>
<td><strong>Univalent Foundations Program @ IAS  &amp; HoTT Book</strong></td>
<td>Voevodsky, et al.</td>
<td>(2012-13)</td>
</tr>
</tbody>
</table>

In the natural world, **convergent evolution** can give creatures analogous structures — even though they cannot mate. But, in the intellectual world, analogous structures can be taken advantage of through interfertilization of areas and in finding new applications.

And that we have seen happen with the $\lambda$-calculus many, many times over the years.
A Closing Thought from Robert Harper

For me, I think it is important to stress the overwhelming influence of the λ-calculus among all other models of computation:

• It codifies not only computation, but also the basic principles of human reason (natural deduction).

• Moreover, it was born fully formed, and is directly and immediately relevant to this day, rather than something that collects dust on the shelf.

Admittedly Turing's model had the advantage of being explicitly psychologically motivated, but on the other hand Church focused on one of the greatest achievements of the human mind, the concept of a variable (= reasoning under hypotheses). Church saw that this was central, and time has born out the significance of his insight.

By contrast, no one cares one bit about the details of a Turing Machine; for, it fails to address the central issue of modularity (logical consequence), which is so important in programming and reasoning. And it does not extend to higher-order computation in anything like a natural or smooth way.

LAMBDA CONQUERS ALL!

Perhaps my good friend and colleague has spoken a little too strongly here, as Turing Machines have had many applications, say in Complexity Theory.

But the study of Programming Languages does not seem to need them today.
A Selective Bibliography

A very helpful review of the subject of the λ-calculus is in the first reference, and the memoirs by Alonzo Church’s two early students are also useful in checking history. The thesis by Rod Adams gives a very careful survey of early literature. A somewhat revisionist view of the history of recursive function theory with many helpful references is found in the Soare paper. Jones and Simonsen fill out ideas related to machine structure. The whole Royal Society volume is devoted to The Turing Legacy. And Plotkin also recently wrote on operational semantics. The older collection edited by Rolf Herken, The Universal Turing Machine: A Half-Century Survey, has many, many excellent historical discussions by Kleene, Gandy, Davis, Feferman, and others. The papers of Davis and Sieg give very detailed historical reviews of the early 1930s. The recent conference Church’s Thesis After 70 Years (Olszewski, et al. eds. 2006) has many interesting discussions.


What follows is a listing of books. Ph.D. theses and conference proceedings have been excluded, for the most part, as well as very elementary text books. A comprehensive survey is impossible, but the current list has tried to indicate some of the history and development of the intertwining strands of \( \lambda \)-calculus, logic, recursive-function theory, category theory, and programming-language semantics.


*And, no, I have not read — or even seen — all these books!*

Suggestions, corrections and additions would be appreciated, so please send e-mail to dana.scott@cs.cmu.edu with the subject heading: Lambda calculus.

The question of finding the most recent edition of a book is vexing, but Amazon.com was quite helpful. Bibliographies of several books and papers were “mined”, and of course all these books themselves also give references to the ever more vast journal literature. There is also the problem — in outlining history — of comparing the date of discovery to the date of publication. Perhaps there are many such confusions in this survey.