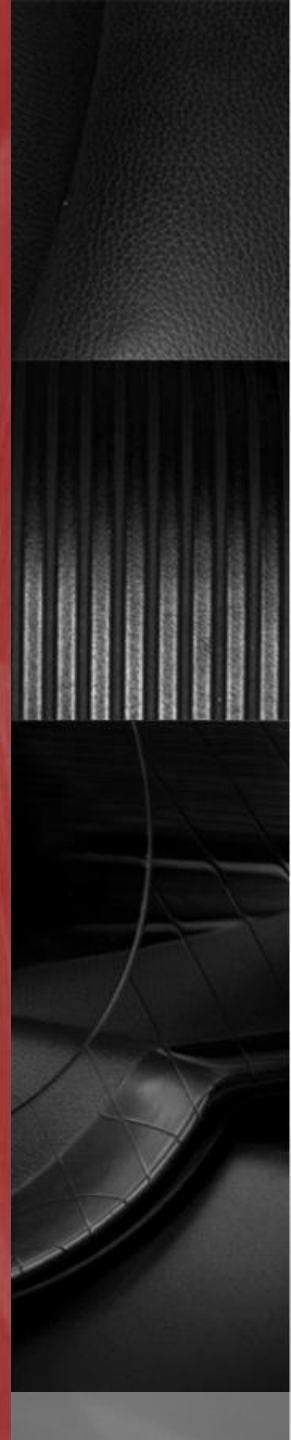
Introduction to the Lambda Calculus

Chris Lomont ~~2010~~ ~~2011~~ 2012

www.lomont.org



Leibniz (1646-1716)

- “Create a universal language in which all possible problems can be stated”

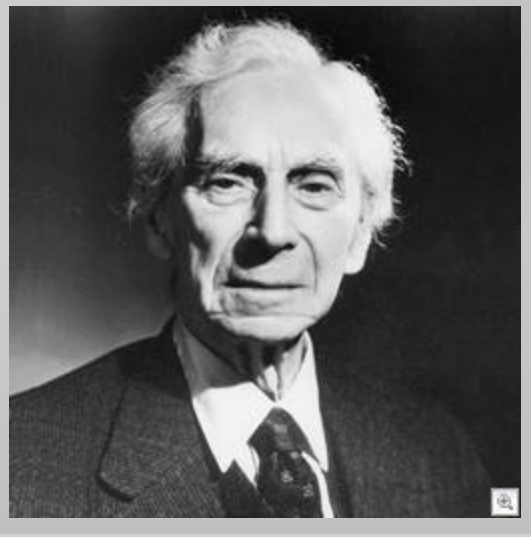
- “Find a decision method to solve all problems stated in the universal language”

The Language

- Set theory coupled with first order logic
  - A formal logical system used in mathematics, philosophy, linguistics, and computer science
  - Many names: first-order predicate calculus, the lower predicate calculus, quantification theory, and predicate logic
  - Different from propositional logic by its use of quantifiers
  - Distinguished from higher-order logics in that quantification is allowed only over atomic entities (individuals but not sets)

- Frege and Russell
- Zermelo
- early 1900's
- Russell paradox 1901 ends naïve sets:

$$R = \{x | x \notin x\} \text{ then } R \in R \Leftrightarrow R \notin R$$



Can all problems be solved?

- Among Hilbert's 23 problems, 1903
- Problems 3, 7, 10, 11, 13, 14, 17, 19, 20, and 21 have a resolution that is accepted by consensus.
- Problems 1, 2, 5, 9, 15, 18+, and 22 have solutions that have partial acceptance, but there exists some controversy as to whether it resolves the problem.
  - (18 is Kepler conjecture, requires computer proof)
- 8: the Riemann hypothesis, 12: abstract number theory
- 4, 6, 16, and 23 are too vague to ever be described as solved.
- **2nd problem: Prove that the axioms of arithmetic are consistent.**
  - Continuing Leibniz dream

Can arithmetic be proven consistent? No!

- Godel Incompleteness Theorems (1931)
- First: “Any effectively generated theory capable of expressing elementary arithmetic cannot be both **consistent** and **complete**.”
- Gödel's second incompleteness theorem, proved in 1931, shows that no proof of its consistency can be carried out within arithmetic itself.
  - “For any formal effectively generated theory *T* including basic arithmetical truths and also certain truths about formal provability, *T* includes a statement of its own consistency if and only if *T* is inconsistent.”
- Leads to Halting Problem in computer science (Turing, 1936)
  - Someday we’ll talk about the Turing Hierarchy ☺

What can then be solved automatically?

- Independently: Alonzo Church and Alan Turing, 1936
- Formalized the notion of “decidable”, which is equivalent to “what is computable.”
- Church (1936) invented the “lambda calculus”, written λ-calculus.
  - A “calculus” is a method of moving symbols, not the undergrad course.
- Turing (1936/37) invented “Turing Machines”.
- Turing showed them to be equivalent.

Outcome

- Modern machines are based on Turing Machines (actually, von Neumann machines, allowing random memory access).
- Modern computer languages are based on concepts from lambda calculus, which allow telling such a machine what to do.
- Imperative languages (Pascal, C, Fortran) are based on the way Turing machine are instructed
- Functional languages like Miranda, Lisp, ML, Haskell, F#, are based on lambda calculus.

Reduction

- Functional program consists of
  - Expression E (represents both algorithm and input)
  - This is subject to rewrite rules
  - Reduction: replace a part P of E by some P', Provided that $P \rightarrow P'$ is one of the rules

$$E[P] \rightarrow E[P']$$

Example:

- first of (sort (append (`dog', `rabbit') (sort ((`mouse', `cat')))))
- ->first of (sort (append (`dog', `rabbit') (`cat', `mouse')))
- ->first of (sort (`dog', `rabbit', `cat', `mouse'))
- -first of (`cat', `dog', `mouse', `rabbit')
- -> `cat':

Properties

- Church-Rosser property: normal form is independent of order of rewrite rules
- Rewrite rules are called combinators

λ-Calculus

- Two operations: application and abstraction
- Application
  - $F \cdot A$ or FA is applying data F as an algorithm to data A considered as input.
  - Theory is type-free, so FF makes sense.
- Abstraction
  - If $M[x]$ is expression containing x, then the function $\lambda x.M[x]$ represents the function $x \longmapsto M[x]$
  - Example: $(\lambda x.2x + 1)3 = 2 * 3 + 1\ (= 7)$
- Called “the smallest programming language of the world”.
- Introduced in 1930 by Alonzo Church to formalize the notion of computability.

Abstraction

- Binds the free variable x in M, that is, $\lambda x.xy$ has x bound and y free.

Multiple arguments:

- $f(x, y)$ depends on 2 variables.
- Can define
- $F_x = \lambda y. f(x, y)$
- $F = \lambda x. F_x$
- Then $(Fx)y = F_x y = f(x, y)$
- Leads to association:
  - $FM_1M_2 \cdots M_n = (\cdots (FM_1)\, M_2) \cdots M_n)$
- Called currying, after Haskell Curry
- Haskell computer language named after him

Notation

Definition 1:

λ-terms: the set Λ built from the infinite set of variables $\Lambda = \{v', v'', v''', \dots\}$ using application and abstraction:

$$x \in V \Rightarrow x \in \Lambda$$
$$M, N \in \Lambda \Rightarrow (MN) \in \Lambda$$
$$M \in \Lambda, x \in V \Rightarrow (\lambda x M) \in \Lambda$$

We use lowercase x, y, z … for variables and caps M, N, L … for Λ-terms

Notation

Definition 2:

The free variables in M, written $FV(M)$ is

$$FV(x) = \{x\}$$
$$FV(MN) = FV(M) \cup FV(N)$$
$$FV(\lambda x M) = FV(M) - \{x\}$$

A not-free variable is bound.

M is a closed λ-term (also called a combinator) if $FV(M) = \emptyset$

Definition 3:

The result of substituting N for the free occurrences of x in M is denoted

$$M[x := N]$$

Λ Calculus

- The principal axiom is

$$(\lambda x M)N = M[x := N]$$

- Logical rules and axioms:
  - Equality

$$M = M$$

$$M = N \Rightarrow N = M$$

$$M = N, N = L \Rightarrow M = L$$

  - Compatibility

$$M = M' \Rightarrow MZ = M'Z$$

$$M = M' \Rightarrow ZM = ZM'$$

$$M = M' \Rightarrow \lambda x.M = \lambda x.M'$$

- Proof: if $M = N'$ is provable in the $\lambda -$ calculus, write $\lambda \vdash M = N$

Standard Combinators

Name	Definition	Action on terms
Identity	$\mathbf{I} = \lambda x.\, x$	$\mathbf{I}M = M$
Left	$\mathbf{K} = \lambda xy.\, x$	$\mathbf{K}MN = M$
Right	$\mathbf{K}_* = \lambda xy.\, y$	$\mathbf{K}MN = N$
	$\mathbf{S} = \lambda xyz.\, xz(yz)$	$\mathbf{S}MNL =$ML(NL)

Used to solve equations

Reduction and Normality

- Three reduction types:
  - α reduction: renaming a bound variable, e.g., $\lambda x.\, x \rightarrow \lambda y.\, y$
  - β reduction: applying functions to arguments , e.g. $(\lambda z.\, z * 2)5 \rightarrow 5 * 2$
  - η reduction: *extensionality*, that is, two functions are the same if and only if they give the same value for all arguments.

Normal Form

- Expression cannot be “rewritten” any further.
- Not all expressions have this property:
  - Consider $\lambda x.\, xxx$. Applied to itself leads to (TODO – keeps expanding)

Type equation here.

- TODO – state it
- TODO – can always be done
- TODO – expand outermost evaluation first
- TODO – some expressions never terminate, have no normal form.

Arithmetic

- Define $F^0(M) \equiv M$, $F^{n+1}(M) \equiv F^n(M)$
- Church numerals: $c_0, c_1, c_2, \ldots$ defined via $c_n \equiv \lambda f x.\, f^n(x)$
- Operations:
$$\mathbf{A}_+ \equiv \lambda xypq.\, xp(ypq)$$
$$\mathbf{A}_* \equiv \lambda xyz.\, x(yz)$$
$$\mathbf{A}_{exp} \equiv \lambda xy.\, yx$$
- Then, for all nonnegative integers n, m
$$\mathbf{A}_+ c_n c_m = c_{n+m}$$
$$\mathbf{A}_* c_n c_m = c_{n*m}$$
$$\mathbf{A}_{exp} = c_{(n^m)}$$
- $\mathbf{A}_+ c_n c_m =$

Booleans

- $\mathbf{true} \equiv \mathbf{K}\ (\equiv \lambda xy.x)$, $\mathbf{false} \equiv \mathbf{K}_*\ (\equiv \lambda xy.y)$
- If B can be considered a boolean (i.e., true or false) then

  "If B then P else Q" can be represented by BPQ
- Ordered pair: $[M, N] \equiv \lambda z.\,\text{if } z \text{ then } M \text{ else } N (\equiv \lambda z.zMN)$

Y-Combinator

- Creates recursion from single steps
- It is a function **Y** such that for any function **g**, **Y(f)=g(Y(g))**.
- **Y** finds fixed-points for functions, **Y(g)=p** where **p=g(p)**
- Does it exist? Yes, there are infinitely many.
  - Y-Combinator **Y** = λf.(λx.f (x x)) (λx.f (x x))
  - Another for amusement:
    $\mathbf{Y}_k$ = (L L L L L L L L L L L L L L L L L L L L L L L L L L) with
    L = λabcdefghijklmnopqstuvwxyzr. (r (t h i s i s a f i x e d p o i n t c o m b i n a t o r))

Y-Combinator

- Definition: **Y** = λf.(λx.f (x x)) (λx.f (x x)). Apply to arbitrary function **g**
- **Y** g= (λf . (λx . f (x x)) (λx . f (x x))) g (by definition of **Y**)
- = (λx . g (x x)) (λx . g (x x)) (β-reduction of λf: applied main function to g)
- = (λy . g (y y)) (λx . g (x x)) (α-conversion: renamed bound variable)
- = g ((λx . g (x x)) (λx . g (x x))) (β-reduction of λy: applied left function to right function)
- = g (**Y** g) (by second equality)
- Repeat, giving g (g (g (g (**Y** g)))) as deep as desired… Recursion.

TODO

- Y-Combinator **Y** = λf.(λx.f (x x)) (λx.f (x x))
- Church numerals, add, exponent,
- True, false, pairs
- Definability, recursive functyions, all computable functions.
- Reduction

Types

- So far the λ-calculus is a type free theory
- Example, $\mathbf{I} = \lambda x.\, x$ can be applied to itself: $\mathbf{II}$
- Typed λ-calculus introduced in Curry(1934) and Church (1940).
- Type is an “thing” assigned to a term.
- Type may be thought of as a units “dimension”, prevents adding meters to mass.

Type benefits

- Allows easier checking for correctness
- Allows easier compilation and running (if type of an expression is arithmetical, can use ALU instead of generalized code).
- Implicit typing, in ML, assigned by compiler
- Explicit typing, common, assigned by programmer
- Not all computable functions can be represented by a typed term!

Typing Results

- Worked out in theory a long time ago
- Computer languages keep playing with various balances between the various flavors of typing.

C# Lambda Functions

- (x, y) => x == y
- TestDelegate myDel = n => { string s = n + " " + "World"; Console.WriteLine(s); }; myDel("Hello");

C++0x Lambda Functions

- [](int x, int y) { return x + y; }

- std::vector<int> some_list; int total = 0; std::for_each(some_list.begin(), some_list.end(), [&total](int x) { total += x; });

F# Lambda Functions

Bibliography

- [1] Introduction to Lambda Calculus, Henk Barendregt and Erik Barendsen, March 2000.
- [2] Wikipedia
- [3] http://en.wikipedia.org/wiki/Currying
- [4] http://www.inf.fu-berlin.de/lehre/WS03/alpi/lambda.pdf, ”A Tutorial Intorduction to the Lambda Calculus”