The Untyped Lambda Calculus: A Simple Functional Programming Language

Paul Gustafson

Math 482 - Texas A&M University

March 5, 2013

Why is the λ-calculus important?

- Computer Science
  - Variable binding in function declarations
  - Scope
  - Type sytems
  - Functional programming languages (Lisp, ML variants, Haskell)
- Logic
  - Computability
  - Constructivism ("Proofs as Programs")
- Linguistics

Why was the λ-calculus developed?

- Formal system of logic developed by Alonzo Church in 1932
- Used to solve Leibniz' *Entscheidungsproblem* ("Decision problem")
  - "Is every statement in first-order logic over a finite set of axioms decidable?"
  - No - Church and Turing, independently

How does the λ-calculus work? (I): λ-terms

- The set of λ-terms, Λ, is built from a countable set of variables $V = \{v, v', v'', \ldots\}$:
  1. $x \in V \implies x \in \Lambda$
  2. $M, N \in \Lambda \implies (MN) \in \Lambda$
  3. $M \in \Lambda, x \in V \implies (\lambda x.M) \in \Lambda$
- Examples of λ-terms
  - v'
  - $(\lambda v.(v'v))$
  - $(((\lambda v.(\lambda v'.(v'v)))v'')v''')$
- Free and bound variables, closed terms

Convenient syntactic assumptions

- Drop outer parentheses
- Lowercase letters are placeholders for arbitrary variables
- Scope of λ extends as far to the right as possible
  - Example: $\lambda x.\lambda y.xy = \lambda x.(\lambda y.(xy))$
- Expressions are left associative by default
  - Example: $xyz = (xy)z$
- Multiple bindings in a row can be contracted.
- Example $\lambda xyz.M = \lambda x.\lambda y.\lambda z.M$.

How does the λ-calculus work? (II): Conversion Rules

- α-conversion: $\lambda x.[...x...] = \lambda y.[...y...]$.
  - "We can rename bound variables."
  - Example: $\lambda a.a = \lambda b.b$
- β-conversion: $\lambda x.[...x...]T = [...T...]$.
  - "Evaluation / substitution."
  - Example: $(\lambda x.x)y = y$.
- η-conversion: $\lambda x.F(x) = F$.
  - "Extensionality - a function is defined by what it does."
  - Example: $\lambda y.\lambda x.yx = \lambda y.y$

Representing booleans

- $\text{true} = \lambda x.\lambda y.x$
- $\text{false} = \lambda x.\lambda y.y$
- if a then b else $c = abc$
  - if true then b else $c = (\lambda x.\lambda y.x)bc = (\lambda y.b)c = b$.
  - if false then b else $c = (\lambda x.\lambda y.y)bc = (\lambda y.y)c = c$.

Church numerals

- A representation of the natural numbers
  - $0 := \lambda f.\lambda x.x$
  - $1 := \lambda f.\lambda x.fx$
  - $2 := \lambda f.\lambda x.f(fx)$
  - $3 := \lambda f.\lambda x.f(f(fx))$
  - $\dots$
  - $n := \lambda fx.f^{(n)}(x)$

Arithmetic with Church numerals (I)

- Successor: $\lambda n.\lambda f.\lambda x.f(nfx)$
  - Example:

$$\begin{aligned} S(1) &= (\lambda nfx.f(nfx))(\lambda fx.fx) \\ &=_\alpha (\lambda nfx.f(nfx))(\lambda gy.gy) \\ &= \lambda fx.f((\lambda gy.gy)fx) \\ &= \lambda fx.f((\lambda y.fy)x) \\ &= \lambda fx.f(f(x)) \\ &= 2. \end{aligned}$$

Arithmetic with Church numerals (II)

- Addition: $\lambda m.\lambda n.\lambda f.\lambda x.mf(nfx)$
- Multiplication: $\lambda m.\lambda n.\lambda f.m(nf)$
- Exponentiation: $\lambda m.\lambda n.nm$
- IsZero: $\lambda n.n(\lambda x.\mathrm{false})\mathrm{true}$
- Predecessor: $\lambda n.\lambda f.\lambda x.n(\lambda g.\lambda h.h(gf))(\lambda u.x)(\lambda u.u)$

The Y-combinator

- Define the Y-combinator by $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$
- Fixed-point Theorem: For any term $g \in \Lambda$, we have $g(Yg) = Yg$.
- Proof:

$$\begin{aligned} Yg &= (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))g \\ &= (\lambda x.g(xx))(\lambda x.g(xx)) \\ &= g((\lambda x.g(xx))(\lambda x.g(xx))) \\ &= g(Yg) \end{aligned}$$

Recursion

- Since $Yg = g(Yg)$, we have
  $Yg = g(Yg) = g(g(Yg) = g(g(g(Yg))) = \ldots.$
- We can use this to implement recursion.

References

- *Lecture Notes on the Lambda Calculus.* Peter Selinger. http://www.mscs.dal.ca/ selinger/papers/lambdanotes.pdf
- *Untyped Lambda Calculus.* Deepak D'Souza. http://drona.csa.iisc.ernet.in/ deepakd/pav/lecture-notes.pdf
- *Introduction to Lambda Calculus.* Barendregt and Barensen. ftp://ftp.cs.ru.nl/pub/CompMath.Found/lambda.pdf
- *Lambda Calculus, Then and Now.* Dana S. Scott. http://www.youtube.com/watch?v=7cPtCpyBPNI