INF 102 ANALYSIS OF PROG. LANGS LAMBDA CALCULUS

Instructors: Kaj Dreef Copyright © Instructors.

History

- Formal mathematical system
- Simplest programming language
- Intended for studying functions, recursion
- Invented in <u>1936</u> by Alonzo Church (1903-1995)

Same year as Turing's paper

Warning

- May seem trivial and/or irrelevant now
- □ Had a tremendous influence in PLs
 □ λ-calculus → Lisp → everything
- Context in the early 60s:
 - Assembly languages
 - Cobol
 - Unstructured programming

What is Calculus?

4

Calculus is a branch of mathematics that deals with limits and the differentiation and integration of functions of one or more variables

Real Definition

- 5
- A calculus is just a bunch of rules for manipulating symbols.
- People can give meaning to those symbols, but that's not part of the calculus.
- Differential calculus is a bunch of rules for manipulating symbols. There is an interpretation of those symbols corresponds with physics, slopes, etc.

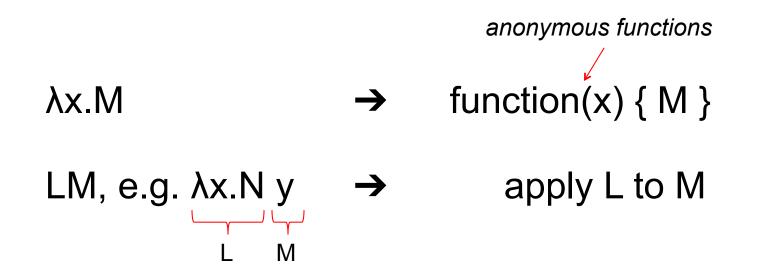


M .:= x	(variable)
λx.M	(abstraction)
MM	(application)

Nothing else!

- No numbers
- No arithmetic operations
- No loops
- No etc.
- Symbolic computation

Syntax reminder



Terminology – bound variables

λx.M

The binding operator λ binds the variable x in the λ -term x.M

- M is called the scope of x
- *x* is said to be a *bound variable*

Terminology – free variables

Free variables are all symbols that aren't bound (duh)

 $FV(x) = \{x\}$ FV(MN) = FV(M) U FV(N) FV(x.M) = FV(M) - x

Renaming of bound variables

$\lambda x.M = \lambda y.([y/x]M)$ if y not in FV(M)

i.e. you can replace x with y aka "renaming"

α-conversion

Operational Semantics

- Evaluating function application: $(\lambda x.e_1) e_2$
 - **Replace every** x in e_1 with e_2
 - Evaluate the resulting term
 - Return the result of the evaluation
- Formally: "β-reduction" (aka "substitution")

 $\Box (\lambda x.e_1) e_2 \rightarrow_{\beta} e_1[e_2/x]$

- A term that can be β-reduced is a redex (reducible expression)
- \square We omit β when obvious

Note again

- Computation = pure symbolic manipulation
 - Replace some symbols with other symbols

Scoping etc.

- Scope of λ extends as far to the right as possible
 $\lambda x.\lambda y.xy$ is $\lambda x.(\lambda y.(x y))$
- Function application is left-associative
 xyz means (xy)z
- Possible syntactic sugar for declarations
 - $\Box (\lambda x.N)M \quad \text{is} \quad \textbf{let } x = M \textbf{ in } N$
 - $(\lambda x.(x + 1))10$ is let x=10 in (x+1)

Multiple arguments

- □ y(x'λ) · 6 is it is it
 - Doesn't exist
- Solution: $\lambda x.\lambda y.e$ [remember, ($\lambda x.(\lambda y.e)$)]
 - A function that takes x and returns another function that takes y and returns e
 - □ (λx . λy .e) a b→(λy .e[a/x]) b→e[a/x][b/y]
 - "Currying" after Curry: transformation of multi-arg functions into higher-order functions
- Multiple argument functions are nothing but syntactic sugar

Boolean Values and Conditionals

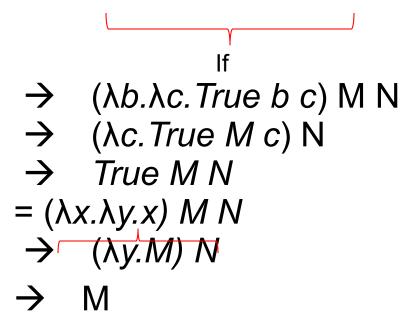
- $\Box \text{ True} = \lambda x.\lambda y.x$
- False = $\lambda x \cdot \lambda y \cdot y$
- □ If-then-else = λa . λb . λc . a b c = a b c
- For example:

□ If-then-else true b c $\rightarrow(\lambda x.\lambda y.x)$ b c $\rightarrow(\lambda y.b)$ c \rightarrow b □ If-then-else false b c

 \rightarrow ($\lambda x.\lambda y.y$) $b \rightarrow (\lambda y.y) \rightarrow c \rightarrow c$

Boolean Values and Conditionals

• If True M N = $(\lambda a.\lambda b.\lambda c.abc)$ True M N

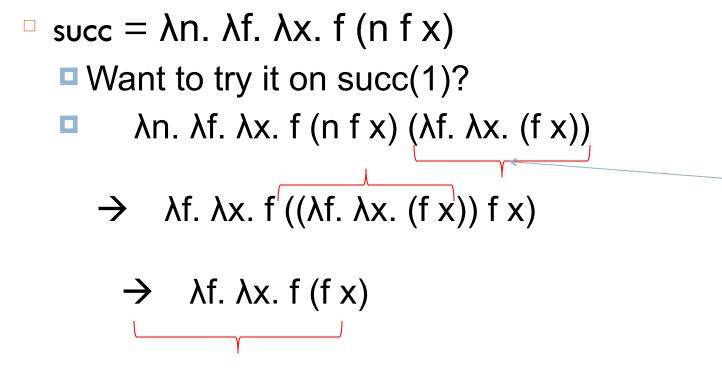




Numbers are counts of things, any things. Like function applications!

Church numerals

Successor



2!



Reading materials

Recursion ???

Recursion – The Y Combinator

$$Y = \lambda t. (\lambda x. t (x x)) (\lambda x. t (x x))$$

Y a =
$$\lambda t.$$
 ($\lambda x.$ t (x x)) ($\lambda x.$ t (x x)) a
= ($\lambda x.$ a (x x)) ($\lambda x.$ a (x x))
= a (($\lambda x.$ a (x x)) ($\lambda x.$ a (x x)))
= a (Y a)

Y *a* = *a* applied to itself!

Y a = a (Y a) = a (a (Y a)) = a (a (a (Y a))) = ...

Factorial again

λn. <mark>λf</mark>.λn. F (if (zero? n) (if (zero? n) (* n (f (sub1 n)))) (* n (f (sub1 n)))) Now it's bound

ΥF

Does it work?

F takes one function and one number as arguments

Points to take home

- Model of computation completely different from Turing Machine
 - pure functions, no commands
- Church-Turing thesis: the two models are equivalent
 What you can compute with one can be computed with the other
- Inspiration behind Lisp (late 1950s)
- Foundation of all "functional programming" languages