Lambda λ Calculus

Maria Raynor, Katie Kessler, Michael von der Lippe
December 5th, 2016
COSC 440

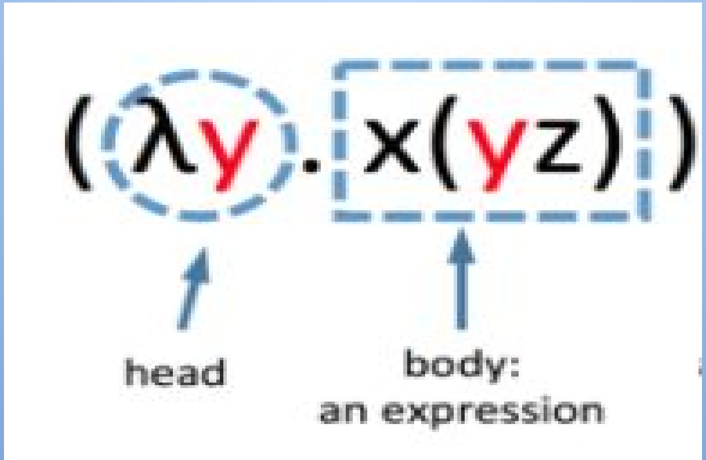
General Intro

- 1930s - Alonzo Church
- 1935 - Kleene and Rosser say inconsistent
- 1936 - published lambda calculus relevant to computation, untyped
- 1940 - published typed lambda calculus
- Language that expresses function abstraction and application

Definition

"A formal system in mathematical logic for expressing computation based on abstraction and application using variable binding and substitution"

- Variables $v_1, v_2, v_3, \ldots, v_n \in Var$
- Parenthesis ()
- Functions-take 1 argument,return 1 value



Syntax

Everything is an expression

1. $E \longrightarrow ID$
2. $E \longrightarrow \lambda\ ID.\ E$
3. $E \longrightarrow E\ E$
4. $E \longrightarrow (E)$

Examples and Non-Examples

- ❑ x
- ❑ λ x . x
- ❑ x y
- ❑ λ λ x . y
- ❑ λ x . y z

What about ambiguous syntax?

There is a set of disambiguation rules:

- E $\longrightarrow$ E E is left associative
  - x y z becomes (x y) z
  - w x y z becomes ((w x) y) z
- λ ID . E extends as far right as possible
  - λ x . x y becomes λ x . (x y)
  - λ x . λ x . x becomes λ x . (λ x . x)
  - λ a . λ b . λ c . a b c becomes λ a . (λ b . (λ c . ((a b) c)))

Simple Examples

Church Booleans: if p then x, else y

- TRUE = λ x . λ y . x
- FALSE = λ x . λ y . y

Identity function λ x . x

Constant function λ x . k

What about λ x . + x 1?

How it relates to theoretical computer science

Mathematical Composition vs Machine Composition

a. Lambda calculus is mathematical by nature.
b. Turing Machines are difficult to applied to programming because there is no natural notation for machines

Curry-Howard Correspondence

This foundation in math allows us to establish functional programming proofs of formal logic.

Category Theory and Curry-Howard Isomorphism:

- A generalization of a graph
- Nodes = “Objects”
- Arrows = “Morphisms”
- Objects connected by Morphisms are Functions

Higher Typed Computability

Church Turing Thesis:

- A Reasonable Machine is a machine that can perform computations
- Something is calculable if and only if a Turing machine or some equivalent construction can compute them.
- Programming is typically composed of higher types

The difference in between Lambda Calculus and TM's

- A Turing Machine can analyze the syntax of an argument
- Lambda calculus can only analyze the lambda symbol its receiving

Realizability Theory

Lambda Calculus Struggles with normal typed functions

- if a function is type N→N, you can't determine the lambda-term structure due to passing it only n's
- This is referred to as lambda is unable to *realize* f:N→N

However, Lambda Calculus performs well with higher typed functions

- Realizability infers that there are some computable problems in which a Turing Machine cannot realize and Lambda Calculus can as well as vice versa
    - Problems that are Lambda Realizable and not Turing Realizable take advantage of the ambiguity of the Lambda Calculus calling functions

John R. Longley - *Notations of Computability at Higher Types*

-Documentation of the many computability methods for higher typed functions
-Computation power is similar, but realizability limits the solution of some methods for some problems
-The research in the survey conveys that TMs are the most realizable machine we have for higher typed functions

ß-Reduction and Reasonability

"Reasonable machines can simulate each other within a polynomially overhead in time" - Weak Invariance Thesis, Slot and van Emde Boas'

Much like we have proven with the 3-CNF Reductions in class we must prove any lambda calculus function can be reduced to another.

ß-Reduction: $(\lambda x.\ M)N \rightarrow_{\beta} M[N/x]$

Process of calculating a result from the application of a function to an expression

- Until recently this was thought to be unrealistic to compute for many problems due costs of computation
- Benjamino Accattoli and Ugo Dal Lago have proven that for all problems within P, and EXP using a standard reduction (leftmost-outermost) are reducible in P and EXP time respectively.

What is possible with this topic?

- Can express anything a TM can express
- Church numerals
  - $0 := \lambda f.\lambda x.x$
  - $1 := \lambda f.\lambda x.f\ x$
  - $2 := \lambda f.\lambda x.f\ (f\ x)$
  - $3 := \lambda f.\lambda x.f\ (f\ (f\ x))$
- Functional programming

Lambda Calculus and Functional Languages

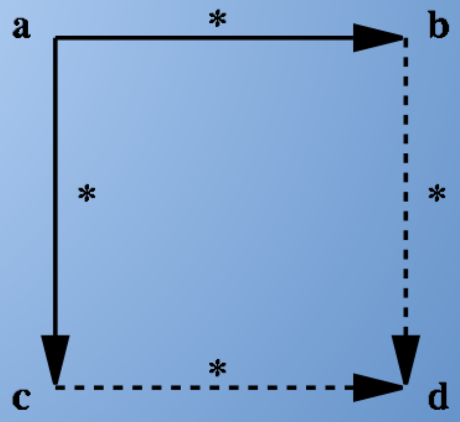
- Functional languages include the creation of functions and the application of functions
- Lambda Calculus can be used to represent all functional programming languages through lambda abstraction and application

Lambda Abstraction

- $E \longrightarrow \lambda\ ID.\ E$ is called a lambda abstraction because it defines a new anonymous function
  - ID - variable of abstraction
  - E - body of abstraction
- EX: f(x) = x + 1 as (λ x . + x 1)

Current Results

- Proof that predicate logic is undecidable
- Church-Rosser Theorem: when applying reduction rules to terms in the lambda calculus, the ordering in which the reductions are chosen does not make a difference to the eventual result



Achievements of Lambda Calculus

- In the 1960s, it was used to create Algol 60
- In 1964, it was used to create CUCH
  - Combinators - higher-order functions that use only function application and earlier defined combinators to define a result from their arguments
- Used to create programming languages like Lisp

How it could be applied to real life

- Lisp has be used for artificial intelligence and coding parts of Emacs
- Can be used to represent any function
- When constants and predefined functions are added, it becomes applied lambda calculus
  - allow λ -term to be a constant
  - add computation rules for replacing constants in expressions

Questions?

Works Cited

- https://docs.google.com/presentation/d/1mJwjUl-fK28oOXBoNQrUeH-awRB5h8Y9UDAk29Z1SEc/edit#slide=id.g191ac13104_0_657
- http://cstheory.stackexchange.com/questions/21705/what-is-the-contribution-of-lambda-calculus-to-the-field-of-theory-of-computatio
- http://homepage.cs.uiowa.edu/~slonnegr/plf/Book/Chapter5.pdf
- http://www.users.waitrose.com/~hindley/SomePapers_PDFs/2006CarHin,HistlamRp.pdf
- http://www.cse.iitd.ac.in/~saroj/LFP/LFP_2013/L18.pdf
- https://en.wikipedia.org/wiki/First-order_logic
- https://en.wikipedia.org/wiki/Church%E2%80%93Rosser_theorem
- http://homepages.inf.ed.ac.uk/jrl/Research/notions1.pdf
- http://www.math.lmu.de/~schwicht/lectures/proofth/ss13/ch2.pdf
- https://78d86768-a-62cb3a1a-s-sites.googlegroups.com/site/beniaminoaccattoli/beta-invariance.pdf?attachauth=ANoY7cp2yHMWFM7I2kDg31sE5vDcRB3Xudg55i35r0xHqvcs_oAyPWMF5XFq_tuc2OUfH9aqj5h9VQ-1dr9GHJKa8fkOCDZT7hMjZ3sAOnQW4Q49asI0T8xYu7OR9UYlur7Bn7MCnulc3ygRrgNKvDVbTKvbFn9vWf590JxvvfU-_CcoqkXpWGP0rcztPswkbSCrzQVt-vjmzqCdfY16TUD6YCIUmCO0bbeVw77YcYCk5Y7EFRxb-Yc%3D&attredirects=0
- http://cstheory.stackexchange.com/questions/1117/realizability-theory-difference-in-power-between-lambda-calculus-and-turing-mac
- **http://www.cs.cornell.edu/courses/cs3110/2008fa/recitations/rec26.html**
- https://www.youtube.com/watch?v=_kYGDJSm0gE