Church's Lambda Calculus

A Quick Overview

CAS 701 Class Presentation
18 November 2008

Mohammad Hosein Yarmand
Department of Computing & Software
McMaster University

Outline

Introduction

- Lambda calculus is a formal system designed to investigate function definition, application and recursion.
- Proposed by Alonzo Church and Stephen Cole Kleene in the 1930s.
- Intended to investigate the foundations of mathematics, but has emerged as a useful tool in the investigation of problems in computability or recursion theory, and forms the basis of functional programming.
- Lambda calculus raised implementation issue for stack-based programming languages as it treats functions as first-class objects.
- Programming languages such as Lisp, Pascal, C++, Smalltalk and Effel have notions to support Lambda calculus.

Grammar

- A **name** or variable can be any letter a, b, c, ...
- The grammar is defined based on **expression**, where expression is:
  $< \mathit{expression} >:=< \mathit{name} > \mid < \mathit{function} > \mid < \mathit{application} >$
  $< \mathit{function} >:= \lambda < \mathit{name} > . < \mathit{expression} >$
  $< \mathit{application} >:=< \mathit{expression} >< \mathit{expression} >$.
- Abbreviation
  - Ex. the function $f(x, y) = x - y$ would be written as $\lambda x.\lambda y.x - y$. A common convention is to abbreviate curried functions as $\lambda xy.x - y$.

Application

- Functions can be applied to expressions.
  - Ex. $(\lambda x.x)y$.
- Function applications are evaluated by reduction rules.
- Function application associates from the left, i.e. the expression
  $E_1 E_2 E_3 \ldots E_n$
  is evaluated as:
  $(\ldots((E_1 E_2) E_3) \ldots E_n)$.

Free and Bound Variables

- A variable <name> is **free** in an expression if one of the following three cases holds:
  - *<name> is free in <name>*
  - *<name> is free in λ <name$_1$>.<exp> if the identifier <name>$\neq$<name$_1$> and <name> is free in <exp>*
  - *<name> is free in $E_1 E_2$ if <name> is free in E_1 or if it is free in E_2.*
- A variable is **bound** if one of two cases holds:
  - *<name> is bound in λ<name$_1$>.<exp> if the identifier <name>$=$<name$_1$> or if <name> is bound in <exp>*
  - *<name> is bound in $E_1 E_2$ if <name> is bound in E_1 or if it is bound in E_2.*

Reduction Rules(1/2)

- Reduction: the process of evaluating a lambda expression.
- α-conversion: allows bound variable names to be changed.
  - Ex. α-conversion of $\lambda x.x$ would be $\lambda y.y$.
- η-conversion: two functions are the same if and only if they give the same result for all arguments. η-conversion converts between $\lambda x.fx$ and f whenever x does not appear free in f.

Reduction Rules(2/2)

- Substitution: perform variable substitution for free variables. The precise definition must be careful in order to avoid accidental variable capture and is recursively defined as follows:
  - $x[x \mapsto N] \equiv N$
  - $y[x \mapsto N] \equiv y$, if $x \neq y$
  - $(M1M2)[x \mapsto N] \equiv (M1[x \mapsto N])(M2[x \mapsto N])$
  - $(\lambda y.M)[x \mapsto N] \equiv \lambda y.(M[x \mapsto N])$, if $x \neq y$ and $!y \in fv(N)$
- β-reduction: expresses the idea of function application. The beta reduction of $((\lambda V.E)E')$ is simply $E[V \mapsto E']$.

Normal Form

- Applying reduction rules does not always stop. The following is an example, which always reduces to itself:
  - $(\lambda x.xx)(\lambda x.xx)$.
- If a sequence of reductions has come to an end where no further reductions are possible, we say that the term has been reduced to **normal form**. As illustrated, not every term has a normal form.

Numbers

- Church numerals define numbers as follows:
  - $0 := \lambda$ f x. x
  - $1 := \lambda$ f x. f x
  - $2 := \lambda$ f x. f (f x)
  - $3 := \lambda$ f x. f (f (f x))
- We define the successor function as:
  - SUCC := λ n f x. f (n f x)

Mohammad Hosein Yarmand

Arithmetic functions

- Addition:
  - λ *m n f x. n f (m f x)*
  - λ *n m. m SUCC n*
- Multiplication:
  - λ *m n f . m (n f)*
  - λ *m n. m (PLUS n) 0*
- Predecessor function is defined as
  PERD = λ *n f x. n (*λ *g h. h (g f)) (*λ *u. x) (*λ *u. u)*.
- Substraction function is defined as
  λ *m n. n PRED m.*

Logic functions

- TRUE := λ x y. x
- FALSE := λ x y. y
- Logical Operators:
  - AND := λ p q. p q p
  - OR := λ p q. p p q
  - NOT := λ p. λ a b. p b a
  - IF THEN ELSE := λ p a b. p a b
- As an example:

AND TRUE FALSE
$\equiv$ (λ p q. p q p) TRUE FALSE $\rightarrow_\beta$ TRUE FALSE TRUE
$\equiv$ (λ x y. x) FALSE TRUE $\rightarrow_\beta$ FALSE

Recursion

- Let us now construct a term for iteration to perform
  I n f x = f (f (f . . . (f x) . . .))
  $I = \lambda\ n\ f\ x.zero?\ n\ x\ (I\ (PRED\ n)\ f\ (f\ x))$
- If n = 0 then zero? n x M will evaluate to x. If n > 0 then we iterate f (n - 1)-times on the argument (f x).
- This definition of I uses I itself in the body. It does nothing else but add one further iteration to an assumed (n - 1)-fold iteration.

Loop

- We change the term on the right into a function which turns“n - 1-iterator” into“n-iterator”:
  Let S be λ *M.(λn f x.zero? n x (M (PRED n) f (f x)))*
- What we now seek is a term which is a **fixpoint** for S ,i.e. *I = S I*
- There are terms Y which construct a fixpoint for any term M, that is, they satisfy *Y M = M (Y M)*

Semantics

- In order to come up with a semantic, a set D should be found that is isomorphic to the function space $D \to D$, of functions on itself.
- The first set-theoratical model for untyped lambda calculus was made by Dana Scott in 1970s
- He introduced **continuos lattices** , that are
  - Algebraically: those complete lattices D where for every $y \in D, y = \bigvee \{\bigwedge U | y \in U \textit{ and } U \textit{ is Scott-open and } U \subseteq D\}$
  - Topologically: those T_0-spaces such that every continuos $f : X \to D$ from a subspace $X \subseteq Y$ can be extended to a continuos $\bar{f} : Y \to D$.
- This work formed the basis for the denotational semantics of programming languages, fixed point combinators, and the domain theory.

Computability & Decidability

- A function $F : N \to N$ of natural numbers is a computable function iff there exists a lambda expression f such that for every pair of x, y in N, $F(x) = y$ iff f x $\to_\beta$ y, where x and y are the Church numerals corresponding to x and y, respectively and $\to_\beta$ meaning equivalence with β-reduction.
- Equivalent to Turing machines. A calculus is **Turing-complete** if it allows one to define all computable functions from N to N.
- Undecidability of equivalence.

Confluence

- **Confluence**: the result of a computation is independent from the order of reduction.
- Theorem (Church-Rosser) If a term M can be reduced (in several steps) to terms N and P, then there exists a term Q to which both N and P can be reduced (in several steps).
- β-reduction is confluent.
- Every λ-term has at most one normal form.

Type Problem

- It is possible to write “sin log”, where the sine function is applied not to a number but to the logarithm function.
- Such terms do not make any sense at all, and any sensible programming language compiler would reject them as ill-formed.
- What is missing in the calculus is a **notion of type**. The type of a term should tell us what kind of arguments the term would accept and what kind of result it will produce.
  - For example, the type of the sine function should be “accepts real numbers and produces real number”.

References

- “A Tutorial Introduction to the Lambda Calculus”, *Raul Rojas*.
- “A short introduction to the Lambda Calculus”, *Achim Jung*.
- “Lecture Notes on the Lambda Calculus”, *Peter Selinger*.
- “History of lambda-calculus and combinatory logic”, *Felice Cardone and J. Roger Hindley*.
- http://www.csse.monash.edu.au/ lloyd/tildeFP/Lambda.
- http://www.wikipedia.org.