Lambda Calculus Part 1

Adam Wyner
CS3518, Spring 2017
University of Aberdeen

Plan of Lecture

- A Brief History of λ-Calculus and initial remarks
- Bibliography
- Syntax of λ-Calculus
- Notational conventions

A Brief History of the λ-Calculus

- The λ-calculus was invented by Alonzo Church in the 1940s as part of an attempt to develop an alternative foundation for mathematics, based on the notion of function rather than sets.
- The resulting foundation was found to be inconsistent (it contained several paradoxes), but the λ-calculus is still useful as a universal model of computation.
- The λ-calculus can be seen as a very simple and pure programming language.
- The underpinnings of functional programming lie in the λ-calculus.

Initial Remarks

- The λ-calculus is a “pure” programming language because it is free from the influence of underlying machine architectures.
- Unlike C or Java which are specifically adapted to a Von Neumann machine architecture.
- Thus, the λ-calculus (and therefore functional programming) is a good basis from which to study computation in the abstract.
- We shall look at the λ-calculus from a practical point of view, just for the aspects that are useful for the later study of Haskell, so most of the technical results (theorems, proofs, etc.) are not explicitly given.

Bibliography

- A. Church, The Calculi of λ-Conversion, Princeton U. Press, 1941.
- H.P. Barendregt, The Lambda-Calculus: its Syntax and Semantics, North Holland, 1984. (available @ QML).
- H. Barendregt and E. Barendsen, Introduction to Lambda-Calculus, Workshop on Implem. of Funct. Langs., Sweden, 1994. (available online).

Syntax of the λ-Calculus

- In mathematics, a function is a mapping from values of a set D (the domain) to values of another set R (the range):

  $f : D \rightarrow R$

- In computing, functions take inputs from D (the arguments), and "process" them onto an element of R (the output or result):

  $f(x) = x2$, where $x \in N$ and $f(x) \in N$

  $g(x,y) = 2 \times (x-y)$, where $x,y \in Z+$,but is $f(x,y) \in Z+$?

- However, in maths we use the same notation to refer both to the definition of the function as well as its result, e.g. x^2.
- The λ-calculus allows us to differentiate these concepts.

Functions and Values in the λ-Calculus

- The λ-calculus syntax allows us to refer without ambiguity to the definition of functions and the results of functions:
  - x2 stands for a value (the result of the function)
  - λx.x2 stands for the definition of the function
- It is also possible to refer to the application of a function:
  - (λx.x2) 3 = 32 = 9, where we apply the function to 3.
- We have a formal definition for the syntax of the λ-calculus.

BNF for the λ-Calculus

- We can provide a simple definition for λ-terms:

  variable ::= a|b|c| ⋯ |z
  λ-term ::= variable | (λ-term λ-term) | (λ variable . λ-term)

- Examples of λ-terms:
  x (xy) (λx.(yx)) ((λx.(yx))z) (((λx.(λy.(yx)))z)w)

- Is (λx.(yz)) a λ-term? Can you prove it?
- We call Λ the set of all λ-terms.

Notational Conventions (1)

- We allow numeric constants (e.g. 2, −900) to appear as λ-terms We also allow mathematical expressions to appear as λ-terms:

  $a + 2$ $(\lambda w.w-10^{w/}\sqrt{w}-4\ )$ $(\lambda x.(\lambda y.x^{y+1}))$ $(x(y + 1(\lambda z.a)))$
- We may use subscripts in variables (e.g. a_1, x_2, . . .). We shall omit outermost parentheses.
- Uppercase letters M, N, L, . . . (possibly subscripted) denote arbitrary λ-terms.
- We say that λx.(yx) has x as bound variable and y as free variable.
- A λ-term without free variables is called a closed term.

Notational Conventions (2)

- $M \equiv N$ means that
  - they are the same λ-term or
  - they can be obtained from each other renaming bound variables (or ...)
- Example: $(\lambda x.x)y \equiv (\lambda z.z)y$
- We use the following abbreviations to eliminate parentheses:
  - $(\cdots((FM_1)M_2)\cdots M_n) \equiv FM_1M_2 \cdots M_n$
  - $\lambda x_1.(\lambda x_2.(\cdots\lambda x_n(M))) \equiv \lambda x_1x_2 \cdots x_n.M$
- Examples: x yx $\lambda x.yx$ $(\lambda x.x2)z$ $\lambda xy.yxzw$

Notational Conventions (3)

- NB: $\lambda xy.yxzw \not\equiv (\lambda xy.yx)\ zw$ (can you see why?)
- λ-terms are also called λ-expressions.
- λ-terms are nameless functions. To name a function (just for easy reference), we can write: function ≡ λ variables . λ-term
- For example:
  - square ≡ $\lambda x.x2$
  - my-function ≡ $\lambda xy.(\sqrt{x} + y - 4xy)$
- The lambda calculus makes it explicit what the function is, which may be obscured by simply a name.

Beta-Conversion

- $(\lambda x.M)\ N \rightarrow M[N/x]$
- Condition: no free variable in N and x occurs bound in M.
- The notation M[N/x] means substitute N for x wherever x occurs in M.

Pitfall 1

- When a free variable in N occurs bound in M.
- Example where condition is not fulfilled:

  $(\lambda x.\ \lambda y.(x+y))(y) \rightarrow \lambda y.(y+y) == \lambda y.2y$
- M = $\lambda y.(x+y)$ N = y, substitute N wherever x occurs in M. That is, put y in for x in $\lambda y.(y+y)$.
- Problem is that the variable y was free in N, but becomes bound in M.

Pitfall 2

- A situation where the rule has an unexpected effect, when x does not occur free in M.
- Example:

  $(\lambda x.y)\ 5 \rightarrow y[5/x] == y$
- N is 5 and M is λx.y. Put 5 in for x wherever it appears in M.
- This conversion is allowed. The argument 5 disappears!

Abstraction

- From an expression to a function:
- Example:

  $P(a) \rightarrow \lambda x.\ P(x)$

- We can make functions from propositions, i.e. predicates with arguments.