Computer Science Theory
COMS W3261
Lecture 23

Alexander Roth

2014 – 12 – 1

Outline

1. Church numerals
2. Arithmetic
3. Logic
4. Other programming language constructs
5. The influence of the lambda calculus on functional languages

1 Church Numerals

- Church numbers are a way of represent the intergers in lambda calculus.
- Church numbers are defined as functions taking two parameters

$$\begin{aligned}
&0 \text{ is defined as } \lambda f.\lambda x.x \\
&1 \text{ is defined as } \lambda f.\lambda x.f\,x \\
&2 \text{ is defined as } \lambda f.\lambda x.f(f\,x) \\
&3 \text{ is defined as } \lambda f.\lambda x.f(f(f\,x)) \\
&n \text{ is defined as } \lambda f.\lambda x.f^n\,x
\end{aligned}$$

- n has the property that for any lambda expressions g and y, $ngy \xrightarrow{*} g^n y$. That is to say, ngy causes g to be applied to y n times.

2 Arithmetic

- In lambda calculus, arithmetic functions can be represented by corresponding operations on Church numerals.
- We can define a successor function *succ* of three arguments that adds one to its first argument:

$$\lambda n.\lambda f.\lambda x.f(n\,f,x)$$

  - Example: Let us evaluate $succ\ 2 =$

$$\begin{aligned}
&(\lambda n.\lambda f.\lambda x.f(n\,f\,x))(\lambda f'.\lambda x'.f'(f'\,x'))\\
&\rightarrow \lambda f.\lambda x.f((\lambda f'.\lambda x'.f'(f'\,x'))f\,x)\\
&\rightarrow \lambda f.\lambda x.f(\lambda x'.f(f\,x')x)\\
&\rightarrow \lambda f.\lambda x.f(f(f\,x))\\
&= 3
\end{aligned}$$

- We can define a function *add* as follows:

$$\lambda m.\lambda n.\lambda f.\lambda x.m\,f(n\,f\,x)$$

  - Example: Let us evaluate $add\ 0\,1 =$

$$\begin{aligned}
&(\lambda m.\lambda n.\lambda f.\lambda x.m\,f(n\,f\,x)0\,1\\
&\rightarrow \lambda n.\lambda f.\lambda x.0\,f\,(n\,f\,x)\,1\\
&\rightarrow \lambda f.\lambda x.0\,f(1\,f\,x)\\
&= \lambda f.\lambda x.(\lambda f'.\lambda x'.x')f(1\,f\,x)\\
&\rightarrow \lambda f.\lambda x.\lambda x'.x'(1\,f\,x)\\
&\rightarrow \lambda f.\lambda x.(1\,f\,x)\\
&= \lambda f.\lambda x.((\lambda f'.\lambda x'.f'\,x')f\,x)\\
&\rightarrow \lambda f.\lambda x.(\lambda x'.f\,x')x\\
&\rightarrow \lambda f.\lambda x.f\,x\\
&= 1
\end{aligned}$$

- We can define a function *mult* as follows:

$$\lambda m.\lambda n.\lambda f.m(n\,f)$$

  - Example: Let us evaluate $mul\ 2\,3 =$

$$\begin{aligned}
&(\lambda m.\lambda n.\lambda f.m(n\,f))2\,3\\
&\rightarrow \lambda n.\lambda f.2(n\,f)3\\
&\rightarrow \lambda f.2(3\,f)\\
&\xrightarrow{*} \lambda f.\lambda x.f(f(f(f(f(f\,x)))))\\
&= 6
\end{aligned}$$

3 Logic

- The boolean value true can be represented by a function of two arguments that always selects its first argument: $\lambda x.\lambda y.x$
- The boolean value false can be represented by a function of two arguments that always selects its second argument: $\lambda x.\lambda y.y$
- An if-then-else statement can be represented by a function of three arguments $\lambda c.\lambda i.\lambda e.c\,i\,e$ that uses its condition c to select either the if-part i or the else-part e.
  - Example: Let us evaluate if true then 1 else 0:

$$\begin{aligned}&(\lambda c.\lambda i.\lambda e.c\,i\,e)\text{true}\,1\,0\\ &\to (\lambda i.\lambda e.\text{true}\,i\,e)1\,0\\ &\to (\lambda e.\text{true}\,1\,e)0\\ &\to \text{true}\,1\,0\\ &\to (\lambda x.\lambda y.x)1\,0\\ &\to (\lambda y.1)0\\ &\to 1\end{aligned}$$

- The boolean operators and, or, and not can be implemented as follows:

$$\begin{aligned}\text{and} &= \lambda p.\lambda q.p\,q\,p\\ \text{or} &= \lambda p.\lambda q.p\,p\,q\\ \text{not} &= \lambda p.\lambda a.\lambda b.p\,b\,a\end{aligned}$$

  - Example: Let us evalue not true:

$$\begin{aligned}&(\lambda p.\lambda a.\lambda b.p\,b\,a)\text{true} && \to \lambda a.\lambda b.\text{true}\,b\,a\\ &= \lambda a.\lambda b.(\lambda x.\lambda y.x)\,b\,a\\ &\to \lambda a.\lambda b.(\lambda y.b)a\\ &\to \lambda a.\lambda b.b\\ &= \text{false (under renaming)}\end{aligned}$$

4 Other Programming Language Constructs

- We can readily implement other programming language constructs in lambda calculus. As an example, here are lambda calculus expressions for various list operations such as cons (constructing a list), head (selecting the first item from a list), and tail (selecting the remainder of a list after the first item):

$$\begin{aligned}
\text{cons} &= \lambda h.\lambda t.\lambda f.f\,h\,t \\
\text{head} &= \lambda l.l(\lambda h.\lambda t.h) \\
\text{tail} &= \lambda l.l(\lambda h.\lambda t.t)
\end{aligned}$$

Class Notes

Church Numerals

$$\begin{aligned}
0 &= \lambda f.\lambda x.x \\
1 &= \lambda f.\lambda x.f\,x \\
2 &= \lambda f.\lambda x.f(f\,x)
\end{aligned}$$

Example

$$(\lambda f.(\lambda x.(f\,x)))E\,F$$

All functions are unary.

$$\begin{aligned}
&\underset{\beta}{\rightarrow} (\lambda x.(E\,x))F \\
&\underset{\beta}{\rightarrow} (E\,F)
\end{aligned}$$

Logic

and true false

$$\begin{aligned}
&((\lambda p.\lambda q.p\,q\,p)\text{true})\text{false}) \\
&\underset{\beta}{\rightarrow} (\lambda q.\text{true}\,q\,\text{true})\text{false} \\
&\underset{\beta}{\rightarrow} \text{true false true} \\
&= ((\lambda x.\lambda y.x)\text{false true}) \\
&= (\lambda y.\text{false})\text{true} \\
&= \text{false}
\end{aligned}$$

Two lambda expressions are equivalent if you can rename one and substitute in the other with bound variables.