Computer Science Theory
COMS W3261
Lecture 22

Alexander Roth

2014 – 11 – 24

Outline

1. Normal form
2. Reduction strategies
3. The Church-Rosser theorems
4. The Y combinator
5. Implementing factorial using the Y combinator

1 Normal Form

- An expression containing no possible beta reductions is said to be in normal form. A normal form expression is one containing no redexes (reducible expressions), that is, one with no subexpressions of the form $(\lambda x.f\,g)$.
- Examples of normal form expressions:
  - x where x is a variable.
  - $x\,e$ where x is a variable and e is a normal form expression.
  - $\lambda x.e$ where x is a variable and e is a normal form expression.
- The expression $(\lambda x.x\,x)(\lambda x.x\,x)$ does not have a normal form because it is a redex that always evaluates to itself. We can think of this expression as a representation for an infinite loop.

2 Reduction Strategies

- A reduction strategy specifies the order in which beta reductions for a lambda expression are made.
- Some reduction orders for a lambda expression may yield a normal form while other orders may not. For example, consider the given expression

  $$(\lambda x.1)((\lambda x.x\,x)(\lambda x.x\,x))$$

  This expression has two redexes:

  1. The entire expression is a redex in which we can apply the function $(\lambda x.1)$ to the argument $((\lambda x.x\,x)(\lambda x.x\,x))$ to yield the normal form 1. This redex is the leftmost outermost redex in the given expression.
  2. The subexpression $((\lambda x.x\,x)(\lambda x.x\,x))$ is also a redex in which we can apply the function $(\lambda x.x\,x)$ to the argument $(\lambda x.x\,x)$. Note that this redex is the leftmost innermost redex in the given expression. But if we evaluate this redex we get same subexpression: $(\lambda x.x\,x)(\lambda x.x\,x) \rightarrow (\lambda x.x\,x)(\lambda x.x\,x)$. Thus, continuing to evaluate the leftmost innermost redex will not terminate and no normal form will result.

- As a second example, consider the expression

  $$(\lambda x.\lambda y.y)((\lambda z.z\,z)(\lambda z.z\,z))$$

  This expression has two redexes:

  1. The entire expression is a redex in which we apply the function $(\lambda x.\lambda y.y)$ to the argument $((\lambda z.z\,z)(\lambda z.z\,z))$ to yield the normal form $(\lambda y.y)$. This redex is the leftmost outermost redex in the given expression.
  2. The subexpression $((\lambda z.z\,z)(\lambda z.z\,z))$ is also a redex in which we apply the function $(\lambda z.z\,z)$ to the argument $(\lambda z.z\,z)$. Note that this redex is the leftmost innermost redex in the given expression. But if we evaluate this redex we get the same subexpression: $((\lambda z.z\,z)(\lambda z.z\,z)) \rightarrow ((\lambda z.z\,z)(\lambda z.z))$. Thus, continuing to evaulate the leftmost innermost redex will not terminate and no normal will result.

- There are two common reduction orders for lambda expression: normal order evaluation and applicative order evaluation

  Normal order evaluation :

  - In normal order evaluation we always reduce the leftmost outermost redex at each step.
  - The first reduction order in each of the two examples above is a normal order evaluation.

- A remarkable property of lambda calculus is that every lambda expression has a unique normal form if one exists. Moreover, if an expression has a normal form, then normal order evaluation will always find it.

Applicative order evaluation :

- In applicative order evaluation we always reduce the leftmost innermost redex at each step.
- Applicative order evaluates the arguments of a function before evaluating the function itself.
- The second reduction order in each of the two examples above isa n applicative order evaluation.
- Thus, even though an expression may have a normal form, applicative order evaluation may fail to find it.

3 The Church-Rosser Theorems

- A remarkable property of lambda calculus is that every expression has a unique normal form if one exists.
- **Church-Rosser Theorem I:** If $e \xrightarrow{*} f$ and $e \xrightarrow{*} g$ by any two reduction orders, then there always exists a lambda expression h such that $f \xrightarrow{*} h$ and $g \xrightarrow{*} h$.
  - A corollary of this theorem is that no lambda expression can be reduced to two distinct normal forms. To see this, suppose f and g are in normal form. The Church-Rosser theorem says there must be an expression h such that f and g are each reducible to h. Since f and g are in normal form, they cannot have any redexes so $f = g = h$.
  - This corollary says that all reduction sequences that terminate will always yield the same result and that result must be a normal form.
  - The term *confluent* is often applied to a rewriting system that has the Church-Rosser property.
- **Church-Rosser Theorem II:** If $e \xrightarrow{*} f$ and f is in normal form, then there exists a normal order reduction sequence from e to f.

4 The Y Combinator

- The Y combinator (sometimes called the paradoxical combinator) is a function that takes a function G as an argument and returns $G(YG)$. With repeated applications we can get $G(G(YG)), G(G(G(YG))), \ldots$.
- We can implement recursive functions using the Y combinator.

- Y is defined as follows:

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

- Let us evaluate YG where G is an expression:

$$\begin{aligned}(\lambda f.(\lambda x.f(x\,x))(\lambda x'.f(x\prime\,x')))G &\rightarrow (\lambda x.G(x\,x))(\lambda x'.G(x'\,x'))\\ &\rightarrow G((\lambda x'.G(x'\,x'))(\lambda x'.G(x'\,x')))\\ &\leftrightarrow G((\lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x)))G)\\ &= G(YG)\end{aligned}$$

- Thus, $YG \xrightarrow{*} G(YG)$; that is, YG reduces to a call of G on (YG).
- We will use Y to implement recursive functions.
- Y is an example of a fixed-point combinator.

5 Implementing Factorial using the Y Combinator

- If we could name lambda abstractions, we could define the factorial function with the following recursive definition:

  $$FAC = (\lambda n.IF\,(=\ n\,0)\,1\,(*\,n\,(FAC\,(-\,n\,1))))$$

  where IF is a conditional function.

- However, functions in lambda calculus cannot be named; they are anonymous.
- But we can express recursion as the fixed-point of a function G. To do this, let us simplify the essence of the problem. We begin with a skeletal recursive definition:

  $$FAC = \lambda n.(\ldots FAC \ldots)$$

- By performing the beta abstraction on FAC, we can transform its definition to:

  $$\begin{aligned}FAC &= (\lambda f.(\lambda n.(\ldots f \ldots)))\,FAC\\ &= G\,FAC\end{aligned}$$

  where

  $$G = \lambda f.\lambda n.IF\,(=\ n\,0)\,1\,(*\,n\,(f\,(-\,n\,1)))$$

  Beta abstraction is just the reverse of beta reduction.

- The equation
$$FAC = G\,FAC$$
says that when the function G is applied to FAC, the result is FAC. That is, FAC is a fixed-point of G.

- We can use the Y combinator to implement FAC :
$$FAC = Y\,G$$

- As an example, let's compute $FAC\,1$:

$$\begin{aligned}
FAC\,1 &= Y\,G\,1 \\
&= G\,(Y\,G)\,1 \\
&= \lambda f.\lambda n.IF\,(=\,n\,0)\,1\,(*\,n\,(f\,(-\,n\,1)))(Y\,G)\,1 \\
&\to \lambda n.IF\,(=\,n\,0)\,1\,(*\,n\,((Y\,G)(-\,n\,1)))\,1 \\
&\to IF(=\,n\,0)\,1\,(*\,n\,((Y\,G)(-\,1\,1))) \\
&\to *\,1\,(Y\,G\,0) \\
&= *\,1\,(G(Y\,G)\,0) \\
&= *\,1((\lambda f.\lambda n.IF\,(=\,n\,0)\,1\,(*\,n\,(f\,(-\,n\,1))))(Y\,G)\,0) \\
&\to *\,1((\lambda n.IF\,(=\,n\,0)\,1\,(*\,n\,((Y\,G)(-\,n\,1)))))0 \\
&\to *\,1(IF\,(=\,0\,0)\,1\,(*\,0\,((Y\,G)(-\,0\,1)))) \\
&\to *\,1\,1 \\
&\to 1
\end{aligned}$$

Class Notes

Beta Reduction

$$\begin{aligned}
&(\lambda x.\lambda y.+\,x\,y)\,1\,2 \\
&(+\,x\,y) \\
&(\lambda y.(\lambda y.(+\,x\,y))) \\
&((\lambda x.(\lambda y.(+\,x\,y)))1)2 \\
&(\lambda y.(+\,1\,y))2 \\
&(+\,1\,2)
\end{aligned}$$

Argument is a Function

$$
\begin{aligned}
&(\lambda f.f\,1)(\lambda x.(+\,x\,2))\\
&\xrightarrow{\beta} (\lambda x.(+\,x\,2))1\\
&\xrightarrow{\beta} (+\,1\,2)
\end{aligned}
$$

Rename to avoid name conflicts

$$
\begin{aligned}
&((\lambda x.(\lambda y.(x\,y)))y)\\
&\xrightarrow{\alpha} ((\lambda x.(\lambda z.(x\,z)))y)\\
&\xrightarrow{\beta} (\lambda z.(y\,z))
\end{aligned}
$$

The Lambda Calculus II

Normal form: A lambda expression with no redexes

Examples

- x
- λy
- $\lambda x.y$

Consider

$$
\begin{aligned}
&(\lambda x.x\,x)(\lambda x.x\,x)\\
&\xrightarrow{\beta} (\lambda x.x\,x)(\lambda x.x\,x)
\end{aligned}
$$

Infinite loop.

Reduction Strategy

The order in which beta reductions are made.

$$e = ((\lambda x.f)g)$$

eh: The lambda in e is to the left of any lambda in h, f, g.