

Lambda-Calculus (cont): Fixpoints, Naming

Lecture 10

CS 565

2/10/08

Recursion and Divergence

- Consider the application:

$$\Omega \equiv ((\lambda x. (x x)) (\lambda x. (x x)))$$

Ω evaluates to itself in one step.

It has no normal form.

A lambda term is in normal form if it does not contain any redex (i.e., a term that is subject to β -reduction)

Now, consider: $Y \equiv ((\lambda x. (f (x x))) (\lambda x. (f (x x))))$

$Y \rightarrow$

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

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

...

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

Normal forms and order of evaluation

- No expression can be converted to two distinct normal forms (Church-Rosser Theorem 1)
- Is there an order of evaluation guaranteed to terminate whenever a particular expression is reducible to normal form?
 - Normal-order: leftmost, outermost reduction: no expression in the argument position of a redex is reduced until the redex is reduced
 - If there is a reduction from A to B and B is in normal form, then there exists a normal order reduction from A to B (Church-Rosser Theorem 2)

Recursion

- The previous definition applies f an infinite number of times
 - Basis for iterated application
 - But, how can we slow its rate of unfolding?

- Consider:

$$\Omega_v \equiv (\lambda y. ((\lambda x. (\lambda y. (x \times y))))$$

$$(\lambda x. (\lambda y. (x \times y)))$$
$$y))$$

Ω_v is in normal form. However, if it is applied to an argument it diverges.

Recursion (cont)

$(\Omega_v v) \rightarrow$

$((\lambda y. ((\lambda x. (\lambda y. (x \times y))))$
 $(\lambda x. (\lambda y. (x \times y))))$
 $y) v) \rightarrow$

$((\lambda y. ((\lambda x. (\lambda y. (x \times y))))$
 $(\lambda x. (\lambda y. (x \times y))))$
 $y) v) \rightarrow$

...

Recursion (cont)

Now, consider

$$Z_f \equiv (\lambda y. ((\lambda x. (f (\lambda y. (x \times y)))) \\ (\lambda x. (f (\lambda y. (x \times y))) \\ y)))$$

If we apply Z_f to an argument:

$$((\lambda y. ((\lambda x. (f (\lambda y. (x \times y)))) \\ (\lambda x. (f (\lambda y. (x \times y))) \\ y)) v) \rightarrow$$

$$(f (\lambda y. ((\lambda x. (f (\lambda y. (x \times y)))) \\ (\lambda x. (f (\lambda y. (x \times y))) \\ y)) v) \rightarrow$$

Since the arguments to f are all values, this expression is equivalent to: $f Z_f v$

Recursion (cont)

How do we apply these insights?

$f \equiv \lambda \text{ fact.}$

$\lambda n. \text{ if } n = 0$
 $\text{ then } 1$
 $\text{ else } n * (\text{fact } (n - 1))$

We can use Z_f to turn f into a real factorial function:

Fixpoints

$Z_f\ 3 \rightarrow$

$f\ Z_f\ 3 \rightarrow$

$(\lambda\ \text{fact.}\ \lambda\ n.\ \dots)\ Z_f\ 3 \rightarrow$

$\text{if } 3 = 0 \text{ then } 1 \text{ else } 3 * (Z_f\ 2) \rightarrow$

$3 * (f\ Z_f\ 2)$

...

We'll stop when $n = 0$

Fixpoints

Define $Z = \lambda f. Zf$

Now, Z defines a fixpoint for any f :

$$Z \equiv \lambda f. (\lambda y. ((\lambda x. (f (\lambda y. (x \ x \ y)))) \\ (\lambda x. (f (\lambda y. (x \ x \ y)))) \\ y))$$

Z computes the least fixpoint of a function.

Fixpoints and order of evaluation

- Consider an alternative definition:
 - $Y \equiv \lambda h. (\lambda x. h(x x)) (\lambda x. h(x x))$
 - What happens if we apply Y to f (the factorial functional) with argument 3?
 - Under normal-order evaluation:
 - $Y f 3 \equiv (\lambda x. f(x x)) (\lambda x. f(x x)) 3 \rightarrow$
 - $f ((\lambda x. f(x x)) (\lambda x. f(x x))) 3$
 - What happens under applicative-order?

Naming and substitution

- Although we claimed that lambda calculus essentially manipulates functions (it does), we've spent a lot of time thinking about variables
 - substitutions
 - free variables
 - equivalence upto renaming
- Implementations must consider these issues seriously
 - Rename bound variables when performing substitutions with "fresh" names.
 - Impose a condition that all bound variables be distinct from each other, and other free variables.
 - Derive a canonical representation that does not require renaming at all.

Terms and Contexts

- De Bruijn indices:
 - Have variable occurrences “point” directly to their binders rather than referring to them by name.
 - Do so by replacing variable occurrences with numbers:
 - number k stands for “the variable bound by the k^{th} enclosing λ -term
 - Example: $\lambda x. \lambda y. x (y x) \equiv \lambda . \lambda . 1 (0 1)$
 - Similar to static offsets in an activation record or display.

Examples

$\text{identity} \equiv \lambda x. x \equiv \lambda . 0$

$\text{true} \equiv \lambda x. \lambda y. x \equiv \lambda . \lambda . 1$

$\text{false} \equiv \lambda x. \lambda y. y \equiv \lambda . \lambda . 0$

$\text{two} \equiv \lambda s. \lambda z. s (s z) \equiv \lambda . \lambda . (1 (1 0))$

Contexts

- How do we replace free variables with their binders?
 - Assume an ordered context listing all free variables that can occur, and map free variables to their index in this context (counting right to left)

- Context: a, b

- $a \mapsto 1, b \mapsto 0$

- $\lambda x. a \equiv \lambda . 2$

- $\lambda x. b \equiv \lambda . 1$

- $\lambda x. b (\lambda y. a) \equiv \lambda . 1(\lambda . 3)$

Shifting and substitution

When substituting into a λ term, indices must be adjusted:

$\lambda y. x [z/x]$ in context x,y,z

$[2 \mapsto 0] \lambda. 2 \equiv \lambda . [3 \mapsto 1] 3 \equiv \lambda . 1$

Key point: context becomes longer when substituting inside an abstraction. Need to be careful to adjust free variables, not bound ones.

$\text{shift}(d,c)(k) = k$ if $k < c$

$k + d$ if $k \geq c$

$\text{shift}(d,c)(\lambda.t) = (\lambda.\text{shift}(d,c+1)(t))$

$\text{shift}(d,c)(t_1 t_2) = (\text{shift}(d,c)(t_1))(\text{shift}(d,c)(t_2))$

Example

□ $\text{shift}(2,0)(\lambda.\lambda. 1 (0 2))$

$\lambda.\lambda. 1 (0 4)$

□ $\text{shift}(2,0)(\lambda. 0 1 (\lambda. 0 1 2))$

$\lambda. 0 3 (\lambda. 0 1 4)$

Substitution

$$[j \mapsto s] k = \begin{cases} s & \text{if } k = j \\ k & \text{otherwise} \end{cases}$$

$$[j \mapsto s](\lambda . t) = \lambda . [j+1 \mapsto \text{shift}(1,0)s] t$$

$$[j \mapsto s](t_1 t_2) = ([j \mapsto s] t_1) ([j \mapsto s] t_2)$$

Beta-reduction:

$$(\lambda t) v \rightarrow \text{shift}(-1,0)([0 \mapsto \text{shift}(1,0)(v)] t)$$

Examples

- Assume context $\langle a, b \rangle$
 - Then, $a \mapsto 1, b \mapsto 0$

- $[a / b] b \lambda x. \lambda y. b$
 $[0 \mapsto 1] 0 \lambda . \lambda . 2$
 $1 \lambda . \lambda . 3 \equiv a \lambda x. \lambda y. a$

- $[(a (\lambda z. a)) / b] (b (\lambda x. b))$
 $[0 \mapsto (1 (\lambda . 2))] (0 \lambda . 1)$
 $1 (\lambda . 2) (\lambda . (2 (\lambda . 3)))$
 $(a (\lambda z. a)) (\lambda x. (a (\lambda z. a)))$

Examples

- $[a/b] (\lambda b. (b a))$
 $[0 \mapsto 1] (\lambda . (0 2))$
 $(\lambda . (0 2))$
 $(\lambda b. (b a))$
- $[a/b] (\lambda a. (b a))$
 $[0 \mapsto 1] \lambda . (1 0)$
 $\lambda . (2 0)$
 $(\lambda a'. a a')$