Lecture 3: Typed & Untyped Lambda Calculus

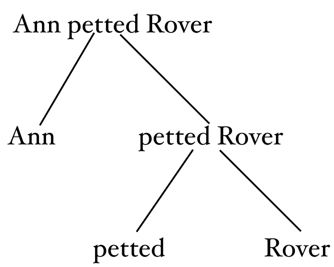
CS 181
Spring 2016
Kim Bruce

Some slide content taken from Unger and Michaelis

Lambda Calculus in Semantics

- Meanings of words and phrases will be functions, with meaning found by evaluating constituent parts and then using function application to determine meaning.

Example



- Ann petted Rover:
  - I (Ann) = *ann*
  - I (Rover) = *rover*
  - I (petted) = $\lambda o.\ \lambda s.\ petted\ (s, o)$
  - where
    - *ann, rover* represent elements of the domain of the model, and
    - *petted(s,o)* represents a binary relation on the model.
  - Then I (petted rover) = $(\lambda o.\ \lambda s.\ petted\ (s, o))\ rover$
    $= \lambda s.\ petted\ (s, rover)$
  - Then I (Ann petted Rover) = $(\lambda s.\ petted\ (s, rover))\ ann$
    $= petted\ (ann, rover)$

Lambda Calculus for Semantics

- Use higher-order functions that, when applied, return other functions.
- While could name the functions representing meanings of words, intermediate functions don't have name.
- Lambda calculus provides a way of writing the functions without naming them.

Pure Lambda Calculus

- Terms of pure lambda calculus
  - M ::= v | (M M) | λv. M   *where v stands for a variable*
  - Pure lambda calculus is Turing-complete
- Left associative: M N P = (M N) P.
- λx,y. M abbreviates λx. λy. M
- Application has higher precedence than abstraction: λx. M N abbreviates λx. (M N)

When are functions the same?

- Which of these are the same?
  - $f_1(x) = (x + 1)^2$
  - $f_2(y) = (y + 1)^2$
  - $f_3(x) = x^2 + 2x + 1$
  - $f_4(y) = y^2 + 2y + 1$
  - All give the same answers for same inputs, but some represent different algorithms
  - Say f_1 and f_2 are the same, as are f_3 and f_4
- Formalize these …

Free Variables

- First look at substitution.
  - Why?
  - Substitution easy to mess up!
- Def: If M is a term, then FV(M), the collection of free variables of M, is defined as follows:
  - FV(x) = { x }
  - FV(M N) = FV(M) ∪ FV(N)
  - FV(λv. M) = FV(M) - {v}

Bound Variables

- In a formula λx. F, the lambda binds all occurrences of x in F that are not already bound by an occurrence of λx in F.
- Examples:
  - (λx. λy. (x z (λx. x w y))) y
    - First λx binds first two x's, next binds last two
    - λy binds first two y's
    - w, z, and last occurrence of y are free

Substitution

- Write [N/x] M to denote result of replacing all free occurrences of x by N in expression M.
- More carefully (& recursively):
  - [N/x] x = N,
  - [N/x] y = y, if $y \neq x$,
  - [N/x] (L M) = ([N/x] L) ([N/x] M),
  - [N/x] (λy. M) = λy. ([N/x] M), if $y \neq x$ and $y \notin FV(N)$,
  - [N/x] (λx. M) = λx. M. *No substitution since no x is free!*

Computation Rules

- Reduction rules for lambda calculus:

  (α) λx. M $\rightarrow_\alpha$ λy. ([y/x] M), if $y \notin FV(M)$.

  change name of parameters if new not capture old

  (β) (λx. M) N $\rightarrow_\beta$ [N/x] M.

  computation by substituting function argument for formal parameter

  (η) λx. (M x) $\rightarrow_\eta$ M.

  Optional rule to get rid of excess λ's

Equivalences

- Generalize: Write $M =_\alpha N$ iff
  1. $M \rightarrow_\alpha N$ or $N \rightarrow_\alpha M$, or
  2. There is an M' s.t. $M \rightarrow_\alpha M'$ or $M' \rightarrow_\alpha M$, and $M' =_\alpha N$
- Equivalently, take reflexive, symmetric, and transitive closure of $\rightarrow_\alpha$
- Similarly for $M =_\beta N$ and $M =_{\alpha\beta} N$

Keeping Out of Trouble!

- Use variable convention: In a term M, ensure:
  - all bound variables are distinct from all free ones, and
  - all lambdas bind different variables
  - E.g. if have (λx.(y (λy.(x (y (λy.(x y))))))), rewrite as: (λu.(y (λv.(u (v (λw.(u w))))))) using α-equivalence before doing any reductions.

Computing w/Lambda Calculus

- Consider terms that are α-equivalent as the same, and compute using β- (and η-) reduction.
- Let $\rightarrow$ abbreviate $\rightarrow_{\beta\eta}$ and define $\Rightarrow$:
  - if M $\rightarrow$ M' then M $\Rightarrow$ M'
  - if M $\Rightarrow$ M' then (M N) $\Rightarrow$ (M' N)
  - if N $\Rightarrow$ N' then (M N) $\Rightarrow$ (M N')
  - if M $\Rightarrow$ M' then (M) $\Rightarrow$ (M' N)
- Called compatible closure and will be used when programming interpreter