Lecture 5: Lambda Calculus

CSC 131
Fall, 2011

Kim Bruce

See lambda calculus cheat sheet on "links" page of course web page!

Pure Lambda Calculus

- Terms of pure lambda calculus
  - M ::= v | (M M) | λv. M
  - Provide theoretical foundations for functional languages.
  - Turing-complete
- Reduction rules for lambda calculus:

  (α) λx. M → λy. ([y/x] M), if y ∉ FV(M).

  (β) (λx. M) N → [N/x] M.

  (η) λx. (M x) → M. *Optional rule*

Computability

- Can encode all computable functions in pure untyped lambda calculus.
  - true = λ u. λ v. u
  - false = λ u. λ v. v
  - cond = λ u. λ v. λ w. u v w

Lambda Encoding

- Pairing:
  - Pair = λ m. λ n. λ b. cond b m n.
  - fst = λp. p true
  - snd = λp. p false

Encoding Integers

- Integers:
  - 0 = λ s. λ z. z.
  - 1 = λ s. λ z. s z.
  - 2 = λ s. λ z. s (s z).
- Integers encode repetition:
  - 2 f x = f (f x)
  - n f x = $f^{(n)}$ (x)

Arithmetic

- Succ = λ n. λ s. λ z. s (n s z)
  - Succ $\underline{n}$ = λ s. λ z. s ($\underline{n}$ s z) = λ s. λ z. s ($s^{(n)}$ z) = $\underline{n+1}$
- Plus = λ n. λ m. λ s. λ z. m s (n s z).
- Mult = λ n. λ m. (m (Plus n) $\underline{0}$).
- isZero = λ n. n (λ x. false) true
- Subtraction is hard!!

Predecessor

- PZero = <0,0> = Pair $\underline{0}$ $\underline{0}$
- PSucc = λ n. Pair (snd n) (Succ (snd n))
  - PSucc PZero = <0,1>
  - n PSucc PZero = <n-1,n> *for n > 0*
- Pred = λ n. fst (n PSucc PZero)
  - Pred n = n - 1, *for n > 0,*
  - Pred $\underline{0}$ = $\underline{0}$

Recursion

- Recursive definitions are handy
  - fact = λn. cond (isZero n) 1 (Mult n (fact (Pred n)))
  - *Not* a definition because can't name functions!
- Let F stand for the closed term:
  - λf. λn. cond (isZero n) 1 (Mult n (f (Pred n)))
  - Notice F(fact) = fact.
  - fact is a *fixed point* of F
  - To find fact, need only find fixed point of F!

Fixed Points

- Several fixed point operators:
  - Ex: Y = λf . (λx. f (xx))(λx. f (xx))
- Claim for all g, Y g = g (Y g)

  Y g = λf . (λx. f (xx))(λx. f (xx)) g

  = (λx. g(xx))(λx. g(xx))

  = g((λx. g(xx))(λx. g(xx)))

  = g (Y g)
- If let x_0 = Y g, then g (x_0) = x_0.

Factorial

- Recursive definition:

- let F = λf. λn. cond (isZero n) 1 (Mult n (f (Pred n)))
- let fact = Y F
- then F(fact) = fact

- Compute:

fact $\underline{0}$ = (F (fact)) $\underline{0}$ *because fact is a fixed point of F*

= cond (isZero $\underline{0}$) $\underline{1}$ (Mult $\underline{0}$ (fact (Pred $\underline{0}$)))

= 1 *by the definition of cond*

Computing Factorials

fact $\underline{1}$ = (F (fact)) $\underline{1}$ *because fact is a fixed point of F*

= (λn. cond (isZero n) 1 (Mult n (fact (Pred n)))) $\underline{1}$ *expanding F*

= cond (isZero $\underline{1}$) $\underline{1}$ (Mult $\underline{1}$ (fact (Pred $\underline{1}$))) *applying it*

= Mult $\underline{1}$ (fact (Pred $\underline{1}$)) *by the definition of cond*

= fact $\underline{0}$ *by the definition of Mult and Pred*

= $\underline{1}$ *by the above calculation*

Some History

Lambda Calculus

- λ-calculus invented in 1928 by Church in Princeton & first published in 1932.
- Goal to provide a foundation for logic
- First to state explicit conversion rules.
- Original version inconsistent, but corrected
- 1933, definition of natural numbers

Collaborators

- 1931-1934: Grad students:
  - J. Barkley Rosser and Stephen Kleene
  - Church-Rosser confluence theorem ensured consistency (earlier version inconsistent)
  - Kleene showed λ-definable functions very rich
    - Equivalent to Herbrand-Gödel recursive functions
    - Equivalent to Turing-computable functions.
    - Founder of recursion theory, invented regular expressions

- Church's thesis:
  - λ-definability ≡ effectively computable

Undecidability

- Convertibility problem for λ-calculus undecidable.
- Validity in first-order predicate logic undecidable.
- Proved independently year later by Turing.
  - First showed halting problem undecidable

Alan Turing

- Turing
  - 1936, in Cambridge, England, definition of Turing machine
  - 1936-38, in Princeton to get Ph.D. under Church.
  - 1937, first published fixed point combinator
  - *Kleene did not use fixed-point operator in defining all recursive functions on natural numbers!*
  - Broke German enigma code in WW2, Turing test AI
  - Persecuted as homosexual, committed suicide in 1954

Combinatory Logic

- Schönfinkel - get rid of bound variables
- Combinators K, S sufficient to write all computable functions:
  - K x y = x
  - S x y z = (x y)(x z)
- CL terms include K, S, variables and all terms created by function application.
- All computable fcns, equivalent to λ-calculus

Typed Lambda Calculus

Types

- Start with base type e and build up types and terms:
  - Type ::= e | Type → Type
  - M ::= v | (M M) | λv : Type. M
- Examples:
  - Types: e, e → e, e → e → e, (e → e) → e, ...
  - Terms: λx: e. x, λf: e → e. λz: e. f(f(z))

Definitions

- Earlier definitions generalize over types t:
  - $true^t$ = λx:t. λy:u. x
  - $\underline{n}^t$ = λs: t→t. λz: t. $s^{(n)}(z)$
- Some untyped terms can't be typed:
  - Ω = (λx. (x x))(λx. (x x))
  - Y = λf . (λx. f (x x))(λx. f (x x))

Totally Awesome!!

- Theorem: If M is a term of the typed lambda calculus, then M has a unique normal form. *I.e., every term of the typed lambda calculus is total.*
- Corollary: The typed lambda calculus does *not* include all computable functions.