

λ Calculus

Madhavan Mukund

Chennai Mathematical Institute

`madhavan@cmi.ac.in`

PLC, 10 March 2017

λ -calculus: syntax

- ▶ Assume a set Var of variables
- ▶ Set Λ of lambda expressions is given by

$$\Lambda = x \mid \lambda x.M \mid MM'$$

where $x \in Var$, $M, M' \in \Lambda$.

- ▶ Basic rule for computing (rewriting) is called β

$$(\lambda x.M)M' \rightarrow_{\beta} M\{x \leftarrow M'\}$$

- ▶ $M\{x \leftarrow M'\}$: substitute free occurrences of x in M by M'
- ▶ When we apply β to MM' , assume that we always rename the bound variables in M to avoid “capturing” free variables from M' .

Encoding arithmetic

Church numerals

$$\begin{aligned}\langle 0 \rangle &= \lambda f x. x \\ \langle n + 1 \rangle &= \lambda f x. f(\langle n \rangle f x)\end{aligned}$$

For instance

$$\langle 1 \rangle = \lambda f x. f(\langle 0 \rangle f x) = \lambda f x. (f((\lambda f x. x) f x))$$

Note that $\langle 0 \rangle g y \rightarrow_{\beta} (\lambda x. x) y \rightarrow_{\beta} y$.

Hence

$$\langle 1 \rangle = \dots = \lambda f x. (f(\underbrace{(\lambda f x. x) f x}_{\text{apply } \beta})) \rightarrow_{\beta} \lambda f x. (f x)$$

So $\langle 1 \rangle g y \rightarrow_{\beta} (\lambda x. (g x)) y \rightarrow_{\beta} g y$

Church numerals . . .

$$\langle 2 \rangle = \lambda fx.f(\langle 1 \rangle fx) = \lambda fx.f(\underbrace{(\lambda fx.(fx)fx)}_{\text{apply } \beta}) \rightarrow_{\beta} \lambda fx.(f(fx))$$

so,

$$\langle 2 \rangle gy \rightarrow_{\beta} \lambda x.(g(gx))y = g(gy)$$

- ▶ Let $g^k y$ denote $g(g(\dots(gy)))$ with k applications of g to y
- ▶ Show by induction that

$$\langle n \rangle = \lambda fx.f(\langle n-1 \rangle fx) \rightarrow_{\beta} \dots \rightarrow_{\beta} \lambda fx.(f^n x)$$

Encoding arithmetic functions ...

Successor

- ▶ $\text{succ}(n) = n + 1$
- ▶ Define as $\lambda p f x. f(p f x)$

$$\begin{aligned} (\lambda p f x. f(p f x)) \langle n \rangle &\rightarrow_{\beta} \lambda f x. f(\langle n \rangle f x) \rightarrow_{\beta} \lambda f x. f(f^n x) = \lambda f x. f^{n+1} x \\ &= \langle n+1 \rangle \end{aligned}$$

plus: $\lambda p q f x. p f (q f x)$.

$$\begin{aligned} (\lambda p q f x. p f (q f x)) \langle m \rangle \langle n \rangle &\rightarrow_{\beta} (\lambda q f x. \langle m \rangle f (q f x)) \langle n \rangle \\ &\rightarrow_{\beta} (\lambda f x. \langle m \rangle f (\langle n \rangle f x)) \\ &\rightarrow_{\beta} (\lambda f x. \langle m \rangle f (f^n x)) \\ &\rightarrow_{\beta} (\lambda f x. f^m (f^n x)) \\ &= (\lambda f x. f^{m+n} x) = \langle m+n \rangle \end{aligned}$$

Encoding arithmetic functions ...

- ▶ If functions are applied to **meaningful** terms we get meaningful answers!

Other functions:

multiplication : $\lambda p q f x. q(p f) x$

exponentiation : $\lambda p q. (p q)$

Computability

- ▶ Church numerals encode $n \in \mathbb{N}$
- ▶ Can we encode computable functions $f : \mathbb{N}^k \rightarrow \mathbb{N}$?
 - ▶ Let $\langle f \rangle$ be the encoding of computable function f
 - ▶ Want $\langle f \rangle \langle n_1 \rangle \langle n_2 \rangle \dots \langle n_k \rangle \rightarrow^* \langle f(n_1, n_2, \dots, n_k) \rangle$
 - ▶ Note! currying ...
- ▶ We must first decide on a syntax for computable functions

Recursive functions

Recursive functions [Gödel]

- ▶ Equivalent to Turing machines, ...

Initial functions

- ▶ Zero: $Z(n) = 0$.
- ▶ Successor: $S(n) = n+1$.
- ▶ Projection: $\Pi_i^k(n_1, n_2, \dots, n_k) = n_i$

Composition Given $f : \mathbb{N}^k \rightarrow \mathbb{N}$ and
 $g_1, g_2, \dots, g_k : \mathbb{N}^h \rightarrow \mathbb{N}$,

$$f \circ (g_1, g_2, \dots, g_k)(n_1, n_2, \dots, n_h) = \\ f(g_1(n_1, n_2, \dots, n_h), g_2(n_1, n_2, \dots, n_h), \dots, g_k(n_1, n_2, \dots, n_h))$$

For instance, $f(n) = n + 2$ is $S \circ S$

Recursive functions ...

Primitive recursion Given $g : \mathbb{N}^k \rightarrow \mathbb{N}$ and
 $h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$

define $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ by primitive recursion as follows:

$$\begin{aligned}f(0, n_1, n_2, \dots, n_k) &= g(n_1, n_2, \dots, n_k) \\f(n+1, n_1, \dots, n_k) &= h(n, f(n, n_1, n_2, \dots, n_k), n_1, \dots, n_k)\end{aligned}$$

Examples

- Define $plus(n, m) = n+m$ from $g = \Pi_1^1$
 $h = S \circ \Pi_2^3$

$$\begin{aligned}plus(0, n) &= g(n) &&= \Pi_1^1(n) \\ & &&= n\end{aligned}$$

$$\begin{aligned}plus(m+1, n) &= h(m, plus(m, n), n) \\ &= S \circ \Pi_2^3(m, plus(m, n), n) = S(plus(m, n))\end{aligned}$$

Recursive functions ...

Primitive recursion

Given $g : \mathbb{N}^k \rightarrow \mathbb{N}$ and

$$h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$$

define $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ by primitive recursion as follows:

$$f(0, n_1, n_2, \dots, n_k) = g(n_1, n_2, \dots, n_k)$$

$$f(n+1, n_1, \dots, n_k) = h(n, f(n, n_1, n_2, \dots, n_k), n_1, \dots, n_k)$$

Examples

- ▶ Define $times(n, m) = n \cdot m$ from $g = Z$
 $h = plus \circ (\Pi_3^3, \Pi_2^3)$

Note Primitive recursive functions are total!

Recursive functions ...

Minimalization

Given $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$, define $f : \mathbb{N}^k \rightarrow \mathbb{N}$ by minimalization from g

$$f(n_1, n_2, \dots, n_k) = \mu n. (g(n, n_1, n_2, \dots, n_k) = 0)$$

where $\mu n.P(n)$ returns the least natural number n such that $P(n)$ holds

Equivalent to computing a while loop

```
n := 0;
while (g(n,n1,n2,...,nk) != 0) {n := n+1};
return n;
```

Define $\log_2 n$ as $\mu k.(n - 2^k)$

- ▶ First k for which $n - 2^k = 0$ is $\log_2 n$

Not defined for all n !

Encoding recursive functions ...

- ▶ $\langle n \rangle \equiv \lambda fx.(f^n x)$.
- ▶ Successor $succ \equiv \lambda nfx.(f(nfx))$ such that $succ \langle n \rangle \rightarrow^* \langle n + 1 \rangle$.
- ▶ Zero $\langle Z \rangle \equiv \lambda x.(\lambda gy.y)$.
- ▶ Projection $\langle \Pi_i^k \rangle \equiv \lambda x_1 x_2 \dots x_k . x_i$.

Composition is easy

Encoding recursive functions . . .

Primitive recursion

- ▶ Assume $f(n+1)$ is defined in terms of g and $h(n, f(n))$
- ▶ Main difficulty is to eliminate recursion
 - ▶ λ -calculus functions are anonymous
 - ▶ Cannot directly use name of f inside definition of f
- ▶ Convert recursion into iteration

Define $t(n) = (n, f(n))$

- ▶ Functions fst and snd extract first and second component of a pair

$$\begin{aligned}t(0) &= (0, f(0)) &= (0, g) \\t(n+1) &= (n+1, f(n+1)) &= (n+1, h(n, f(n))) \\& &= (succ(fst(t(n))), \\& & \quad h(fst(t(n)), snd(t(n))))\end{aligned}$$

- ▶ Clearly, $f(n) = snd(t(n))$

Primitive recursion ...

- ▶ We will evaluate $t(n)$ bottom up
 - ▶ Much like dynamic programming for recursive functions
- ▶ Define a function $step$ that does the following

$$step(n, f(n)) = (n+1, f(n+1))$$

i.e.

$$step(t(n)) = t(n+1)$$

- ▶ So, $t(n) = step^n(0, f(0)) = step^n(0, g) \dots$
- ▶ ... and $f(n) = snd(t(n)) = snd(step^n(0, g))$
- ▶ Will require constructions for building pairs and decomposing them using fst and snd

Primitive recursion ...

An encoding for pairs

- ▶ $\langle pair \rangle \equiv \lambda xyz.(zxy).$
- ▶ $\langle fst \rangle \equiv \lambda p.(p(\lambda xy.x)).$
- ▶ $\langle snd \rangle \equiv \lambda p.(p(\lambda xy.y)).$

Thus

- ▶ $\langle pair \rangle a b \rightarrow^* \lambda z.zab$
- ▶ $\langle fst \rangle(\langle pair \rangle ab) \rightarrow_{\beta} \lambda p.(p(\lambda xy.x))(\lambda z.zab)$
 $\rightarrow_{\beta} (\lambda z.zab)(\lambda xy.x) \rightarrow_{\beta} (\lambda xy.x)ab \rightarrow_{\beta} (\lambda y.a)b \rightarrow_{\beta} a$
- ▶ $\langle snd \rangle(\langle pair \rangle ab) \rightarrow_{\beta} \lambda p.(p(\lambda xy.y))(\lambda z.zab)$
 $\rightarrow_{\beta} (\lambda z.zab)(\lambda xy.y) \rightarrow_{\beta} (\lambda xy.y)ab \rightarrow_{\beta} (\lambda y.y)b \rightarrow_{\beta} b$

Primitive recursion ...

$f(n+1)$ is defined in terms of g and $h(n, f(n))$

$$\begin{aligned}t(0) &= (0, f(0)) &= (0, g) \\t(n+1) &= (n+1, f(n+1)) &= (n+1, h(n, f(n))) \\&= (\text{succ}(\text{fst}(t(n))), \\&\quad h(\text{fst}(t(n)), \text{snd}(t(n))))\end{aligned}$$

$$\text{step}(t(n)) = \text{step}(n, f(n)) = (n+1, f(n+1)) = t(n+1)$$

Use primitive recursive defn of $t(n)$ to encode step

$$\langle \text{step} \rangle \equiv \lambda x. \langle \text{pair} \rangle (\langle \text{succ} \rangle (\langle \text{fst} \rangle x)) (\langle h \rangle (\langle \text{fst} \rangle x) (\langle \text{snd} \rangle x)).$$

Primitive recursion ...

$\langle \text{step} \rangle \equiv \lambda x. \langle \text{pair} \rangle (\langle \text{succ} \rangle (\langle \text{fst} \rangle x)) (\langle h \rangle (\langle \text{fst} \rangle x) (\langle \text{snd} \rangle x))$.

Check that $\langle \text{step} \rangle (\langle \text{pair} \rangle \langle n \rangle \langle f(n) \rangle) \rightarrow^* \langle \text{pair} \rangle \langle n+1 \rangle \langle f(n+1) \rangle$

$\langle \text{step} \rangle (\langle \text{pair} \rangle \langle n \rangle \langle f(n) \rangle)$
 $\rightarrow \langle \text{pair} \rangle (\langle \text{succ} \rangle (\langle \text{fst} \rangle (\langle \text{pair} \rangle \langle n \rangle \langle f(n) \rangle)))$
 $\quad (\langle h \rangle (\langle \text{fst} \rangle (\langle \text{pair} \rangle \langle n \rangle \langle f(n) \rangle)) (\langle \text{sec} \rangle (\langle \text{pair} \rangle \langle n \rangle \langle f(n) \rangle)))$
 $\rightarrow \langle \text{pair} \rangle (\langle \text{succ} \rangle \langle n \rangle) (\langle h \rangle \langle n \rangle \langle f(n) \rangle)$
 $\rightarrow \langle \text{pair} \rangle (\langle \text{succ} \rangle \langle n \rangle) \langle h(n, f(n)) \rangle$
 $\rightarrow \langle \text{pair} \rangle \langle n+1 \rangle \langle h(n, f(n)) \rangle$
 $\rightarrow \langle \text{pair} \rangle \langle n+1 \rangle \langle f(n+1) \rangle$.

Primitive recursion ...

Recall that $t(n) = \text{step}^n(0, g)$

Hence

$$\langle t \rangle \equiv \lambda y. y \langle \text{step} \rangle (\langle \text{pair} \rangle \langle 0 \rangle \langle g \rangle)$$

Check that for all n , $\langle t \rangle \langle n \rangle \rightarrow^* \text{pair} \langle n \rangle \langle f(n) \rangle$.

$$\langle t \rangle \langle 0 \rangle \rightarrow \langle 0 \rangle \text{step} (\text{pair} \langle 0 \rangle \langle g \rangle) \rightarrow \text{pair} \langle 0 \rangle \langle g \rangle$$

$$\langle t \rangle \langle n+1 \rangle \rightarrow \langle n+1 \rangle \text{step} (\text{pair} \langle 0 \rangle \langle g \rangle)$$

$$\rightarrow \text{step} (\langle n \rangle \text{step} (\text{pair} \langle 0 \rangle \langle g \rangle))$$

$$\rightarrow \text{step} (\text{pair} \langle n \rangle \langle f(n) \rangle) \text{ (by ind. hyp.)}$$

$$\rightarrow \text{pair} \langle n+1 \rangle \langle f(n+1) \rangle \text{ (by what has been proved above)}$$

Primitive recursion ...

Clearly $f(n) = \text{snd}(t(n))$

$$\langle f \rangle \equiv \lambda y. \langle \text{snd} \rangle (\langle t \rangle y)$$

Collapsing all the steps we have seen above, we have an expression $\langle PR \rangle$ that constructs a primitive recursive definition from the functions g and h :

$$\langle PR \rangle \equiv \lambda hgy. \langle \text{snd} \rangle (y (\lambda x. \langle \text{pair} \rangle (\langle \text{succ} \rangle (\langle \text{fst} \rangle x)) \\ (h(\langle \text{fst} \rangle x) (\langle \text{snd} \rangle x)))) (\langle \text{pair} \rangle \langle 0 \rangle g)$$