λ Calculus: Lecture 2

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

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

  - $M\{x \leftarrow M'\}$: 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

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

For instance

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

Note that $\langle 0 \rangle gy \to_\beta (\lambda x.x)y \to_\beta y$.
Hence

$$\langle 1 \rangle = \ldots = \lambda fx.(f(\underbrace{(\lambda fx.x)fx}_{\text{apply } \beta})) \to_\beta \lambda fx.(fx)$$

So $\langle 1 \rangle gy \to_\beta (\lambda x.(gx))y \to_\beta gy$

Church numerals ...

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

so,

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

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

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

Encoding arithmetic functions . . .

Successor

- $succ(n) = n + 1$
- Define as $\lambda pfx.f(pfx)$

$$(\lambda pfx.f(pfx))\langle n\rangle \to_\beta \lambda fx.f(\langle n\rangle fx) \to_\beta \lambda fx.f(f^n x) = \lambda fx.f^{n+1}x$$
$$= \langle n{+}1\rangle$$

plus: $\lambda pqfx.pf(qfx)$.

$$\begin{array}{rcl}(\lambda pqfx.pf(qfx))\langle m\rangle\langle n\rangle & \to_\beta & (\lambda qfx.\langle m\rangle f(qfx))\langle n\rangle \\ & \to_\beta & (\lambda fx.\langle m\rangle f(\langle n\rangle fx)) \\ & \to_\beta & (\lambda fx.\langle m\rangle f(f^n x)) \\ & \to_\beta & (\lambda fx.f^m(f^n x)) \\ & = & (\lambda fx.f^{m+n}x) = \langle m{+}n\rangle\end{array}$$

Encoding arithmetic functions . . .

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

Other functions:

multiplication	:	$\lambda pqfx.q(pf)x$
exponentiation	:	$\lambda pq.(pq)$

Computability

- Church numerals encode $n \in \mathbb{N}$
- Can we encode computable functions $f : \mathbb{N}^k \to \mathbb{N}$?
  - Let $\langle f \rangle$ be the encoding of computable function f
  - Want $\langle f \rangle \langle n_1 \rangle \langle n_2 \rangle \ldots \langle n_k \rangle \to^* \langle f(n_1, n_2, \ldots, n_k) \rangle$
  - 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: $\Pi_i^k(n_1, n_2, \ldots, n_k) = n_i$

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

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

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

Recursive functions . . .

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

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

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

Examples

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

$$\begin{aligned} plus(0, n) &= g(n) &&= \Pi_1^1(n) \\ &&&= n \\ plus(m{+}1, n) &= h(m, plus(m, n), n) \\ &= S \circ \Pi_2^3(m, plus(m, n), n) &&= S(plus(m, n)) \end{aligned}$$

Recursive functions . . .

Primitive recursion

Given $g : \mathbb{N}^k \to \mathbb{N}$ and
$h : \mathbb{N}^{k+2} \to \mathbb{N}$

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

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

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

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

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 $succ\langle n \rangle \to^* \langle n+1 \rangle$.
- Zero $\langle Z \rangle \equiv \lambda x.(\lambda gy.y)$.
- Projection $\langle \Pi_i^k \rangle \equiv \lambda x_1 x_2 \ldots x_k.x_i$.

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

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

- 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, $t(n) = step^n(0, f(0)) = step^n(0, g)$ . . .
- . . . 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

- $\langle pair \rangle\ a\ b \to^* \lambda z.zab$
- $\langle fst \rangle(\langle pair \rangle ab) \to_\beta \lambda p.(p(\lambda xy.x))(\lambda z.zab)$
  $\to_\beta (\lambda z.zab)(\lambda xy.x) \to_\beta (\lambda xy.x)ab \to_\beta (\lambda y.a)b \to_\beta a$
- $\langle snd \rangle(\langle pair \rangle ab) \to_\beta \lambda p.(p(\lambda xy.y))(\lambda z.zab)$
  $\to_\beta (\lambda z.zab)(\lambda xy.y) \to_\beta (\lambda xy.y)ab \to_\beta (\lambda y.y)b \to_\beta$

Primitive recursion . . .

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

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

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

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

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

Primitive recursion ...

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

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

$$\begin{array}{ll}
\langle step \rangle\ (\langle pair \rangle\ \langle n \rangle\ \langle f(n) \rangle) & \\
\rightarrow & \langle pair \rangle\ \ (\langle succ \rangle(\langle fst \rangle(\langle pair \rangle\ \langle n \rangle\ \langle f(n) \rangle))) \\
 & \qquad\quad (\langle h \rangle(\langle fst \rangle(\langle pair \rangle\ \langle n \rangle\ \langle f(n) \rangle))(\langle sec \rangle(\langle pair \rangle \langle n \rangle\ \langle f(n) \rangle))) \\
\rightarrow & \langle pair \rangle\ (\langle succ \rangle\ \langle n \rangle)\ (\langle h \rangle \langle n \rangle \langle f(n) \rangle) \\
\rightarrow & \langle pair \rangle\ (\langle succ \rangle\ \langle n \rangle)\ \langle h(n, f(n)) \rangle \\
\rightarrow & \langle pair \rangle\ \langle n{+}1 \rangle\ \langle h(n, f(n)) \rangle \\
\rightarrow & \langle pair \rangle\ \langle n{+}1 \rangle\ \langle f(n{+}1) \rangle.
\end{array}$$

Primitive recursion ...

Recall that $t(n) = step^n(0, g)$
Hence

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

Check that for all n, $\langle t \rangle \langle n \rangle \to^* pair\ \langle n \rangle\ \langle f(n) \rangle$.

$$\begin{aligned}
\langle t \rangle \langle 0 \rangle &\to \langle 0 \rangle\ step\ (pair\ \langle 0 \rangle\ \langle g \rangle) \to pair\ \langle 0 \rangle\ \langle g \rangle \\
\langle t \rangle \langle n{+}1 \rangle &\to \langle n{+}1 \rangle\ step\ (pair\ \langle 0 \rangle\ \langle g \rangle) \\
&\to step\ (\langle n \rangle\ step\ (pair\ \langle 0 \rangle\ \langle g \rangle)) \\
&\to step\ (pair\ \langle n \rangle\ \langle f(n) \rangle) \text{ (by ind. hyp.)} \\
&\to pair\ \langle n{+}1 \rangle\ \langle f(n{+}1) \rangle \text{ (by what has been proved above)}
\end{aligned}$$

Primitive recursion ...

Clearly $f(n) = snd(t(n))$

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

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:

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