Programming Language Concepts: Lecture 16

Madhavan Mukund

Chennai Mathematical Institute

madhavan@cmi.ac.in

http://www.cmi.ac.in/~madhavan/courses/pl2009

PLC 2009, Lecture 16, 23 March 2009

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

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'

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

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

Encoding arithmetic

Church numerals

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

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

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

Encoding arithmetic functions . . .

Successor

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

Encoding arithmetic functions . . .

Successor

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

$(\lambda pfx.f(pfx))\langle n\rangle$

Encoding arithmetic functions . . .

Successor

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

$$(\lambda pfx.f(pfx))\langle n\rangle \rightarrow_{\beta} \lambda fx.f(\langle n\rangle fx)$$

Encoding arithmetic functions ...

Successor

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

$$(\lambda pfx.f(pfx))\langle n\rangle \rightarrow_\beta \lambda fx.f(\langle n\rangle fx) \rightarrow_\beta \lambda fx.f(f^n x)$$

Encoding arithmetic functions . . .

Successor

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

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

Encoding arithmetic functions . . .

Successor

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

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

Encoding arithmetic functions . . .

Successor

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

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

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

Encoding arithmetic functions ...

Successor

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

$$(\lambda pfx.f(pfx))\langle n\rangle \rightarrow_\beta \lambda fx.f(\langle n\rangle fx) \rightarrow_\beta \lambda fx.f(f^n x) \begin{aligned}[t] &= \lambda fx.f^{n+1}x \\ &= \langle n{+}1\rangle \end{aligned}$$

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

$$(\lambda pqfx.pf(qfx))\langle m\rangle\langle n\rangle$$

Encoding arithmetic functions . . .

Successor

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

$$(\lambda pfx.f(pfx))\langle n\rangle \rightarrow_\beta \lambda fx.f(\langle n\rangle fx) \rightarrow_\beta \lambda fx.f(f^n x) \begin{aligned}[t] &= \lambda fx.f^{n+1}x \\ &= \langle n{+}1\rangle \end{aligned}$$

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

$$(\lambda pqfx.pf(qfx))\langle m\rangle\langle n\rangle \quad \rightarrow_\beta \quad (\lambda qfx.\langle m\rangle f(qfx))\langle n\rangle$$

Encoding arithmetic functions ...

Successor

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

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

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

$$\begin{aligned}(\lambda pqfx.pf(qfx))\langle m\rangle\langle n\rangle &\rightarrow_\beta (\lambda qfx.\langle m\rangle f(qfx))\langle n\rangle \\ &\rightarrow_\beta (\lambda fx.\langle m\rangle f(\langle n\rangle fx))\end{aligned}$$

Encoding arithmetic functions ...

Successor

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

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

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

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

Encoding arithmetic functions . . .

Successor

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

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

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

$$
\begin{aligned}
(\lambda pqfx.pf(qfx))\langle m\rangle\langle n\rangle \quad &\rightarrow_\beta \quad (\lambda qfx.\langle m\rangle f(qfx))\langle n\rangle \\
&\rightarrow_\beta \quad (\lambda fx.\langle m\rangle f(\langle n\rangle fx)) \\
&\rightarrow_\beta \quad (\lambda fx.\langle m\rangle f(f^n x)) \\
&\rightarrow_\beta \quad (\lambda fx.f^m(f^n x))
\end{aligned}
$$

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

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

Encoding arithmetic functions ...

Successor

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

$$(\lambda pfx.f(pfx))\langle n\rangle \rightarrow_\beta \lambda fx.f(\langle n\rangle fx) \rightarrow_\beta \lambda fx.f(f^n x) \begin{array}{l} = \lambda fx.f^{n+1}x \\ = \langle n{+}1\rangle \end{array}$$

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

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

Encoding arithmetic functions . . .

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

Encoding arithmetic functions . . .

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

Other functions:

$$\begin{array}{lcl} \text{multiplication} & : & \lambda pqfx.q(pf)x \\ \text{exponentiation} & : & \lambda pq.(pq) \end{array}$$

One step reduction

- Can have other reduction rules like β

One step reduction

- Can have other reduction rules like β
- Observe that $\lambda x.(Mx)$ and M are equivalent with respect to β-reduction

One step reduction

- Can have other reduction rules like β
- Observe that $\lambda x.(Mx)$ and M are equivalent with respect to β-reduction
- New reduction rule η

$$\lambda x.(Mx) \rightarrow_{\eta} M$$

One step reduction

- Can have other reduction rules like β
- Observe that $\lambda x.(Mx)$ and M are equivalent with respect to β-reduction
- New reduction rule η

$$\lambda x.(Mx) \rightarrow_{\eta} M$$

- Given basic rules $\beta, \eta, \ldots,$ we are allowed to use them "in any context"

One step reduction

- Can have other reduction rules like β
- Observe that $\lambda x.(Mx)$ and M are equivalent with respect to β-reduction
- New reduction rule η

$$\lambda x.(Mx) \rightarrow_\eta M$$

- Given basic rules β, η, ..., we are allowed to use them "in any context"
- Define a one step reduction relation $\rightarrow$ inductively

$$\frac{M \rightarrow_x M''}{M \rightarrow M}$$

$$x \in \{\beta, \eta, \ldots\}$$

One step reduction

- Can have other reduction rules like β
- Observe that $\lambda x.(Mx)$ and M are equivalent with respect to β-reduction
- New reduction rule η

$$\lambda x.(Mx) \rightarrow_{\eta} M$$

- Given basic rules β, η, ..., we are allowed to use them "in any context"
- Define a one step reduction relation $\rightarrow$ inductively

$$\frac{M \rightarrow_x M''}{M \rightarrow M} \qquad \frac{M \rightarrow M'}{\lambda x.M \rightarrow \lambda x.M'}$$

$x \in \{\beta, \eta, \ldots\}$

One step reduction

- Can have other reduction rules like β
- Observe that $\lambda x.(Mx)$ and M are equivalent with respect to β-reduction
- New reduction rule η

$$\lambda x.(Mx) \rightarrow_\eta M$$

- Given basic rules β, η, ..., we are allowed to use them "in any context"
- Define a one step reduction relation $\rightarrow$ inductively

$$\frac{M \rightarrow_x M''}{M \rightarrow M} \qquad \frac{M \rightarrow M'}{\lambda x.M \rightarrow \lambda x.M'} \qquad \frac{M \rightarrow M'}{MN \rightarrow M'N}$$

$$x \in \{\beta, \eta, \ldots\}$$

One step reduction

- Can have other reduction rules like β
- Observe that $\lambda x.(Mx)$ and M are equivalent with respect to β-reduction
- New reduction rule η

$$\lambda x.(Mx) \rightarrow_\eta M$$

- Given basic rules β, η, . . . , we are allowed to use them "in any context"
- Define a one step reduction relation $\rightarrow$ inductively

$$\frac{M \rightarrow_x M''}{M \rightarrow M} \qquad \frac{M \rightarrow M'}{\lambda x.M \rightarrow \lambda x.M'} \qquad \frac{M \rightarrow M'}{MN \rightarrow M'N} \qquad \frac{N \rightarrow N'}{MN \rightarrow MN'}$$

$x \in \{\beta, \eta, \ldots\}$

Computability

- Church numerals encode $n \in \mathbb{N}$

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

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 $\langle f \rangle \langle n_1 \rangle \langle n_2 \rangle \ldots \langle n_k \rangle \to^* \langle f(n_1, n_2, \ldots, n_k) \rangle$

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 ...

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, . . .

Recursive functions

Recursive functions [Gödel]

- Equivalent to Turing machines, ...

Initial functions

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

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
$g_1, g_2, \ldots, g_k : \mathbb{N}^h \to \mathbb{N}$,

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

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:

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

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:

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

Examples

- Define $plus(n, m) = n+m$ from $g = \Pi^1_1$
  $h = S \circ \Pi^3_2$

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 $g = \Pi^1_1$
  $h = S \circ \Pi^3_2$

$$\begin{aligned} plus(0, n) &= g(n) && = \Pi^1_1(n) \\ & && = n \\ plus(m+1, n) &= h(m, plus(m, n), n) \\ &= S \circ \Pi^3_2(m, plus(m, n), n) && = S(plus(m, n)) \end{aligned}$$

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:

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

Examples

- Define $times(n, m) = n \cdot m$ from $g = Z$
  $h = plus \circ (\Pi^3_3, \Pi^3_2)$

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:

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

Examples

- Define $times(n, m) = n \cdot m$ from $g = Z$, $h = plus \circ (\Pi^3_3, \Pi^3_2)$

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

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

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

Recursive functions ...

Minimalization

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

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

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;
```

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 $\mu n.(n - 2^k)$

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

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 $\mu n.(n - 2^k)$

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

Not defined for all n!

Encoding recursive functions . . .

- $\langle n \rangle \equiv \lambda fx.(f^n x)$.

Encoding recursive functions . . .

- $\langle n \rangle \equiv \lambda fx.(f^n x)$.
- Successor $succ \equiv \lambda nfx.(f(nfx))$ such that $succ\langle n \rangle \rightarrow^* \langle n+1 \rangle$.

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 $\langle Z \rangle \equiv \lambda x.(\lambda gy.y)$.

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 $\langle \Pi_i^k \rangle \equiv \lambda x_1 x_2 \ldots x_k . x_i$.

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 $\langle \Pi_i^k \rangle \equiv \lambda x_1 x_2 \ldots x_k.x_i$.

Composition is easy

Encoding recursive functions . . .

Primitive recursion

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

Encoding recursive functions ...

Primitive recursion

- Assume $f(n+1)$ is defined in terms of g and $h(n, f(n))$
- Convert recursion into iteration

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

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

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

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

Recursive functions . . .

Primitive Recursion

- We will evaluate $t(n)$ bottom up
  - Much like dynamic programming for recursive functions

Recursive functions . . .

Primitive Recursion

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

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

Recursive functions . . .

Primitive Recursion

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

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

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

Recursive functions . . .

Primitive Recursion

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

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

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

In the next class, we will provide a λ-calculus translation for *step*

- Will require constructions for building pairs and decomposing them using *fst* and *snd*

Recursive functions . . .

Minimalization

- To evaluate

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

  we go back to the idea of computing a while loop

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

- Implement the while loop using recursion

```
f(n1,n2,...,nk) =   check(0,n1,n2...nk)
where
  check(n,n1,n2...nk){
    if (iszero(g(n,n1,n2,...,nk))  {return n;}
    else  {check(n+1,n1,n2,...,nk);}
  }
```

- Need a mechanism to encode booleans, if-then-else in λ-calculus