Programming Language Concepts: Lecture 17

Madhavan Mukund

Chennai Mathematical Institute

madhavan@cmi.ac.in

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

PLC 2009, Lecture 17, 25 March 2009

Recursive functions

Recursive functions [Gödel]

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]

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 ...

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:

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

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:

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

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

Encoding recursive functions . . .

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

Encoding recursive functions ...

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

Encoding recursive functions . . .

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

Encoding recursive functions . . .

- $\langle n \rangle \equiv \lambda fx.(f^n x)$.
- Successor $\langle succ \rangle \equiv \lambda nfx.(f(nfx))$ 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 $h(n, f(n))$
- Main difficulty is to eliminate recursion
  - λ-calculus functions are anonymous
  - Cannot directly use name of f inside definition of f

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

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

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

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

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

- 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

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

i.e.

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

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

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

i.e.

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

- 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

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

i.e.

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

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

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

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

  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*

Recursive functions . . .

An encoding for pairs

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

Recursive functions . . .

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

- $\langle pair \rangle\ a\ b \rightarrow^* \lambda z.zab$
- $\langle fst \rangle(\langle pair \rangle ab)$

Recursive functions . . .

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

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

Recursive functions . . .

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

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

Recursive functions . . .

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

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

Recursive functions ...

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

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

Recursive functions . . .

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

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

Recursive functions . . .

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

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

Recursive functions ...

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

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

Recursive functions ...

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

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

Recursive functions . . .

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

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

Recursive functions ...

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

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

Back to primitive recursion . . .

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

Back to primitive recursion ...

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

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

Back to primitive recursion ...

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

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

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

Back to primitive recursion ...

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

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

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

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

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

Back to primitive recursion . . .

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

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

$\langle \mathit{step} \rangle\ (\langle \mathit{pair} \rangle\ \langle n \rangle\ \langle f(n) \rangle)$

Back to 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 $\langle step \rangle\ (\langle pair \rangle \langle n \rangle \langle f(n) \rangle) \to^* \langle pair \rangle\ \langle n{+}1 \rangle \langle f(n{+}1) \rangle$

$$\begin{array}{l} \langle step \rangle\ (\langle pair \rangle\ \langle n \rangle\ \langle f(n) \rangle) \\ \quad \to \quad \langle pair \rangle \quad (\langle succ \rangle(\langle fst \rangle(\langle pair \rangle\ \langle n \rangle\ \langle f(n) \rangle))) \end{array}$$

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

$$\begin{array}{lll} \multicolumn{3}{l}{\langle step\rangle\ (\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle)} \\ \quad\rightarrow & \langle pair\rangle & (\langle succ\rangle(\langle fst\rangle(\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle))) \\ & & (\langle h\rangle(\langle fst\rangle(\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle))(\langle sec\rangle(\langle pair\rangle\langle n\rangle\ \langle f(n)\rangle))) \end{array}$$

Back to 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 $\langle step \rangle \ (\langle pair \rangle \langle n \rangle \langle f(n) \rangle) \to^* \langle pair \rangle \ \langle n{+}1 \rangle \langle f(n{+}1) \rangle$

$$\begin{array}{lll} \multicolumn{3}{l}{\langle step \rangle \ (\langle pair \rangle \ \langle n \rangle \ \langle f(n) \rangle)} \\ \to & \langle pair \rangle & (\langle succ \rangle(\langle fst \rangle(\langle pair \rangle \ \langle n \rangle \ \langle f(n) \rangle))) \\ & & (\langle h \rangle(\langle fst \rangle(\langle pair \rangle \ \langle n \rangle \ \langle f(n) \rangle))(\langle sec \rangle(\langle pair \rangle \langle n \rangle \ \langle f(n) \rangle))) \\ \to & \langle pair \rangle & (\langle succ \rangle \ \langle n \rangle) \ (\langle h \rangle \langle n \rangle \langle f(n) \rangle) \end{array}$$

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

$$\begin{array}{ll}
\multicolumn{2}{l}{\langle step\rangle\ (\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle)} \\
\rightarrow & \langle pair\rangle\quad (\langle succ\rangle(\langle fst\rangle(\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle)))) \\
 & \qquad\qquad (\langle h\rangle(\langle fst\rangle(\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle))(\langle sec\rangle(\langle pair\rangle\langle n\rangle\ \langle f(n)\rangle))) \\
\rightarrow & \langle pair\rangle\ (\langle succ\rangle\ \langle n\rangle)\ (\langle h\rangle\langle n\rangle\langle f(n)\rangle) \\
\rightarrow & \langle pair\rangle\ (\langle succ\rangle\ \langle n\rangle)\ \langle h(n, f(n))\rangle
\end{array}$$

Back to 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 $\langle step\rangle\ (\langle pair\rangle\langle n\rangle\langle f(n)\rangle) \to^* \langle pair\rangle\ \langle n{+}1\rangle\langle f(n{+}1)\rangle$

$$\begin{array}{ll} \multicolumn{2}{l}{\langle step\rangle\ (\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle)} \\ \to & \langle pair\rangle\quad (\langle succ\rangle(\langle fst\rangle(\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle))) \\ & \qquad\qquad (\langle h\rangle(\langle fst\rangle(\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle))(\langle sec\rangle(\langle pair\rangle\langle n\rangle\ \langle f(n)\rangle))) \\ \to & \langle pair\rangle\ (\langle succ\rangle\ \langle n\rangle)\ (\langle h\rangle\langle n\rangle\langle f(n)\rangle) \\ \to & \langle pair\rangle\ (\langle succ\rangle\ \langle n\rangle)\ \langle h(n,f(n))\rangle \\ \to & \langle pair\rangle\ \langle n{+}1\rangle\ \langle h(n,f(n))\rangle \end{array}$$

Back to 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 $\langle step\rangle\ (\langle pair\rangle\langle n\rangle\langle f(n)\rangle) \to^* \langle pair\rangle\ \langle n{+}1\rangle\langle f(n{+}1)\rangle$

$$\begin{array}{ll}
\multicolumn{2}{l}{\langle step\rangle\ (\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle)} \\
\to & \langle pair\rangle\quad (\langle succ\rangle(\langle fst\rangle(\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle))) \\
 & \qquad\qquad (\langle h\rangle(\langle fst\rangle(\langle pair\rangle\ \langle n\rangle\ \langle f(n)\rangle))(\langle sec\rangle(\langle pair\rangle\langle n\rangle\ \langle f(n)\rangle))) \\
\to & \langle pair\rangle\ (\langle succ\rangle\ \langle n\rangle)\ (\langle h\rangle\langle n\rangle\langle f(n)\rangle) \\
\to & \langle pair\rangle\ (\langle succ\rangle\ \langle n\rangle)\ \langle h(n, f(n))\rangle \\
\to & \langle pair\rangle\ \langle n{+}1\rangle\ \langle h(n, f(n))\rangle \\
\to & \langle pair\rangle\ \langle n{+}1\rangle\ \langle f(n{+}1)\rangle.
\end{array}$$

Back to 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, $\langle t \rangle \langle n \rangle \to^* pair \ \langle n \rangle \ \langle f(n) \rangle$.

$$\langle t \rangle \langle 0 \rangle \to \langle 0 \rangle \ step \ (pair \ \langle 0 \rangle \ \langle g \rangle) \to pair \ \langle 0 \rangle \ \langle g \rangle$$

Back to 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, $\langle t \rangle \langle n \rangle \rightarrow^* pair \ \langle n \rangle \ \langle f(n) \rangle$.

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

Back to 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, $\langle t \rangle \langle n \rangle \to^* pair \ \langle n \rangle \ \langle f(n) \rangle$.

$$\begin{aligned}
\langle t \rangle \langle 0 \rangle &\to \langle 0 \rangle \ step \ (pair \ \langle 0 \rangle \ \langle g \rangle) \to pair \ \langle 0 \rangle \ \langle g \rangle \\
\langle t \rangle \langle n{+}1 \rangle \ &\to \langle n{+}1 \rangle \ step \ (pair \ \langle 0 \rangle \ \langle g \rangle) \\
&\to step \ (\langle n \rangle \ step \ (pair \ \langle 0 \rangle \ \langle g \rangle))
\end{aligned}$$

Back to 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, $\langle t \rangle \langle n \rangle \rightarrow^* pair \ \langle n \rangle \ \langle f(n) \rangle$.

$$\begin{aligned}
\langle t \rangle \langle 0 \rangle &\rightarrow \langle 0 \rangle \ step \ (pair \ \langle 0 \rangle \ \langle g \rangle) \rightarrow pair \ \langle 0 \rangle \ \langle g \rangle \\
\langle t \rangle \langle n{+}1 \rangle &\rightarrow \langle n{+}1 \rangle \ step \ (pair \ \langle 0 \rangle \ \langle g \rangle) \\
&\rightarrow step \ (\langle n \rangle \ step \ (pair \ \langle 0 \rangle \ \langle g \rangle)) \\
&\rightarrow step \ (pair \ \langle n \rangle \ \langle f(n) \rangle) \ \text{(by ind. hyp.)}
\end{aligned}$$

Back to 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, $\langle t\rangle\langle n\rangle \to^* pair\ \langle n\rangle\ \langle f(n)\rangle$.

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

Back to 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:

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

Minimalization

- A direct encoding of primitive recursion would require us to define $f(n+1)$ in terms of $f(n)$

Minimalization

- A direct encoding of primitive recursion would require us to define $f(n+1)$ in terms of $f(n)$
- Going via $t(n)$ and *step* avoided such a recursive definition

Minimalization

- A direct encoding of primitive recursion would require us to define $f(n+1)$ in terms of $f(n)$
- Going via $t(n)$ and $step$ avoided such a recursive definition
- Recursive definitions arise again in our translation of minimilation

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;
```

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);}
  }
```

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.

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. Also need a mechanism for recursion.

Recursive definitions

Suppose $F = \lambda x_1 x_2 \ldots x_n E$, where where E contains an occurrence of F

Recursive definitions

Suppose $F = \lambda x_1 x_2 \ldots x_n E$, where where E contains an occurrence of F

- Choose a new variable f
- Convert E to E^* replacing every F in E by ff
  - If E is of the form $\cdots F \cdots F \cdots$ then E^* is $\cdots (ff) \cdots (ff) \cdots$.

Recursive definitions

Suppose $F = \lambda x_1 x_2 \ldots x_n E$, where where E contains an occurrence of F

- Choose a new variable f
- Convert E to E^* replacing every F in E by ff
  - If E is of the form $\cdots F \cdots F \cdots$ then E^* is $\cdots (ff) \cdots (ff) \cdots$.

Now write

$$
\begin{aligned}
G &= \lambda f x_1 x_2 \ldots x_n . E^* \\
&= \lambda f x_1 x_2 \ldots x_n . \cdots (ff) \cdots (ff) \cdots
\end{aligned}
$$

Recursive definitions

Suppose $F = \lambda x_1 x_2 \dots x_n E$, where where E contains an occurrence of F

- Choose a new variable f
- Convert E to E^* replacing every F in E by ff
  - If E is of the form $\cdots F \cdots F \cdots$ then E^* is $\cdots (ff) \cdots (ff) \cdots$.

Now write

$$\begin{aligned} G &= \lambda f x_1 x_2 \dots x_n . E^* \\ &= \lambda f x_1 x_2 \dots x_n . \cdots (ff) \cdots (ff) \cdots \end{aligned}$$

Then

$$GG = \lambda x_1 x_2 \dots x_n . \cdots (GG) \cdots (GG) \cdots$$

Recursive definitions

Suppose $F = \lambda x_1 x_2 \dots x_n E$, where where E contains an occurrence of F

- Choose a new variable f
- Convert E to E^* replacing every F in E by ff
  - If E is of the form $\cdots F \cdots F \cdots$ then E^* is $\cdots (ff) \cdots (ff) \cdots$.

Now write

$$\begin{aligned} G &= \lambda f x_1 x_2 \dots x_n . E^* \\ &= \lambda f x_1 x_2 \dots x_n . \cdots (ff) \cdots (ff) \cdots \end{aligned}$$

Then

$$GG = \lambda x_1 x_2 \dots x_n . \cdots (GG) \cdots (GG) \cdots$$

- GG satisfies the equation defining F
- Write $F = GG$, where $G = \lambda f x_1 x_2 \dots x_n . E^*$.

Minimalization

```
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);}
  }
```

Minimalization

```
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);}
  }
```

Encoding Booleans

- $\langle True \rangle \equiv \lambda xy.x$
- $\langle False \rangle \equiv \lambda xy.y$
- $\langle if - b - then - x - else - y \rangle \equiv \lambda bxy.\ bxy$

Minimalization

- $\langle True \rangle \equiv \lambda xy.x$
- $\langle False \rangle \equiv \lambda xy.y$
- $\langle if - b - then - x - else - y \rangle \equiv \lambda bxy.\ bxy$

Minimalization

- $\langle True \rangle \equiv \lambda xy.x$
- $\langle False \rangle \equiv \lambda xy.y$
- $\langle if - b - then - x - else - y \rangle \equiv \lambda bxy.\ bxy$

$$
\begin{array}{rcl}
(\lambda bxy.bxy)\langle True \rangle fg & \to & \lambda xy.((\lambda xy.x)xy)fg \\
& \to & \lambda y.((\lambda xy.x)fy)g \\
& \to & (\lambda xy.x)fg \\
& \to & (\lambda y.f)g \\
& \to & f
\end{array}
$$

Minimalization

- $\langle True \rangle \equiv \lambda xy.x$
- $\langle False \rangle \equiv \lambda xy.y$
- $\langle if - b - then - x - else - y \rangle \equiv \lambda bxy.\ bxy$

$$
\begin{array}{rcl}
(\lambda bxy.bxy)\langle True \rangle fg & \to & \lambda xy.((\lambda xy.x)xy)fg \\
& \to & \lambda y.((\lambda xy.x)fy)g \\
& \to & (\lambda xy.x)fg \\
& \to & (\lambda y.f)g \\
& \to & f
\end{array}
$$

$$
\begin{array}{rcl}
(\lambda bxy.bxy)\langle False \rangle fg & \to & \lambda xy.((\lambda xy.y)xy)fg \\
& \to & \lambda y.((\lambda xy.y)fy)g \\
& \to & (\lambda xy.y)fg \\
& \to & (\lambda y.y)g \\
& \to & g
\end{array}
$$

Minimalization

- Want to define $f : \mathbb{N}^k \to \mathbb{N}$ by minimalization from a $g : \mathbb{N}^{k+1} \to \mathbb{N}$
- Already have an encoding $\langle g \rangle$ for g

Minimalization

- Want to define $f : \mathbb{N}^k \to \mathbb{N}$ by minimalization from a $g : \mathbb{N}^{k+1} \to \mathbb{N}$
- Already have an encoding $\langle g \rangle$ for g

Define F as follows:

$$\begin{aligned} F = \lambda n x_1 x_2 \ldots x_k.\ & \text{if } \langle \mathit{iszero} \rangle (\langle g \rangle\ n\ x_1\ x_2\ \ldots\ x_k) \\ & \text{then } n \\ & \text{else } F(\langle \mathit{succ} \rangle n)\ x_1\ x_2\ \ldots\ x_k \end{aligned}$$

Minimalization

- Want to define $f : \mathbb{N}^k \to \mathbb{N}$ by minimalization from a $g : \mathbb{N}^{k+1} \to \mathbb{N}$
- Already have an encoding $\langle g \rangle$ for g

Define F as follows:

$$
\begin{aligned}
F = \lambda n x_1 x_2 \ldots x_k.\ & \text{if } \langle iszero \rangle (\langle g \rangle\ n\ x_1\ x_2\ \ldots\ x_k) \\
& \text{then } n \\
& \text{else } F(\langle succ \rangle n)\ x_1\ x_2\ \ldots\ x_k
\end{aligned}
$$

- $\tilde{F}$: the lambda term for F after unravelling the recursive definition
- $\langle f \rangle$ is then $\tilde{F}\ \langle 0 \rangle$.

Minimalization

- Want to define $f : \mathbb{N}^k \to \mathbb{N}$ by minimalization from a $g : \mathbb{N}^{k+1} \to \mathbb{N}$
- Already have an encoding $\langle g \rangle$ for g

Define F as follows:

$$
\begin{aligned}
F = \lambda n x_1 x_2 \ldots x_k.\ & \text{if } \langle \mathit{iszero} \rangle (\langle g \rangle\ n\ x_1\ x_2\ \ldots\ x_k) \\
& \text{then } n \\
& \text{else } F(\langle \mathit{succ} \rangle n)\ x_1\ x_2\ \ldots\ x_k
\end{aligned}
$$

- $\tilde{F}$: the lambda term for F after unravelling the recursive definition
- $\langle f \rangle$ is then $\tilde{F}\ \langle 0 \rangle$.

Still need to define $\langle \mathit{iszero} \rangle$

Minimalization

$\langle \mathit{iszero} \rangle = \lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle$

Minimalization

$$\langle iszero \rangle = \lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle$$

$$\langle iszero \rangle \; \langle 0 \rangle$$

Minimalization

$$\langle iszero \rangle = \lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle$$

$$\langle iszero \rangle\ \langle 0 \rangle \quad = \quad (\lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle)\ (\lambda fx.x)$$

Minimalization

$\langle iszero \rangle = \lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle$

$$\begin{aligned}
\langle iszero \rangle\ \langle 0 \rangle &= (\lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle)\ (\lambda fx.x) \\
&\to_\beta (\lambda fx.x)(\lambda z.\langle false \rangle)\langle true \rangle
\end{aligned}$$

Minimalization

$$\langle iszero \rangle = \lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle$$

$$\begin{aligned}
\langle iszero \rangle\ \langle 0 \rangle &= (\lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle)\ (\lambda fx.x) \\
&\to_\beta (\lambda fx.x)(\lambda z.\langle false \rangle)\langle true \rangle \\
&\to_\beta (\lambda x.x)\langle true \rangle
\end{aligned}$$

Minimalization

$$\langle iszero \rangle = \lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle$$

$$\begin{aligned} \langle iszero \rangle\ \langle 0 \rangle &= (\lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle)\ (\lambda fx.x) \\ &\to_\beta (\lambda fx.x)(\lambda z.\langle false \rangle)\langle true \rangle \\ &\to_\beta (\lambda x.x)\langle true \rangle \\ &\to_\beta \langle true \rangle \end{aligned}$$

Minimalization

$$\langle iszero \rangle = \lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle$$

$$\begin{aligned} \langle iszero \rangle\ \langle 0 \rangle &= (\lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle)\ (\lambda fx.x) \\ &\to_\beta (\lambda fx.x)(\lambda z.\langle false \rangle)\langle true \rangle \\ &\to_\beta (\lambda x.x)\langle true \rangle \\ &\to_\beta \langle true \rangle \end{aligned}$$

$$\langle iszero \rangle\ \langle 1 \rangle$$

Minimalization

$$\langle \mathit{iszero} \rangle = \lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle$$

$$\begin{array}{rcl}
\langle \mathit{iszero} \rangle \ \langle 0 \rangle & = & (\lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle)\ (\lambda fx.x) \\
& \to_\beta & (\lambda fx.x)(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle \\
& \to_\beta & (\lambda x.x)\langle \mathit{true} \rangle \\
& \to_\beta & \langle \mathit{true} \rangle
\end{array}$$

$$\langle \mathit{iszero} \rangle \ \langle 1 \rangle = (\lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle)\ (\lambda fx.fx)$$

Minimalization

$$\langle \mathit{iszero} \rangle = \lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle$$

$$\begin{array}{rcl}
\langle \mathit{iszero} \rangle\ \langle 0 \rangle & = & (\lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle)\ (\lambda fx.x) \\
& \rightarrow_\beta & (\lambda fx.x)(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle \\
& \rightarrow_\beta & (\lambda x.x)\langle \mathit{true} \rangle \\
& \rightarrow_\beta & \langle \mathit{true} \rangle
\end{array}$$

$$\begin{array}{rcl}
\langle \mathit{iszero} \rangle\ \langle 1 \rangle & = & (\lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle)\ (\lambda fx.fx) \\
& \rightarrow_\beta & (\lambda fx.fx)(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle
\end{array}$$

Minimalization

$$\langle iszero\rangle = \lambda n.n(\lambda z.\langle false\rangle)\langle true\rangle$$

$$\begin{aligned}
\langle iszero\rangle\ \langle 0\rangle &= (\lambda n.n(\lambda z.\langle false\rangle)\langle true\rangle)\ (\lambda fx.x) \\
&\to_\beta (\lambda fx.x)(\lambda z.\langle false\rangle)\langle true\rangle \\
&\to_\beta (\lambda x.x)\langle true\rangle \\
&\to_\beta \langle true\rangle
\end{aligned}$$

$$\begin{aligned}
\langle iszero\rangle\ \langle 1\rangle &= (\lambda n.n(\lambda z.\langle false\rangle)\langle true\rangle)\ (\lambda fx.fx) \\
&\to_\beta (\lambda fx.fx)(\lambda z.\langle false\rangle)\langle true\rangle \\
&\to_\beta (\lambda x.(\lambda z.\langle false\rangle)\ x)\langle true\rangle
\end{aligned}$$

Minimalization

$$\langle \mathit{iszero} \rangle = \lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle$$

$$\begin{array}{rcl}
\langle \mathit{iszero} \rangle\ \langle 0 \rangle & = & (\lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle)\ (\lambda fx.x) \\
& \rightarrow_\beta & (\lambda fx.x)(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle \\
& \rightarrow_\beta & (\lambda x.x)\langle \mathit{true} \rangle \\
& \rightarrow_\beta & \langle \mathit{true} \rangle
\end{array}$$

$$\begin{array}{rcl}
\langle \mathit{iszero} \rangle\ \langle 1 \rangle & = & (\lambda n.n(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle)\ (\lambda fx.fx) \\
& \rightarrow_\beta & (\lambda fx.fx)(\lambda z.\langle \mathit{false} \rangle)\langle \mathit{true} \rangle \\
& \rightarrow_\beta & (\lambda x.(\lambda z.\langle \mathit{false} \rangle)\ x)\langle \mathit{true} \rangle \\
& \rightarrow_\beta & (\lambda z.\langle \mathit{false} \rangle)\ \langle \mathit{true} \rangle
\end{array}$$

Minimalization

$\langle iszero \rangle = \lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle$

$$\begin{aligned}
\langle iszero \rangle\ \langle 0 \rangle &= (\lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle)\ (\lambda fx.x) \\
&\to_\beta (\lambda fx.x)(\lambda z.\langle false \rangle)\langle true \rangle \\
&\to_\beta (\lambda x.x)\langle true \rangle \\
&\to_\beta \langle true \rangle
\end{aligned}$$

$$\begin{aligned}
\langle iszero \rangle\ \langle 1 \rangle &= (\lambda n.n(\lambda z.\langle false \rangle)\langle true \rangle)\ (\lambda fx.fx) \\
&\to_\beta (\lambda fx.fx)(\lambda z.\langle false \rangle)\langle true \rangle \\
&\to_\beta (\lambda x.(\lambda z.\langle false \rangle)\ x)\langle true \rangle \\
&\to_\beta (\lambda z.\langle false \rangle)\ \langle true \rangle \\
&\to_\beta \langle false \rangle
\end{aligned}$$

By induction, for $n > 0$ ...

$$\langle iszero \rangle\ \langle n \rangle \to_\beta^* (\lambda z.\langle false \rangle)^n\ \langle true \rangle \to_\beta^* \langle false \rangle$$