Encoding recursion, fixpoints

Madhavan Mukund, **S P Suresh**

Programming Language Concepts
Lecture 19, 27 March 2025

Encoding recursive functions

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

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,\text{«}n_1\text{»}\,\cdots\,\text{«}n_k\text{»} \xrightarrow{*}_{\beta} \text{«}m\text{»}$ iff $f(n_1, \cdots, n_k) = m$

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ iff $f(n_1, \cdots, n_k) = m$
  - If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle f(n_1, \cdots, n_k) \rangle\!\rangle$

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ iff $f(n_1, \cdots, n_k) = m$
  - If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle f(n_1, \cdots, n_k) \rangle\!\rangle$
  - Shown in detail in this lecture

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,«n_1» \cdots «n_k» \xrightarrow{*}_\beta «m»$ iff $f(n_1, \cdots, n_k) = m$
  - If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,«n_1» \cdots «n_k» \xrightarrow{*}_\beta «f(n_1, \cdots, n_k)»$
  - Shown in detail in this lecture
  - If $\mathbf{f}\,«n_1» \cdots «n_k» \xrightarrow{*}_\beta «m»$ and $\mathbf{f}\,«n_1» \cdots «n_k» \xrightarrow{*}_\beta «p»$, then $m = p$

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ iff $f(n_1, \cdots, n_k) = m$
  - If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle f(n_1, \cdots, n_k) \rangle\!\rangle$
  - Shown in detail in this lecture
  - If $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ and $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle p \rangle\!\rangle$, then $m = p$
  - Follows from more general theorems proved later

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ iff $f(n_1, \cdots, n_k) = m$
  - If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle f(n_1, \cdots, n_k) \rangle\!\rangle$
  - Shown in detail in this lecture
  - If $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ and $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle p \rangle\!\rangle$, then $m = p$
  - Follows from more general theorems proved later
  - If $f(n_1, \ldots, n_k)$ is undefined, then $\neg\left(\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle\right)$ for any m

Encoding recursive functions

- Church numerals encode $n \in \mathbb{N}$
- Can we encode recursive functions $f : \mathbb{N}^k \to \mathbb{N}$?
- Let **f** be the encoding of f
- We want $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ iff $f(n_1, \cdots, n_k) = m$
  - If $f(n_1, \cdots, n_k)$ is defined, then $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle f(n_1, \cdots, n_k) \rangle\!\rangle$
  - Shown in detail in this lecture
  - If $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle$ and $\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle p \rangle\!\rangle$, then $m = p$
  - Follows from more general theorems proved later
  - If $f(n_1, \ldots, n_k)$ is undefined, then $\neg\left(\mathbf{f}\,\langle\!\langle n_1 \rangle\!\rangle \cdots \langle\!\langle n_k \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle m \rangle\!\rangle\right)$ for any m
  - Brief discussion at the end of this lecture

Encoding recursive functions

- $\ll n \gg = \lambda f\, x \cdot f^{\,n} x$

Encoding recursive functions

- $\ll n \gg = \lambda f\, x \cdot f^{\,n} x$
- **Zero**: $\mathbf{Z} = \lambda x \cdot \ll 0 \gg$

Encoding recursive functions

- $\ll n \gg = \lambda f\ x \cdot f^{\,n} x$
- **Zero**: $\mathbf{Z} = \lambda x \cdot \ll 0 \gg$
- **Successor**: $\mathbf{succ} = \lambda p\ f\ x \cdot f\ (p\ f\ x)$

Encoding recursive functions

- $\ll n \gg = \lambda f\, x \cdot f^{\,n} x$
- **Zero**: $\mathbf{Z} = \lambda x \cdot \ll 0 \gg$
- **Successor**: $\mathbf{succ} = \lambda p\, f\, x \cdot f\ (p\, f\, x)$
- **Projection**: $\mathbf{proj}_i^k = \lambda x_1 x_2 \cdots x_k \cdot x_i$

Encoding recursive functions

- $\ll n \gg = \lambda f\, x \cdot f^{\,n} x$
- **Zero**: $\mathbf{Z} = \lambda x \cdot \ll 0 \gg$
- **Successor**: $\mathbf{succ} = \lambda p\, f\, x \cdot f\ (p\, f\, x)$
- **Projection**: $\mathbf{proj}_i^k = \lambda x_1 x_2 \cdots x_k \cdot x_i$
- **Composition**: If $f : \mathbb{N}^k \to \mathbb{N}$ is defined by $f = g \circ (h_1, \ldots, h_m)$

$$\mathbf{f} = \lambda \vec{x} \cdot \mathbf{g}\ (\mathbf{h}_1\, \vec{x})\ \cdots\ (\mathbf{h}_m\, \vec{x})$$

Encoding recursive functions: primitive recursion

- **Primitive recursion**: Suppose f is defined via primitive recursion from $g : \mathbb{N} \to \mathbb{N}$ and $h : \mathbb{N}^3 \to \mathbb{N}$

$$f\ (0, n) = g(n)$$
$$f\ (i + 1, n) = h(i, f\ (i, n), n)$$

Encoding recursive functions: primitive recursion

- **Primitive recursion**: Suppose f is defined via primitive recursion from $g : \mathbb{N} \to \mathbb{N}$ and $h : \mathbb{N}^3 \to \mathbb{N}$

$$f\ (0, n) = g(n)$$
$$f\ (i + 1, n) = h(i, f\ (i, n), n)$$

- We need to eliminate recursion

Encoding recursive functions: primitive recursion

- **Primitive recursion**: Suppose f is defined via primitive recursion from $g : \mathbb{N} \to \mathbb{N}$ and $h : \mathbb{N}^3 \to \mathbb{N}$

$$f\ (0, n) = g(n)$$
$$f\ (i + 1, n) = h(i, f\ (i, n), n)$$

- We need to eliminate recursion
  - λ-calculus functions are anonymous

Encoding recursive functions: primitive recursion

- **Primitive recursion**: Suppose f is defined via primitive recursion from $g : \mathbb{N} \to \mathbb{N}$ and $h : \mathbb{N}^3 \to \mathbb{N}$

$$f\ (0, n) = g(n)$$
$$f\ (i + 1, n) = h(i, f\ (i, n), n)$$

- We need to eliminate recursion
  - λ-calculus functions are anonymous
  - Cannot directly use name of f inside definition of f

Encoding recursive functions: primitive recursion

- **Primitive recursion**: Suppose f is defined via primitive recursion from $g : \mathbb{N} \to \mathbb{N}$ and $h : \mathbb{N}^3 \to \mathbb{N}$

$$f\ (0, n) = g(n)$$
$$f\ (i + 1, n) = h(i, f\ (i, n), n)$$

- We need to eliminate recursion
  - λ-calculus functions are anonymous
  - Cannot directly use name of f inside definition of f
- We convert recursion into iteration

Encoding primitive recursion

- Given m and n, generate a sequence of pairs

$$(0, f(0, n)), (1, f(1, n)), \ldots, (m, f(m, n))$$

Encoding primitive recursion

- Given m and n, generate a sequence of pairs

$$(0, f(0, n)), (1, f(1, n)), \ldots, (m, f(m, n))$$

- Generate the sequence by the following recursion

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

Encoding primitive recursion

- Given m and n, generate a sequence of pairs

$$(0, f(0, n)), (1, f(1, n)), \ldots, (m, f(m, n))$$

- Generate the sequence by the following recursion

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

  - *fst* and *snd* return the first and second components of a pair

Encoding primitive recursion

- Given m and n, generate a sequence of pairs

$$(0, f(0,n)), (1, f(1,n)), \ldots, (m, f(m,n))$$

- Generate the sequence by the following recursion

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

  - fst and snd return the first and second components of a pair
  - $f\ (m,n)$ can be retrieved as $\mathit{snd}(t(m))$

Encoding recursive functions

- Generate the sequence by the following recursion

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

Encoding recursive functions

- Generate the sequence by the following recursion

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

- We generate the $t(i)$'s by iteration

Encoding recursive functions

- Generate the sequence by the following recursion

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

- We generate the $t(i)$'s by iteration
- Define $A = t(0) = (0, g(n))$ and $S : t(i) \mapsto t(i+1)$

Encoding recursive functions

- Generate the sequence by the following recursion

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

- We generate the $t(i)$'s by iteration
- Define $A = t(0) = (0, g(n))$ and $S : t(i) \mapsto t(i+1)$
- So $t(m) = S^m(A)$ ...

Encoding recursive functions

- Generate the sequence by the following recursion

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

- We generate the $t(i)$'s by iteration
- Define $A = t(0) = (0, g(n))$ and $S : t(i) \mapsto t(i+1)$
- So $t(m) = S^m(A)$ ...
- ...and $f(m, n) = snd(t(m)) = snd(S^m(A))$

Encoding pairs, *fst* and *snd*

- **pair** $= \lambda xyz \cdot zxy$

Encoding pairs, *fst* and *snd*

- **pair** $= \lambda xyz \cdot zxy$
- **pair** $a\ b \xrightarrow{*}_{\beta} \lambda z \cdot zab$

Encoding pairs, *fst* and *snd*

- **pair** $= \lambda xyz \cdot zxy$
- **pair** $a\ b \xrightarrow{*}_{\beta} \lambda z \cdot zab$
- **fst** $= \lambda p.p(\lambda xy \cdot x)$

Encoding pairs, *fst* and *snd*

- **pair** $= \lambda xyz \cdot zxy$
- **pair** $a\ b \xrightarrow{*}_\beta \lambda z \cdot zab$
- **fst** $= \lambda p.p(\lambda xy \cdot x)$
- **fst** (**pair** $a\ b) \xrightarrow{*}_\beta (\lambda p \cdot p(\lambda xy \cdot x))(\lambda z \cdot zab) \longrightarrow_\beta (\lambda z \cdot zab)(\lambda xy \cdot x)$
  $\longrightarrow_\beta (\lambda xy \cdot x)ab \longrightarrow_\beta (\lambda y \cdot a)b \longrightarrow_\beta a$

Encoding pairs, *fst* and *snd*

- **pair** $= \lambda xyz \cdot zxy$
- **pair** $a\ b \xrightarrow{*}_\beta \lambda z \cdot zab$
- **fst** $= \lambda p.p(\lambda xy \cdot x)$
- **fst** (**pair** $a\ b) \xrightarrow{*}_\beta (\lambda p \cdot p(\lambda xy \cdot x))(\lambda z \cdot zab) \longrightarrow_\beta (\lambda z \cdot zab)(\lambda xy \cdot x)$
  $\longrightarrow_\beta (\lambda xy \cdot x)ab \longrightarrow_\beta (\lambda y \cdot a)b \longrightarrow_\beta a$
- **snd** $= \lambda p \cdot (p(\lambda xy \cdot y))$

Encoding pairs, *fst* and *snd*

- **pair** $= \lambda xyz \cdot zxy$
- **pair** $a\ b \xrightarrow{*}_\beta \lambda z \cdot zab$
- **fst** $= \lambda p.p(\lambda xy \cdot x)$
- **fst** (**pair** $a\ b) \xrightarrow{*}_\beta (\lambda p \cdot p(\lambda xy \cdot x))(\lambda z \cdot zab) \longrightarrow_\beta (\lambda z \cdot zab)(\lambda xy \cdot x)$
  $\longrightarrow_\beta (\lambda xy \cdot x)ab \longrightarrow_\beta (\lambda y \cdot a)b \longrightarrow_\beta a$
- **snd** $= \lambda p \cdot (p(\lambda xy \cdot y))$
- **snd** (**pair** $a\ b) \xrightarrow{*}_\beta (\lambda p \cdot p(\lambda xy \cdot y))(\lambda z \cdot zab) \longrightarrow_\beta (\lambda z \cdot zab)(\lambda xy \cdot y)$
  $\longrightarrow_\beta (\lambda xy \cdot y)ab \longrightarrow_\beta (\lambda y \cdot y)b \longrightarrow_\beta b$

Encoding primitive recursion

- $A = (0, g(n))$

Encoding primitive recursion

- $A = (0, g(n))$
- $S(t(i)) = (i + 1, f(i + 1, n)) = (succ(\ fst(t(i))),\quad h(\ fst(t(i)), snd(t(i)), n))$

Encoding primitive recursion

- $\mathsf{A} = (0, g(n))$
- $\mathsf{S}(t(i)) = (i+1, f(i+1, n)) = (succ(\ fst(t(i))),\quad h(\ fst(t(i)), snd(t(i)), n))$
- $t(m) = \mathsf{S}^m(\mathsf{A})$ and $f(m, n) = snd(t(m))$

Encoding primitive recursion

- $\mathsf{A} = (0, g(n))$
- $\mathsf{S}(t(i)) = (i+1, f(i+1, n)) = (succ(\ fst(t(i))),\quad h(\ fst(t(i)), snd(t(i)), n))$
- $t(m) = \mathsf{S}^m(\mathsf{A})$ and $f(m, n) = snd(t(m))$
- n is "free" in the above, but in the lambda encoding we carry an extra argument

Encoding primitive recursion

- $\mathsf{A} = (0, g(n))$
- $\mathsf{S}(t(i)) = (i+1, f(i+1, n)) = (succ(\ fst(t(i))),\quad h(\ fst(t(i)), snd(t(i)), n))$
- $t(m) = \mathsf{S}^m(\mathsf{A})$ and $f(m, n) = snd(t(m))$
- n is "free" in the above, but in the lambda encoding we carry an extra argument
- $\mathbf{A} = \lambda x \cdot \mathbf{pair}\ \mathbf{«}\,0\,\mathbf{»}\ (\mathbf{g}\ x)$

Encoding primitive recursion

- $\mathsf{A} = (0, g(n))$
- $\mathsf{S}(t(i)) = (i+1, f(i+1, n)) = (succ(\ fst(t(i))),\quad h(\ fst(t(i)), snd(t(i)), n))$
- $t(m) = \mathsf{S}^m(\mathsf{A})$ and $f(m, n) = snd(t(m))$
- n is "free" in the above, but in the lambda encoding we carry an extra argument
- $\mathbf{A} = \lambda x \cdot \mathbf{pair}\ \mathbf{«}\ 0\ \mathbf{»}\ (\mathbf{g}\ x)$
- $\mathbf{S} = \lambda x\ p \cdot \mathbf{pair}\quad (\mathbf{succ}\ (\mathbf{fst}\ p))\quad (\mathbf{h}\ (\mathbf{fst}\ p)\ (\mathbf{snd}\ p)\ x)$

Encoding primitive recursion

- $\mathsf{A} = (0, g(n))$
- $\mathsf{S}(t(i)) = (i+1, f(i+1, n)) = (succ(\ fst(t(i))),\quad h(\ fst(t(i)), snd(t(i)), n))$
- $t(m) = \mathsf{S}^m(\mathsf{A})$ and $f(m, n) = snd(t(m))$
- n is "free" in the above, but in the lambda encoding we carry an extra argument
- $\mathbf{A} = \lambda x \cdot \mathbf{pair}\ \mathbf{«}\ 0\ \mathbf{»}\ (\mathbf{g}\ x)$
- $\mathbf{S} = \lambda x\ p \cdot \mathbf{pair}\quad (\mathbf{succ}\ (\mathbf{fst}\ p))\quad (\mathbf{h}\ \ (\mathbf{fst}\ p)\ \ (\mathbf{snd}\ p)\ \ x)$
- $\mathbf{f} = \lambda y x \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$

Encoding primitive recursion

- $\mathbf{f} = \lambda yx \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$

Encoding primitive recursion

- $\mathbf{f} = \lambda yx \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$
- $\mathbf{f}\ «m»\ «n» \xrightarrow{*}_{\beta} \mathbf{snd}\ («m»\ (\mathbf{S}\ «n»)\ (\mathbf{A}\ «n»)) \xrightarrow{*}_{\beta} \mathbf{snd}\ ((\mathbf{S}\ «n»)^m\ (\mathbf{A}\ «n»))$

Encoding primitive recursion

- $\mathbf{f} = \lambda yx \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$
- $\mathbf{f}\ «m»\ «n» \xrightarrow{*}_{\beta} \mathbf{snd}\ («m»\ (\mathbf{S}\ «n»)\ (\mathbf{A}\ «n»)) \xrightarrow{*}_{\beta} \mathbf{snd}\ ((\mathbf{S}\ «n»)^m\ (\mathbf{A}\ «n»))$
- Check that $\mathbf{S}\ «n»\ (\mathbf{pair}\ «i»\ «f(i,n)») \xrightarrow{*}_{\beta} \mathbf{pair}\ «i+1»\ «f(i+1,n)»$

Encoding primitive recursion

- Check that $\mathbf{S}\ \ulcorner n \urcorner\ (\mathbf{pair}\ \ulcorner i \urcorner\ \ulcorner f(i,n) \urcorner) \xrightarrow{*}_{\beta} \mathbf{pair}\ \ulcorner i+1 \urcorner\ \ulcorner f(i+1,n) \urcorner$

Encoding primitive recursion

- Check that $\mathbf{S}\,\langle\!\langle n \rangle\!\rangle\,(\mathbf{pair}\,\langle\!\langle i \rangle\!\rangle\,\langle\!\langle f(i,n) \rangle\!\rangle) \xrightarrow{*}_{\beta} \mathbf{pair}\,\langle\!\langle i+1 \rangle\!\rangle\,\langle\!\langle f(i+1,n) \rangle\!\rangle$
- $\mathbf{S}\,\langle\!\langle n \rangle\!\rangle\,(\mathbf{pair}\,\langle\!\langle i \rangle\!\rangle\,\langle\!\langle f(i,n) \rangle\!\rangle)$

$$
\begin{aligned}
&\xrightarrow{*}_{\beta} && \mathbf{pair}\ (\mathbf{succ}\ (\mathbf{fst}\ (\mathbf{pair}\,\langle\!\langle i \rangle\!\rangle\,\langle\!\langle f(i,n) \rangle\!\rangle)))\\
& && (\mathbf{h}\ (\mathbf{fst}\ (\mathbf{pair}\,\langle\!\langle i \rangle\!\rangle\,\langle\!\langle f(i,n) \rangle\!\rangle))\\
& && \quad (\mathbf{snd}\ (\mathbf{pair}\,\langle\!\langle i \rangle\!\rangle\,\langle\!\langle f(i,n) \rangle\!\rangle))\\
& && \quad \langle\!\langle n \rangle\!\rangle)\\
&\xrightarrow{*}_{\beta} && \mathbf{pair}\quad (\mathbf{succ}\,\langle\!\langle i \rangle\!\rangle)\ (\mathbf{h}\,\langle\!\langle i \rangle\!\rangle\,\langle\!\langle f(i,n) \rangle\!\rangle\,\langle\!\langle n \rangle\!\rangle)\\
&\xrightarrow{*}_{\beta} && \mathbf{pair}\,\langle\!\langle i+1 \rangle\!\rangle\,\langle\!\langle h(i,f(i,n),n) \rangle\!\rangle\\
&= && \mathbf{pair}\,\langle\!\langle i+1 \rangle\!\rangle\,\langle\!\langle f(i+1,n) \rangle\!\rangle
\end{aligned}
$$

Encoding primitive recursion

- Check that $(\mathbf{S}\ \ulcorner n \urcorner)^i\ (\mathbf{A}\ \ulcorner n \urcorner) \xrightarrow{*}_\beta \mathbf{pair}\ \ulcorner i \urcorner\ \ulcorner f(i, n) \urcorner$

Encoding primitive recursion

- Check that $(\mathbf{S}\,\ulcorner n \urcorner)^i\,(\mathbf{A}\,\ulcorner n \urcorner) \xrightarrow{*}_\beta \mathbf{pair}\,\ulcorner i \urcorner\,\ulcorner f(i,n) \urcorner$
- $(\mathbf{S}\,\ulcorner n \urcorner)^0\,(\mathbf{A}\,\ulcorner n \urcorner) \xrightarrow{*}_\beta \mathbf{A}\,\ulcorner n \urcorner = \mathbf{pair}\,\ulcorner 0 \urcorner\,(\mathbf{g}\,\ulcorner n \urcorner) = \mathbf{pair}\,\ulcorner 0 \urcorner\,\ulcorner f(0,n) \urcorner$

Encoding primitive recursion

- Check that $(\mathbf{S}\,\langle\!\langle n \rangle\!\rangle)^i\,(\mathbf{A}\,\langle\!\langle n \rangle\!\rangle) \xrightarrow{*}_\beta \mathbf{pair}\,\langle\!\langle i \rangle\!\rangle\,\langle\!\langle f(i,n) \rangle\!\rangle$
- $(\mathbf{S}\,\langle\!\langle n \rangle\!\rangle)^0\,(\mathbf{A}\,\langle\!\langle n \rangle\!\rangle) \xrightarrow{*}_\beta \mathbf{A}\,\langle\!\langle n \rangle\!\rangle = \mathbf{pair}\,\langle\!\langle 0 \rangle\!\rangle\,(\mathbf{g}\,\langle\!\langle n \rangle\!\rangle) = \mathbf{pair}\,\langle\!\langle 0 \rangle\!\rangle\,\langle\!\langle f(0,n) \rangle\!\rangle$
- $$\begin{aligned}(\mathbf{S}\,\langle\!\langle n \rangle\!\rangle)^{i+1}\,(\mathbf{A}\,\langle\!\langle n \rangle\!\rangle) \quad &= \quad \mathbf{S}\,\langle\!\langle n \rangle\!\rangle\,((\mathbf{S}\langle\!\langle n \rangle\!\rangle)^i\,(\mathbf{A}\,\langle\!\langle n \rangle\!\rangle)) \\ &\xrightarrow{*}_\beta \quad \mathbf{S}\,\langle\!\langle n \rangle\!\rangle\,(\mathbf{pair}\,\langle\!\langle i \rangle\!\rangle\,\langle\!\langle f(i,n) \rangle\!\rangle) \quad \textbf{(ind. hyp.)} \\ &\xrightarrow{*}_\beta \quad \mathbf{pair}\,\langle\!\langle i+1 \rangle\!\rangle\,\langle\!\langle f(i+1,n) \rangle\!\rangle \quad \textbf{(previous slide)}\end{aligned}$$

Encoding primitive recursion

- $\mathbf{f} = \lambda yx \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$

Encoding primitive recursion

- $\mathbf{f} = \lambda yx \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$
- $\mathbf{f}\ «m»\ «n» \xrightarrow{*}_{\beta} \mathbf{snd}\ («m»\ (\mathbf{S}\ «n»)\ (\mathbf{A}\ «n»)) \xrightarrow{*}_{\beta} \mathbf{snd}((\mathbf{S}\ «n»)^m\ (\mathbf{A}\ «n»))$

Encoding primitive recursion

- $\mathbf{f} = \lambda yx \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$
- $\mathbf{f}\ \ll m \gg\ \ll n \gg \xrightarrow{*}_{\beta} \mathbf{snd}\ (\ll m \gg\ (\mathbf{S}\ \ll n \gg)\ (\mathbf{A}\ \ll n \gg)) \xrightarrow{*}_{\beta} \mathbf{snd}((\mathbf{S}\ \ll n \gg)^m\ (\mathbf{A}\ \ll n \gg))$
- $(\mathbf{S}\ \ll n \gg)^m\ (\mathbf{A}\ \ll n \gg) \xrightarrow{*}_{\beta} \mathbf{pair}\ \ll m \gg\ \ll f\,(m, n) \gg$

Encoding primitive recursion

- $\mathbf{f} = \lambda yx \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$
- $\mathbf{f}\ «m»\ «n» \xrightarrow{*}_{\beta} \mathbf{snd}\ («m»\ (\mathbf{S}\ «n»)\ (\mathbf{A}\ «n»)) \xrightarrow{*}_{\beta} \mathbf{snd}((\mathbf{S}\ «n»)^m\ (\mathbf{A}\ «n»))$
- $(\mathbf{S}\ «n»)^m\ (\mathbf{A}\ «n») \xrightarrow{*}_{\beta} \mathbf{pair}\ «m»\ «f(m,n)»$
- So $\mathbf{f}\ «m»\ «n» \xrightarrow{*}_{\beta} \mathbf{snd}\ (\mathbf{pair}\ «m»\ «f(m,n)») \xrightarrow{*}_{\beta} «f(m,n)»$

Encoding primitive recursion

- $\mathbf{f} = \lambda yx \cdot \mathbf{snd}\ (y\ (\mathbf{S}\ x)\ (\mathbf{A}\ x))$
- $\mathbf{f}\ «m»\ «n» \xrightarrow{*}_\beta \mathbf{snd}\ («m»\ (\mathbf{S}\ «n»)\ (\mathbf{A}\ «n»)) \xrightarrow{*}_\beta \mathbf{snd}((\mathbf{S}\ «n»)^m\ (\mathbf{A}\ «n»))$
- $(\mathbf{S}\ «n»)^m\ (\mathbf{A}\ «n») \xrightarrow{*}_\beta \mathbf{pair}\ «m»\ «f(m,n)»$
- So $\mathbf{f}\ «m»\ «n» \xrightarrow{*}_\beta \mathbf{snd}\ (\mathbf{pair}\ «m»\ «f(m,n)») \xrightarrow{*}_\beta «f(m,n)»$
- The expression **PR** encodes the schema of primitive recursion

$$\mathbf{PR} = \lambda hgyx \cdot \mathbf{snd}\ (y\ ((\lambda x\ p \cdot \mathbf{pair}\ (\mathbf{succ}\ (\mathbf{fst}\ p))\ (h\ (\mathbf{fst}\ p)\ (\mathbf{snd}\ p)\ x))\ \ x)\ ((\lambda x \cdot \mathbf{pair}\ «0»\ (g\ x))\ \ x)$$

Encoding μ-recursion

- Suppose $f : \mathbb{N} \to \mathbb{N}$ is obtained from $g : \mathbb{N}^2 \to \mathbb{N}$ by μ-recursion

$$f(n) = \begin{cases} i & \text{if } g(i,n) = 0 \text{ and } \forall j < i : g(j,n) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- f can be expressed as the following (potentially unbounded) **`while`** loop:

```
i = 0;
while (g(i, n) > 0) {i = i + 1;}
return i;
```

Encoding μ-recursion

- f can be expressed as the following (potentially unbounded) **while** loop:

```
i = 0;
while (g(i, n) > 0) {i = i + 1;}
return i;
```

- Implement the **while** loop using recursion:

```
int search(i, n) {
    if (iszero(g(i, n))) return i;
    else return search(i+1, n);
}
f(n) = search(0, n);
```

Encoding booleans

- Need ways to encode booleans, if-then-else, test for zero and recursive definitions in λ-calculus

Encoding booleans

- Need ways to encode booleans, if-then-else, test for zero and recursive definitions in λ-calculus
- **true** $= \lambda xy \cdot x$

Encoding booleans

- Need ways to encode booleans, if-then-else, test for zero and recursive definitions in λ-calculus
- **true** $= \lambda xy \cdot x$
- **false** $= \lambda xy \cdot y$

Encoding booleans

- Need ways to encode booleans, if-then-else, test for zero and recursive definitions in λ-calculus
- **true** $= \lambda xy \cdot x$
- **false** $= \lambda xy \cdot y$
- **if-then-else** $= \lambda bxy \cdot bxy$

Encoding booleans

- Need ways to encode booleans, if-then-else, test for zero and recursive definitions in λ-calculus
- **true** $= \lambda xy \cdot x$
- **false** $= \lambda xy \cdot y$
- **if-then-else** $= \lambda bxy \cdot bxy$
- **Syntactic sugar**: **if-then-else** $b\ f\ g$ is written as **if** b **then** f **else** g

$$
\begin{array}{lcll}
\textbf{if true then } f \textbf{ else } g & = & (\lambda bxy \cdot bxy)(\lambda xy \cdot x)fg & \longrightarrow_\beta \quad (\lambda xy \cdot (\lambda xy \cdot x)xy)fg \\
 & \longrightarrow_\beta & (\lambda y \cdot (\lambda xy \cdot x)fy)g & \longrightarrow_\beta \quad (\lambda xy \cdot x)fg \\
 & \longrightarrow_\beta & (\lambda y \cdot f)g & \longrightarrow_\beta \quad f \\
\textbf{if false then } f \textbf{ else } g & = & (\lambda bxy \cdot bxy)(\lambda xy \cdot y)fg & \overset{*}{\longrightarrow}_\beta \quad (\lambda xy \cdot y)fg \\
 & \longrightarrow_\beta & (\lambda y \cdot y)g & \longrightarrow_\beta \quad g
\end{array}
$$

Encoding test for zero

- **iszero** $:= \lambda x \cdot x(\lambda z \cdot \textbf{false})\textbf{true}$

Encoding test for zero

- $\textbf{iszero} := \lambda x \cdot x(\lambda z \cdot \textbf{false})\textbf{true}$
- $\textbf{iszero}\ \ll n \gg \longrightarrow_\beta \ll n \gg (\lambda z \cdot \textbf{false})\textbf{true} \xrightarrow{*}_\beta (\lambda z \cdot \textbf{false})^n \textbf{true}$

Encoding test for zero

- $\textbf{iszero} := \lambda x \cdot x(\lambda z \cdot \textbf{false})\textbf{true}$
- $\textbf{iszero}\ \ll n \gg \longrightarrow_\beta \ll n \gg (\lambda z \cdot \textbf{false})\textbf{true} \xrightarrow{*}_\beta (\lambda z \cdot \textbf{false})^n \textbf{true}$
- $(\lambda z \cdot \textbf{false})^0 \textbf{true} = \textbf{true}$

Encoding test for zero

- $\textbf{iszero} := \lambda x \cdot x(\lambda z \cdot \textbf{false})\textbf{true}$
- $\textbf{iszero}\ \ll n \gg \longrightarrow_\beta \ll n \gg (\lambda z \cdot \textbf{false})\textbf{true} \xrightarrow{*}_\beta (\lambda z \cdot \textbf{false})^n \textbf{true}$
- $(\lambda z \cdot \textbf{false})^0 \textbf{true} = \textbf{true}$
- For $n > 0$, $(\lambda z \cdot \textbf{false})^n \textbf{true} = (\lambda z \cdot \textbf{false})((\lambda z \cdot \textbf{false})^{n-1} \textbf{true}) \longrightarrow_\beta \textbf{false}$

Encoding test for zero

- $\textbf{iszero} := \lambda x \cdot x(\lambda z \cdot \textbf{false})\textbf{true}$
- $\textbf{iszero}\ \ll n \gg \longrightarrow_\beta \ll n \gg (\lambda z \cdot \textbf{false})\textbf{true} \xrightarrow{*}_\beta (\lambda z \cdot \textbf{false})^n \textbf{true}$
- $(\lambda z \cdot \textbf{false})^0 \textbf{true} = \textbf{true}$
- For $n > 0$, $(\lambda z \cdot \textbf{false})^n \textbf{true} = (\lambda z \cdot \textbf{false})((\lambda z \cdot \textbf{false})^{n-1} \textbf{true}) \longrightarrow_\beta \textbf{false}$
- Thus

Encoding test for zero

- $\textbf{iszero} := \lambda x \cdot x(\lambda z \cdot \textbf{false})\textbf{true}$
- $\textbf{iszero}\ \ll n \gg \longrightarrow_\beta \ll n \gg (\lambda z \cdot \textbf{false})\textbf{true} \xrightarrow{*}_\beta (\lambda z \cdot \textbf{false})^n \textbf{true}$
- $(\lambda z \cdot \textbf{false})^0 \textbf{true} = \textbf{true}$
- For $n > 0$, $(\lambda z \cdot \textbf{false})^n \textbf{true} = (\lambda z \cdot \textbf{false})((\lambda z \cdot \textbf{false})^{n-1} \textbf{true}) \longrightarrow_\beta \textbf{false}$
- Thus
  - $\textbf{iszero}\ \ll 0 \gg \xrightarrow{*}_\beta \textbf{true}$

Encoding test for zero

- $\textbf{iszero} := \lambda x \cdot x(\lambda z \cdot \textbf{false})\textbf{true}$
- $\textbf{iszero}\ \ll n \gg \longrightarrow_\beta \ll n \gg (\lambda z \cdot \textbf{false})\textbf{true} \xrightarrow{*}_\beta (\lambda z \cdot \textbf{false})^n \textbf{true}$
- $(\lambda z \cdot \textbf{false})^0 \textbf{true} = \textbf{true}$
- For $n > 0$, $(\lambda z \cdot \textbf{false})^n \textbf{true} = (\lambda z \cdot \textbf{false})((\lambda z \cdot \textbf{false})^{n-1} \textbf{true}) \longrightarrow_\beta \textbf{false}$
- Thus
  - $\textbf{iszero}\ \ll 0 \gg \xrightarrow{*}_\beta \textbf{true}$
  - $\textbf{iszero}\ \ll n \gg \xrightarrow{*}_\beta \textbf{false}$ for $n > 0$

Recursive definitions

- $f(n) = \mu i : g(i, n)$ is expressed as follows:

```
int search(i, n) {
    if (iszero(g(i, n))) return i;
    else return search(i+1, n);
}
f(n) = search(0, n);
```

- The λ-expression **search** encoding `search` satisfies the following property:

  search « m » « n » $\xrightarrow{*}_{\beta}$ **if** (**iszero** (**g** « m » « n »)) **then** « m » **else** (**search** « $m + 1$ » « n »)

Recursive definitions

- Suppose **search** satisfies the following property:

$$\textbf{search} \xrightarrow{*}_{\beta} (\lambda cyx \cdot \textbf{if } (\textbf{iszero}(\textbf{g}\, y\, x)) \textbf{ then } y \textbf{ else } (c\ (\textbf{succ}\, y)\ x))\ \ \textbf{search}$$

Recursive definitions

- Suppose **search** satisfies the following property:

  $$\textbf{search} \xrightarrow{*}_{\beta} (\lambda cyx \cdot \textbf{if}\ (\textbf{iszero}(\textbf{g}\ y\ x))\ \textbf{then}\ y\ \textbf{else}\ (c\ (\textbf{succ}\ y)\ x))\ \ \textbf{search}$$

- Then it satisfies the following:

  $$\textbf{search}\ \langle\!\langle m \rangle\!\rangle\ \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_{\beta} \textbf{if}\ (\textbf{iszero}(\textbf{g}\ \langle\!\langle m \rangle\!\rangle\ \langle\!\langle n \rangle\!\rangle))\ \textbf{then}\ \langle\!\langle m \rangle\!\rangle\ \textbf{else}\ (\textbf{search}\ \langle\!\langle m+1 \rangle\!\rangle\ \langle\!\langle n \rangle\!\rangle)$$

Recursive definitions

- Suppose **search** satisfies the following property:

  $$\textbf{search} \xrightarrow{*}_{\beta} (\lambda cyx \cdot \textbf{if } (\textbf{iszero}(\textbf{g}\, y\, x)) \textbf{ then } y \textbf{ else } (c\ (\textbf{succ}\, y)\ x))\ \ \textbf{search}$$

- Then it satisfies the following:

  $$\textbf{search} \ll m \gg \ll n \gg \xrightarrow{*}_{\beta} \textbf{if } (\textbf{iszero}(\textbf{g} \ll m \gg \ll n \gg)) \textbf{ then } \ll m \gg \textbf{ else } (\textbf{search} \ll m+1 \gg \ll n \gg)$$

- So with $F := \lambda cyx \cdot \textbf{if } (\textbf{iszero } (\textbf{g}\, y\, x)) \textbf{ then } y \textbf{ else } (c(\textbf{succ}\, y)\ x)$ ...

Recursive definitions

- Suppose **search** satisfies the following property:

  $$\textbf{search} \xrightarrow{*}_{\beta} (\lambda cyx \cdot \textbf{if } (\textbf{iszero}(\textbf{g}\, y\, x)) \textbf{ then } y \textbf{ else } (c\ (\textbf{succ}\, y)\ x))\ \ \textbf{search}$$

- Then it satisfies the following:

  $$\textbf{search}\ \langle\!\langle m \rangle\!\rangle\ \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_{\beta} \textbf{if } (\textbf{iszero}(\textbf{g}\ \langle\!\langle m \rangle\!\rangle\ \langle\!\langle n \rangle\!\rangle)) \textbf{ then } \langle\!\langle m \rangle\!\rangle \textbf{ else } (\textbf{search}\ \langle\!\langle m+1 \rangle\!\rangle\ \langle\!\langle n \rangle\!\rangle)$$

- So with $F := \lambda cyx \cdot \textbf{if } (\textbf{iszero } (\textbf{g}\, y\, x)) \textbf{ then } y \textbf{ else } (c(\textbf{succ}\, y)\ x)$ ...
- ...we want $\textbf{search} \xrightarrow{*}_{\beta} F\ \textbf{search}$

Recursive definitions and fixed points

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

Recursive definitions and fixed points

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Taking $C = (\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$, we have

$$C \xrightarrow{*}_{\beta} F\ C$$

Recursive definitions and fixed points

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Taking $C = (\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$, we have

$$C \xrightarrow{*}_{\beta} F\ C$$

- Such a C is a **fixed point** of F

Recursive definitions and fixed points

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Taking $C = (\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$, we have

$$C \xrightarrow{*}_{\beta} F\ C$$

- Such a C is a **fixed point** of F
- **Recall**: x is a fixed point of a function f if $f\ (x) = x$

Recursive definitions and fixed points

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Taking $C = (\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$, we have

$$C \xrightarrow{*}_{\beta} F\ C$$

- Such a C is a **fixed point** of F
- **Recall**: x is a fixed point of a function f if $f\ (x) = x$
- Can we abstract the process of finding a fixed point?

Recursive definitions and fixed points

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Taking $C = (\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$, we have

$$C \xrightarrow{*}_{\beta} F\ C$$

- Such a C is a **fixed point** of F
- **Recall**: x is a fixed point of a function f if $f\ (x) = x$
- Can we abstract the process of finding a fixed point?
- Enter **fixed-point combinators!**

Recursive definitions and the **Y** combinator

- Define $\mathbf{Y} = \lambda f \cdot (\lambda x \cdot f\,(x x))\,(\lambda x \cdot f\,(x x))$

Recursive definitions and the **Y** combinator

- Define $\mathbf{Y} = \lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx))$
- $\mathbf{Y}\, F \longrightarrow_\beta (\lambda x \cdot F(xx))(\lambda x \cdot F(xx)) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$

Recursive definitions and the **Y** combinator

- Define $\mathbf{Y} = \lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx))$
- $\mathbf{Y}\, F \longrightarrow_\beta (\lambda x \cdot F(xx))(\lambda x \cdot F(xx)) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$
- $F(\mathbf{Y}\, F) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$

Recursive definitions and the **Y** combinator

- Define $\mathbf{Y} = \lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx))$
- $\mathbf{Y}\, F \longrightarrow_\beta (\lambda x \cdot F(xx))(\lambda x \cdot F(xx)) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$
- $F(\mathbf{Y}\, F) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$
- So there is a G such that $\mathbf{Y}\, F \overset{*}{\longrightarrow}_\beta G$ and $F(\mathbf{Y}\, F) \overset{*}{\longrightarrow}_\beta G$

Recursive definitions and the **Y** combinator

- Define $\mathbf{Y} = \lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx))$
- $\mathbf{Y}\, F \longrightarrow_\beta (\lambda x \cdot F(xx))(\lambda x \cdot F(xx)) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$
- $F(\mathbf{Y}\, F) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$
- So there is a G such that $\mathbf{Y}\, F \overset{*}{\longrightarrow}_\beta G$ and $F(\mathbf{Y}\, F) \overset{*}{\longrightarrow}_\beta G$
- We say that $\mathbf{Y}\, F =_\beta F\ (\mathbf{Y}\, F)$

Recursive definitions and the **Y** combinator

- Define $\mathbf{Y} = \lambda f \cdot (\lambda x \cdot f(xx))(\lambda x \cdot f(xx))$
- $\mathbf{Y}\ F \longrightarrow_\beta (\lambda x \cdot F(xx))(\lambda x \cdot F(xx)) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$
- $F(\mathbf{Y}\ F) \longrightarrow_\beta F(\lambda x \cdot F(xx))(\lambda x \cdot F(xx))$
- So there is a G such that $\mathbf{Y}\ F \overset{*}{\longrightarrow}_\beta G$ and $F(\mathbf{Y}\ F) \overset{*}{\longrightarrow}_\beta G$
- We say that $\mathbf{Y}\ F =_\beta F\ (\mathbf{Y}\ F)$
- For any F, $\mathbf{Y}\ F$ is a C such that $C =_\beta F\ C$

Recursive definitions and the Θ combinator

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

Recursive definitions and the Θ combinator

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Define $\Theta = (\lambda xy \cdot y(xxy))(\lambda xy \cdot y(xxy))$

Recursive definitions and the Θ combinator

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Define $\Theta = (\lambda xy \cdot y(xxy))(\lambda xy \cdot y(xxy))$
- $\Theta\ F = (\lambda xy \cdot y(xxy))(\lambda xy \cdot y(xxy))\ F \longrightarrow_{\beta} (\lambda y \cdot y(\ (\lambda xy \cdot y(xxy))\ (\lambda xy \cdot y(xxy))\ y)\ )\ F \longrightarrow_{\beta}$
  $F\ (\ (\lambda xy \cdot y(xxy))\ (\lambda xy \cdot y(xxy))\ F) = F\ (\Theta\ F)$

Recursive definitions and the Θ combinator

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Define $\Theta = (\lambda xy \cdot y(xxy))(\lambda xy \cdot y(xxy))$
- $\Theta\ F = (\lambda xy \cdot y(xxy))(\lambda xy \cdot y(xxy))\ F \longrightarrow_{\beta} (\lambda y \cdot y(\ (\lambda xy \cdot y(xxy))\ (\lambda xy \cdot y(xxy))\ y)\ )\ F \longrightarrow_{\beta}$
  $F\ (\ (\lambda xy \cdot y(xxy))\ (\lambda xy \cdot y(xxy))\ F) = F\ (\Theta\ F)$
- Thus $\Theta\ F \xrightarrow{*}_{\beta} F\ (\Theta\ F)$

Recursive definitions and the Θ combinator

- Given a λ-expression F, find an expression C such that

$$C \xrightarrow{*}_{\beta} F\ C$$

- Define $\Theta = (\lambda xy \cdot y(xxy))(\lambda xy \cdot y(xxy))$
- $\Theta\ F = (\lambda xy \cdot y(xxy))(\lambda xy \cdot y(xxy))\ F \longrightarrow_{\beta} (\lambda y \cdot y(\ (\lambda xy \cdot y(xxy))\ (\lambda xy \cdot y(xxy))\ y)\ )\ F \longrightarrow_{\beta}$ $F\ (\ (\lambda xy \cdot y(xxy))\ (\lambda xy \cdot y(xxy))\ F) = F\ (\Theta\ F)$
- Thus $\Theta\ F \xrightarrow{*}_{\beta} F\ (\Theta\ F)$
- For any F, $\Theta\ F$ is a C such that $C \xrightarrow{*}_{\beta} F\ C$

Back to μ-recursion

- $f(n) = \mu i : g(i, n)$ is encoded as follows:
  - $F = \lambda cyx \cdot$ **if** (**iszero**(**g** y x)) **then** y **else** (c(**succ** y)x)
  - **search** $= \Theta F$, and so **search** $\xrightarrow{*}_{\beta}$ F **search**
  - **f** $=$ **search** « 0 »
- **f** « n » $=$ **search** « 0 » « n »

Encoding μ-recursion

- Suppose $g(i, n) = 0$

Encoding μ-recursion

- Suppose $g(i, n) = 0$
  - Then $\mathbf{g}\ \ulcorner i \urcorner\ \ulcorner n \urcorner \xrightarrow{*}_{\beta} \ulcorner 0 \urcorner$

Encoding μ-recursion

- Suppose $g(i, n) = 0$
  - Then $\mathbf{g}\ \ll i \gg \ll n \gg \xrightarrow{*}_{\beta} \ll 0 \gg$
  - So $\mathbf{iszero}(\mathbf{g}\ \ll i \gg \ll n \gg) \xrightarrow{*}_{\beta} \mathbf{iszero}\ \ll 0 \gg \xrightarrow{*}_{\beta} \mathbf{true}$

Encoding μ-recursion

- Suppose $g(i, n) = 0$
  - Then $\mathbf{g}\ \ll i \gg \ll n \gg \xrightarrow{*}_\beta \ll 0 \gg$
  - So $\mathbf{iszero}(\mathbf{g}\ \ll i \gg \ll n \gg) \xrightarrow{*}_\beta \mathbf{iszero}\ \ll 0 \gg \xrightarrow{*}_\beta \mathbf{true}$
  - So

$$
\begin{aligned}
\mathbf{search}\ \ll i \gg \ll n \gg &\xrightarrow{*}_\beta F\ \mathbf{search}\ \ll i \gg \ll n \gg \\
&\xrightarrow{*}_\beta \mathbf{if}\ (\mathbf{iszero}(\mathbf{g}\ \ll i \gg \ll n \gg)) \\
&\quad\ \mathbf{then}\ \ll i \gg \\
&\quad\ \mathbf{else}\ (\mathbf{search}(\mathbf{succ}\ \ll i \gg)\ \ll n \gg) \\
&\xrightarrow{*}_\beta \mathbf{if\ true\ then}\ \ll i \gg \mathbf{\ else}\ (\mathbf{search}\ (\mathbf{succ}\ \ll i \gg)\ \ll n \gg) \\
&\xrightarrow{*}_\beta \ll i \gg
\end{aligned}
$$

Encoding μ-recursion

- Suppose $g(i, n) = k > 0$

Encoding μ-recursion

- Suppose $g(i, n) = k > 0$
  - Then $\mathbf{g}\,\ulcorner i \urcorner\,\ulcorner n \urcorner \xrightarrow{*}_{\beta} \ulcorner k \urcorner$

Encoding μ-recursion

- Suppose $g(i, n) = k > 0$
  - Then $\mathbf{g}\ \ulcorner i \urcorner\ \ulcorner n \urcorner \xrightarrow{*}_{\beta} \ulcorner k \urcorner$
  - So $\mathbf{iszero}(\mathbf{g}\ \ulcorner i \urcorner\ \ulcorner n \urcorner) \xrightarrow{*}_{\beta} \mathbf{iszero}\ \ulcorner k \urcorner \xrightarrow{*}_{\beta} \mathbf{false}$

Encoding μ-recursion

- Suppose $g(i, n) = k > 0$
  - Then $\mathbf{g}\ \ll i \gg \ll n \gg \xrightarrow{*}_\beta \ll k \gg$
  - So $\mathbf{iszero}(\mathbf{g}\ \ll i \gg \ll n \gg) \xrightarrow{*}_\beta \mathbf{iszero}\ \ll k \gg \xrightarrow{*}_\beta \mathbf{false}$
  - So

$$
\begin{aligned}
\mathbf{search}\ \ll i \gg \ll n \gg &\xrightarrow{*}_\beta F\ \mathbf{search}\ \ll i \gg \ll n \gg \\
&\xrightarrow{*}_\beta \mathbf{if}\ (\mathbf{iszero}(\mathbf{g}\ \ll i \gg \ll n \gg)) \\
&\quad\ \mathbf{then}\ \ll i \gg \\
&\quad\ \mathbf{else}\ (\mathbf{search}(\mathbf{succ}\ \ll i \gg)\ \ll n \gg) \\
&\xrightarrow{*}_\beta (\mathbf{if}\ \mathbf{false}\ \mathbf{then}\ \ll i \gg \mathbf{else}\ (\mathbf{search}\ (\mathbf{succ}\ \ll i \gg)\ \ll n \gg) \\
&\xrightarrow{*}_\beta \mathbf{search}\ (\mathbf{succ}\ \ll i \gg)\ \ll n \gg \\
&\xrightarrow{*}_\beta \mathbf{search}\ \ll i + 1 \gg\ \ll n \gg
\end{aligned}
$$

Encoding μ-recursion

- If $g(i, n) = 0$ then **search** $\langle\!\langle i \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_{\beta} \langle\!\langle i \rangle\!\rangle$

Encoding μ-recursion

- If $g(i, n) = 0$ then **search** $\langle\!\langle i \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_{\beta} \langle\!\langle i \rangle\!\rangle$
- If $g(i, n) > 0$ then **search** $\langle\!\langle i \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_{\beta}$ **search** $\langle\!\langle i + 1 \rangle\!\rangle \langle\!\langle n \rangle\!\rangle$

Encoding μ-recursion

- If $g(i, n) = 0$ then **search** $\langle\!\langle i \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_{\beta} \langle\!\langle i \rangle\!\rangle$
- If $g(i, n) > 0$ then **search** $\langle\!\langle i \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_{\beta}$ **search** $\langle\!\langle i + 1 \rangle\!\rangle \langle\!\langle n \rangle\!\rangle$
- Suppose now that $g(b, n) = 0$ and $g(a, n) > 0$ for all $a < b$

Encoding μ-recursion

- If $g(i, n) = 0$ then **search** $\langle\!\langle i \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle i \rangle\!\rangle$
- If $g(i, n) > 0$ then **search** $\langle\!\langle i \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_\beta$ **search** $\langle\!\langle i + 1 \rangle\!\rangle \langle\!\langle n \rangle\!\rangle$
- Suppose now that $g(b, n) = 0$ and $g(a, n) > 0$ for all $a < b$
- **search** $\langle\!\langle 0 \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_\beta$ **search** $\langle\!\langle 1 \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_\beta$ **search** $\langle\!\langle 2 \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_\beta \ldots \xrightarrow{*}_\beta$ **search** $\langle\!\langle b \rangle\!\rangle \langle\!\langle n \rangle\!\rangle \xrightarrow{*}_\beta \langle\!\langle b \rangle\!\rangle$

Encoding μ-recursion

- If $g(i,n) = 0$ then **search** $\ll i \gg \ll n \gg \xrightarrow{*}_\beta \ll i \gg$
- If $g(i,n) > 0$ then **search** $\ll i \gg \ll n \gg \xrightarrow{*}_\beta$ **search** $\ll i+1 \gg \ll n \gg$
- Suppose now that $g(b,n) = 0$ and $g(a,n) > 0$ for all $a < b$
- **search** $\ll 0 \gg \ll n \gg \xrightarrow{*}_\beta$ **search** $\ll 1 \gg \ll n \gg \xrightarrow{*}_\beta$ **search** $\ll 2 \gg \ll n \gg \xrightarrow{*}_\beta \ldots \xrightarrow{*}_\beta$ **search** $\ll b \gg \ll n \gg \xrightarrow{*}_\beta \ll b \gg$
- Thus **f** $\ll n \gg \xrightarrow{*}_\beta$ **search** $\ll 0 \gg \ll n \gg \xrightarrow{*}_\beta \ll b \gg$ where $b = \mu i : g(i, \vec{n})$

Encoding μ-recursion

- If $g(i,n) = 0$ then **search** $\ll i \gg \ll n \gg \xrightarrow{*}_\beta \ll i \gg$
- If $g(i,n) > 0$ then **search** $\ll i \gg \ll n \gg \xrightarrow{*}_\beta$ **search** $\ll i+1 \gg \ll n \gg$
- Suppose now that $g(b,n) = 0$ and $g(a,n) > 0$ for all $a < b$
- **search** $\ll 0 \gg \ll n \gg \xrightarrow{*}_\beta$ **search** $\ll 1 \gg \ll n \gg \xrightarrow{*}_\beta$ **search** $\ll 2 \gg \ll n \gg \xrightarrow{*}_\beta \ldots \xrightarrow{*}_\beta$ **search** $\ll b \gg \ll n \gg \xrightarrow{*}_\beta \ll b \gg$
- Thus **f** $\ll n \gg \xrightarrow{*}_\beta$ **search** $\ll 0 \gg \ll n \gg \xrightarrow{*}_\beta \ll b \gg$ where $b = \mu i : g(i, \vec{n})$
- The expression **Mu** $= \lambda g \cdot \Theta G \ll 0 \gg$ encodes the schema of μ-recursion where

$$G = \lambda cyx \cdot \textbf{if } (\textbf{iszero } (g\ y\ x)) \textbf{ then } y \textbf{ else } (c(\textbf{succ} y)x)$$

Encoding μ-recursion

- Suppose $g(a, n)$ is defined and > 0 for all a

Encoding μ-recursion

- Suppose $g(a, n)$ is defined and > 0 for all a
- **search** « 0 » « n » $\xrightarrow{*}_\beta$ **search** « 1 » « n » $\xrightarrow{*}_\beta$ **search** « 2 » « n » $\xrightarrow{*}_\beta$ **search** « 3 » « n » ...

Encoding μ-recursion

- Suppose $g(a, n)$ is defined and > 0 for all a
- **search** « 0 » « n » $\xrightarrow{*}_\beta$ **search** « 1 » « n » $\xrightarrow{*}_\beta$ **search** « 2 » « n » $\xrightarrow{*}_\beta$ **search** « 3 » « n » ...
- Thus no reduction sequence starting from **f** « n » terminates

Encoding μ-recursion

- Suppose $g(a, n)$ is defined and > 0 for all a
- **search** «0» «n» $\xrightarrow{*}_\beta$ **search** «1» «n» $\xrightarrow{*}_\beta$ **search** «2» «n» $\xrightarrow{*}_\beta$ **search** «3» «n» ...
- Thus no reduction sequence starting from **f** «n» terminates
- Suppose $g(b, n)$ is undefined for some b, and $g(a, n) \neq 0$ for all $a < b$

Encoding μ-recursion

- Suppose $g(a, n)$ is defined and > 0 for all a
- **search** « 0 » « n » $\xrightarrow{*}_\beta$ **search** « 1 » « n » $\xrightarrow{*}_\beta$ **search** « 2 » « n » $\xrightarrow{*}_\beta$ **search** « 3 » « n » ...
- Thus no reduction sequence starting from **f** « n » terminates
- Suppose $g(b, n)$ is undefined for some b, and $g(a, n) \neq 0$ for all $a < b$
- Thus **g** « b » « n » has no terminating reduction sequence, and similarly for **f** « n »