Functional Programming and Scheme

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

Church Encoding

Natural numbers

n-fold composition

Church numerals: $\underline{n} \triangleq \lambda f\ z. f^{n}\ z$

Note: $\underline{n}$ is the encoding of number *n*

$\underline{0} \triangleq \lambda f\ z. f^{0}\ z = \lambda f\ z. z$
$\underline{1} \triangleq \lambda f\ z. f^{1}\ z = \lambda f\ z. (f\ z)$
$\underline{2} \triangleq \lambda f\ z. f^{2}\ z = \lambda f\ z. (f\ (f\ z))$
...

Church Encoding

Natural numbers

$$\underline{n} \triangleq \lambda f\ z.\, f^{n}\ z$$

Encoding of “+ 1”?

Goal: $\mathrm{SUCC}\ \ \underline{n}\ = \lambda f\ z.\, f^{n+1}\ z$

Definition $\mathrm{SUCC} \triangleq\ \lambda n\ f\ z.\ \ (f\ (n\ f\ z))$

Church Encoding

Natural numbers

$$\underline{n} \triangleq \lambda f\ z.\, f^{n}\ z$$

Encoding of "+"?

Goal: $\mathrm{PLUS}\ \ \underline{n_1}\ \underline{n_2} = \lambda f\ z.\, f^{n_1+n_2}\ z$

Definition $\mathrm{PLUS} \triangleq \lambda n_1\ n_2.\ \ (n_1\ \mathrm{SUCC}\ n_2)$

Encoding of "×" ? (Check solution in Note 2)

Church Encoding: Example

Natural numbers

$$\underline{n} \triangleq \lambda f\ z.\, f^{n}\ z$$

Definition $\mathrm{PLUS} \triangleq \lambda n_1\ n_2.\, (n_1\ \mathrm{SUCC}\ n_2)$

Check that $\mathrm{PLUS}\ \underline{1}\ \underline{2}\ = \underline{3}$ (Note 2)

Named Functions

Use definition $\mathrm{SUCC} \triangleq \lambda n\, f\, z.\ \left(f\ (n\ f\ z)\right)$
in term (SUCC (SUCC $\underline{1}$))?

Syntax: **let** (name def) body (or, **let** name = def **in** body)
E.g., **let** SUCC $(\lambda n\, f\, z.\ \left(f\ (n\ f\ z)\right))$ (SUCC (SUCC $\underline{1}$))

let (name def) body
is just a shorthand for $(\lambda \mathrm{name}.\ \mathrm{body})\ \mathrm{def}$

Pure vs. Applied λ-Calculus

Pure λ-Calculus: the calculus discussed so far

Applied λ-Calculus:

- Built-in values and data structures
  (e.g., 1, 2, 3, true, false, (1 2 3))
- Built-in functions
  (e.g., +, $*$, /, and, or)
- Named functions
- Recursion

All features can be encoded in the pure λ-Calculus!

Functional Languages