

Church Encoding

COMP3610 – Principles of Programming Languages

Ben Lippmeier

Australian National University
Semester 2, 2009

Derived Forms

The λ -calculus has textual substitution as its primitive operation.

We can define **let**-expressions in terms of features we already have.

$$\mathbf{let } x = e_1 \mathbf{ in } e_2 \stackrel{\text{def}}{=} (\lambda x. e_2) e_1$$

To evaluate a program with **let**-expressions, simply translate it to the pure lambda calculus and evaluate it with our original rules.

The **let**-expression is *syntactic sugar* which makes a more primitive mechanism (substitution) easier for the programmer to work with.

Alonzo Church

- Lived 1903 - 1995.
- One of the founders of computer science.
- Introduced the λ -calculus.
- Church-Rosser Theorem.
 λ -calculus is confluent.
- Church-Turing Theorem.
The system of basic arithmetic and predicate calculus is undecidable.
- Church-Turing Thesis.
Every effectively calculable function is a computable function.



Defining Functions

$$\mathbf{let } f \overline{x_n} = e_1 \mathbf{ in } e_2 \stackrel{\text{def}}{=} \mathbf{let } f = (\lambda \overline{x_n}. e_1) \mathbf{ in } e_2$$
$$\stackrel{\text{def}}{=} (\lambda f. e_2) (\lambda \overline{x_n}. e_1)$$

$$\mathbf{let } fst \ x_1 \ x_2 = x_1$$
$$\mathbf{in } \mathbf{fst } \ \mathbf{cat } \ \mathbf{dog}$$
$$\xrightarrow{\text{def}} (\lambda \mathbf{fst}. \mathbf{fst } \ \mathbf{cat } \ \mathbf{dog}) (\lambda x_1 \ x_2. x_1)$$
$$\longrightarrow (\lambda x_1 \ x_2. x_1) \ \mathbf{cat } \ \mathbf{dog}$$
$$\longrightarrow (\lambda x_2. \mathbf{cat}) \ \mathbf{dog}$$
$$\longrightarrow \mathbf{cat}$$

Booleans and choice

A Boolean value expresses a choice between two options:

$$\text{true} \stackrel{\text{def}}{=} \lambda x y. x$$
$$\text{false} \stackrel{\text{def}}{=} \lambda x y. y$$

true *cat dog*

$$\xrightarrow{\text{def}} \underline{(\lambda x y. x) \text{ cat dog}}$$
$$\longrightarrow (\lambda y. \text{cat}) \text{ dog}$$
$$\longrightarrow \text{cat}$$

false *cat dog*

$$\xrightarrow{\text{def}} \underline{(\lambda x y. y) \text{ cat dog}}$$
$$\longrightarrow (\lambda y. y) \text{ dog}$$
$$\longrightarrow \text{dog}$$

Boolean operators

$$e_1 \text{ and } e_2 \stackrel{\text{def}}{=} \text{if } e_1 \text{ then } e_2 \text{ else false}$$

true and false

$$\xrightarrow{\text{def}} \text{if true then false else false}$$
$$\xrightarrow{\text{def}} \text{true false false}$$
$$\longrightarrow (\lambda x y. x) (\lambda x y. y) (\lambda x y. y)$$
$$\longrightarrow (\lambda y. (\lambda x y. y)) (\lambda x y. y)$$
$$\longrightarrow (\lambda x y. y)$$
$$\xrightarrow{\text{def}} \text{false}$$

if-then-else

$$\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \stackrel{\text{def}}{=} e_1 e_2 e_3$$

if true then *cat* else *dog*

$$\xrightarrow{\text{def}} \text{true cat dog}$$
$$\xrightarrow{\text{def}} \underline{(\lambda x y. x) \text{ cat dog}}$$
$$\longrightarrow (\lambda y. \text{cat}) \text{ dog}$$
$$\longrightarrow \text{cat}$$

What is computation? (reloaded)

Representation, Propagation, Interaction

true and false

$$\xrightarrow{\text{def}} \text{if true then false else false} \quad (\text{change representation})$$
$$\xrightarrow{\text{def}} \text{true false false} \quad (\text{change representation})$$
$$\longrightarrow (\lambda x y. x) (\lambda x y. y) (\lambda x y. y) \quad (\text{propagate second term into the first})$$
$$\longrightarrow (\lambda y. (\lambda x y. y)) (\lambda x y. y) \quad (\text{terms interact...})$$
$$\longrightarrow (\lambda x y. y) \quad (\text{...affecting subsequent propagation})$$
$$\xrightarrow{\text{def}} \text{false} \quad (\text{change representation})$$

Church Numerals

Define natural numbers inductively:

- **zero** is a natural number.
- if n is a natural number then (**succ** n) is also a natural number.

$0 = \mathbf{zero}$
 $1 = \mathbf{succ\ zero}$
 $2 = \mathbf{succ\ (succ\ zero)}$
 $3 = \mathbf{succ\ (succ\ (succ\ zero))}$
 $4 = \dots$

But how do we define **succ** and **zero**?

Successors

The **succ** function takes a number n and adds another s application.

$$\mathbf{succ} \equiv \lambda n. \lambda s z. s (n s z)$$

$$\begin{aligned} \mathbf{succ} \ c_2 & \\ \xrightarrow{\text{def}} & (\lambda n. \lambda s z. s (n s z)) \ (\lambda s z. s (s z)) \\ \xrightarrow{\alpha} & (\lambda n. \lambda s z. s (n s z)) \ (\lambda a b. a (a b)) \\ \longrightarrow & (\lambda s z. s ((\lambda a b. a (a b)) s z)) \\ \longrightarrow & (\lambda s z. s ((\lambda b. s (s b)) z)) \\ \longrightarrow & (\lambda s z. s (s (s z))) \\ \xrightarrow{\text{def}} & c_3 \end{aligned}$$

We are more interested in *interaction* than *representation*.

Booleans embody the idea of *choosing between two things*.

Numbers embody the idea of *doing something multiple times*.

We can make the *something* abstract:

$c_0 \stackrel{\text{def}}{=} \lambda s z. z$
 $c_1 \stackrel{\text{def}}{=} \lambda s z. s z$
 $c_2 \stackrel{\text{def}}{=} \lambda s z. s (s z)$
 $c_3 \stackrel{\text{def}}{=} \lambda s z. s (s (s z))$
 $c_4 \stackrel{\text{def}}{=} \dots$

Now it is up to the caller to decide what the 'something' is.

Addition

$$\mathbf{plus} \equiv \lambda m. \lambda n. \lambda s z. m s (n s z)$$

$$\begin{aligned} \mathbf{plus} \ c_2 \ c_3 & \\ \xrightarrow{\text{def}} & (\lambda m. \lambda n. \lambda s z. m s (n s z)) \ c_2 \ c_3 \\ \xrightarrow{*} & (\lambda s z. c_2 s (c_3 s z)) \\ \xrightarrow{\text{def}} & (\lambda s z. c_2 s ((\lambda s z. s (s (s z))) s z)) \\ \longrightarrow & (\lambda s z. c_2 s (s (s (s z)))) \\ \xrightarrow{\text{def}} & (\lambda s z. (\lambda s z. s (s z)) s (s (s (s z)))) \\ \xrightarrow{*} & (\lambda s z. (s (s (s (s (s z)))))) \\ \xrightarrow{\text{def}} & c_5 \end{aligned}$$

Recursion

$$Y \stackrel{\text{def}}{=} \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$$

$Y \text{ fun}$

$$\xrightarrow{\text{def}} (\lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))) \text{ fun}$$

$$\longrightarrow (\lambda x. \text{fun} (x x)) (\lambda x. \text{fun} (x x))$$

$$\longrightarrow \text{fun} ((\lambda x. \text{fun} (x x)) (\lambda x. \text{fun} (x x)))$$

$$\xrightarrow{\text{def}} \text{fun} (Y \text{ fun})$$

$$\xrightarrow{*} \text{fun} (\text{fun} (Y \text{ fun}))$$

$$\xrightarrow{*} \dots$$