

## History

- In 1936,
  - Alan Turing invented Turing machines, defined a notion of computable functions
  - Alonzo Church invented  $\lambda$ -calculus, defined a notion of computable functions
  - Definitions of computability turn out to be the same.

## Un(i)typed $\lambda$ -calculus

April 3, 2002

CS 131: Programming Languages

## Syntax

- Pure lambda calculus:

$M, N ::= x$	<i>variables</i>
$\lambda x. M$	<i>functions</i>
$M N$	<i>applications</i>

- That's it!

## Note on Syntax

- Anonymous function values  $\lambda x. M$  show up under many different guises in functional languages.

$x \mapsto M$	(Math)
$(x) \Rightarrow M$	(rex)
$(\text{lambda } (x) M)$	(Scheme)

$\text{fn } x \Rightarrow M$	(SML)
$\backslash x \rightarrow M$	(Haskell)

## Conventions

- Terms differing only in names of bound variables are considered the same term
  - In the term  $\lambda x.M$ , variable  $x$  is bound in  $M$
- Application associates leftward

$$xyzw = ((xy)z)w$$

- Function bodies are as large as possible.

$$\lambda x.yx = \lambda x.(yx) \neq (\lambda x.y)x$$

## $\beta$ -Reduction Summarized

- Look for the pattern

$$(\lambda x.M)N$$

occurring anywhere, and replace it with

$$M[x \rightarrow N]$$

## One-step $\beta$ -Reduction

- The relation  $\rightarrow_\beta$  is defined by:

capture-free substitution!

$$\frac{}{(\lambda x.M)N \rightarrow_\beta M[x \rightarrow N]}$$

$$\frac{M \rightarrow_\beta M'}{MN \rightarrow_\beta M'N}$$

$$\frac{N \rightarrow_\beta N'}{MN \rightarrow_\beta MN'}$$

$$\frac{M \rightarrow_\beta M'}{\lambda x.M \rightarrow_\beta \lambda x.M'}$$

## A Worked Example

$$\underline{(\lambda b. (\lambda x. (\lambda y. ((b y) x))) (\lambda w. (\lambda z. w)))}$$

$$\rightarrow_\beta \lambda x. (\lambda y. (((\lambda w. (\lambda z. w)) \underline{y}) x))$$

$$\rightarrow_\beta \lambda x. (\lambda y. ((\lambda z. y) x))$$

$$\rightarrow_\beta \lambda x. (\lambda y. y)$$

$$\underline{(\lambda b. \lambda x. \lambda y. b y x) (\lambda w. \lambda z. w)}$$

$$\rightarrow_\beta \lambda x. \lambda y. (\lambda w. \lambda z. w) \underline{y} x$$

$$\rightarrow_\beta \lambda x. \lambda y. (\lambda z. y) \underline{x}$$

$$\rightarrow_\beta \lambda x. \lambda y. y$$

## Exercise

- Reduce the following term:  
 $(\lambda x. x \ x) ((\lambda y. y) (\lambda z. z))$

## Other Relations

- The relation  $\rightarrow_{\beta}^*$  is defined to be the reflexive, transitive closure of  $\rightarrow_{\beta}$ 
  - i.e., 0 or more  $\rightarrow_{\beta}$  steps.
- The relation  $\leftrightarrow_{\beta}^*$  is defined to be the reflexive, transitive, *symmetric* closure of  $\rightarrow_{\beta}$ .
  - Often referred to as *conversion*
  - Provides a notion of program equivalence

## Conversion Formalized

$$\frac{}{M \leftrightarrow_{\beta}^* M} \qquad \frac{M_1 \rightarrow_{\beta} M_2}{M_1 \leftrightarrow_{\beta}^* M_2}$$

$$\frac{M_2 \leftrightarrow_{\beta}^* M_1}{M_1 \leftrightarrow_{\beta}^* M_2}$$

$$\frac{M_1 \leftrightarrow_{\beta}^* M_2 \quad M_2 \leftrightarrow_{\beta}^* M_3}{M_1 \leftrightarrow_{\beta}^* M_3}$$

## Programming in $\lambda$ -Calculus

- Want terms in the  $\lambda$ -calculus that "act like"
  - booleans
  - numbers
  - conditionals
  - arithmetic operations
  - pairs and projections
  - etc.
- Many different ways to do encodings
  - I'll just show one example of each

## Encoding Booleans

- We use the following definition:

**tt** :=  $\lambda x. \lambda y. x$   
**ff** :=  $\lambda x. \lambda y. y$   
**if** :=  $\lambda z. z$

**if tt** M N  $\leftrightarrow_{\beta}^*$   
**if ff** M N  $\leftrightarrow_{\beta}^*$

## Exercises

- Find a term **not** such that  
**not tt**  $\leftrightarrow_{\beta}^*$  **ff**  
**not ff**  $\leftrightarrow_{\beta}^*$  **tt**
- Define **and** and **or**

## Pairs

- We use the following definition:

**<M,N>** :=  $\lambda f. f M N$   
**fst** :=  $\lambda p. (p (\lambda x. \lambda y. x))$   
**snd** :=  $\lambda p. (p (\lambda x. \lambda y. y))$

- Show that

**fst** **<M,N>**  $\leftrightarrow_{\beta}^*$  M  
**snd** **<M,N>**  $\leftrightarrow_{\beta}^*$  N

## Natural Numbers

- Church numerals:

**0** :=  $\lambda f. \lambda b. b$   
**1** :=  $\lambda f. \lambda b. f(b)$   
**2** :=  $\lambda f. \lambda b. f(f(b))$

...

**n** :=  $\lambda f. \lambda b. f^n(b)$

$$\boxed{\begin{array}{l} \mathbf{n} \ x \ y \\ \leftrightarrow_{\beta}^* \ x^n(y) \end{array}}$$

**succ** :=  $\lambda m. \lambda f. \lambda b. f(m f b)$

## Is this Reasonable?

- Yes!

```
succ 'n'  
= (λm.λf.λb.f(m f b)) 'n'  
↔β* λf.λb.f('n' f b)  
↔β* λf.λb.f(fn b)  
= λf.λb.fn+1 b  
= 'n+1'
```

## Exercises

- Find an alternative definition for **succ**  
- Hint:  $1+n = n+1$
- Find terms **plus** and **times** such that  
**plus** 'm' 'n' ↔<sub>β</sub>\* 'm+n'  
**times** 'm' 'n' ↔<sub>β</sub>\* 'mn'
- Find a term **iszero** such that  
**iszero** '0' ↔<sub>β</sub>\* **tt**  
**iszero** 'n' ↔<sub>β</sub>\* **ff** for any  $n > 0$