History

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

Syntax

- Pure lambda calculus:
  \[
  M, N ::= x \quad \text{variables} \\
  \quad | \lambda x. M \quad \text{functions} \\
  \quad | M N \quad \text{applications}
  \]

- That's it!

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 as as large as possible.
  \[
  \lambda x.yx = \lambda x.(yx) \neq (\lambda x.y)x
  \]
One-step $\beta$-Reduction

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

$$
\begin{array}{c}
(\lambda x . M) N \rightarrow_\beta M[x \rightarrow N] \\
M \rightarrow_\beta M' \\
M N \rightarrow_\beta M' N \\
N \rightarrow_\beta N' \\
M N \rightarrow_\beta M' N' \\
M \rightarrow_\beta M' \\
\lambda x . M \rightarrow_\beta \lambda x . M'
\end{array}
$$

β-Reduction

- 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 $\rightarrow_\beta$.

Example

- Reduce the following term:

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

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:

\[
\begin{align*}
\text{tt} & := \lambda x.\lambda y.x \\
\text{ff} & := \lambda x.\lambda y.y
\end{align*}
\]

\[
\begin{align*}
\text{tt} \ M \ N & \leftrightarrow^* \ ?? \\
\text{ff} \ M \ N & \leftrightarrow^* \ ??
\end{align*}
\]

Exercises

- Find a term not such that
  \[
  \text{not tt} \leftrightarrow^* \text{ff}
  \]
  \[
  \text{not ff} \leftrightarrow^* \text{tt}
  \]

- Define and and or

Pairs

- We use the following definition:

\[
\begin{align*}
\langle M, N \rangle & := \lambda f. f \ M \ N \\
\text{fst} & := \lambda p. (p \ \text{tt}) \\
\text{snd} & := \lambda p. (p \ \text{ff})
\end{align*}
\]

- Show that

\[
\begin{align*}
\text{fst} \ \langle M, N \rangle & \leftrightarrow^* M \\
\text{snd} \ \langle M, N \rangle & \leftrightarrow^* N
\end{align*}
\]

Exercises

- The following holds when \( M \) is a pair.

\[
\langle \text{fst} \ M, \text{snd} \ M \rangle \leftrightarrow^* M
\]

- Is this true for all \( M \)?
  - Don't actually have mechanism to prove this yet
  - But do you have a guess?
Natural Numbers

- Church numerals:
  
  \begin{align*}
  0 & := \lambda f. \lambda b. b \\
  1 & := \lambda f. \lambda b. f(b) \\
  2 & := \lambda f. \lambda b. f(f(b)) \\
  3 & := \lambda f. \lambda b. f(f(f(b))) \\
  \vdots \\
  n & := \lambda f. \lambda b. f^n(b) \\
  \end{align*}

  \text{succ} := \lambda n. \lambda f. \lambda b. f(n \ f \ b)

Exercises

- Find an alternative definition for \text{succ}
  - Hint: \(1+n = n+1\)

- Find a term \text{iszero} such that
  \[
  \text{iszero} 0 \leftrightarrow \text{tt} \\
  \text{iszero} n \leftrightarrow \text{ff} \quad \text{for any } n>0
  \]

- Find terms \text{plus} and \text{times} such that
  \[
  \begin{align*}
  \text{plus} m n & \leftrightarrow (m+n) \\
  \text{times} m n & \leftrightarrow (mn)
  \end{align*}
  \]