Un(i)typed λ-calculus

April 2, 2001

CS 131: Programming Languages

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          variables
      |  λx.M       functions
      |  M N        applications
```

- That's it!

Conventions

- Terms differing only in names of bound variables are considered the same term
  In the term `λx.M`, variable `x` is bound in `M`
- Application associates leftward
  `xyzw   =   ((xy)z)w`

- Function bodies as as large as possible.
  `λx.yx = λx.(yx)  ≠  (λx.y)x`

One-step β-Reduction

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

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

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

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

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

Programming in λ-Calculus

- Want terms in the λ-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:

$\mathbf{tt} := \lambda x.\lambda y.x$
$\mathbf{ff} := \lambda x.\lambda y.y$

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

Exercises

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

- Define **and** and **or**

Pairs

- We use the following definition:

$<M,N> := \lambda f.f\ M\ N$
$\mathbf{fst} := \lambda p.(p\ \mathbf{tt})$
$\mathbf{snd} := \lambda p.(p\ \mathbf{ff})$

- Show that
  $\mathbf{fst}\ <M,N> \leftrightarrow_\beta^* M$
  $\mathbf{snd}\ <M,N> \leftrightarrow_\beta^* N$

Exercises

- The following holds when M is a pair.

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

- 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:

```
0  :=   λf.λb.b
1  :=   λf.λb.f(b)
2  :=   λf.λb.f(f(b))
3  :=   λf.λb.f(f(f(b)))
...
n  :=   λf.λb.f^n(b)

succ := λn.λf.λb.f(n f b)
```

Exercises

- Find an alternative definition for `succ`
  - Hint: `1+n = n+1`
- Find a term **iszero** such that
  iszero 0 $\leftrightarrow_\beta^*$ **tt**
  iszero n $\leftrightarrow_\beta^*$ **ff** for any n>0
- Find terms **plus** and **times** such that
  plus m n $\leftrightarrow_\beta^*$ **(m+n)**
  times m n $\leftrightarrow_\beta^*$ **(mn)**