More Lambda Calculus: Arithmetic and Fixed Points

April 4, 2001

CS 131: Programming Languages

Review: Syntax

- Pure lambda calculus:

M,N ::=	x		*variables*
	|	λx.M	*functions*
	|	M N	*applications*

- That's it!

Review: Parenthesis 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

Review: One-step β-Reduction

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

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

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

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

Review: Reduction, Conversion

- 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 $\rightarrow_\beta$.

Review: Encodings

- Last class we showed how to *encode* booleans and pairs
  - Terms **tt** and **ff** such that

    tt M N $\leftrightarrow_\beta^*$ M

    ff M N $\leftrightarrow_\beta^*$ N
  - Terms **fst** and **snd** and **<**M,N**>** such that

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

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

Review: Encodings

- Also defined Church numerals

  $\ulcorner 0 \urcorner$:= $\lambda f.\lambda b.b$

  $\ulcorner 1 \urcorner$:= $\lambda f.\lambda b.f(b)$

  $\ulcorner 2 \urcorner$:= $\lambda f.\lambda b.f(f(b))$

  ...

  $\ulcorner n \urcorner$:= $\lambda f.\lambda b.f^n(b)$
- Operations

  succ $\ulcorner n \urcorner$ $\leftrightarrow_\beta^*$ $\ulcorner n+1 \urcorner$

  iszero $\ulcorner 0 \urcorner$ $\leftrightarrow_\beta^*$ **tt**

  iszero $\ulcorner n \urcorner$ $\leftrightarrow_\beta^*$ **ff** (for all n>0)

Review: Reduction

$\underline{(\lambda b.\lambda x.\lambda y.b\ y\ x)(\lambda w.\lambda z.w)}$ [**not tt**]

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

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

$\rightarrow_\beta \lambda x.\lambda y.y$ [**ff**]

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

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

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

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

Addition and Multiplication

- Find terms **plus** and **times** such that

  plus ⌜m⌝ ⌜n⌝ $\leftrightarrow_\beta^*$ ⌜m+n⌝

  times ⌜m⌝ ⌜n⌝ $\leftrightarrow_\beta^*$ ⌜mn⌝

Predecessor

- Tricky to subtract with Church numerals
- Key idea: consider the sequence

  <⌜0⌝,⌜0⌝**>** **<**⌜0⌝,⌜1⌝**>** **<**⌜1⌝,⌜2⌝**>** **<**⌜2⌝,⌜3⌝**>** ...

  pred' := λn.(n (λp.**<snd** p,**succ**(**snd** p)**>**)
  <0,0**>)**

  pred := λn.(**fst** (**pred'** n))

Fixed Points

1. For *every* λ-calculus term M, there exists N such that

   M(N) $\leftrightarrow_\beta^*$ N

   (including M=**not** and M=**succ**)

2. A term N with this property is called a *fixed point* of M.

3. Fixed points can be found *uniformly*. That is, there is a term **Y** such that

   M(**Y**(M)) $\leftrightarrow_\beta^*$ **Y**(M)

   Namely,

   Y := λf.(λx.f(xx))(λx.f(xx))

Exercises

- Show that

  M(**Y**(M)) $\leftrightarrow_\beta^*$ **Y**(M)

Definitions

- The definitions we have seen so far have all been *abbreviations* or *macros*
- Convenient shorthand, letting us write
  not tt
  instead of
  (λb.λx.λy.b y x)(λw.λz.w).
- But, we can always get rid of all abbreviations by replacing them with their definitions.

Recursive Definitions

- But what about defining recursive functions, like factorial?
- A definition like

```
fact := λn.(iszero n) ⌜1⌝
              (times n (fact (pred n)))
```

  is not a simple macro, but a circular definition!
- There's no way to write `fact ⌜7⌝` without referring to `fact`.
- Why should we believe this defines anything at all?

A Higher-Order Function

- Consider the following *non-circular* definition:

```
F :=  λg.λn.(iszero n) ⌜1⌝
              (times n (g (pred n)))
```

- So

```
F(f) ↔β* λn.(iszero n) ⌜1⌝
              (times n (f (pred n)))
```

- If the argument to **F** was *already* the factorial function, we'd get the same thing back.
- Thus the factorial function is a fixed point of **F**.
- We know that **Y**(**F**) is a fixed point of **F**.
- Hence **Y**(**F**) is the factorial function. (!?)

Applying Factorial

```
F    :=  λf.λn.(iszero n) ⌜1⌝
                (times n (f (pred n)))
fact := Y(F)

fact(⌜n⌝)
  ↔β* (F(fact))(⌜n⌝)
  ↔β* (iszero ⌜n⌝) ⌜1⌝
          (times ⌜n⌝ (fact (pred ⌜n⌝)))
```

Applying Factorial

`fact(⌜2⌝)` $\leftrightarrow_{\beta}^{*}$

Recursive Definitions

- Consider the SML definition
  `fun g(n) = ...code involving g...`
- This is convenient syntax for
  `val rec g = (fn n => ...code involving g...)`
- The function we want is a solution to this *equation*
  `g == (fn n => ...code involving g...)`
- Hence the function we want is a fixed point of
  `fn g => (fn n => ...code involving g...)`

Fibonacci Example

```
FIB :=  λg.λn.(iszero n) ⌜0⌝
                (iszero (pred n) ⌜1⌝
                  (plus (g (pred n))
                        (g (pred (pred n)))))
fib := Y(FIB)
```

Fixed Points

- Every term has at least one fixed point.
  - Even **succ** and **not**!
- Some terms have many fixed points
  - For example, λn.n
  - Or, **FIB**
    - Recall the `fib` and `intfib` functions from Assignment 1!
  - **Y** picks out the unique *least* fixed point.
    - Which turns out to be the one we "expect".
    - Yields answers as infrequently as possible.