CSE114A lecture 2

lambda calculus!

the Greek letter λ

a system of reasoning

1936 - both Turing machines and λ-calculus were invented!

A very simple programming language

variables

~~types~~

~~loops~~

~~state/assignment~~

recursion (encodable)

functions abstractions

operators (encodable)

conditionals (encodable)

~~memory model~~

All we've got:

- variables
- functions
  - way to define them
  - way to call them

$$
\begin{aligned}
e ::= \;& x, y, z && \text{(variables)} \\
\mid\;& \lambda \underline{x}.\ \underline{e} && \text{(function definition)} \\
\mid\;& \underline{e_1}\ e_2 && \text{(function application, aka function call)}
\end{aligned}
$$

e_1: function you're calling

e_2: argument to that function

formal parameter

```
def f(x):
    return x+3
```

body

```
> f(5)
8
```

Computation in λ-calculus all boils down to substitution.

The simplest function: the identity function

$\lambda x.\ x$ (on-paper syntax)

`\x -> x` (Elsa syntax)

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

function application

$$\cancel{\lambda x. (x}\ y)$$

```
def f(y):
    return y
```

```
> f(z)
z
```

The essence of computation in λ-calculus is substitution, via β-reduction

$$\underbrace{(\lambda \underline{x}. \overbrace{e_1}^{\text{body}})}_{\text{function}} \; \underbrace{e_2}_{\text{argument}} \longrightarrow_\beta \underbrace{e_1[\underline{x} := e_2]}$$

"e_1, but with all* occurrences* of x replaced with e_2"

* note: this is a lie

$$(\lambda x. \underbrace{x}) \; \underbrace{(\lambda y. y)}_{\text{argument}} \longrightarrow_\beta \lambda y. y$$

$$x[x := \lambda y. y] = \underline{\lambda y. y}$$

$$\left.\begin{array}{l}\lambda x. x\\ \lambda y. y\end{array}\right\}\ \text{these expressions are } \alpha\text{-equivalent}$$

Cool thing:

λ-calculus is a naturally parallelizable model of computation

(because you can pick anywhere in an expression to evaluate)

Let's program with λ-calc!

let TRUE = $\lambda x\ y.\ x$

let FALSE = $\lambda x\ y.\ y$

How do we encode 'if'?

let ITE = $\lambda b\ x\ y.\ \boxed{b\ x\ y}$

condition you're checking

value if b evals to true

value if b evals to false

$$\text{ITE} \; \underline{\text{TRUE}} \; \text{FALSE} \; \text{TRUE}$$

$$\text{let ITE} = \lambda b \; x \; y . \; b \; x \; y$$

$$\text{let TRUE} = \lambda x \; y . \; x$$

$$\text{let FALSE} = \lambda x \; y . \; y$$

recall: $(\lambda x. \underbrace{e_1}) \; e_2 \longrightarrow_\beta e_1[x := e_2]$

$$(\lambda \underline{b} \; x \; y . \underbrace{b \; x \; y}) \; \underbrace{\text{TRUE}} \; \text{FALSE} \; \text{TRUE}$$

$$\longrightarrow_\beta$$

$$(\lambda \underline{x} \; y . \; \text{TRUE} \; x \; y) \; \text{FALSE} \; \text{TRUE}$$

$$\longrightarrow_\beta$$

$$(\lambda \; y . \; \text{TRUE} \; \text{FALSE} \; y) \; \text{TRUE}$$

$$\longrightarrow_\beta$$

$$\text{TRUE} \; \text{FALSE} \; \text{TRUE}$$

$$\longrightarrow_{def}$$

$$(\lambda \underline{x} \; y . \; x) \; \text{FALSE} \; \text{TRUE}$$

$$\longrightarrow_\beta$$

$$(\lambda y . \; \text{FALSE}) \; \text{TRUE}$$

$$\longrightarrow_\beta$$

$$\text{FALSE}$$

$\underbrace{(\lambda x.\, x)}_{\text{function}}\ \underbrace{y}_{\text{arg}} \longrightarrow_\beta\ y$

$\overline{(\lambda \underline{x}.\, \underbrace{(\lambda y.\, \underline{x})}\,)}\ \ y$

$\longrightarrow_\beta$ in a naive way:

$\lambda y.\, y$

This is wrong! :(

To solve this problem,
we need capture-avoiding substitution

$\left.\begin{array}{l} \lambda \underline{x}.\, \underline{x} \\ \lambda \underline{y}.\, \underline{y} \end{array}\right\}$ α-equivalent

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

$\longrightarrow_\alpha$

$(\lambda \underline{x}.\, \underbrace{(\lambda z.\, \underline{x})})\ y$

$\longrightarrow_\beta$

$\lambda z.\, y$

We replaced all free occurrences of the bound variable in the body.