Bonus lecture: The Lambda Calculus

OH COOL, EXCEL IS ADDING A LAMBDA FUNCTION, 50 YOU CAN RECURSIVELY DEFINE FUNCTIONS.







Intro: python lambdas

- ▶ In many languages, a "lambda" function is a function that can be treated like an object (usually with captured variables)
- However, python has a different interpretation!

```
a = lambda x: x ** 2
print(a(5))
# >> 25

plus = lambda lhs: lambda rhs: lhs + rhs
print(plus(123)(321))
# >> 444
```

- ▶ Generally: lambda b: f(b) is a formula which, when applied on input x, returns f(x)
 - b is the thing that will be replaced
 - Everything after the colon is the text in which b will be replaced

Church's Version

- Alonzo Church was Turing's PhD advisor at Princeton
- Followed the work of Gödel, Peano, Principia Mathematica in formalizing mathematics
- Also following the string-rewriting-rule formalizations of Thue
- ▶ He invented "the lambda calculus", or λ -calculus

Def: λ -calculus. Let M[x := N] mean "string M, but with any occurrence of x replaced with N".

- 1. $\lambda x.M$ is a **lambda abstraction**
- 2. If M is a lambda abstraction, MN is an **application**
- 3. $\lambda x.M$ and $\lambda y.M[x:=y]$ are equivalent (α -conversion, used to avoid collisions)
- 4. $(\lambda x.M)(N) = M[x := N]$ is the result of an application $(\beta$ -reduction, represents computation)

Example

Ex:
$$(\lambda x.\sqrt{x})((\lambda x.x^2)5)$$

- 1. Input: $(\lambda x.\sqrt{x})((\lambda x.x^2)5)$
- 2. β -reduction: $\sqrt{((\lambda x.x^2)5)}$
- 3. β -reduction: $\sqrt{5^2}$
- 4. Arithmetic: 5

Currying

- ▶ What if we want to have multiple arguments?
- ▶ In python, lambda x, y: f(x, y) is legal
 - ▶ This is **not** legal in λ -calculus!
 - ► However, abstractions are allowed to return abstractions!
- We can make a function that takes the first argument and returns a function taking the second argument

```
mean = lambda a: lambda b: lambda c: (a + b + c) / 3

# mean(1) is `lambda b: lambda c: (1 + b + c) / 3`
# mean(1)(2) is `lambda c: (1 + 2 + c) / 3`
# mean(1)(2)(3) is `(1 + 2 + 3) / 3`

print(mean(1)(2)(3))
# >> 2
```

- This is called Currying
- Named after Haskell Curry (also the namesake of the Haskell language)

Currying Example

Ex:
$$[\lambda x. \lambda y. \lambda z. \frac{x}{y} + z](15)(3)(37)$$

1. Input: $[\lambda x. \lambda y. \lambda z. \frac{x}{v} + z](15)(3)(37)$

2. β -reduction: $[\lambda y.\lambda z.\frac{15}{v} + z](3)(37)$

3. β -reduction: $\left[\lambda z.\frac{15}{3}+z\right](37)$

4. β -reduction: $\frac{15}{3} + 37$

5. Arithmetic: $\frac{15}{3} + 37 = 5 + 37 = 42$

THAT'S NOTHING. OUR OH. HUH. CALIFORNIA PASSED A LAW GIVING COLLEGE ATHLETES FULL RIGHTS TO THEIR NAMES AND IMAGES. GOOD I THINK?

STATE GAVE COLLEGE PLAYERS RIGHTS TO USE THE NAMES AND IMAGES OF ANY CALIFORNIA ATHLETES. IT DID NOT.

SURE IT DID! THAT'S HOW OUR SCHOOL FIELDED A BASKETBALL TEAM MADE UP FNTIRELY OF STEPH CURRYS. OR IS THE PLURAL "STEPHS CURRY"?

THEY DIDN'T ALL COPY THE ORIGINAL STEPH, THOUGH. ONE PLAYER GOT THE RIGHTS TO HIS NAME, THEN THE NEXT PLAYER GOT IT FROM THEM, AND SO ON. THIS PROCESS IS KNOWN AS "CURRYING" ... I HATE YOU SO MUCH.

The Y-Combinator

Consider the classical "paradox" discovered by Curry:

$$Y = \lambda f.(\lambda x. f(xx))(\lambda x. f(xx))$$

Puzzle: What does $Yg \beta$ -reduce to?

Y-Combinator β -reduction

- 1. Yg (input)
- 2. = $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))g$ (by definition of Y)
- 3. = $(\lambda x.g(xx))(\lambda x.g(xx))$ (by β -reduction)
- 4. = $g((\lambda x.g(xx))(\lambda x.g(xx)))$ (by β -reduction)
- 5. = g(Yg) (by equivalency of steps 1 and 3)
- 6. $= g(g(Yg)) = g(g(g(Yg))) = \dots$ (ad infinitum)

This can be used to create loops in λ -calculus!

Turing-Equivalency

- ► Church and Turing proved λ -calculus is equivalent to Turing machines
- Since both Peano arithmetic and λ -calculus are Turing-equivalent, they must be equivalent to each other!
 - ► They are both equally valid characterizations of unbounded arithmetic on the natural numbers