Welcome to CS 704

The greatest class on the principles of programming languages since 2015

Aws Albarghouthi
PhD from UToronto

Does program P satisfy some property Q?

deadlock-freed
program equivalence
memory safety issues
termination
space [$O(n^2)$ / constant time
fairness constraints
privacy

How do we enforce P satisfies Q?

How can we check that P satisfies Q?

(1) By programming language design + types (A)

(2) By static analysis (B)
take a program and analyze it

Systems / FS

Robotics / robotics / visualization

ML / ML for code / ML for verification / verifying ML

Theory / complexity of certain verification problems

DB / many connections

Optim / used in static analysis

Net / network verification + programming

λ calculus

a model of computation developed by Alonzo Church in the 1920s

two constructs : function application
" definition

Turing showed that λ calculus ≡ Turing machines

In mid 1960s, Peter Landin
"The next 700 programming languages"
"correspondence between ALGOL 60 and Church's λ calculus

McCarthy developed LISP
(1) a simple, stripped-down FPL
(2) we can reason about such programs

Syntax

Everything is a function

term

$t = x$ variable

$\mid \lambda x.t$ abstraction → def f(x) {t}

$\mid t\ t$ applying t to t

Given $\lambda x.t$

we say x is bound in t

λx is a binder whose scope is t

a variable that is not bound is free

$(\lambda x.\ x)$ identity combinator

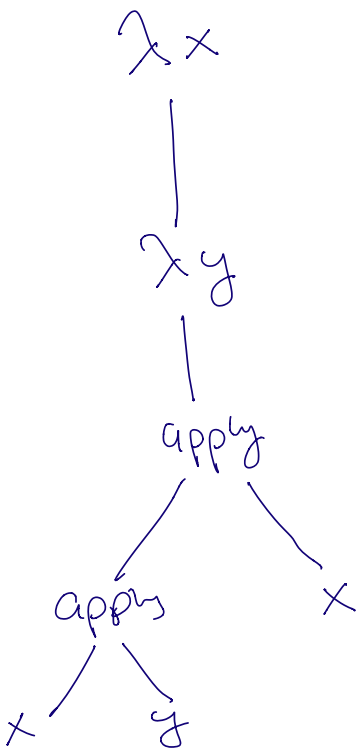
```
def f(x)
    return x
```

$\lambda x . \lambda y . x\ y\ x$

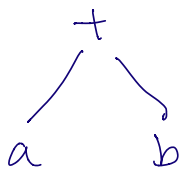
$\lambda x . (\lambda y . (x\ y)\ x)$

Bodies of abstractions extend to the right

Application is left associative



a + b



Semantics

abstraction $\lambda x.t$ as a one argument function

application $M\ N$ applying M to "data" N

Simplification = computation

$$1 + 2x + 3x$$
$$= 1 + 5x$$

$(\lambda x.t)\ t_2 \longrightarrow [x \mapsto t_2]t$

$(\lambda x.t)$ function, t_2 input, $\longrightarrow$ simplification

```
def f(x)
    return x+1
```

$f(10)$ → $10+1 \to 11$, $[x \mapsto 10]$

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

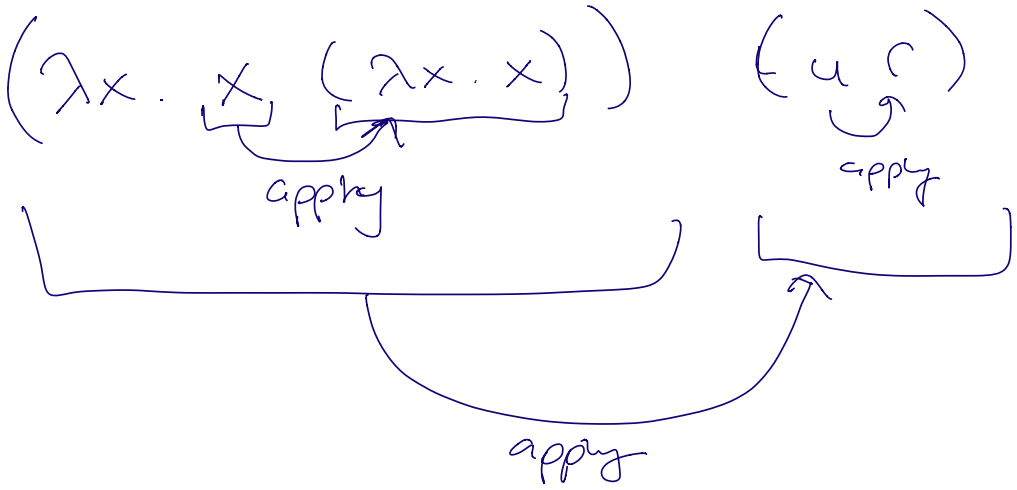
$x \sqcup y$

u

$(\lambda x.\lambda x.x\ (\lambda x.x))\ (u\ r)$

$\longrightarrow (u\ r)\ (\lambda x.x)$

β-reduction


(λx. x (λx. x))
(u c)
apply
apply
apply