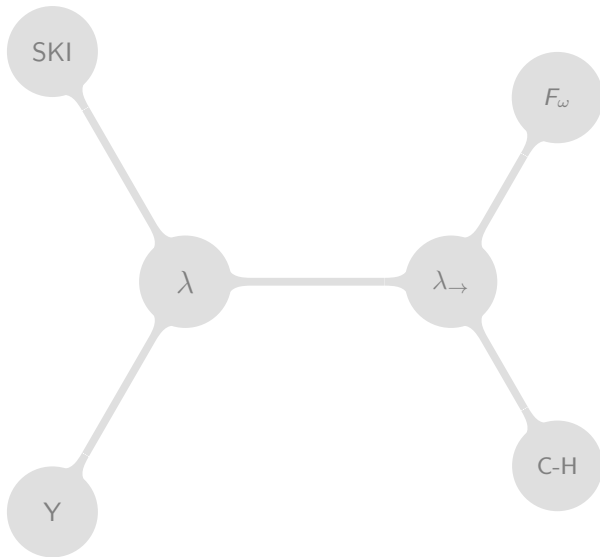


# Introduction to Lambda Calculus

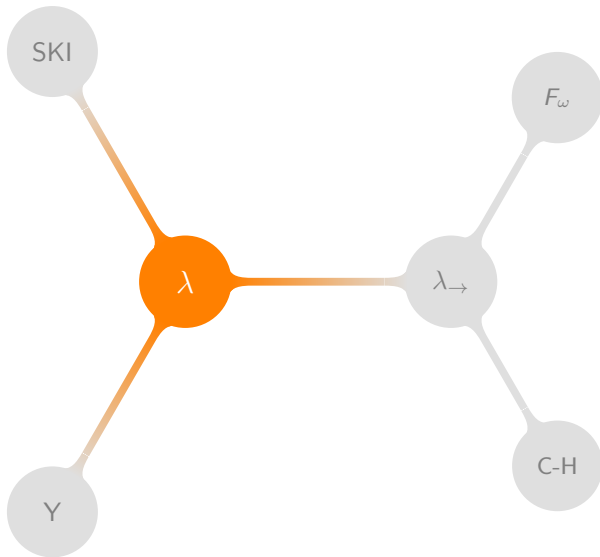
Maciek Makowski (@mmakowski)

3rd November 2014

# The Plan



# Basic Lambda Calculus



# Syntax

$\langle term \rangle ::= x$	(variable)
$(\lambda x. \langle term \rangle)$	(abstraction)
$(\langle term \rangle \langle term \rangle)$	(application)

where  $x \in \mathbb{X}$  – the set of variables

# Syntax

$v_1$

# Syntax

$v_1$

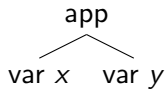
var  $v_1$

# Syntax

$x\ y$

# Syntax

$x\ y$





# Syntax

$\lambda a.b$

# Syntax

$\lambda a.b$

abs  $a$   
|  
var  $b$

# Syntax

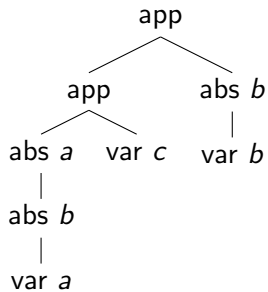
$(\lambda a. \lambda b. a) \ c \ (\lambda b. b)$

# Syntax

$\langle term \rangle ::= x$	(variable)
$(\lambda x. \langle term \rangle)$	(abstraction)
$(\langle term \rangle \langle term \rangle)$	(application)

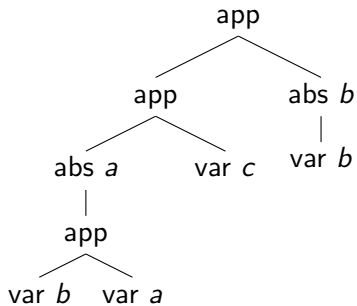
# Syntax

$(\lambda a. \lambda b. a) c (\lambda b. b)$



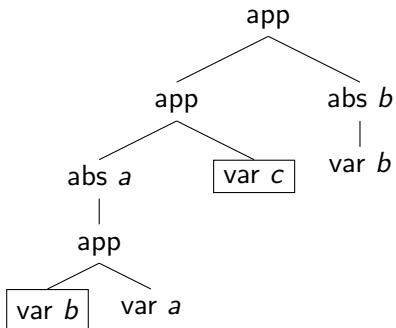
# Syntax

$(\lambda a. b \ a) \ c \ (\lambda b. b)$



# Syntax

$(\lambda a. \underline{b} \ a) \ \underline{c} \ (\lambda b. b)$



# Syntax

- ▶ terms: trees consisting of
  - ▶ variables
  - ▶ abstractions
  - ▶ applications
- ▶ variables are *bound* by abstraction; otherwise *free*



# Rewriting

$\alpha$ -conversion

$$(\lambda x.x\ y)\ (\lambda x.x) \longleftrightarrow_{\alpha} (\lambda a.a\ y)\ (\lambda b.b)$$

# Rewriting

$\beta$ -reduction

$$(\lambda x.M) N \longrightarrow_{\beta} M[x/N]$$

# Rewriting

$\beta$ -reduction

$$(\lambda x.M) N \longrightarrow_{\beta} M[x/N]$$

---

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

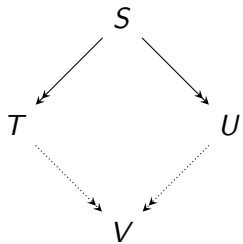
# Rewriting

$\beta$ -reduction

- ▶ *call-by-value*: start with innermost redex, do not reduce under abstraction
- ▶ *call-by-name*: start with outermost redex, do not reduce under abstraction

# Rewriting

Church-Rosser



# Semantics

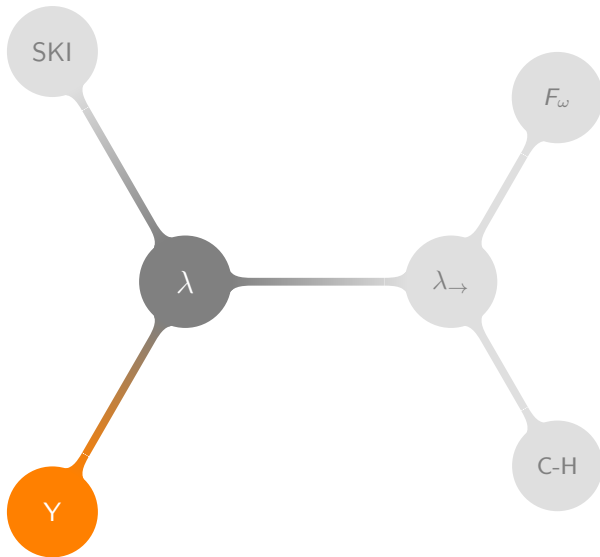
$$f(x) = 3 * x + 2$$

# Semantics

$$f(x) = 3 * x + 2$$

$$\lambda x. + (* 3 x) 2$$

# Programming in Lambda Calculus





# Conditionals

if  $C$  then  $T$  else  $F$

---

# Conditionals

if  $C$  then  $T$  else  $F$

---

$\text{true} = \lambda t. \lambda f. t$

$\text{false} = \lambda t. \lambda f. f$

# Conditionals

if  $C$  then  $T$  else  $F$

---

$\text{true} = \lambda t. \lambda f. t$

$\text{false} = \lambda t. \lambda f. f$

$\text{test} = \lambda c. \lambda t. \lambda f. c \ t \ f$

$\text{if } C \text{ then } T \text{ else } F = \text{test } C \ T \ F$

# Numbers

$$0 = \lambda s. \lambda z. z$$

$$\text{succ} = \lambda n. \lambda s. \lambda z. s (n s z)$$

# Numbers

$$0 = \lambda s. \lambda z. z$$

$$\text{succ} = \lambda n. \lambda s. \lambda z. s (n s z)$$

$$0 = \lambda s. \lambda z. z$$

$$1 = \text{succ } 0 = \lambda s. \lambda z. s z$$

$$2 = \text{succ } 1 = \lambda s. \lambda z. s (s z)$$

$$3 = \text{succ } 2 = \lambda s. \lambda z. s (s (s z))$$

$\vdots$

$$n = \lambda s. \lambda z. \underbrace{s (\dots s (s z) \dots)}_n$$

# Numbers

$$0 = \lambda s. \lambda z. z$$

$$\text{succ} = \lambda n. \lambda s. \lambda z. s (n s z)$$

$$\text{plus} = \lambda m. \lambda n. \lambda s. \lambda z. m s (n s z)$$

$$\text{times} = \lambda m. \lambda n. m (\text{plus } n) 0$$

# Recursion

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ n * (n - 1)! & \text{otherwise.} \end{cases}$$

# Recursion

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

---



# Recursion

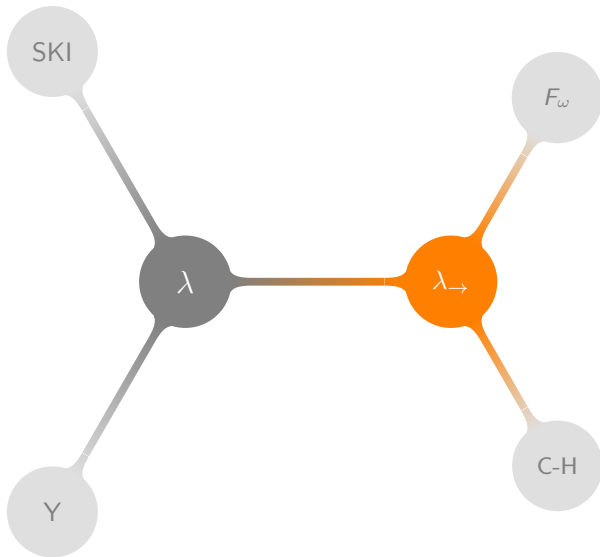
$$Y = \lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x))$$

---

$g = \lambda f.\lambda n.\text{if eq } n\ 0 \text{ then } 1 \text{ else } (\text{times } n\ (f\ (\text{pred } n)))$

`factorial = Y g`

# Simple Types



# Simple Types

$$\begin{array}{l} \langle type \rangle ::= \sigma \\ \quad | \quad (\langle type \rangle \rightarrow \langle type \rangle) \end{array}$$

# Simple Types

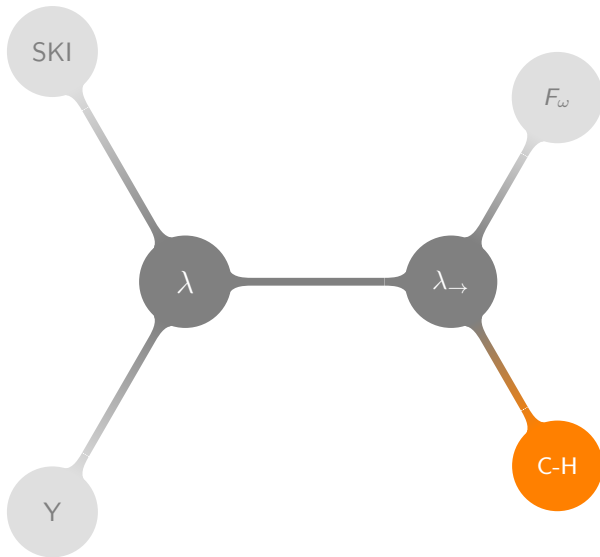
$$\frac{M : \sigma \rightarrow \tau \quad N : \sigma}{M N : \tau}$$

(application)

$$\frac{\begin{array}{c} \cancel{x : \sigma} \\ \vdots \\ M : \tau \end{array}}{\lambda x. M : \sigma \rightarrow \tau}$$

(abstraction)

# Curry-Howard Correspondence



# Curry-Howard Correspondence

$$\frac{\phi \Rightarrow \psi \quad \phi}{\psi}$$

(implication elimination)

$$\frac{\begin{array}{c} \phi \\ \vdots \\ \psi \end{array}}{\phi \Rightarrow \psi}$$

(implication introduction)

# Curry-Howard Correspondence

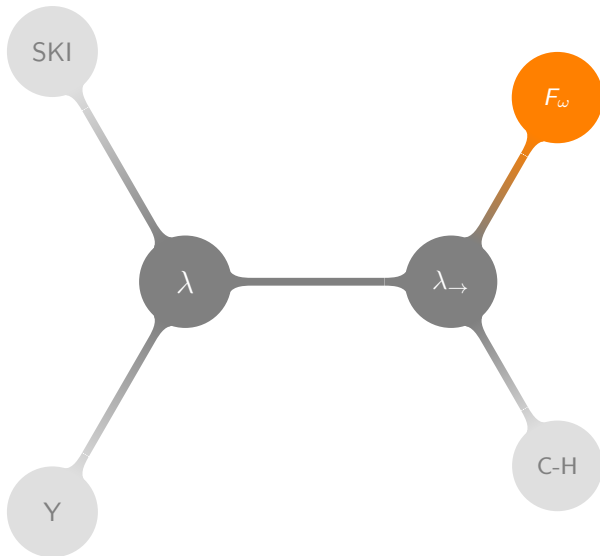
$$\frac{\phi \Rightarrow \psi \quad \phi}{\psi}$$

$$\frac{M : \sigma \rightarrow \tau \quad N : \sigma}{M N : \tau}$$

$$\frac{\begin{array}{c} \cancel{\phi} \\ \vdots \\ \psi \end{array}}{\phi \Rightarrow \psi}$$

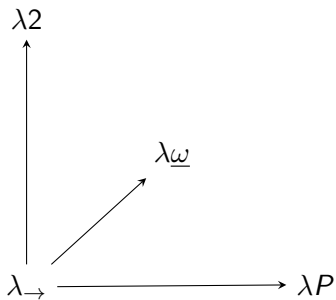
$$\frac{\begin{array}{c} \cancel{x : \sigma} \\ \vdots \\ M : \tau \end{array}}{\lambda x. M : \sigma \rightarrow \tau}$$

## More Types

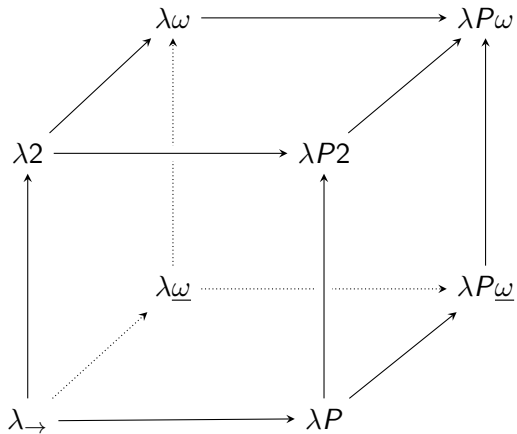




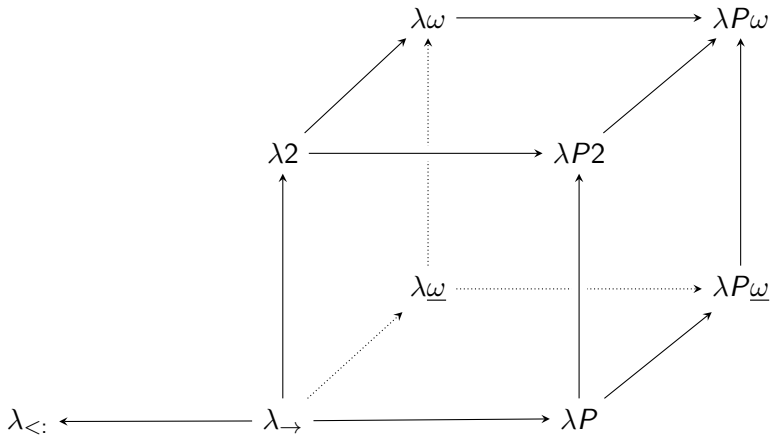
# The Lambda Cube



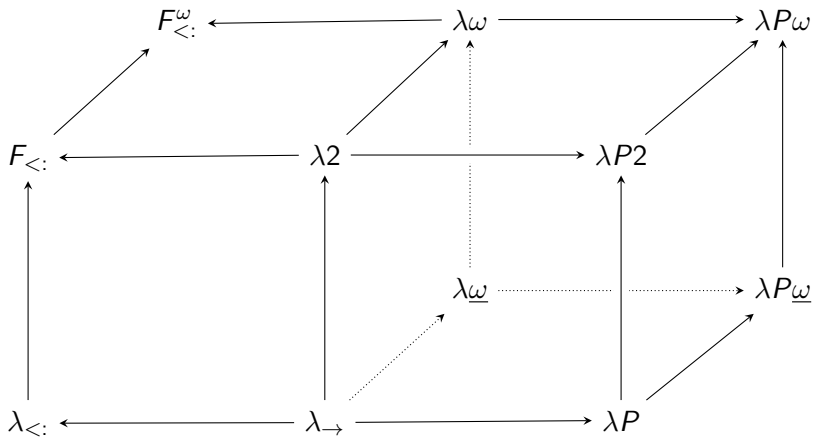
# The Lambda Cube



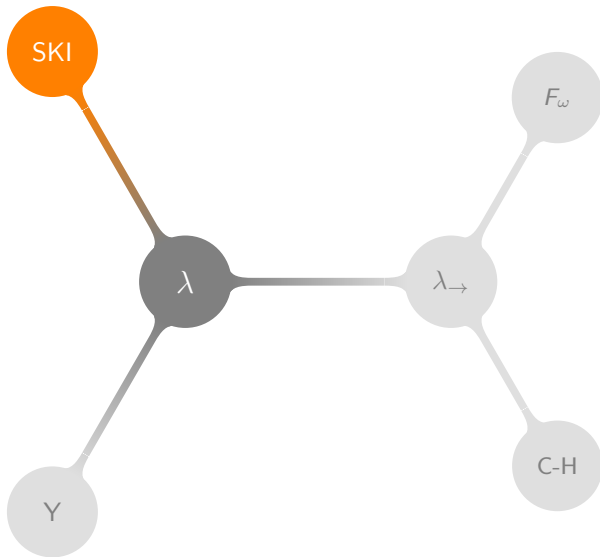
# Subtyping



# Subtyping



# Combinatory Logic



# Combinatory Logic

$$K = \lambda x. \lambda y. x$$

$$S = \lambda x. \lambda y. \lambda z. xz(yz)$$

$$I = SKK$$

---

# Combinatory Logic

$$K = \lambda x. \lambda y. x$$

$$S = \lambda x. \lambda y. \lambda z. xz(yz)$$

$$I = SKK$$

---

$$\lambda x. \lambda y. yx = S(K(SI))(S(KK)I)$$

# Combinatory Logic

$$X = \lambda x.x SK$$

---



# Combinatory Logic

$$X = \lambda x. x SK$$

---

$$K = X(X(XX))$$

$$S = X(X(X(XX)))$$

## Further Reading

- ▶ Benjamin C. Pierce, *Types and Programming Languages*
- ▶ Morten Heine B. Sørensen, Paweł Urzyczyn, *Lectures on the Curry-Howard Isomorphism*
- ▶ Henk Berendregt, Erik Barendsen, *Introduction to Lambda Calculus*
- ▶ Henk Berendregt *The Lambda Calculus, its Syntax and Semantics*

# Notes

- ▶ <https://github.com/mmakowski/introlambda/blob/baml/notes.pdf>
- ▶ <https://github.com/mmakowski/introlambda/blob/baml/slides.pdf>