Untyped λ-Calculus

Harley Eades

A Core Calculus

- (1960s) Landin showed: complex PL = Tiny Core + Derived Forms
- The core calculus used by Landin was the λ-calculus.
- In widespread use ever since.
- Can be viewed as a simple PL and a mathematical object about which rigorous statements can be proved.

λ-Calculus

- Can be enriched in a variety of ways.
  - Concrete syntax for numbers, tuples, records, etc.
  - Complex features: mutable references, nonlocal exception handling, etc.
  - Require heavy translations.
  - Lead to: ML, Haskell, and Scheme.

λ-Calculus

- PLs often provide a means of writing procedures that allow the programmer to reuse computations.
- Procedural (or functional) abstractions abstract computations over one or more named parameters.
  - Computations are then written generically.

λ-Calculus

Example:

$$\text{sum}(n) = \text{if iszero } n \text{ then } 0 \text{ else } n + \text{sum}(n-1)$$

$$\text{sum}(3) + \text{sum}(2) + \text{sum}(1) = (3 + 2 + 1) + (2 + 1) + 1$$

λ-Calculus

Example:

$$\text{sum} = \lambda n \,.\, \text{if iszero } n \text{ then } 0 \text{ else } n + (\text{sum } (n - 1))$$

$$(\text{sum } 3) + (\text{sum } 2) + (\text{sum } 1) = (3 + 2 + 1) + (2 + 1) + 1$$

λ-Calculus

$$
\begin{array}{lll}
t ::= & & \\
\quad | & x & \text{Variables} \\
\quad | & \lambda x \,.\, t & \lambda\text{-abstraction (unary function)} \\
\quad | & t_1 \; t_2 & \text{application (function application)}
\end{array}
$$

λ-Calculus: Abstract Syntax

$$
\begin{aligned}
t ::= & \\
& \mid x \\
& \mid \lambda x(t) \\
& \mid @(t_1, t_2)
\end{aligned}
$$

Used by compiler:

- Easier to compute with.
- Eliminates ambiguity.
- Harder to read by humans.

λ-Calculus: Abstract Syntax

Concrete (external)

$$t ::= \mid x \mid \lambda x\,.\,t \mid t_1\,t_2$$

parsing

$\cong$

pretty printing

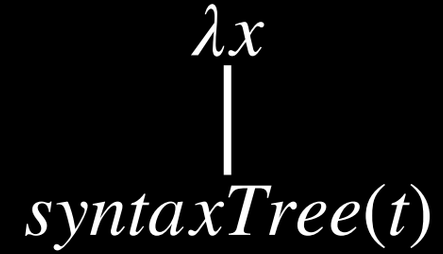
Abstract (internal)

$$\mathrm{at} ::= \mid x \mid \lambda x(at) \mid @(at_1, at_2)$$

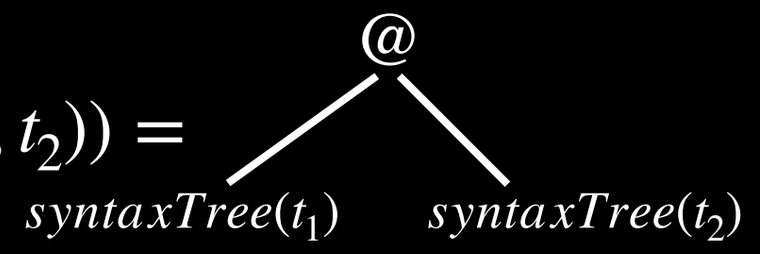
λ-Calculus: Syntax Trees

$$syntaxTree(x) = x$$

$$syntaxTree(\lambda x(t)) =$$



$$syntaxTree(@(t_1, t_2)) =$$

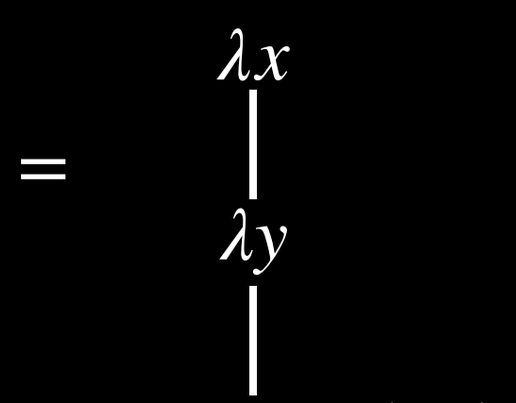


Graphical Representation

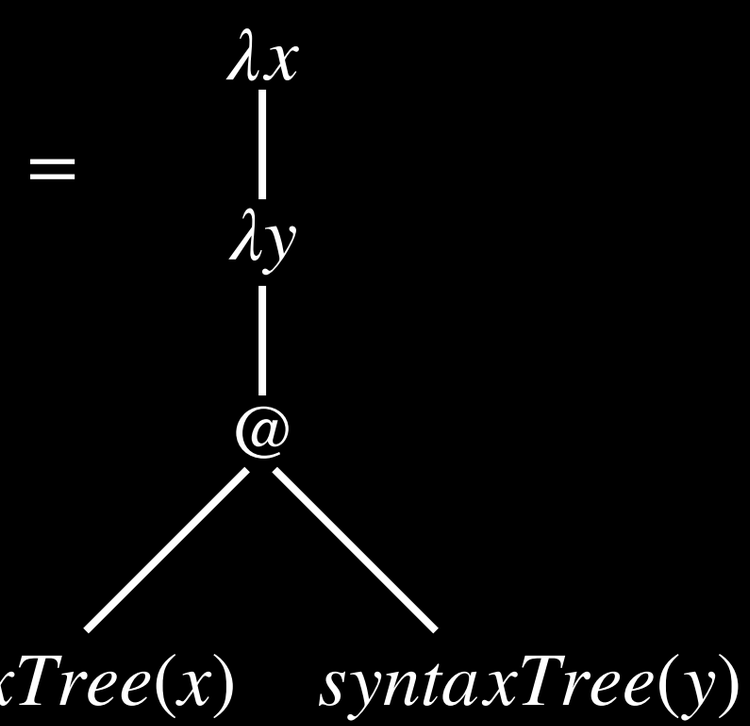
λ-Calculus: Syntax Trees

Example:

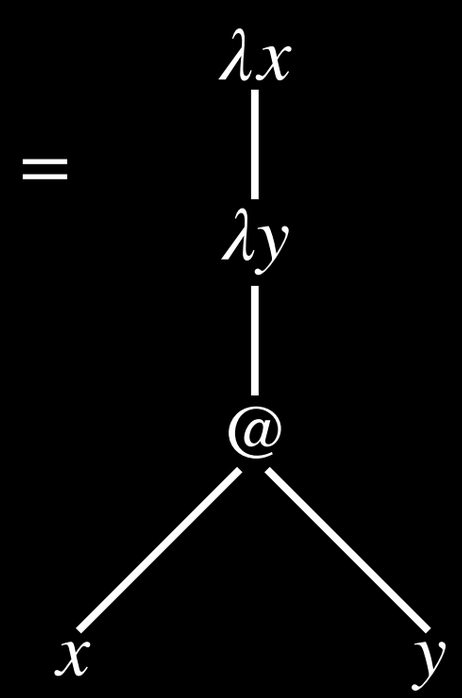
$$syntaxTree(\lambda x\,.\,(\lambda y\,.\,x\;y)) = \begin{array}{c}\lambda x\\ |\\ syntaxTree(\lambda y\,.\,x\;y)\end{array}$$



=



=



λ-Calculus: Free Variables

$$FV(x) = \{x\}$$

$$FV(t_1\, t_2) = FV(t_1) \cup FV(t_2)$$

$$FV(\lambda x\,.\,t) = \mathsf{FV}(t) - \{x\}$$

λ-Calculus: Bound Variables

Definition: A variable x in a term t is <u>bound</u> if $x \notin \mathrm{FV}(t)$.

A bound variable is always associated with a binder. The only binder in the λ-calculus is the λ-binder λz binder at the head of a λ-abstraction.

λ-Calculus: Free Variables

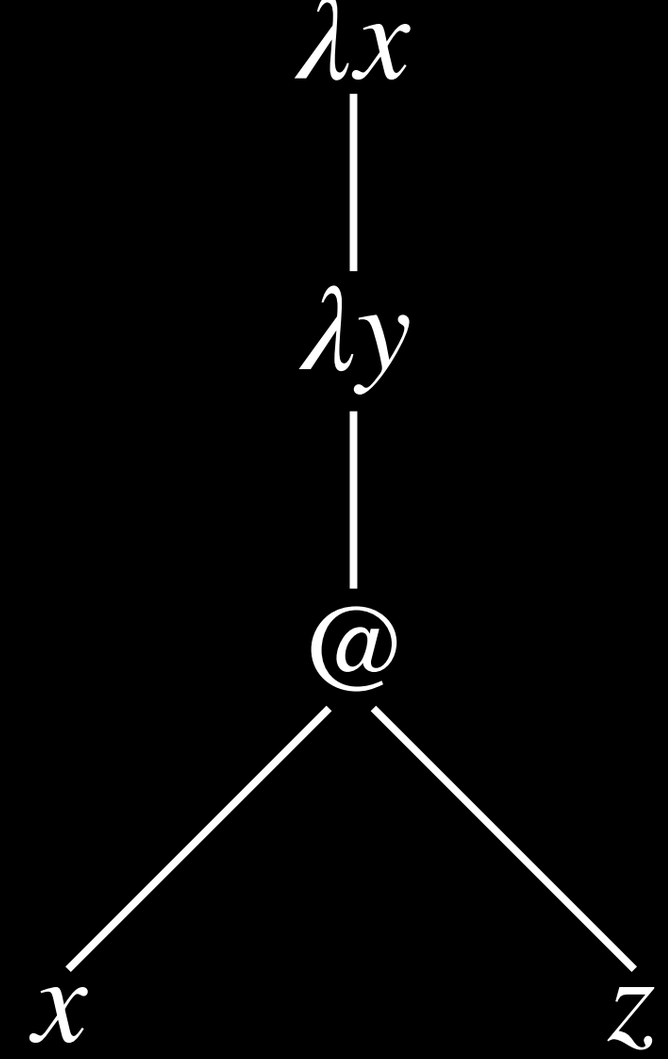
$$BV(x) = \varnothing$$

$$BV(t_1\, t_2) = BV(t_1) \cup BV(t_2)$$

$$BV(\lambda x\,.\,t) = \{x\} \cup \mathsf{BV}(t)$$

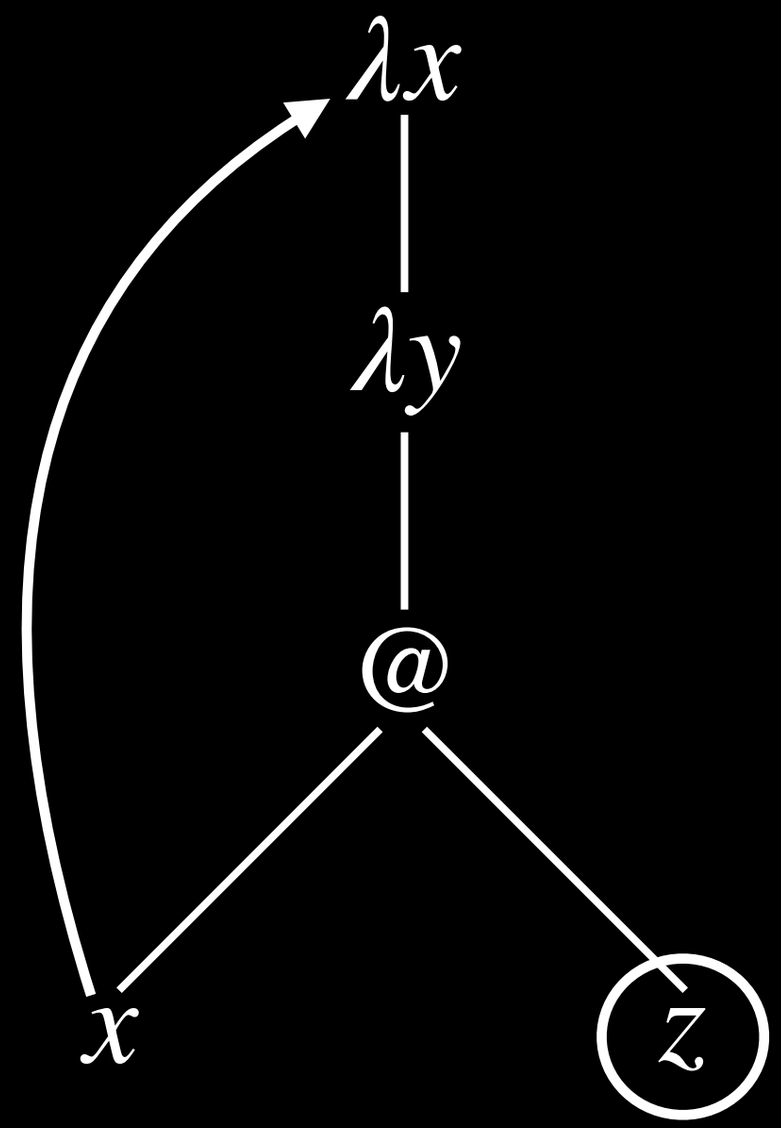
λ-Calculus: Bound Variables

Circle all free variables and draw lines connecting bound variables to their binder.



λ-Calculus: Bound Variables

Circle all free variables and draw arrows connecting bound variables to their binder.



λ-Calculus: Variables

Lemma: For any term t, $\mathsf{FV}(t) \cap \mathsf{BV}(t) = \varnothing$.

That is, a variable is either free or bound, but not both.

Definition: The set of all variables is defined as follows:

$$\mathsf{Vars}(t) = \mathsf{FV}(t) \cup \mathsf{BV}(t)$$

λ-Calculus: Substitution

Variables are replaced using an meta-operation called capture avoiding substitution.

Substitution simply replaces free variables with terms.

λ-Calculus: Substitution

$[x \mapsto t]x = t$

$[x \mapsto t]y = y$, if $x \neq y$

$[x \mapsto t](t_1\, t_2) = [x \mapsto t]t_1\, [x \mapsto t]t_2$

$[x \mapsto t](\lambda x\,.\,t') = \lambda x\,.\,t'$

$[x \mapsto t](\lambda y\,.\,t') = \lambda y\,.\,[x \mapsto t]t'$

λ-Calculus: Substitution

$$[x \mapsto t]\left(\begin{array}{c} x \end{array}\right) = t$$

$$[x \mapsto t]\left(\begin{array}{c} \lambda x \\ | \\ t' \end{array}\right) = \begin{array}{c} \lambda x \\ | \\ t' \end{array}$$

$$[x \mapsto t]\left(\begin{array}{c} y \end{array}\right) = y, \text{ if } x \neq y$$

$$[x \mapsto t]\left(\begin{array}{c} \lambda y \\ | \\ t' \end{array}\right) = \begin{array}{c} \lambda x \\ | \\ [x \mapsto t]t' \end{array}, \text{ if } x \neq y$$

$$[x \mapsto t]\left(\begin{array}{ccc} & @ & \\ t_1 & & t_2 \end{array}\right) = \begin{array}{ccc} & @ & \\ [x \mapsto t]t_1 & & [x \mapsto t]t_2 \end{array}$$