Untyped Lambda Calculus: Syntax

$t ::=$		terms:
	x	variables
	$\lambda x.\, t$	abstraction
	$t\, t$	application

$v ::=$		values:
	$\lambda x.\, t$	abstraction value

Application associates to the .

Abstractions extend as far to the right as possible.

$\lambda x.\, t$ binds x in t. Terms that differ only in the names of bound variables are interchangeable in all contexts.

Untyped Lambda Calculus: Syntax

$t ::=$		terms:
	x	variables
	$\lambda x.\, t$	abstraction
	$t\, t$	application

$v ::=$		values:
	$\lambda x.\, t$	abstraction value

Application associates to the left.

Abstractions extend as far to the right as possible.

$\lambda x.\, t$ binds x in t. Terms that differ only in the names of bound variables are interchangeable in all contexts.

Untyped Lambda Calculus: Free Variables and Substitution

The *free variables* of a term are defined recursively as follows:

$$\begin{aligned} \mathsf{FV}(x) &= \{x\}, \\ \mathsf{FV}(\lambda x.\, t_1) &= \mathsf{FV}(t_1) \setminus \{x\}, \\ \mathsf{FV}(t_1\, t_2) &= \mathsf{FV}(t_1) \cup \mathsf{FV}(t_2). \end{aligned}$$

A term is *closed* iff it has no free variables; otherwise it is *open*.

The *substitution of* s *for the free occurrences of* x *in* t is defined recursively by:

$$\begin{aligned} [x \mapsto s]x &= s, \\ [x \mapsto s]y &= y, \text{ if } y \neq x, \\ [x \mapsto s](\lambda y.\, t_1) &= \lambda y.\, [x \mapsto s]t_1, \text{ if } y \neq x \text{ and} \\ [x \mapsto s](t_1\, t_2) &= ([x \mapsto s]t_1\, [x \mapsto s]t_2). \end{aligned} \quad ,$$

Untyped Lambda Calculus: Free Variables and Substitution

The *free variables* of a term are defined recursively as follows:

$$\begin{aligned} \mathsf{FV}(x) &= \{x\}, \\ \mathsf{FV}(\lambda x.\, t_1) &= \mathsf{FV}(t_1) \setminus \{x\}, \\ \mathsf{FV}(t_1\, t_2) &= \mathsf{FV}(t_1) \cup \mathsf{FV}(t_2). \end{aligned}$$

A term is *closed* iff it has no free variables; otherwise it is *open*.

The *substitution of* s *for the free occurrences of* x *in* t is defined recursively by:

$$\begin{aligned} [x \mapsto s]x &= s, \\ [x \mapsto s]y &= y, \text{ if } y \neq x, \\ [x \mapsto s](\lambda y.\, t_1) &= \lambda y.\, [x \mapsto s]t_1, \text{ if } y \neq x \text{ and } y \notin \mathsf{FV}(s), \\ [x \mapsto s](t_1\, t_2) &= ([x \mapsto s]t_1\, [x \mapsto s]t_2). \end{aligned}$$

Untyped Lambda Calculus: Evaluation

Evaluation: $\boxed{t \to t'}$

$$\frac{t_1 \to t_1'}{t_1\, t_2 \to t_1'\, t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{v_1\, t_2 \to v_1\, t_2'} \qquad \text{(E-App2)}$$

$$(\lambda x.\, t_{12}) v_2 \to [x \mapsto v_2] t_{12} \qquad \text{(E-AppAbs)}$$

Untyped Lambda Calculus: Evaluation

Evaluation: $\boxed{t \to t'}$

$$\frac{t_1 \to t_1'}{t_1\, t_2 \to t_1'\, t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{v_1\, t_2 \to v_1\, t_2'} \qquad \text{(E-App2)}$$

$$(\lambda x.\, t_{12}) v_2 \to [x \mapsto v_2] t_{12} \qquad \text{(E-AppAbs)}$$

It doesn't make much sense to try to evaluate open terms.

Untyped Lambda Calculus: Evaluation

Evaluation: $\boxed{t \to t'}$

$$\frac{t_1 \to t_1'}{t_1\, t_2 \to t_1'\, t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{v_1\, t_2 \to v_1\, t_2'} \qquad \text{(E-App2)}$$

$$(\lambda x.\, t_{12}) v_2 \to [x \mapsto v_2] t_{12} \qquad \text{(E-AppAbs)}$$

It doesn't make much sense to try to evaluate open terms. Thus the subtleties of substitution don't really come into play in this evaluation relation.