□ Questionnaire

	yes	no
Are you familiar with (untyped) λ–calculus?	□	□
Are you familiar with simply typed λ–calculus?	□	□
Are you familiar with parametric polymorphism?	□	□
Have you programmed in Scheme?	□	□
Have you programmed in ML (i.e. SML or OCaml)?	□	□
Have you programmed with Haskell?	□	□
Are you familiar with continuations and CPS?	□	□
Are you familiar with monads?	□	□

□ Advanced Functional Programming with Haskell

Course Outline

- Review of basic lambda calculus
- Simple types
- Polymorphic types
- Higher-order functions and combinators
- Monads
- Monad transformers
- Generic programming

□ Lambda Calculus Expressions

Expressions

$$e \quad ::= \quad x \mid \lambda x.e \mid e\,e$$

by convention: $e_1\,e_2\,e_3 \equiv (e_1\,e_2)\,e_3$

Examples

$$\lambda x.x \qquad \lambda f.\lambda x.x$$

$$\lambda x.\lambda y.x \qquad \lambda f.\lambda x.f\,x$$

$$\lambda f.\lambda g.\lambda x.f\,x\,(g\,x) \qquad \lambda f.\lambda x.f\,(f\,x)$$

$$(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)) \qquad \lambda m.\lambda n.\lambda f.\lambda x.m\,f\,(n\,f\,x)$$

□ Free Variables

Free Variables: $\mathcal{FV}(e)$

$$\begin{array}{rcl} \mathcal{FV}(x) & = & \{x\} \\ \mathcal{FV}(\lambda x.e) & = & \mathcal{FV}(e) - \{x\} \\ \mathcal{FV}(e_1\, e_2) & = & \mathcal{FV}(e_1) \cup \mathcal{FV}(e_2) \end{array}$$

Examples

$$\begin{array}{rcl} \mathcal{FV}(\lambda x.\lambda y.x) & = & \emptyset \\ \mathcal{FV}(\lambda x.\lambda f.f\,(g\,x)\,x) & = & \{g\} \end{array}$$

Closed Expressions: e is closed iff $\mathcal{FV}(e) = \emptyset$

□ Substitution

Substitution: $e_1[x := e_2]$

$$
\begin{array}{rcl}
x[x := e] & = & e \\
x_1[x_2 := e] & = & x_1 \quad (x_1 \neq x_2) \\
(\lambda x.e_1)[x := e_2] & = & \lambda x.e_1 \\
(\lambda x_1.e_1)[x_2 := e_2] & = & \lambda x_1.e_1[x_2 := e_2] \quad (x_1 \neq x_2 \text{ and } x_1 \notin \mathcal{FV}(e_2)) \\
(e_1\, e_2)[x := e_3] & = & e_1[x := e_3]\, e_2[x := e_3]
\end{array}
$$

Examples

$$
\begin{array}{rcl}
(f\, x)[f := \lambda y.y] & = & (\lambda y.y)\, x \\
((\lambda x.f\, x)\, x)[x := \lambda y.y] & = & (\lambda x.f\, x)\, (\lambda y.y) \\
((\lambda y.f\, y)\, (g\, f))[f := z\, y] & = & \text{stuck since } y \in \mathcal{FV}(z\, y)
\end{array}
$$

□ Alpha Equivalence

Alpha Equivalence: $=_\alpha$

$$\frac{}{x =_\alpha x} \qquad \frac{e_1 =_\alpha e_2[x_2 := x_1]}{\lambda x_1.e_1 =_\alpha \lambda x_2.e_2} \qquad \frac{e_1 =_\alpha e_3 \quad e_2 =_\alpha e_4}{e_1\, e_2 =_\alpha e_3\, e_4}$$

Examples

$$\begin{array}{rcl} \lambda x.x & =_\alpha & \lambda y.y \\ \lambda x.\lambda y.x & =_\alpha & \lambda y.\lambda x.y \\ \lambda x.\lambda y.y & =_\alpha & \lambda x.\lambda x.x \\ \\ \lambda x.\lambda y.x & \neq_\alpha & \lambda y.\lambda x.x \\ \lambda x.\lambda y.x & \neq_\alpha & \lambda x.\lambda x.x \end{array}$$

Standard equality on lambda expressions includes alpha equivalence.

□ Capture Avoiding Substitution

$$(\lambda x_1.e_1)[x_2 := e_2] \quad = \quad \lambda x_1.e_1[x_2 := e_2] \quad (x_1 \neq x_2 \text{ and } x_1 \notin \mathcal{FV}(e_2))$$

Side condition $x_1 \notin \mathcal{FV}(e_2)$ is always satisfiable since we can rename the bound variable, x_1, in e_1 to avoid capture.

$$((\lambda y.f\, y)\,(g\, f))[f := z\, y] \quad = \quad (\lambda x.z\, y\, x)\,(g\,(z\, y))$$

Note that: $\lambda y.f\, y \;=_\alpha\; \lambda x.f\, x$

□ Beta Equivalence

Beta Equivalence: $=_\beta$

$$(\lambda x.e_1)\, e_2 \quad =_\beta \quad e_1[x := e_2]$$

A function applied to an argument is a redex.

Changing a redex to its beta equivalent form is called reduction.

Examples

$$\begin{aligned}(\lambda x.\lambda f.f\, x\, x)\, (\lambda y.y) &=_\beta \lambda f.f\, (\lambda y.y)\, (\lambda y.y)\\ \lambda f.(\lambda x.x\, (f\, x))\, (\lambda y.y) &=_\beta \lambda f.(\lambda y.y)\, (f\, (\lambda y.y))\end{aligned}$$

Standard equality on lambda expressions includes beta equivalence.

□ Normal Form

Lambda expressions are in normal form if they contain no redex.

Examples of normal forms:

$$\lambda x.x$$

$$\lambda f.f\,(x\,x)$$

$$f\,(g\,(\lambda x.\lambda y.x))\,(\lambda x.\lambda y.y)$$

Non-normal forms:

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

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

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

□ Normalization

Normalization is removing every redex from a term.

Use following abbreviations:

$$\mathsf{one} \triangleq \lambda f.\lambda x.f\,x \qquad \mathsf{plus} \triangleq \lambda m.\lambda n.\lambda f.\lambda x.m\,f\,(n\,f\,x)$$

Then

$$\begin{array}{lll} \mathsf{plus\,one\,one} & =_\beta & (\lambda n.\lambda f.\lambda x.\mathsf{one}\,f\,(n\,f\,x))\,\mathsf{one} \\ & =_\beta & \lambda f.\lambda x.\mathsf{one}\,f\,(\mathsf{one}\,f\,x) \\ & =_\beta & \lambda f.\lambda x.\mathsf{one}\,f\,(f\,x) \\ & =_\beta & \lambda f.\lambda x.f\,(f\,x) \end{array}$$

More than one way to normalize an expression.

□ Perpetual Expressions

Some lambda expressions do not have a normal form.

$$
\begin{aligned}
(\lambda x.x\,x)\,(\lambda x.x\,x) &=_\beta (\lambda x.x\,x)\,(\lambda x.x\,x) \\
&=_\beta (\lambda x.x\,x)\,(\lambda x.x\,x) \\
&=_\beta (\lambda x.x\,x)\,(\lambda x.x\,x) \\
&=_\beta \cdots
\end{aligned}
$$

$$
\begin{aligned}
(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)) &=_\beta f\,((\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))) \\
&=_\beta f\,(f\,((\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)))) \\
&=_\beta f\,(f\,(f\,((\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))))) \\
&=_\beta \cdots
\end{aligned}
$$

□ Reduction Strategies

Reduction strategy is a method for reducing an expression to normal form.

Some reduction strategies don't lead to normal form.

Always reducing argument redex never reaches normal form:

$$\begin{aligned}(\lambda x.\lambda y.y)\,((\lambda x.x\,x)\,(\lambda x.x\,x)) &=_\beta (\lambda x.\lambda y.y)\,((\lambda x.x\,x)\,(\lambda x.x\,x))\\ &=_\beta (\lambda x.\lambda y.y)\,((\lambda x.x\,x)\,(\lambda x.x\,x))\\ &=_\beta \cdots\end{aligned}$$

Reducing the main redex leads to normal form:

$$(\lambda x.\lambda y.y)\,((\lambda x.x\,x)\,(\lambda x.x\,x)) \quad =_\beta \quad \lambda y.y$$

Reducing the leftmost, outermost redex leads to normal form when it exists.

□ Fixpoints

When

$$e_1\, e_2 \quad =_\beta \quad e_2$$

e_2 is a fixpoint of e_1.

Every lambda expression has a fixpoint.

$$Y \quad \triangleq \quad \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

Y is known as the fixpoint combinator since

$$Y\, e \quad =_\beta \quad e\,(Y\, e)$$

$Y\, e$ is the fixpoint of e.

For any e

$$Y\, e \;\; =_\beta \;\; (\lambda x.e\,(x\,x))\,(\lambda x.e\,(x\,x)) \;\; =_\beta \;\; e\,((\lambda x.e\,(x\,x))\,(\lambda x.e\,(x\,x)))$$

□ Homework 1

1. [10 points]
   Using the following definitions

   $$K \triangleq \lambda x.\lambda y.x \quad S \triangleq \lambda f.\lambda g.\lambda x.f\,x\,(g\,x)$$

   Reduce $S\,K\,K$ to normal form.

2. [30 points]
   We can encode natural numbers in pure lambda calculus as follows:

   $$\text{zero} \triangleq \lambda f.\lambda x.x \quad \text{one} \triangleq \lambda f.\lambda x.f\,x \quad \text{two} \triangleq \lambda f.\lambda x.f\,(f\,x) \quad \ldots$$

   In general

   $$n \triangleq \lambda f.\lambda x.f\,(f\,\cdots\,(f\,x)\cdots)$$

   with n applications of f. This is known as the Church encoding.

   We can define addition as:

   $$\text{plus} \triangleq \lambda m.\lambda n.\lambda f.\lambda x.m\,f\,(n\,f\,x)$$

   (a) Show: plus one one $=_\beta$ two.

   (b) Define multiplication and show: mult two three $=_\beta$ six.

   (c) Define exponentiation and show: exp two three $=_\beta$ eight.