Lecture 6

Examples

- $(\lambda x.\ x)$ **denotes the 'identity function'**
  - $((\lambda x.\ x)\ E) = E$
  - **"=" defined later**
- $(\lambda x.\ (\lambda f.\ (f\ \ x)))$ **denotes the function:**
  - **which when applied to** E
  - **yields** $(\lambda f.\ (f\ \ x))\texttt{[}E/x\texttt{]}\ \ =\ \ (\lambda f.\ (f\ \ E))$
- $(\lambda f.\ (f\ \ E))$ **is the function**
  - **which when applied to** E'
  - **yields** $(f\ \ E)\texttt{[}E'/f\texttt{]}\ \ =\ \ (E'\ E)$
- **Thus**

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

and

$$((\lambda f.\ (f\ \ E))\ E') = (E'\ E)$$

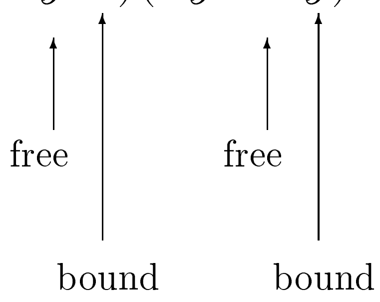
Notational conventions

- **Function application associates to the left**
  - $E_1\ E_2$ **means** $(E_1\ E_2)$
  - $E_1\ E_2\ E_3$ **means** $((E_1\ E_2)E_3)$
  - $E_1\ E_2\ E_3\ E_4$ **means** $(((E_1\ E_2)E_3)E_4)$
  - $E_1\ E_2\ \cdots\ E_n$ **means** $((\ \cdots\ (E_1\ E_2)\ \cdots\ )\ E_n)$
- $\lambda V.\ E_1\ E_2\ \ldots\ E_n$ **means** $(\lambda V.\ (E_1\ E_2\ \ldots\ E_n))$
  - **Scope of 'λV' extends as far right as possible**
- $\lambda V_1\ \cdots\ V_n.\ E$ **means** $(\lambda V_1.\ (\ \cdots\ .\ (\lambda V_n.\ E)\ \cdots\ ))$
  - $\lambda x\ y.\ E$ **means** $(\lambda x.\ (\lambda y.\ E))$
  - $\lambda x\ y\ z.\ E$ **means** $(\lambda x.\ (\lambda y.\ (\lambda z.\ E)))$
  - $\lambda x\ y\ z\ w.\ E$ **means** $(\lambda x.\ (\lambda y.\ (\lambda z.\ (\lambda w.\ E))))$

Free and bound variables

- **Occurrence of V is *free* if**
  - **it is not within the scope of a 'λV'**
  - **otherwise it is *bound***
- **Example:**

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



- **E is *closed* if it contains no free variables**
- **Convention: will use `bold` names for particular closed terms**

Conversion rules

- **λ-expressions can represent data objects like numbers, strings etc**
  - **$(2 + 3) \times 5$ can be represented as a λ-expression**
  - **its 'value' 25 can also be represented**
  - **details later**

- **Notation: underlining denotes representation as λ-expression**
  - **$\underline{3}$ is λ-expression denoting 3**

- **The process of 'simplifying' $(2 + 3) \times 5$ to 25 will be represented by a process called *conversion* (or *reduction*)**

- **Rules of λ-conversion are very general:**
  - **when applied to λ-expressions representing arithmetic expressions they do arithmetical evaluation**

Kinds of λ-conversion

- **Three kinds of λ-conversion;**
  - **α-conversion – renaming bound variables**
  - **β-conversion – function application rule**
  - **η-conversion – extensionality**
- **Notation: E[E'/V] denotes**
  - **the result of substituting E'**
  - **for each *free* occurrence of V in E**
- **The substitution is *valid* if and only if:**
  - **no free variable in E' becomes bound in E[E'/V]**
- **Substitution is described in more detail later**

Rules of λ-conversion

- **α-conversion**
  - $\lambda V.\ E$ **can be converted to** $\lambda V'.\ E[V'/V]$
  - **provided the substitution of** V' **for** V **in** E **is valid**
  - $E_1 \underset{\alpha}{\longrightarrow} E_2$ **means** E_1 α**-converts to** E_2
- **β-conversion**
  - $(\lambda V.\ E_1)\ E_2$ **can be converted to** $E_1[E_2/V]$
  - **provided the substitution of** E_2 **for** V **in** E_1 **is valid**
  - $E_1 \underset{\beta}{\longrightarrow} E_2$ **means** E_1 β**-converts to** E_2
- **η-conversion**
  - $\lambda V.\ (E\ V)$ **can be converted to** E
  - **provided** V **has no free occurrence in** E
  - $E_1 \underset{\eta}{\longrightarrow} E_2$ **means** E_1 η**-converts to** E_2

Remarks on conversion rules

- **β-conversion is most important**
  - **it can simulate arbitrary evaluation mechanisms**
  - $\underline{(2+3)\times 5} \underset{\beta}{\longrightarrow} \underline{25}$
  - **details later**

- **α-conversion concerns the technical manipulation of bound variables**

- **η-conversion forces functions that always give the same results on the same arguments to be equal**
  - **this is called "extensionality"**

- **N.B.** **"conversion" and "reduction" are used interchangeably**

α-conversion

- **A λ-expression to which α-reduction can be applied is called an *α-redex***
  - **necessarily an abstraction**

- **The term "redex" abbreviates "reducible expression"**

- **α-conversion says that bound variables can be renamed**
  - **provided no 'name-clashes' occur**

Examples of α-conversion

- $\lambda x.\ x \xrightarrow[\alpha]{} \lambda y.\ y$
- $\lambda x.\ \mathtt{f}\ x \xrightarrow[\alpha]{} \lambda y.\ \mathtt{f}\ y$
- **It is *not* the case that**

$$\lambda x.\ \lambda y.\ \mathtt{f}\ x\ y \xrightarrow[\alpha]{} \lambda y.\ \lambda y.\ \mathtt{f}\ y\ y$$

  - **the substitution** $(\lambda y.\ \mathtt{f}\ x\ y)\mathtt{[}y/x\mathtt{]}$ **is not valid**
  - **since the** y **that replaces** x **becomes bound**

β-conversion

- **A λ-expression to which β-reduction can be applied is called a** β*-redex*
  - **necessarily an application**
- β**-conversion is like the evaluation of a function call in a programming language**
  - $(\lambda V.\ E_1)\ E_2 \underset{\beta}{\longrightarrow} E_1[E_2/V]$
  - **the body E_1 of the function $\lambda V.\ E_1$ is evaluated**
  - **with V is bound to E_2**

Examples of β-conversion

- $(\lambda x.\ \texttt{f}\ x)\ E \underset{\beta}{\longrightarrow} \texttt{f}\ E$

- $(\lambda x.\ (\lambda y.\ \texttt{f}\ x\ y))\ \underline{3} \underset{\beta}{\longrightarrow} \lambda y.\ \texttt{f}\ \underline{3}\ y$

- $(\lambda y.\ \texttt{f}\ \underline{3}\ y)\ \underline{4} \underset{\beta}{\longrightarrow} \texttt{f}\ \underline{3}\ \underline{4}$

- **It is *not* the case that**

$$(\lambda x.\ (\lambda y.\ \texttt{f}\ x\ y))\ (\texttt{g}\ y) \underset{\beta}{\longrightarrow} \lambda y.\ \texttt{f}\ (\texttt{g}\ y)\ y$$

  - **the substition** $(\lambda y.\ \texttt{f}\ x\ y)\texttt{[}(\texttt{g}\ y)/x\texttt{]}$ **is not valid**
  - y **is free in** $(\texttt{g}\ y)$
  - **becomes bound after substitution for** x **in** $(\lambda y.\ \texttt{f}\ x\ y)$

Identifying β-redexes

- **Consider the application:**

$$(\lambda x.\ \lambda y.\ \texttt{f}\ \ x\ y)\ \underline{3}\ \underline{4}$$

- **bracketting according to conventions yields:**

$$(((\lambda x.\ (\lambda y.\ ((\texttt{f}\ \ x)\ y)))\ \underline{3})\ \underline{4})$$

- **which has the form:**

$$((\lambda x.\ E)\ \underline{3})\ \underline{4}$$

  where

$$E =\ (\lambda y.\ \texttt{f}\ \ x\ y)$$

  $(\lambda x.\ E)\ \underline{3}$ **is a β-redex and could be reduced to** $E\,\texttt{[}\underline{3}/x\texttt{]}$

η-conversion

- **A λ-expression to which η-reduction can be applied is called an *η-redex***
  - **necessarily an abstraction**
- **η-conversion expresses *extensionality***
  - **two functions are equal if they give the same results when applied to the same arguments**
- **$\lambda V.\ (E\ V)$ denotes the function which:**
  - **when applied to an argument E'**
  - **returns $(E\ V)[E'/V]$**
- **If V does not occur free in E**
  - **then $(E\ V)[E'/V] = (E\ E')$**
  - **Thus $\lambda V.\ E\ V$ and E both yield the same result, namely $E\ E'$, when applied to the same arguments**
  - **hence they denote the same function**

Examples of η-conversion

- $\lambda x.\ \texttt{f}\ x \xrightarrow[\eta]{} \texttt{f}$

- $\lambda y.\ \texttt{f}\ x\ y \xrightarrow[\eta]{} \texttt{f}\ x$

- **It is *not* the case that**

$$\lambda x.\ \texttt{f}\ x\ x \xrightarrow[\eta]{} \texttt{f}\ x$$

  because x is free in $\texttt{f}\ x$

Generalized conversions

- $\underset{\alpha}{\longrightarrow}$, $\underset{\beta}{\longrightarrow}$ **and** $\underset{\eta}{\longrightarrow}$ **can be generalized:**
  - $E_1 \underset{\alpha}{\longrightarrow} E_2$ **if** E_2 **can be got from** E_1 **by** α**-converting any subterm**
  - $E_1 \underset{\beta}{\longrightarrow} E_2$ **if** E_2 **can be got from** E_1 **by** β**-converting any subterm**
  - $E_1 \underset{\eta}{\longrightarrow} E_2$ **if** E_2 **can be got from** E_1 **by** η**-converting any subterm**

- **Examples:** $((\lambda x.\ \lambda y.\ \texttt{f}\ \ x\ y)\ \underline{3})\ \underline{4} \underset{\beta}{\longrightarrow} (\lambda y.\ \texttt{f}\ \ \underline{3}\ y)\ \underline{4}$
  - **subexpression** $(\lambda x.\ \lambda y.\ \texttt{f}\ \ x\ y)\underline{3}$ **is** β**-reduced**

- **Notation for a sequence of conversions:**

$$((\lambda x.\ \lambda y.\ \texttt{f}\ \ x\ y)\ \underline{3})\ \underline{4} \underset{\beta}{\longrightarrow} (\lambda y.\ \texttt{f}\ \ \underline{3}\ y)\ \underline{4} \underset{\beta}{\longrightarrow} \texttt{f}\ \ \underline{3}\ \underline{4}$$

More example reductions

(i) $(\lambda x.\ x)\ \underline{1} \underset{\beta}{\longrightarrow} \underline{1}$

(ii) $(\lambda y.\ y)\ ((\lambda x.\ x)\ \underline{1}) \underset{\beta}{\longrightarrow} (\lambda y.\ y)\underline{1} \underset{\beta}{\longrightarrow} \underline{1}$

(iii) $(\lambda y.\ y)\ ((\lambda x.\ x)\ \underline{1}) \underset{\beta}{\longrightarrow} (\lambda x.\ x)\ \underline{1} \underset{\beta}{\longrightarrow} \underline{1}$

- **(ii) & (iii) start with the same λ-expression**
  - **but reduce redexes in different orders**
- **An important property of β-reductions:**
  - **no matter in which order one does reductions**
  - **one always ends up with equivalent results**
- **Some reduction sequences may never terminate**

Equality of λ-expressions

- **Conversion rules preserve the meaning of λ-expressions**
  - **i.e. if E_1 can be converted to E_2**
  - **then E_1 and E_2 denote the same function**
- **This property of conversion should be intuitively clear**
- **Can give a mathematical definition of the function denoted by a λ-expression**
  - **then to prove that this is unchanged by α-, β- or η-conversion**
  - **doing this is surprisingly difficult**

Definition of equality

- **We** *define* **two λ-expressions to be equal if they can be transformed into each other by a sequence of (forwards or backwards) λ-conversions**

- **Must distinguish** *equality* **and** *identity*

  - **λ-expressions are identical if they consist of** *exactly* **the same sequences of characters**

  - **they are equal if one can be converted to the other**

  - $\lambda x.\ x$ **is equal to** $\lambda y.\ y$

  - **but not identical to it**

- **Notation:**

  - $E_1 \equiv E_2$ **means** E_1 **and** E_2 **are identical**

  - $E_1 = E_2$ **means** E_1 **and** E_2 **are equal**

Formal definition of equality

- **If E and E' are λ-expressions, then $E = E'$ if**
  - $E \equiv E'$
  - **or there exist expressions E_1, E_2, ..., E_n such that:**
    1. $E \equiv E_1$
    2. $E' \equiv E_n$
    3. **For each i either**
       (a) $E_i \underset{\alpha}{\longrightarrow} E_{i+1}$ **or** $E_i \underset{\beta}{\longrightarrow} E_{i+1}$ **or** $E_i \underset{\eta}{\longrightarrow} E_{i+1}$ **or**
       (b) $E_{i+1} \underset{\alpha}{\longrightarrow} E_i$ **or** $E_{i+1} \underset{\beta}{\longrightarrow} E_i$ **or** $E_{i+1} \underset{\eta}{\longrightarrow} E_i$**.**

- **Examples:**
  - $(\lambda x.\ x)\ \underline{1}\ =\ \underline{1}$
  - $(\lambda x.\ x)\ ((\lambda y.\ y)\ \underline{1})\ =\ \underline{1}$
  - $(\lambda x.\ \lambda y.\ \mathtt{f}\ x\ y)\ \underline{3}\ \underline{4}\ =\ \mathtt{f}\ \underline{3}\ \underline{4}$

Properties of equality

- $E = E$ **for any** E
  - **equality is** *reflexive*

- **If** $E = E'$**, then** $E' = E$
  - **equality is** *symmetric*

- **If** $E = E'$ **and** $E' = E''$**, then** $E = E''$
  - **equality is** *transitive*

- **If a relation is reflexive, symmetric and transitive then it is called an** *equivalence relation*
  - **thus** $=$ **is an equivalence relation**

Leibnitz' Law

- **If** $E_1 = E_2$
- **And if** E_1' **and** E_2' **only differ in that:**
  - **where one contains** E_1 **the other contains** E_2
- **Then** $E_1' = E_2'$
- **This property is called** *Leibnitz's law*
  - **It holds because the same sequence of reduction for getting from** E_1 **to** E_2 **can be used for getting from** E_1' **to** E_2'
  - **For example, if** $E_1 = E_2$**, then by Leibnitz's law** $\lambda V.\ E_1 = \lambda V.\ E_2$

Extensionality

- **Suppose:**
  - $E_1\ V = E_2\ V$
  - V **not free in** E_1 **or** E_2

- **By Leibnitz's law**

$$\lambda V.\ E_1\ V = \lambda V.\ E_2\ V$$

and by η-reduction applied to both sides

$$E_1 = E_2$$

- **Useful for proving λ-expressions equal:**
  - **to prove** $E_1 = E_2$
  - **prove** $E_1\ V = E_2\ V$ **for some** V **not occuring free in** E_1 **or** E_2

- **Such proofs are** *by extensionality*
  - **e.g.** $(\lambda f\ g\ x.\ f\ x\ (g\ x))\ (\lambda x\ y.\ x)\ (\lambda x\ y.\ x)\ =\ \lambda x.\ x$

Need for valid substitutiions

- **Suppose** $\lambda x.\ (\lambda y.\ x) \xrightarrow[\alpha]{} \lambda y.\ (\lambda y.\ y)$
  - y **becomes bound after substitution for** x **in** $\lambda y.\ x$.
- **Then it would follow by the definition of $=$ that:**

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

- **But then for** *any* E_1 **and** E_2

$$(\lambda x.\ (\lambda y.\ x))\ E_1\ E_2 \xrightarrow[\beta]{} (\lambda y.\ E_1)\ E_2 \xrightarrow[\beta]{} E_1$$

and

$$(\lambda y.\ (\lambda y.\ y))\ E_1\ E_2 \xrightarrow[\beta]{} (\lambda y.\ y)\ E_2 \xrightarrow[\beta]{} E_2$$

one would be forced to conclude that $E_1 = E_2$

- **So all λ-expressions would be equal!**

The $\longrightarrow$ relation

- $E = E'$ **means:**
  - E' **can be obtained from** E
  - **by a sequence of forwards** *or backwards* **conversions**

- $E \longrightarrow E'$ **means:**
  - E' **can be got from** E **using only forwards conversions**
  - **if** $E \equiv E'$ **or there exist expressions** E_1, E_2, $\ldots$, E_n **such that:**

    1. $E \equiv E_1$
    2. $E' \equiv E_n$
    3. **For each** i **either:**
       - $E_i \underset{\alpha}{\longrightarrow} E_{i+1}$ **or**
       - $E_i \underset{\beta}{\longrightarrow} E_{i+1}$ **or**
       - $E_i \underset{\eta}{\longrightarrow} E_{i+1}$