Pure Lambda Calculus

Lecture 17
Lambda Calculus

- Lambda Calculus (λ-calculus) is a functional notation introduced by Alonzo Church in the early 1930s to formalize the notion of computability.
- Pure λ-calculus is an abstract model of computation and is used to study the computation with functions.
- It is mainly concerned with
  - functional applications and
  - the evaluation of λ-expressions by techniques of substitution.
- The λ-calculus is an expressive and sufficiently powerful simple language with few constructs and a simple semantics and can be used to express all computable functions.
Pure λ-calculus does not have any built-in functions or constants but these are included in applied λ-calculus.

Different languages are generated from different choices of functions and constants.

Functional programming languages (e.g., LISP, SCHEME, POP-2, SML) are based on applied λ-calculus.

Calculation in the λ-calculus is by rewriting (reducing) a λ-expression to a normal form.
  - For pure λ-calculus, λ-expressions are reduced by substitution.
  - Every occurrences of the parameter in the body are replaced with (copies of) the argument. In extended λ-calculus, we also apply the usual reduction rules.
Pure $\lambda$-Calculus

- Mainly three constructs to define $\lambda$-term
  - variables,
  - function application and
  - function abstraction.

- Notational conventions
  - A variable denoted by $x, y, z, f, g, \ldots$
  - $\lambda$-term denoted by $M, N, P, Q, \ldots$
\[\text{\textit{\textbf{\lambda-term}}}\]

- All variables are \lambda-terms and are called \textit{atoms}.
- If \(M\) and \(N\) are arbitrary \lambda-terms, then \((MN)\) is a \lambda-term, called \textit{function application}.
  - More usual notation for function application is \(M(N)\) but historically \((MN)\) has become standard in \lambda-calculus.
- If \(M\) is any \lambda-term and \(x\) is any variable, then \((\lambda x.M)\) is a \lambda-term.
  - This is called an \textit{function abstraction}.
Formally, the grammar of $\lambda$-term in pure $\lambda$-calculus is defined as:

$$<\lambda\text{-term}> ::= x \mid (MN) \mid (\lambda x . M)$$

**Examples:** Following are examples of $\lambda$-terms.

1. $x$
2. $(x y)$
3. $(\lambda x . x)$
4. $(\lambda x . (y z))$
5. $((\lambda x . x) y)$
6. $(x (\lambda y . y))$
7. $(\lambda x . (\lambda y . (x y)))$
8. $(\lambda x . y) (\lambda y . z)$
Informal Interpretation of \( \lambda \)-term

- A term \((\lambda x \cdot M)\) represents a function whose value at an argument \(N\) such as \((\lambda x \cdot M)N\) is calculated by substituting \(N\) for all free occurrence of \(x\) in \(M\).

**Example:** \((\lambda f \cdot f(f \ y))\) represents an operation of applying a function twice to an object denoted by \(y\).

\[\text{e.g., } (\lambda f \cdot f(f \ y)) \ N = N(N \ y)\]

- If \(M\) has been interpreted as a function, then \((MN)\) is interpreted as the result of applying \(M\) to an argument \(N\) provided the result is meaningful.

**Example:**

If \(M = (\lambda x \cdot x \ y)\), then \(MN = (\lambda x \cdot x \ y)N = N \ y\)
Definition: The $\lambda$-expression of the form $(\lambda x . y)$ is called a constant function which when applied to any argument $N$ gives $y$ as a result
- e.g., $(\lambda x . y) \ N = y$ or $(\lambda x . y) \ M = y$.

Definition: The $\lambda$-expression of the form $(\lambda x . x)$ is called identity function which when applied to any argument $N$ gives itself e.g., $(\lambda x . x) \ N = N$.

Pure lambda calculus is untyped and functions can be applied freely.
- The $\lambda$-term $(x \ x)$ is valid, where variable $x$ is applied to itself. But in notion of computation, it may not be meaningful.
- For the sake of simplicity, the following syntactic conventions are used to minimize brackets.
- $MNPQ$ is same as $(((MN)P)Q)$ - (left associative).
- $\lambda x . MN$ is same as $(\lambda x . (MN))$
Currying of Function

- Currying of function means \( \lambda \)-function with more arguments can be expressed as a function of less number of arguments.
- We can write
  \[
  \lambda x_1 . (\lambda x_2 . (\ldots . (\lambda x_n . M) \ldots)) = \lambda x_1 x_2 \ldots x_n . M
  \]
- \( \lambda \)-function with more variables is right associative.
  - if \( (\lambda xy . x y) \) is applied to a single argument say, \( z \) then \( (\lambda xy . x y) z \) results in \( \lambda y . z y \)
  - if \( (\lambda xyz . x y z) \) is applied to two argument say, \( p & q \) then \( (\lambda xyz . x y z) p q \) results in \( \lambda z . p q z \)
- \( \lambda \)-function with \( n \) arguments is when applied on
  - one argument, then it reduces to a function of \( (n-1) \) arguments
  - two arguments, then it reduces to a function of \( (n-2) \) arguments and so on.
The scope of various $\lambda$’s in the following function are given below:

$$\lambda_1 y \cdot y x (\lambda_2 x \cdot y (\lambda_3 y \cdot z)x) v w$$

- Scope of $\lambda_1$ is $y x (\lambda_2 x \cdot y (\lambda_3 y \cdot z)x)$
- Scope of $\lambda_2$ is $y(\lambda_3 y \cdot z)x$
- Scope of $\lambda_3$ is $z$

(v w) does not fall under the scope of any $\lambda$.
Definitions

**Definition:** An occurrence of a variable x in a \( \lambda \)-term P is **bound** iff it is in a part of P with the form \((\lambda x . M)\) otherwise x is said to be **free** variable.

– The set of all such variables is denoted by FV(P).

**Definition:** A term is **closed** if it does not have free variables.

**Substitutions:**

- The notation \( M [N/x] \) means to get a result after substituting N for each free occurrence of x in a term M.
- The substitution is said to be **valid** if no free variable in N becomes bound after substitution.
Substitution Rules

There are few substitution rules:

1. $x [N/x] = N$
2. $y [N/x] = y$, $\forall y \neq x$
3. $(P \ q) [N/x] = (P [N/x]) (Q [N/x])$
4. $(\lambda x. M) [N/x] = \lambda x. M$
   (since $x$ is bound in $M$ thus can not be replaced by $N$)
5. $(\lambda y. M) [N/x] = \lambda y. M [N/x]$, $\mbox{if } y \neq x \mbox{ and } y \notin \mbox{FV}(N)$
6. $(\lambda y. M) [N/x] = \lambda z. ((M [z/y]) [N/x])$, $\mbox{if } y \neq x \mbox{ and } y \in \mbox{FV}(N) \mbox{ and } z \notin \mbox{FV}(MN)$.

Some obvious results:

- $M [x/x] = M$
- $M [N/x] = M$, $\mbox{if } x \notin \mbox{FV}(M)$
- $\mbox{FV}(M [N/x]) = \mbox{FV}(N) \cup (\mbox{FV}(M) - \{x\})$, $\mbox{if } x \in \mbox{FV}(M)$
- $\mbox{FV}(MN) = \mbox{FV}(M) \cup \mbox{FV}(N)$
- $\mbox{FV}(\lambda x. M) = \mbox{FV}(M) - \{x\}$
- $\mbox{FV}(xy) = \{x, y\}$
Examples

Evaluate the following $\lambda$-expressions (constant functions in following examples) using substitution rules.

1. $(\lambda y . x) [z/x]$
   
   Using rule (5), we get
   
   $(\lambda y . x) [z/x] = \lambda y . x [z/x]$
   
   Further, using rule (1), we get
   
   $\lambda y . z$
   
   Hence, $(\lambda y . x) [z/x] = \lambda y . z$

2. $(\lambda y . x) [y/x]$
   
   Here rule (5) can not be applied as $y \in \text{FV}(N) = \{y\}$. Using rule (6), we get
   
   $(\lambda y . x) [y/x] = \lambda z . ((x [z/y]) [y/x])$
   
   $\quad = \lambda z . x [y/x] \{\text{using rule (2)}\}$
   
   $\quad = \lambda z . y \{\text{using rule (1)}\}$
   
   Hence,
   
   $(\lambda y . x) [y/x] = \lambda z . y$
3. \((\lambda y \cdot x \ (\lambda x \cdot x)) \ [(\lambda y \cdot x \ y)/x]\)

Here \(N = (\lambda y \cdot x \ y), \ FV(N) = \{x\}\)

\(M = x \ (\lambda x \cdot x)\)

Since \(y \notin FV(N)\), using rule 5, we get

\((\lambda y \cdot x \ (\lambda x \cdot x) \ ) \ [(\lambda y \cdot x \ y)/x]\)

\[= \lambda y \cdot (x \ (\lambda x \cdot x)) \ [(\lambda y \cdot x \ y)/x]\]

Using (3), we get

\[= \lambda y \cdot x \ [(\lambda y \cdot x \ y)/x] \ (\lambda x \cdot x) \ [(\lambda y \cdot x \ y)/x]\]

Using (1) and (4)

\[= \lambda y \cdot (\lambda y \cdot x \ y) \ (\lambda x \cdot x)\]

Hence,

\((\lambda y \cdot x \ (\lambda x \cdot x)) \ [(\lambda y \cdot x \ y)/x]\)

\[= \lambda y \cdot (\lambda y \cdot x \ y) \ (\lambda x \cdot x)\]
The $\lambda$-expression ($\lambda$-term) is simplified by using conversion or reduction rules. There are three types of $\lambda$ - conversions rules. These make use of substitution explained above. 

- $\alpha$-conversion (alpha)
- $\beta$-conversion (beta)
- $\eta$-conversion (eta)
Few Definitions

**Definition:** Let a term $P$ contains an occurrence of $(\lambda x . M)$ and let $y \notin \text{FV}(M)$. Then the act renaming $x$ by $y$ is replacing $(\lambda x . M)$ by $(\lambda y . M[y/x])$. It is called a change of bound variable in $P$.

**Definition:** $P$ $\alpha$-converts to $Q$ iff $Q$ has been obtained from $P$ by finite series of changes of bound variables.

- The terms $P$ and $Q$ have identical interpretations and play identical roles in any application of $\lambda$-calculus.

**Definition:** Two terms $M$ and $N$ are congruent if $M$ $\alpha$-converts to $N$. It is denoted by

$$M \rightarrow_\alpha N.$$
\(\alpha\)-conversion rule

- Any abstraction of the form \(\lambda x . M\) can be converted to \(\lambda y . M [y/x]\) provided the substitution is valid.
- This is called \(\alpha\)-conversion or \(\alpha\)-reduction rule.

\(\alpha\)-reduction Rule: \(\lambda x . M \rightarrow_\alpha \lambda y . M [y/x]\)

**Examples:**

1. \(\lambda x . x y \rightarrow_\alpha \lambda z . z y,\) whereas \(\lambda x . (x y) \neq_\alpha \lambda y . (y y)\)

2. \(\lambda x y . x(x y) \rightarrow_\alpha \lambda u z . u(u z)\)

**Proof:**

\[
\begin{align*}
\lambda x y . x(x y) &= \lambda x . (\lambda y . x(x y)) \\
&= \lambda x . (\lambda z . x(x z)) \\
&\rightarrow_\alpha \lambda u . (\lambda z . u(u z)) \\
&\rightarrow_\alpha \lambda u z . u(u z)
\end{align*}
\]
**β-conversion rule**

- A λ-expression \((\lambda x . M)N\) represents a function \((\lambda x . M)\) applied to an argument \(N\).
- Any λ-expression of the form \((\lambda x . M)N\) is reduced to \(M[N/x]\) provided the substitution of \(N\) for \(x\) in \(M\) is valid.
- This type of reduction is called β-conversion or β-reduction rule.
- It is the most important conversion rule.
Definition: Any term of the form \((\lambda x . M) N\) is called a \(\beta\)-redex (redex stands for reducible expression) and the corresponding term \(M[N/x]\) is called its contractum.

Definition: If a term \(P\) contains an occurrence of \((\lambda x . M) N\) and if we replace that occurrence by \(M[N/x]\) to obtain a result \(Q\), then we say that we have contracted the redex occurrence in \(P\) or \(P\ \beta\)-contracts to \(Q\) (denoted as \(P \rightarrow_\beta Q\)).

Alternatively, \(P\ \beta\)-reduces to \(Q\) (\(P \rightarrow_\beta Q\)) iff \(Q\) is obtained from \(P\) by finite (perhaps empty) series of \(\beta\)-contractions.

\(\beta\)-reduction Rule: \((\lambda x . M) N \rightarrow_\beta M[N/x]\)
Examples $\beta$-reduction

1. $(\lambda x . x) y \quad \rightarrow \quad x [y/x] \quad \rightarrow \quad y$
   Hence, $(\lambda x . x) y \quad \rightarrow_{\beta} \quad y$

2. $(\lambda x . x(x y)) N \quad \rightarrow \quad x(x y) [N/x]$
   $\quad \rightarrow \quad N(N y)$
   Hence $(\lambda x . x(x y)) N \quad \rightarrow_{\beta} \quad N(N y)$

3. $(\lambda x . y) N \quad \rightarrow \quad y [N/x] \quad \rightarrow \quad y$
   Hence $(\lambda x . y) N \quad \rightarrow_{\beta} \quad y$
4. Show that

\[(\lambda x. (\lambda y. y x) z) \ v \rightarrow_\beta z \ v\]

Proof:

\[(\lambda x. (\lambda y. y x) z) \ v \rightarrow ((\lambda y. y x)z) \ [v/x]\]
\[\rightarrow (\lambda y. y v) \ z\]
\[\rightarrow (y v) \ [z/y]\]
\[\rightarrow z \ v\]

Hence

\[(\lambda x. (\lambda y. y x) z) \ v \rightarrow_\beta z \ v\]
η-conversion

- Any abstraction of the form \((\lambda x . M x)\) is reduced to \(M\) if \(x\) is not free in \(M\).
- A \(\lambda\)-expression (abstraction form) to which \(\eta\)-reduction can be applied is called \(\eta\)-redex.
- The rule of \(\eta\)-conversion expresses the property that two functions are equal if they give the same results when applied to the same arguments.

\[\eta\text{-Reduction Rule: } (\lambda x . M x) \rightarrow M\]
Equality of $\lambda$-expressions

- Two $\lambda$-expressions $M$ and $N$ are equal if they can be transformed into each other by finite sequence of $\lambda$-conversions.
- The meaning of $\lambda$-expressions are preserved after applying any conversion rule.
- For example, $\lambda x . x$ is same as $\lambda z . z$.
- We represent $M$ equal to $N$ by $M = N$. 
Properties of Equality

Equality satisfies the following properties (equivalence relation in set theory):

- **Idempotence**: A term $M$ is equal to itself e.g., $M = M$.
- **Commutativity**: If $M$ is equal to $N$, then $N$ is also equal to $M$ i.e., if $M = N$, then $N = M$.
- **Transitivity**: If $M$ is equal to $N$ and $N$ is equal to $P$ then $M$ is equal to $P$ i.e., if $M = N$ and $N = P$ then $M = P$. 
Identical Expressions

**Definition:** Two $\lambda$-expressions $M$ and $N$ are identical, if they consists of same sequence of characters.

**Example:**

$\lambda x .xy$ and $\lambda z .zy$ are not identical but they are equal.

– So, identical expressions are equal but converse may not be true.
Computation with Pure $\lambda$-Term

- Computation in the $\lambda$-calculus is done by rewriting (reducing) a $\lambda$-expression into a simple form as far as possible.
- The result of computation is independent of the order in which reduction is applied.
- A reduction is any sequence of $\lambda$-conversions.
- If a term can not be further reduced, then it is said to be in a normal form.
Normal Form

**Definition:** A $\lambda$-expression is said to be in *normal form* if no beta-redex (e.g. a sub exp of the form $(\lambda x. M)N$) occurs in it.

**Example:** The normal form of the $\lambda$-expression

$$(\lambda x . (\lambda y . y x) z) v \rightarrow z v$$

It is not necessary that all $\lambda$-expressions have the normal forms because $\lambda$-expression may not be terminating.
Example of \( \lambda \)-expression with no Normal Form

**Example:** \( \lambda \)-expression \((\lambda x . xxy) \ (\lambda x . xxy)\) does not have a normal form because it is non terminating.

\[
(\lambda x . xxy) \ (\lambda x . xxy)
\]

\[
\rightarrow (xxy) \ [(\lambda x . xxy)/x]
\]

\[
\rightarrow (\lambda x . xxy) \ (\lambda x . xxy) \ y
\]

\[
\rightarrow (xxy) \ [(\lambda x . xxy)/x] \ y
\]

\[
\rightarrow (\lambda x . xxy) \ (\lambda x . xxy) \ yy
\]

\[
\vdots
\]

\[
\vdots
\]

- This reduction is nonterminating
Reduction Order

- $\lambda$-conversion rules provide the mechanism to reduce a $\lambda$-expression to normal form but do not tell us what order to apply the reductions when more than one redex is available.

- Mathematician Curry proved that if an expression has a normal form, then it can be found by leftmost reduction.
  - The leftmost reduction is called *lazy reduction* because it does not first evaluate the arguments but substitutes the arguments directly into the expression.
  - Curried functions ($f \times y z$) use lazy reduction.
  - *Eager reduction* is one where the arguments are evaluated before substitution.
  - Function $f(x,y,z)$ use eager reduction.