Lambda Calculus for Advanced Programming

Based on Dr MC du Plessis' Lectures

June 2016

Introduction

In Lambda Calculus

- $< expression >::=< name > \mid < function > \mid < application >$
- $< function >::= \lambda < name > . < expression >$
- $< application >::= (< function > \quad < expression >)$

Examples of valid Lambda expressions

- $\lambda x.x$
- $\lambda first.\lambda second.first$
- $\lambda f.\lambda a.(f\ a)$

β-reductions

Everything done when evaluating Lambda expressions are called β-reductions.

Steps: Remove brackets, take away first name and replace with second part of application.

Examples

- $(\lambda a.\lambda b.(a \quad b) \quad z) = \lambda b.(z \quad b)$
- $(\lambda x.x \quad \lambda y.y) = \lambda y.y$

Identity function example

- $(\lambda x.x \quad \lambda x.x) = \lambda x.x$

More functions

Identity function	$\lambda x.x$

Self-application function	$\lambda s.(s \quad s)$

E.g. Apply identity function to self-application function

- $(\lambda x.x \quad \lambda s.(s \quad s)) = \lambda s.(s \quad s)$

E.g. Apply self-application function to identity function

- $(\lambda s.(s \quad s) \quad \lambda x.x) = (\lambda x.x \quad \lambda x.x) = \lambda x.x$

More functions

E.g. Apply self-application function to self-application function

$$(\lambda s.(s \quad s) \quad \lambda s.(s \quad s))$$

$$\downarrow$$

$$(\lambda s.(s \quad s) \quad \lambda s.(s \quad s))$$

$$\downarrow$$

$$\vdots$$

$$\downarrow$$

$$(\lambda s.(s \quad s) \quad \lambda s.(s \quad s))$$

More functions

Function application function	$\lambda func.\lambda arg.(func \quad arg)$

E.g. Apply function application function to identity function and self-application function

$$((\lambda func.\lambda arg.(func \quad arg) \quad \lambda x.x) \quad \lambda s.(s \quad s))$$
$$\downarrow$$
$$(\lambda arg.(\lambda x.x \quad arg) \quad \lambda s.(s \quad s))$$
$$\downarrow$$
$$(\lambda x.x \quad \lambda s.(s \quad s))$$
$$\downarrow$$
$$\lambda s.(s \quad s)$$

α-conversion

Context of variables are local to their expressions, but having multiple variable of the same name may be confusing.
α-conversion is the renaming of variables.

Without α-conversion:

$$
\begin{aligned}
&(\lambda f.\lambda g.(\lambda f.(g \quad f) \quad f) \quad \lambda p.\lambda q.p) \\
&= (\lambda f.\lambda g.(g \quad f) \quad \lambda p.\lambda q.p) \\
&= \lambda g.(g \quad \lambda p.\lambda q.p)
\end{aligned}
$$

With α-conversion:

$$
\begin{aligned}
&(\lambda f.\lambda g.(\lambda f_1.(g \quad f_1) \quad f) \quad \lambda p.\lambda q.p) \\
&= (\lambda f.\lambda g.(g \quad f) \quad \lambda p.\lambda q.p) \\
&= \lambda g.(g \quad \lambda p.\lambda q.p)
\end{aligned}
$$

β-reduction example

$$
\begin{aligned}
&(((\lambda x.\lambda y.\lambda z.((x \quad y) \quad z) \quad \lambda f.\lambda a.(f \quad a)) \quad \lambda i.i) \quad \lambda j.j) \\
&= ((\lambda y.\lambda z.((\lambda f.\lambda a.(f \quad a) \quad y) \quad z) \quad \lambda i.i) \quad \lambda j.j) \\
&= (\lambda z.((\lambda f.\lambda a.(f \quad a) \quad \lambda i.i) \quad z) \quad \lambda j.j) \\
&= ((\lambda f.\lambda a.(f \quad a) \quad \lambda i.i) \quad \lambda j.j) \\
&= (\lambda a.(\lambda i.i \quad a) \quad \lambda j.j) \\
&= (\lambda i.i \quad \lambda j.j) \\
&= \lambda j.j
\end{aligned}
$$

Named Functions

Define named functions with: **def** $<$ **name** $> = <$ **func** $>$ to make it easier to write out.

Some named functions:

def self_apply $= \lambda$**s**.(**s** **s**)

def identity $= \lambda$**x**.**x**

def apply $= \lambda$**func**.λ**arg**.(**func** **arg**)

Named Functions

Let $\textbf{def identity}_2 = \lambda \mathbf{x}.((\textbf{apply} \quad \textbf{identity}) \quad \mathbf{x})$, then:

$$
\begin{aligned}
(identity_2 \quad identity) &= (\lambda x.((apply \quad identity) \quad x) \quad identity) \\
&= ((apply \quad identity) \quad identity) \\
&= ((\lambda f.\lambda a.(f \quad a) \quad identity) \quad identity) \\
&= ((\lambda a.(identity \quad a) \quad identity) \\
&= (identity \quad identity) \\
&= (\lambda x.x \quad \lambda x.x) \\
&= \lambda x.x
\end{aligned}
$$

Named Functions

Example

Prove that the application function applies two arguments where the first is the function and the second its expression.

$$\begin{aligned}((apply \quad F) \quad AR) &= ((\lambda f.\lambda a.(f \quad a) \quad F) \quad AR) \\ &= (\lambda a.(F \quad a) \quad AR) \\ &= (F \quad AR)\end{aligned}$$

Named Functions

Example

Let $\textbf{def self\_apply}_2 = \lambda \mathbf{s}.((\textbf{apply} \quad \mathbf{s}) \quad \mathbf{s})$. Prove that this named function is equivalent to the normal apply function.

$$
\begin{aligned}
(self\_apply_2 \quad z) &= (\lambda s.((apply \quad s) \quad s) \quad z) \\
&= ((apply \quad z) \quad z) \\
&= ((\lambda f.\lambda a.(f \quad a) \quad z) \quad z) \\
&= (\lambda a.(z \quad a) \quad z) \\
&= (z \quad z)
\end{aligned}
$$

Named Functions

More named functions:

$$\textbf{def select\_first} = \lambda\textbf{first}.\lambda\textbf{second}.\textbf{first}$$

$$\textbf{def select\_second} = \lambda\textbf{first}.\lambda\textbf{second}.\textbf{second}$$

Select first example:

Notice how *identity* can be replaced with A and similarly you'd get A as result.

$$\begin{aligned} ((\mathit{select\_first}\ \mathit{identity})\ \mathit{apply}) &= ((\lambda \mathit{first}.\lambda \mathit{second}.\mathit{first}\ \mathit{identity})\ \mathit{apply}) \\ &= (\lambda \mathit{second}.\mathit{identity}\ \mathit{apply}) \\ &= \mathit{identity} \end{aligned}$$

Named Functions

Select second example:

$$
\begin{aligned}
((select\_second \quad A) \quad B) &= ((\lambda first.\lambda second.second \quad A) \quad B) \\
&= (\lambda second.second \quad B) \\
&= B
\end{aligned}
$$

Observe:

$$
\begin{aligned}
&(select\_first\ identity) \\
&= (\lambda first.\lambda second.firstidentity) \\
&= \lambda second.identity \\
&= \lambda second.\lambda x.x \qquad\qquad\qquad \text{(Renaming variables)} \\
&= \lambda first.\lambda second.second \\
&\equiv select\_second
\end{aligned}
$$

Named Functions

$$\textbf{def make\_pair} = \lambda \textbf{first}.\lambda \textbf{second}.\lambda \textbf{func}.((\textbf{func first})\ \textbf{second})$$

make_pair acts like a tuple.

$$\begin{aligned} &((\mathit{make\_pair}\ A)\ B) \\ &= ((\lambda \mathit{first}.\lambda \mathit{second}.\lambda \mathit{func}.((\mathit{func}\ \mathit{first})\ \mathit{second})\ \mathit{second})\ A)\ B) \\ &= (\lambda \mathit{second}.\lambda \mathit{func}((\mathit{func}\ A)\ \mathit{second})\ B) \\ &= \lambda \mathit{func}.((\mathit{func}\ A)\ B) \end{aligned}$$

Named Functions

Example application of *make_pair* result:

$$(\lambda func.((func \quad A) \quad B) \quad select\_first)$$
$$= ((select\_first \quad A) \quad B)$$

Similarly:

$$(\lambda func.((func \quad A) \quad B) \quad select\_second)$$
$$= ((select\_second \quad A) \quad B)$$

If Statement

We want `if` statements to be able to do more interesting things.

$$\textbf{def cond} = \lambda \mathbf{e_1}.\lambda \mathbf{e_2}.\lambda \mathbf{c}((\mathbf{c} \quad \mathbf{e_1}) \quad \mathbf{e_2})$$

$$\textbf{def true} = \lambda \mathbf{a}.\lambda \mathbf{b}.\mathbf{a}$$

$$\textbf{def false} = \lambda \mathbf{a}.\lambda \mathbf{b}.\mathbf{b}$$

Notice how *true* and *false* come from *select_first* and *select_second*.

If Statement

Example

$$
\begin{aligned}
&(((cond \quad x) \quad y) \quad true) \\
&= ((((\lambda e_1.\lambda e_2.\lambda c((c \quad e_1) \quad e_2) \quad x) \quad y) \quad true) \\
&= ((\lambda e_2.\lambda c((c \quad x) \quad e_2) \quad y) \quad true) \\
&= (\lambda c((c \quad x) \quad y) \quad true) \\
&= ((true \quad x) \quad y) \\
&= ((\lambda a.\lambda b.a \quad x) \quad y) \\
&= (\lambda b.x \quad y) \\
&= x
\end{aligned}
$$

Read like a ternary `if` statement, i.e.: `cond ?  x :  y`. In this case `true` is passed as the **condition**.

NOT Operator

Don't have operators yet so we can't get `true` or `false` out of conditions. Consider a truth table for NOT:

X	NOT X
T	F
F	T

Where `T` is `true` and `F` is `false`. How do you define a function for this?

NOT Operator

Solution lies in utilizing the ternary `if` statement: `x ? false : true`.

$$\textbf{def not} = \lambda \mathbf{x}.(((\textbf{cond} \quad \textbf{false}) \quad \textbf{true}) \quad \mathbf{x})$$

Derivation

$$
\begin{aligned}
&\lambda x.(((cond \quad false) \quad true) \quad x) \\
&= \lambda x.((((\lambda e_1.\lambda e_2.\lambda c.((c \quad e_1) \quad e_2) \quad false) \quad true) \quad x) \\
&= \lambda x.((\lambda e_2.\lambda c.((c \quad false) \quad e_2) \quad true) \quad x) \\
&= \lambda x.(\lambda c.((c \quad false) \quad true) \quad x) \\
&= \lambda x.((x \quad false) \quad true)
\end{aligned}
$$

Now we can pass `x` as an argument and get a result...

NOT Operator

Test the result

Passing `true` as argument:

$$
\begin{aligned}
&(not \quad true) \\
&= (\lambda x.(\lambda x.((c \quad false) \quad true) \quad x) \quad true) \\
&= \lambda x.((x \quad false) \quad true) \\
&= ((true \quad false) \quad true) \\
&= ((\lambda a.\lambda b.a \quad false) \quad true) && \text{(def. of true)} \\
&= (\lambda b.false \quad true) \\
&= false
\end{aligned}
$$

AND Operator

Truth table for the AND operator:

X	Y	X AND Y
T	T	T
T	F	F
F	T	F
F	F	F

Where `T` is `true` and `F` is `false`. How do you define a function for this?

AND Operator

Comes from `x ? y : false`.

$$\textbf{def and} = \lambda \mathbf{x}.\lambda \mathbf{y}.((\mathbf{x} \quad \mathbf{y}) \quad \textbf{false})$$

Derivation

$$\begin{aligned}
&\lambda x.\lambda y.(((cond \quad y) \quad false) \quad x) \\
&= \lambda x.\lambda y.(((\lambda e_1.\lambda e_2.\lambda c.((c \quad e_1) \quad e_2) \quad y) \quad false) \quad x) \\
&= \lambda x.\lambda y.((\lambda e_2.\lambda c.((c \quad y) \quad e_2) \quad false) \quad x) \\
&= \lambda x.\lambda y.(\lambda c.((c \quad y) \quad false) \quad x) \\
&= \lambda x.\lambda y.((x \quad y) \quad false)
\end{aligned}$$

Now we can pass `x` and `y` as arguments and get a result...

AND Operator

Test the result

Passing `true` and `false` as arguments:

$$
\begin{aligned}
&((\mathit{and} \quad \mathit{true}) \quad \mathit{false}) \\
&= ((\lambda x.\lambda y.((x \quad y) \quad \mathit{false}) \quad \mathit{true}) \quad \mathit{false}) \\
&= (\lambda y.((\mathit{true} \quad y) \quad \mathit{false}) \quad \mathit{false}) \\
&= ((\mathit{true} \quad \mathit{false}) \quad \mathit{false}) \\
&= ((\lambda a.\lambda b.a \quad \mathit{false}) \quad \mathit{false}) && \text{(def. of true)} \\
&= (\lambda b.\mathit{false} \quad \mathit{false}) \\
&= \mathit{false}
\end{aligned}
$$

AND Operator

Test the result

Passing `true` and `true` as arguments:

$$
\begin{aligned}
&((\mathit{and} \quad \mathit{true}) \quad \mathit{true}) \\
&= ((\lambda x.\lambda y.((x \quad y) \quad \mathit{false}) \quad \mathit{true}) \quad \mathit{true}) \\
&= (\lambda y.((\mathit{true} \quad y) \quad \mathit{false}) \quad \mathit{true}) \\
&= ((\mathit{true} \quad \mathit{true}) \quad \mathit{false}) \\
&= ((\lambda a.\lambda b.a \quad \mathit{true}) \quad \mathit{false}) && \text{(def. of true)} \\
&= (\lambda b.\mathit{true} \quad \mathit{false}) \\
&= \mathit{true}
\end{aligned}
$$

OR Operator

Truth table for the OR operator:

X	Y	X OR Y
T	T	T
T	F	T
F	T	T
F	F	F

Where `T` is `true` and `F` is `false`. How do you define a function for this?

OR Operator

Comes from `x ? true : y`.

$$\textbf{def or} = \lambda \mathbf{x}.\lambda \mathbf{y}.((\mathbf{x} \quad \mathbf{true}) \quad \mathbf{y})$$

Derivation

$$\begin{aligned}
&\lambda x.\lambda y.(((cond \quad true) \quad y) \quad x) \\
&= \lambda x.\lambda y.(((\lambda e_1.\lambda e_2.\lambda c.((c \quad e_1) \quad e_2) \quad true) \quad y) \quad x) \\
&= \lambda x.\lambda y.((\lambda e_2.\lambda c.((c \quad true) \quad e_2) \quad y) \quad x) \\
&= \lambda x.\lambda y.(\lambda c.((c \quad true) \quad y) \quad x) \\
&= \lambda x.\lambda y.((x \quad true) \quad y)
\end{aligned}$$

Now we can pass `x` and `y` as arguments and get a result...

OR Operator

Test the result

Passing `true` and `false` as arguments:

$$
\begin{aligned}
&((or \quad true) \quad false) \\
&= ((\lambda x.\lambda y.((x \quad true) \quad y) \quad true) \quad false) \\
&= (\lambda y.((true \quad true) \quad y) \quad false) \\
&= ((true \quad true) \quad false) \\
&= ((\lambda a.\lambda b.a \quad true) \quad false) && \text{(def. of true)} \\
&= (\lambda b.true \quad false) \\
&= true
\end{aligned}
$$

Conditions & Operators

Exercise

Use Lambda Calculus to calculate the following:

$$
\begin{aligned}
&\textit{if } ((T \vee F) \wedge (F \vee T))\ \{ \\
&\quad T \wedge F \\
&\}\ \textit{else}\ \{ \\
&\quad T \\
&\}
\end{aligned}
$$

Conditions & Operators

Solution

Start with a basic outline:

$$(((cond\ x)\ y)\ z)$$

where z is the actual condition you're passing. Similar to the ternary `if` statement `z ? x : y`. Now:

$$x = ((and\ T)\ F),$$

$$y = T,$$

and

$$z = ((and\ ((or\ T)\ F))\ ((or\ F)\ T)).$$

Substituting yields:

$$(((cond\ ((and\ T)\ F))\ T)\ ((and\ ((or\ T)\ F))\ ((or\ F)\ T))) \tag{1}$$

Conditions & Operators

Solution continued

Breaking up the work into parts:

$$\begin{aligned}((or \quad T) \quad F) &= ((\lambda a.\lambda b.((a \quad T) \quad b) \quad T) \quad F) \\ &= (\lambda b.((T \quad T) \quad b) \quad F) \\ &= ((T \quad T) \quad F) \\ &= ((\lambda a.\lambda b.a \quad T) \quad F) \\ &= (\lambda b.T \quad F) \\ &= T\end{aligned} \tag{2}$$

Conditions & Operators

Solution continued

$$
\begin{aligned}
((or \quad F) \quad T) &= ((\lambda a.\lambda b.((a \quad T) \quad b) \quad F) \quad T) \\
&= (\lambda b.((F \quad T) \quad b) \quad T) \\
&= ((F \quad T) \quad T) \\
&= ((\lambda a.\lambda b.b \quad T) \quad T) \\
&= (\lambda b.b \quad T) \\
&= T
\end{aligned}
\tag{3}
$$

Conditions & Operators

Solution continued

Now combining the results from (2) and (3):

$$
\begin{aligned}
((and \quad T) \quad T) &= ((\lambda x.\lambda y.((x \quad y) \quad F) \quad T) \quad T) \\
&= (\lambda y.((T \quad y) \quad F) \quad T) \\
&= ((T \quad T) \quad F) \\
&= ((\lambda a.\lambda b.a \quad T) \quad F) \\
&= (\lambda b.T \quad F) \\
&= T
\end{aligned}
\tag{4}
$$

Conditions & Operators

Solution continued

$$
\begin{aligned}
((and \quad T) \quad F) &= ((\lambda a.\lambda b.((a \quad b) \quad F) \quad T) \quad F) \\
&= (\lambda b.((T \quad b) \quad F) \quad F) \\
&= ((T \quad F) \quad F) \\
&= ((\lambda a.\lambda b.a \quad F) \quad F) \\
&= (\lambda b.F \quad F) \\
&= F
\end{aligned} \tag{5}
$$

Conditions & Operators

Solution continued

Finally combining results from (4) and (5) into (1), yields:

$$
\begin{aligned}
(((cond\ F)\ T)\ T) &= (((\lambda e_1.\lambda e_2.\lambda c.((c\ e_1)\ e_2)\ F)\ T)\ T) \\
&= ((\lambda e_2.\lambda c.((c\ F)\ e_2)\ T)\ T) \\
&= (\lambda c.((c\ F)\ T)\ T) \\
&= ((T\ F)\ T) \\
&= ((\lambda a.\lambda b.a\ F)\ T) \\
&= (\lambda b.F\ T) \\
&= F \quad \square
\end{aligned} \tag{6}
$$

Numbers

How are numbers defined?

$$\begin{aligned} 1 &= \text{ successor of } 0 \\ 2 &= \text{ successor of } 1 \\ &\vdots \end{aligned}$$

$\boxed{\textbf{def zero} = \lambda \mathbf{x}.\mathbf{x} \qquad \text{(identity function)}}$

$\boxed{\textbf{def succ} = \lambda \mathbf{n}.\lambda \mathbf{s}.((\mathbf{s} \quad \textbf{false}) \quad \mathbf{n})}$

Now we can define **def one** $=$ (**succ** **zero**) etc. for larger numbers.

Numbers

Numbers can be derived as follows:

$$
\begin{aligned}
one &= (succ \quad zero) \\
&= (\lambda n.\lambda s.((s \quad false) \quad n) \quad zero) \\
&= \lambda s.((s \quad false) \quad zero) \\
two &= (succ \quad one) \\
&= (\lambda n.\lambda s.((s \quad false) \quad n) \quad one) \\
&= \lambda s.((s \quad false) \quad one) \\
&= \lambda s.((s \quad false) \quad \lambda s.((s \quad false) \quad zero))
\end{aligned}
$$

Notice that in general any positive natural number ζ can be represented by:

$$\lambda s.((s \quad false) \quad \zeta - 1), \quad \ni \zeta \in \{one, two, \dots\}$$

Numbers

$$\textbf{def is\_zero} = \lambda \mathbf{n}.(\mathbf{n} \quad \textbf{select\_first})$$

Test

$$\begin{aligned}(is\_zero \quad zero) &= (\lambda n.(n \quad select\_first) \quad \lambda x.x) \\ &= (\lambda x.x \quad select\_first) \\ &= select\_first \\ &= true\end{aligned}$$

Numbers

Test

$$
\begin{aligned}
&(is\_zero \quad \lambda s.((s \quad false) \quad \zeta - 1)) \\
&= (\lambda n.(n \quad select\_first) \quad \lambda s.((s \quad false) \quad \zeta - 1)) \\
&= (\lambda s.((s \quad false) \quad \zeta - 1) \quad select\_first) \\
&= ((select\_first \quad false) \quad \zeta - 1) \\
&= false
\end{aligned}
$$

$\therefore$ is_zero is *false* for any number not zero.

Numbers

$$\textbf{def pred}_1 = \lambda \mathbf{n}.(\mathbf{n} \quad \textbf{select\_second})$$

Consider:

$$\begin{aligned}(\mathit{pred}_1 \quad \mathit{zero}) &= (\lambda n.(n\mathit{select\_second}) \quad \lambda x.x) \\ &= (\lambda x.x \quad \mathit{select\_second}) \\ &= \mathit{select\_second} \\ &= \mathit{false}\end{aligned}$$

Which clearly isn't correct. So pred_1 doesn't work for *zero*.

Numbers

But what about other numbers? Let $\zeta = \lambda s.((s\ \mathit{false})\ \zeta - 1)$ be some positive natural number $\therefore$

$$\zeta \in \{\mathit{one}, \mathit{two}, \dots\}$$

Then:

$$\begin{aligned}
(\mathit{pred}_1 \quad \zeta) &= (\mathit{pred}_1 \quad \lambda s.((s \quad \mathit{false}) \quad \zeta - 1)) \\
&= (\lambda n.(n \quad \mathit{select_second}) \quad \lambda s.((s \quad \mathit{false}) \quad \zeta - 1)) \\
&= (\lambda s.((s \quad \mathit{false}) \quad \zeta - 1) \quad \mathit{select_second}) \\
&= ((\mathit{select_second} \quad \mathit{false}) \quad \zeta - 1) \\
&= ((\lambda x.\lambda y.y \quad \mathit{false}) \quad \zeta - 1) \\
&= (\lambda y.y \quad \zeta - 1) \\
&= \zeta - 1
\end{aligned}$$

Which is correct.

Numbers

To get around the problem with *zero* consider the function:

$$\lambda n.(((cond \quad zero) \quad (pred_1 \quad n)) \quad (is\_zero \quad n))$$

Which basically reads: `n == zero ?  zero :  n - 1`.
Simplifying:

$$\begin{aligned}
&\lambda n.(((cond \quad zero) \quad (pred_1 \quad n)) \quad (is_zero \quad n)) \\
&= \lambda n.(((\lambda e_1.\lambda e_2.\lambda c.((c\ e_1)\ e_2)\ zero)\ (pred_1\ n))\ (is_zero\ n)) \\
&= \lambda n.((\lambda e_2.\lambda c.((c\ zero)\ e_2)\ (pred_1\ n))\ (is_zero\ n)) \\
&= \lambda n.(\lambda c.((c\ zero)\ (pred_1\ n))\ (is_zero\ n)) \\
&= \lambda n.(((is_zero\ n)\ zero)\ (pred_1\ n)) \\
&= \lambda n.(((is_zero\ n)\ zero)\ (\lambda q.(q\ select_second)\ n)) \\
&= \lambda n.(((is_zero\ n)\ zero)\ (n\ select_second))
\end{aligned}$$

Numbers

From the previous result we can now define:

$$\textbf{def pred} = \lambda \mathbf{n}.(((\textbf{is\_zero n})\ \textbf{zero})\ (\textbf{n select\_second}))$$

Now: how to do addition? Consider:

$$\lambda x.\lambda y.(((\mathit{cond}\ x)\ ((\mathit{add}\ (\mathit{succ}\ x))\ (\mathit{pred}\ y)))\ (\mathit{is\_zero}\ y)).$$

But this is defined recursively, which breaks Lambda Calculus. Therefore...

Numbers

```
let rec add x y = if is_zero y then x else add (succ x) (pred y)
```