The Austalian National University Semester 1, 2024
Research School of Computer Science
Dr. Ranald Clouston

COMP1130: Lambda Calculus - Worksheet 2

- **Combinators**

  For the following expressions, identify which are combinators. (Recall that a combinator is a lambda expression with no free variables.)

  1. $\lambda x.x$
     Solution. The only variable is x which is bound. Therefore, this is a combinator.
  2. $\lambda x.\lambda y.y$
     Solution. All variables are bound. Therefore, this is a combinator.
  3. $\lambda x.xy$
     Solution. The variable y is free. Therefore, this is not a combinator.
  4. $\lambda y.\lambda x.yx(\lambda z.a)y$
     Solution. The variable a is free. Therefore, this is not a combinator.

- **Free Variables and Alpha Equivalence**

  For the following expressions, identify which variables are bound, and which are free. Afterwards, rewrite each expression using alpha conversion where necessary, such that no variable is bound by a λ more than once.

  1. x
     Solution. x is free, as $\mathrm{FV}(x) = \{x\}$.
  2. $\lambda x.x$
     Solution. No free variables, the x is bound to the $\lambda x.$.
  3. $\lambda y.x$
     Solution. x is free, as the $\lambda y.$ does not bound x.
  4. $\lambda x.xy$
     Solution. x is bound, y is free.
  5. $\lambda x.\lambda x.x$
     Solution. x is bound to the inner most λx. We can alpha convert to $\lambda y.\lambda x.x$.
  6. $x(\lambda x.\lambda x.x)$
     Solution. The outer x is free, and the inner x is bound to the inner most $\lambda x.$. We can alpha convert to $x(\lambda z.\lambda y.y)$.
  7. $(\lambda x.x)(\lambda x.x)$
     Solution. No free variables. We could alpha convert to $(\lambda y.y)(\lambda x.x)$ if desired.
  8. $\lambda x.x\lambda x.x$
     Solution. No free variables. The inner x is bound to the inner $\lambda x.$, and the outer x to the other λx. We could alpha convert to $\lambda y.y\lambda x.x$.
  9. $z\lambda x.\lambda y.\lambda z.\lambda x.xyzx$
     Solution. The z on the outside is free, and all the innermost variables are bound. We could alpha convert to $z\lambda a.\lambda b.\lambda c.\lambda d.dbcd$

10. $y(\lambda x.\lambda z.xz)z$
    Solution. The outermost y and z are free. The x and z in the middle are bound. We could alpha convert to $y(\lambda x.\lambda w.xw)z$.
11. $\lambda x.\lambda y.yz(\lambda x.x)x$
    Solution. z is free. We could alpha convert to $\lambda x.\lambda y.yz(\lambda w.w)x$

- **Substitution**

  Apply the following substitutions. If alpha converting is required for the substitution, indicate as such.

  1. $x[x := y]$
     Solution. y
  2. $x[x := \lambda z.z]$
     Solution. $\lambda z.z$
  3. $\lambda x.x[x := y]$
     Solution. Here, x is bound, so by alpha converting to $\lambda z.z[x := y]$, there is no x to replace, so we just get $\lambda z.z$ or equivalentally $\lambda x.x$.
  4. $y(\lambda y.y)[y := \lambda z.zx]$
     Solution. Alpha convert to $y(\lambda w.w)[y := \lambda z.zx]$, and then substitute afterwards to get $(\lambda z.zx)(\lambda w.w)$. Note that $\lambda z.zx(\lambda w.w)$ would be incorrect.
  5. $x(\lambda x.y)[y := z]$
     Solution. $x(\lambda x.z)$
  6. $(\lambda x.xyx)[y := \lambda x.xx]$
     Solution. Alpha convert to $(\lambda z.zyz)[y := \lambda x.xx]$ to get $\lambda z.z(\lambda x.xx)z$. Again, note the brackets, and that $\lambda z.z\lambda x.xxz = \lambda z.z\lambda x.(xxz)$ is different.
  7. $(\lambda x.x(\lambda y.z)x)[x := \lambda z.\lambda y.z]$
     Solution. Here, x is bound, so by alpha converting to $(\lambda w.w(\lambda y.z)w)[x := \lambda z.\lambda y.z]$, there is no x to replace, so we just get $\lambda w.w(\lambda y.z)w$.
  8. $\lambda x.\lambda y.z[y := zx]$
     Solution. There is no y variable to replace, so nothing happens. We get $\lambda x.\lambda y.z$.
  9. $(\lambda x.\lambda y.w(\lambda y.zwy))[z := \lambda x.\lambda y.w(\lambda y.zwy)]$
     Solution. Alpha convert to $(\lambda a.\lambda b.w(\lambda c.zwc))[z := \lambda x.\lambda y.w(\lambda y.zwy)]$, and replace z to obtain
     $$\lambda a.\lambda b.w(\lambda c.(\lambda x.\lambda y.w(\lambda y.zwy))wc)$$

- **Beta Reduction**

  Perform a single beta reduction on the following expressions. Make it explicit where you've had to perform an alpha conversion, if required.

  For example,

  $$(\lambda x.xx)x \underset{\alpha}{=} (\lambda y.yy)x \underset{\beta}{\to} xx$$

  1. $(\lambda x.x)y$
     Solution. $(\lambda x.x)y \underset{\beta}{\to} y$
  2. $(\lambda x.x)x$
     Solution. $(\lambda x.x)x \underset{\alpha}{=} (\lambda y.y)x \underset{\beta}{\to} x$

3. $(\lambda x.\lambda y.xy)xy$
   Solution. $(\lambda x.\lambda y.xy)xy \underset{\alpha}{=} (\lambda a.\lambda y.ay)xy \underset{\beta}{\to} (\lambda y.xy)y$

4. $(\lambda x.yx)(\lambda x.x)$
   Solution. $(\lambda x.yx)(\lambda x.x) \underset{\alpha}{=} (\lambda z.yz)(\lambda x.x) \underset{\beta}{\to} y(\lambda x.x)$

5. $(\lambda x.xx)(\lambda x.xx)$
   Solution. $(\lambda x.xx)(\lambda x.xx) \underset{\alpha}{=} (\lambda y.yy)(\lambda x.xx) \underset{\beta}{\to} (\lambda x.xx)(\lambda x.xx)$ Note that this expression doesn't change under a beta reduction, and attempting to fully beta reduce will not terminate.

- **Encoding Booleans**

  Recall from lectures that

$$\texttt{true} := \lambda x.\lambda y.x$$

$$\texttt{false} := \lambda x.\lambda y.y$$

$$\texttt{if ... then ... else ...} := \lambda x.\lambda y.\lambda z.xyz$$

  For the following questions in this section, try to make your answers as simple as possible, reducing if required.

  1. Find a lambda expression for logical `and`.
     Verify that `and true false` $\underset{\beta}{\to}$ `false`
     (Hint: First try to find an expression for `and` using `if`).
     Solution. We can encode logical `and` of two statements p and q as

$$\texttt{and } p \ q \texttt{ = if } p \texttt{ then } q \texttt{ else false}$$

     as the only case it evaluates to `true` is when both `p` and `q` are true. We can then abstract the p and q over,

$$\texttt{and} = \lambda p.\lambda q.\text{if } p \text{ then } q \text{ else false}$$

     and then insert the definition of `if..then..else` and `false` to obtain a pure lambda calculus expression, without any other built up definitions.

$$\texttt{and} = \lambda p.\lambda q.pq(\lambda x.\lambda y.y)$$

     Now, to verify `and true false` $\underset{\beta}{\to}$ `false`

$$\begin{aligned}
\texttt{and true false} &= (\lambda p.\lambda q.pq(\lambda x.\lambda y.y))(\lambda x.\lambda y.x)(\lambda x.\lambda y.y) \\
&\underset{\beta}{\to} (\lambda q.(\lambda x.\lambda y.x)q(\lambda x.\lambda y.y))(\lambda x.\lambda y.y) \\
&\underset{\beta}{\to} (\lambda x.\lambda y.x)(\lambda x.\lambda y.y)(\lambda x.\lambda y.y) \\
&\underset{\beta}{\to} (\lambda y.(\lambda x.\lambda y.y))(\lambda x.\lambda y.y) \\
&\underset{\beta}{\to} (\lambda x.\lambda y.y) = \texttt{false}
\end{aligned}$$

  2. Find a lambda expression for logical `or`.
     Verify that `or false true` $\underset{\beta}{\to}$ `true`.
     Solution.

We can encode logical `or` of two statements p and q as

$$\texttt{or } p \ q \texttt{ = if } p \texttt{ then true else } q$$

as the only case it evaluates to `false` is when both `p` and `q` are false. We can then abstract the p and q over,

$$\texttt{or} = \lambda p.\lambda q.\texttt{if } p \texttt{ then true else } q$$

and then insert the definition of `if..then..else` and `false` to obtain a pure lambda calculus expression, without any other built up definitions.

$$\texttt{or} = \lambda p.\lambda q.p(\lambda x.\lambda y.x)q$$

Now, to verify that $\texttt{or false true} \underset{\beta}{\to} \texttt{true}$.

$$\begin{aligned}
\texttt{or false true} &= (\lambda p.\lambda q.p(\lambda x.\lambda y.x)q)(\lambda x.\lambda y.y)(\lambda x.\lambda y.x) \\
&\underset{\beta}{\to} (\lambda q.(\lambda x.\lambda y.y)(\lambda x.\lambda y.x)q)(\lambda x.\lambda y.x) \\
&\underset{\beta}{\to} (\lambda x.\lambda y.y)(\lambda x.\lambda y.x)(\lambda x.\lambda y.x) \\
&\underset{\beta}{\to} (\lambda y.y)(\lambda x.\lambda y.x) \\
&\underset{\beta}{\to} (\lambda x.\lambda y.x) = \texttt{true}
\end{aligned}$$

3. Find a lambda expression for logical `not`.
   Verify that $\texttt{not (not } x\texttt{)} \underset{\beta}{\to} x$ for all booleans x.
   (Hint: It may help to evaluate $\texttt{not (not } x\texttt{)}$ for a generic variable x, and then substitute `true` and `false` afterwards.)
   Solution.
   We can encode logical `not` of a statement p

$$\texttt{not } p \texttt{ = if } p \texttt{ then false else true}$$

We can then abstract the p over,

$$\texttt{not} = \lambda p.\texttt{if } p \texttt{ then false else true}$$

and then insert the definition of `if..then..else`, `false` and `true` to obtain a pure lambda calculus expression, without any other built up definitions.

$$\texttt{not} = \lambda p.p(\lambda x.\lambda y.y)(\lambda x.\lambda y.x)$$

Now, to verify that $\texttt{not (not } x\texttt{)} \underset{\beta}{\to} x$.

$$\begin{aligned}
\texttt{not (not } x\texttt{)} &= (\lambda p.p(\lambda x.\lambda y.y)(\lambda x.\lambda y.x))\,((\lambda p.p(\lambda x.\lambda y.y)(\lambda x.\lambda y.x))x) \\
&\underset{\beta}{\to} (\lambda p.p(\lambda x.\lambda y.y)(\lambda x.\lambda y.x))(x(\lambda x.\lambda y.y)(\lambda x.\lambda y.x)) \\
&\underset{\beta}{\to} (x(\lambda x.\lambda y.y)(\lambda x.\lambda y.x))(\lambda x.\lambda y.y)(\lambda x.\lambda y.x)) \\
&= x(\lambda x.\lambda y.y)(\lambda x.\lambda y.x)(\lambda x.\lambda y.y)(\lambda x.\lambda y.x))
\end{aligned}$$

Now, suppose that x is `true`.

$$\begin{aligned}
&\underset{\beta}{\to} (\lambda x.\lambda y.x)(\lambda x.\lambda y.y)(\lambda x.\lambda y.x)(\lambda x.\lambda y.y)(\lambda x.\lambda y.x) \\
&\underset{\beta}{\to} (\lambda y.(\lambda x.\lambda y.y))(\lambda x.\lambda y.x)(\lambda x.\lambda y.y)(\lambda x.\lambda y.x)) \\
&\underset{\beta}{\to} (\lambda x.\lambda y.y)(\lambda x.\lambda y.y)(\lambda x.\lambda y.x) \\
&\underset{\beta}{\to} (\lambda y.y)(\lambda x.\lambda y.x) \\
&\underset{\beta}{\to} (\lambda x.\lambda y.x) \\
&= \texttt{true}
\end{aligned}$$

For the other case, we note the properties

$$\texttt{true}\ x\ y = x$$

$$\texttt{false}\ x\ y = y$$

and use that and left associativity to evaluate directly.

```
not (not false)
→β false false true false true
= (false false true) false true
→β true false true
→β false
```

4. Find a lambda expression for logical `xor`.
(Hint: Use your previously defined logical operators to help you.)
Solution. There are many ways to do this, the shortest I could come up with was that using the value of p to conditionally negate q. `xor` requires that precisely one of the inputs are true, so if p is true, we verify if q is false, and if p is false, we verify if q is true. Hence, the following will work.

$$\texttt{xor}\ p\ q = \texttt{if}\ p\ \texttt{then (not}\ q\texttt{) else}\ q$$

Substituting the definition of `not`, and abstracting over the p and q gives

$$\texttt{xor} = \lambda p.\lambda q.p\ (\ (\lambda p.p(\lambda x.\lambda y.x)(\lambda x.\lambda y.y))\ q)\ q$$

which, when simplified via beta reduction, gives

$$\texttt{xor} = \lambda p.\lambda q.p(q(\lambda x.\lambda y.x)(\lambda x.\lambda y.y))q$$