1 Expressions

- Our basic notions here are **expression** and **evaluation**.

  Definition 1.1 (Expression)
  Let $A = \{a, b, \ldots, z\}$ be the set of names.

  i. all names are expressions;
  ii. if ω_1 and ω_2 are expressions, so is $(\omega_1\omega_2)$;
  iii. if ω is an expression and α is a name, then $(\lambda\alpha.\omega)$ is an expression.
  iv. nothing else is an expression.

- Some example expressions:

$$
\begin{array}{cccc}
x & (xy) & (x(yz)) & ((xy)z) \\
(\lambda x.x) & (\lambda y.(\lambda x.x)) & (\lambda z.(x(\lambda y.(yz)))) & (x(\lambda z.(\lambda y.(yz)))) \\
x(\lambda x.x) & (\lambda y.(\lambda x.x))(\lambda x.x) & (\lambda y.(\lambda x.x))(\lambda x.x)(xy) & (x(yz))((xy)z)
\end{array}
$$

- Note that in lambda calculus we write fx rather than the usual $f(x)$ to represent the application of function f to the argument x.

  Notational conventions:

  i. Omit the outermost parentheses:

$$
\begin{array}{cccc}
x & xy & x(yz) & (xy)z \\
\lambda x.x & \lambda y.(\lambda x.x) & \lambda z.(x(\lambda y.(yz))) & x(\lambda z.(\lambda y.(yz)))
\end{array}
$$

  ii. Parentheses associate to left:

$$fxyz \equiv (((fx)y)z)$$

  iii. Stacked lambdas:

$$
\begin{aligned}
(\lambda f.(\lambda x.(f(fx)))) &\equiv \lambda f\lambda x.(f(fx)) \\
&\equiv \lambda fx.(f(fx))
\end{aligned}
$$

- The notions of bondage, freedom and substitution we covered in predicate logic apply here as well.

2 β-reduction

- An expression of the form $(\lambda\alpha.\gamma)\omega$ is called a β**-redex**.
- It can be β**-reduced** to $\gamma[\omega/\alpha]$, called a **reduct**, if ω is free for α in γ.
- A lambda expression can be reduced by turning all the redexes to reducts, which results in a β**-normal** form.
- Some example reductions:

$$
\begin{array}{rcl}
(\lambda f.fx)g & \rightarrow_\beta & gx \\
(\lambda f.fx)ga & \rightarrow_\beta & gxa \\
(\lambda f.fx)(ga) & \rightarrow_\beta & gax \\
(\lambda f\lambda x.fx)ga & \rightarrow_\beta & ga
\end{array}
$$

- There may be more than one redex in a expression:

$$
\begin{array}{rclcl}
(\lambda x.y)((\lambda z.zz)(\lambda w.w)) & \rightarrow_\beta & (\lambda x.y)((\lambda w.w)(\lambda w.w)) & & \\
& \rightarrow_\beta & (\lambda x.y)(\lambda w.w) & \rightarrow_\beta & y
\end{array}
$$

- Another reduction of the same expression would be:

$$(\lambda x.y)((\lambda z.zz)(\lambda w.w)) \quad \rightarrow_\beta \quad y$$

- The first is called the **applicative order** reduction; the second is called the **normal order** reduction.
- Reduce the following expressions:

$$
\begin{aligned}
(\lambda x.mx)j \\
(\lambda y.yj)m \\
(\lambda x.\lambda y.y(yx))jm \\
(\lambda y.yj)(\lambda x.mx) \\
(\lambda x.xx)(\lambda y.yyy)
\end{aligned}
$$

3 Lambda calculus in action: some examples

3.1 Logic

- Let's define the truth values:

$$
\begin{aligned}
\mathbf{T} &\equiv \lambda x\lambda y.x \\
\mathbf{F} &\equiv \lambda x\lambda y.y
\end{aligned}
$$

- Verify that $\lambda x\lambda y.yxy$ behaves like *and* in prefix notation (i.e. $\wedge pq$).
- Can you think of lambda expressions for $\vee$ and $-$?
- What about an if-then-else function that applies to the test, a function to execute if the test is true and a function to execute if the test is false.

3.2 Arithmetic

- Number can be represented as lambda expressions in the following way:

$$
\begin{aligned}
0 &\equiv \lambda f\lambda x.x \\
1 &\equiv \lambda f\lambda x.fx \\
2 &\equiv \lambda f\lambda x.f(fx) \\
3 &\equiv \lambda f\lambda x.f(f(fx)) \\
&\vdots
\end{aligned}
$$

- A successor function which returns $n+1$ given n is:

$$
\mathbf{S} \equiv \lambda a\lambda f\lambda x.f(afx)
$$

- Here are addition and multiplication (again in prefix notation); verify that they do what they are meant to do.

$$
\begin{aligned}
+ &\equiv \lambda a\lambda b\lambda f\lambda x.af(bfx) \\
\times &\equiv \lambda a\lambda b\lambda f.a(bf)
\end{aligned}
$$