Questions

5.2.1 Define logical `or` and `not` functions

$$\text{or} = \lambda b.\ \lambda c.\ b \text{ tru } c;$$
$$\text{not} = \lambda b.\ \lambda t.\ \lambda f.\ b\ f\ t;$$

Tests:

$$\text{or tru tru} = \text{tru tru tru} = \text{tru}$$
$$\text{or fls tru} = \text{fls tru tru} = \text{tru}$$
$$\text{or tru fls} = \text{tru tru fls} = \text{tru}$$
$$\text{or fls fls} = \text{fls tru fls} = \text{fls}$$
$$\text{not tru} = \lambda t.\ \lambda f.\ \text{tru } f\ t$$
$$= \lambda t.\ \lambda f.\ f$$
$$= \text{fls}$$
$$\text{not fls} = \lambda t.\ \lambda f.\ \text{fls } f\ t$$
$$= \lambda t.\ \lambda f.\ t$$
$$= \text{tru}$$

5.2.4 Define a term for raising one number to the power of another.

$$\text{power} = \lambda m.\ \lambda n.\ m\ (\text{times } n)\ c_0;$$

5.2.7 Write a function `equal` that tests two numbers for equality and returns a Church boolean. For example,

$$\text{equal}\ \ c_3\ c_3;$$
$$> (\lambda t.\ \lambda f.\ t)$$
$$\text{equal}\ \ c_3\ c_2;$$
$$> (\lambda t.\ \lambda f.\ f)$$

Answer:
$$\text{equal} = \lambda c_1.\ \lambda c_2.\ \text{and}\ (\text{iszro}(m\ \text{prd } n))\ (\text{iszro}(n\ \text{prd } m));$$

5.2.8 A list can be represented in the lambda calculus by its `fold` function. (OCaml's name for this function is `fold_left`; it is also sometimes called `reduce`.) For example, the list `[x,y,z]` becomes a function that takes two arguments `c` and `n` and returns `c x (c y (c z n)))`. What would the representation of `nil` be? Write a function `cons` that takes an element `h` and a list (that is, a `fold` function) `t` and returns a similar representation of the list formed by prepending `h` to `t`. Write `isnil` and `head` functions, each taking a list parameter. Finally, write a `tail` function for this representation of lists (this is quite a bit harder and requires a trick analogous to the one used to define `prd` for numbers).

5.2.11 Use `fix` and the encoding of lists from Exercise 5.2.8 to write a function that sums lists of Church numerals

TODO

5.3.8 Exercise 4.2.2 introduced a "big-step" style of evaluation for arithmetic expressions, where the basic evaluation relation is "term `t` evaluates to final result `v`." Show how to formulate the evaluation rules for lambda-terms in the big-step style.

$$\frac{t_1 \Downarrow (\lambda x.\ t_{12}) \quad t_2 \Downarrow v_2}{t_1\ t_2 \Downarrow t_{12}[v_2/x]}$$