
Questions

5.2.1 Define logical `or` and `not` functions

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

Tests:

$$\begin{aligned}\text{or tru tru} &= \text{tru tru tru} = \text{tru}\\ \text{or fls tru} &= \text{fls tru tru} = \text{tru}\\ \text{or tru fls} &= \text{tru tru fls} = \text{tru}\\ \text{or fls fls} &= \text{fls tru fls} = \text{fls}\\ \text{not tru} &= \lambda t.\ \lambda f.\ \text{tru}\ f\ t\\ &= \lambda t.\ \lambda f.\ f\\ &= \text{fls}\\ \text{not fls} &= \lambda t.\ \lambda f.\ \text{fls}\ f\ t\\ &= \lambda t.\ \lambda f.\ t\\ &= \text{tru}\end{aligned}$$

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,

$$\begin{aligned}
&\quad \text{equal}\ \ c_3\ c_3;\\
&> (\lambda t.\ \lambda f.\ t)\\
&\quad \text{equal}\ \ c_3\ c_2;\\
&> (\lambda t.\ \lambda f.\ f)
\end{aligned}$$

Answer:

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

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).

$$\begin{aligned}\texttt{nil} &= \lambda c.\ \lambda n.\ n;\\ \texttt{cons} &= \lambda h.\ \lambda t.\ \lambda c.\ \lambda n.\ (c\ h\ (t\ c\ n));\end{aligned}$$

$$\begin{aligned}\texttt{isnil} &= \lambda l.\ l\ (\lambda h.\ \lambda t.\ \text{fls})\ \text{tru}\\ \texttt{head} &= \lambda l.\ l\ (\lambda h.\ \lambda t.\ h)\ \texttt{nil};\\ \texttt{tail} &= \lambda l.\ \texttt{fst}\ (l\\ &\qquad (\lambda h.\ \lambda t.\ (\texttt{pair}\ (\texttt{snd}\ t)\ (\texttt{cons}\ h\ (\texttt{snd}\ t)))\\ &\qquad (\texttt{pair}\ \texttt{nil}\ \texttt{nil})\\ &\ );\end{aligned}$$

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

$$\begin{aligned}\text{sum\_impl} &= \lambda\text{fct}.\ \lambda l.\\ &\qquad \texttt{if realbool}\ (\texttt{isnil}\ l)\ \texttt{then}\\ &\qquad\quad c_0\\ &\qquad \texttt{else}\\ &\qquad\quad \texttt{plus}\ \text{fct}\ (\texttt{tail}\ l)\\ \text{sum} &= \texttt{fix}\ \text{sum\_impl}\end{aligned}$$

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.

$$\frac{\mathrm{t}_1 \Downarrow (\lambda \mathrm{x}.\ \mathrm{t}_{12}) \quad \mathrm{t}_2 \Downarrow \mathrm{v}_2}{t_1\ t_2 \Downarrow \mathrm{t}_{12}[\mathrm{v}_2/\mathrm{x}]}$$