Exercise 1 (Fixed-point Combinator)

a) In the last tutorial, we came up with an encoding for lists together with the functions nil, cons, null, hd, and tl. Use a fixed-point combinator to compute the length of a list in this encoding.

b) In the last homework, we encoded lists with the fold encoding, i.e. a list \([x, y, z]\) is represented as \(\lambda c\ n. c\ x\ (c\ y\ (c\ z\ n))\). Define a length function for lists in this encoding.

Exercise 2 (\(\beta\)-reduction on de Bruijn Preserves Substitution)

We consider an alternative representation of \(\lambda\)-terms that is due to de Bruijn. In this representation, \(\lambda\)-terms are defined according to the following grammar:

\[ d ::= i \in \mathbb{N}_0 \mid d_1\ d_2 \mid \lambda\ d \]

a) Convert the terms \(\lambda x\ y.\ x\) and \(\lambda x\ y\ z.\ x\ z\ (y\ z)\) into terms according to de Bruijn.

b) Convert the term \(\lambda\ ((\lambda\ (1\ (\lambda\ 1)))\ (\lambda\ (2\ 1)))\) into our usual representation.

c) Define substitution and \(\beta\)-reduction on de Bruijn terms.

d) Now restate Lemma 1.2.5 for de Bruijn terms and prove it:

\[ s \rightarrow_\beta s' \implies s[u/x] \rightarrow_\beta s'[u/x] \]
Homework 3 (Multiplication)
Define multiplication using fix and prove its correctness. You can assume that you are given a predecessor function pred such that:

- \( \text{pred} \ 0 \rightarrow \beta 0 \)
- \( \text{pred} \ (\text{succ} \ n) \rightarrow \beta n \)

Homework 4 (Efficient Substitution on de Bruijn)
We define a new lifting operator \(- \uparrow \-\):

\[
i \uparrow^n_l = \begin{cases} 
i, & \text{if } i < l \\
i + n, & \text{if } i \geq l
\end{cases}
\]

\[
(d_1 \ d_2) \uparrow^n_l = d_1 \uparrow^n_l \ d_2 \uparrow^n_l
\]

\[
(\lambda \ d) \uparrow^n_l = \lambda \ d \uparrow^n_{l+1}
\]

Use \(- \uparrow \-\) to define a more efficient version of substitution for de Bruijn terms that only applies lifting in the case that a variable is actually replaced by a term. Prove that \(t[s/0]\) yields the same result for both, your new version and the version from the tutorial. \textit{Hint:} Find a suitable generalization first.

Homework 5 (Expanding Lets)
We have a language with \texttt{let}-expressions, i.e.:

\[
t ::= v \mid t \mid \texttt{let} \ v = t \ \texttt{in} \ t
\]

Write a program which expands all \texttt{let}-expressions. The \texttt{let}-semantics are:

\[
(\texttt{let} \ v = t_1 \ \texttt{in} \ t_2) = (\lambda v. \ t_2) \ t_1
\]

You can find a Haskell template for this exercise here.