Assignment III (30 pts)

Burak Ekici

Assigned : December the 1st, 13h30
Due : December the 8th, 23h55

1 Pairs

A **pair** of terms in pure (untyped) λ-Calculus (UTLC) is represented by the following λ-term

$$\texttt{pair} := \lambda x.\lambda y.\lambda f.f\ x\ y$$

along with the projections

$$\begin{aligned}\texttt{first} &:= \lambda p.p\ \texttt{true}\\ \texttt{second} &:= \lambda p.p\ \texttt{false}.\end{aligned}$$

2 Lists

Similarly, a **list** of terms in UTLC could be encoded with the constructors stated below.

$$\begin{aligned}\texttt{cons} &:= \lambda x.\lambda l.\texttt{pair false (pair } x\ l\texttt{)}\\ \texttt{nil} &:= \lambda l.l\end{aligned}$$

To obtain the **head** and the **tail** of a given list, one could employ the following λ-terms:

$$\begin{aligned}\texttt{hd} &:= \lambda l.\texttt{first (second } l\texttt{)}\\ \texttt{tl} &:= \lambda l.\texttt{second (second } l\texttt{)}.\end{aligned}$$

In addition, the **nullary check** (checking whether a given list is empty) could be captured by the λ-term

$$\texttt{isNull} := \texttt{first}.$$

One can then develop **list operations** such as

$$\begin{aligned}\texttt{length} &:= \lambda f.\lambda l.\texttt{ite (isNull } l\texttt{) (zero) (addition (one) (}f\ \texttt{(tl } l\texttt{)))}\\ \texttt{append} &:= \lambda f.\lambda l_1.\lambda l_2\texttt{ite (isNull } l_1\texttt{) (}l_2\texttt{) (cons (hd } l_1\texttt{) (}f\ \texttt{(tl } l_1\texttt{) } l_2\texttt{))}\\ \texttt{reverse} &:= \lambda f.\lambda l_1.\lambda l_2.\texttt{ite (isNull } l_1\texttt{) (}l_2\texttt{) (}f\ \texttt{(tl } l_1\texttt{) (cons (hd } l_1\texttt{) (}l_2\texttt{)))}.\end{aligned}$$

thanks to the λ-terms listed above and those imported from the provided modules.

Note.

Note that the term `reverse` is implemented in a tail recursive fashion, therefore the parameter l_2 must be `nil`.

3 Tasks

Implement in Haskell, the following UTLC terms.

1. (**10 pts**) `length :: Term -> Term`
2. (**10 pts**) `append :: Term -> Term -> Term`
3. (**10 pts**) `reverse :: Term -> Term`

Important.

Download the accompanying library `assignment3.zip` from the course DYS page and include your code into the files `Pairs.hs` and `Lists.hs`.

Do not remove the topmost lines that generate modules (e.g., `module Pairs where`), and those importing previously implemented modules. E.g., `import Booleans`, `import Church`, and etc.

4 Sanity Check

To sanity check your implementations, you could execute below commands and compare obtained results with the expected ones:

```
let t1 = (cons one (cons four (cons two nil)))
let t2 = (cons three (cons one nil))
```

Command	Expected Output
`refl_trans_beta t1`	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s z]))]`
	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s [s [s [s z]]]]))]`
	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s [s z]]))]`
	`(λl.l)])])])])`
`refl_trans_beta (Lists.length t1)`	`(λs.(λx.[s [s [s x]]]))`
`refl_trans_beta (Lists.reverse t1)`	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s [s z]]))]`
	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s [s [s [s z]]]]))]`
	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s z]))]`
	`(λl.l)])])])])`

`refl_trans_beta (Lists.append t1 t2)`	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s z]))]`
	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s [s [s z]]]))]`
	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s [s z]]))]`
	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s [s z]]))]`
	`(λf.[[f (λx.(λy.y))](λf.[[f (λs.(λz.[s z]))]`
	`(λl.l)])])])])])])])])

5 Submission Policy

Please only submit `Pairs.hs` and `Lists.hs` files, not the entire library. Also, make sure that your code interprets fine with `ghci`.

Important Notice.

- Collaboration is strictly and positively prohibited; lowers your score to 0 if detected.
- Any submission after 23h55 on December the 8th will NOT be accepted. Please beware and respect the deadline!