Assignment 2

Ethan Poole
LING 185A: Comp. Ling. I
Due: 13 April 2020

Instructions: Download `Recursion.hs` and `Assignment02.hs` from the course website into the same directory on your computer. The **`import`** line near the top of `Assignment02.hs` imports all of the definitions from `Recursion.hs`, which you may then use in your assignment. Please submit your assignment as a modified version of `Assignment02.hs`, making sure not to change the file name.

On this assignment, you may *not* use any of Haskell's predefined list functions, e.g. **`map`**, **`length`**, **`filter`**, etc. or any of their synonyms.

1 *15 points*

Please write the following recursive functions on the `Numb` type:

(1) a. `times :: Numb -> Numb -> Numb` which computes the product of two numbers using the existing `add` function.

Example usage:
`times two three` $\Longrightarrow^*$ `S (S (S (S (S E))))`
`times E three` $\Longrightarrow^*$ `E`

b. `equal :: Numb -> Numb -> Bool` which returns **`True`** if the two numbers given are equal and **`False`** otherwise.

Example usage:
`equal two three` $\Longrightarrow^*$ **`False`**
`equal two two` $\Longrightarrow^*$ **`True`**

c. `bigger :: Numb -> Numb -> Numb` which takes two numbers and returns the larger one.

Example usage:
`bigger two three` $\Longrightarrow^*$ `S (S (S E))`
`bigger three two` $\Longrightarrow^*$ `S (S (S E))`

2 *15 points*

Please write the following recursive functions on lists:

(2) a. `count :: (a -> Bool) -> [a] -> Numb` which returns the number of elements in the given list for which the given function returns **`True`**.

Example usage:

`count (\x -> x > 3) [2,5,8,11,14]` $\Longrightarrow^*$ `S (S (S (S E)))`

`count isOne [one, one, two, three]` $\Longrightarrow^*$ `S (S E)`

`count not [True, False, True, True]` $\Longrightarrow^*$ `S E`

b. `remove :: (a -> Bool) -> [a] -> [a]` such that `remove f l` returns a list which is just like `l` but with those elements removed for which `f` returns **`True`**.

Example usage:

`remove (\x -> x > 3) [2,5,8,11,14]` $\Longrightarrow^*$ `[2]`

`remove isOne [one, one, two, three]` $\Longrightarrow^*$ `[S (S E), S (S (S E))]`

`remove not [True, False, True, True]` $\Longrightarrow^*$ `[True ,True ,True]`

c. `prefix :: Numb -> [a] -> [a]` such that `prefix n list` returns the list containing the first `n` elements of `list`; or if `n` is greater than the length of `list`, returns `list` as is. (*Hint:* The function will need to work recursively on both arguments.)

Example usage:

`prefix one [2,5,8]` $\Longrightarrow^*$ `[2]`

`prefix two [2,5,8]` $\Longrightarrow^*$ `[2,5]`

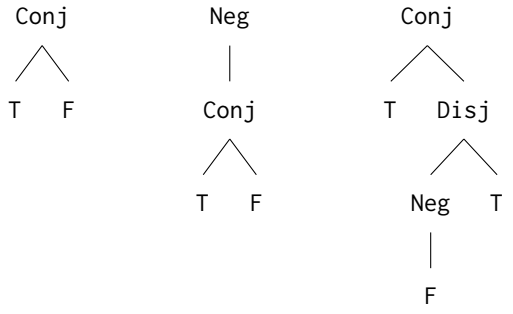
`prefix four [2,5,8]` $\Longrightarrow^*$ `[2,5,8]`

3

10 points

The structure of `WFF` expressions from the previous assignment can be represented as tree structures, as illustrated below:

(3)



Please write a function `depth :: WFF -> Numb` which returns the length of the longest root-to-leaf sequence of nodes in the tree for the given `WFF`, i.e. the depth of the most deeply-embedded leaf of the tree. The `bigger` function that you wrote above will be useful here.

For example, with the `WFF` trees in (3), `depth` should return two, three, and four respectively (as `Numb` expressions).