CSCC24 2023 Summer – Assignment 4

Due: Wednesday, August 9, 11:59PM

This assignment is worth 10% of the course grade.

Question 1: Parametricity

This question is about the parametricity theorem. The starter file and the file to hand in is a4-parametricity.txt.

(a) [5 marks] Prove that $f :: \forall a.[a] \to [a]$ satisfies

$$map\, h\, (f\, xs) = f\, (map\, h\, xs)$$

(for all A_L, A_R, $h :: A_L \to A_R$, $xs :: [A_L]$)

Curiosity remark: Some examples of f are: *reverse*, *take* 5, *drop* 5. So we are saying, for example,

$$map\, h\, (reverse\, xs) = reverse\, (map\, h\, xs)$$

(b) [10 marks] Prove that $e :: \forall b.(Int \to b \to b) \to b \to b$ satisfies

$$foldr\, op_R\, z_R\, (e\, (:)\, [\,]) = e\, op_R\, z_R$$

(for all B_R, $z_R :: B_R$, $op_R :: Int \to B_R \to B_R$)

It is simpler to use a function for the relation for $\langle b \rangle$. The appendix shows how I would redo the lecture example $a \to (a \to a) \to a$ using a function for the relation $\langle a \rangle$.

Curiousity remark: An example of e could be

```
example :: (Int -> b -> b) -> b -> b
example op z = op 3 (op 1 (op 4 z))
i.e.,
example op z = foldr op z [3, 1, 4]
```

so `example` merely performs `foldr` on some encapsulated list of Ints. Note that `example (:) []` exposes the encapsulated list (so `b = [Int]`).

We are proving that every e must be like that, except with a different list inside.

(c) [3 marks] If `f :: a -> [a]` and a test case shows `f () = [(), (), ()]`, then we know `f x = [x, x, x]` in general.

Some programmers who really like Java opine that it is super convenient that *every type* has a `toString()` method. We will see why it sacrifices parametricity. (Therefore, Haskell stands firm and makes you write `Show a => a -> [a]` explicitly.)

Implement in both Java and Haskell

```
<T> LinkedList<T> bad(T x)
bad :: Show a => a -> [a]
```

such that `bad` returns a list of length 3 for some inputs, and the empty list for some other input, by exploiting `toString()` in Java and `show` in Haskell.

Question 2: Toy Interpreter [10 marks]

In this question, you will complete an interpreter for a toy language. The starter file and the file to hand in is Turbo.hs.

Code quality is worth 10% of the marks.

Turbo Graphics

There are some educational programming languages that draw simple pictures. Their model is described to children as: Your program controls a turtle that can move around and draw. It has a current position, a current direction, and a pen in one of two states: down (touches paper) and up (away from paper). (More elaborate versions also have colours and stroke widths.) There are commands to tell the turtle to:

- change the pen state
- turn counterclockwise by a number of degrees (change direction)
- move forward by a distance (whether this draws a line segment or not depends on the pen state)

In this assignment, you are given the abstract syntax tree of such a language—call it Turbo—and you are to implement an interpreter for it. For simplicity, Turbo adds only variables, arithmetic, and for-loops.

The type of the abstract syntax tree is `Stmt`, with an accompanying `RealExpr` for the expressions. The interpreter model is represented by the type `Turbo`, which is a state transition function. Please see TurboDef.hs for the definitions and some descriptions.

The interpreter returns a list of `SVGPathCmd` (explained below) to represent the resulting moving and drawing. Naturally, many statements don't move or draw—so just return the empty list. `Forward` is the only statement kind that causes outputting one command (in a list). For-loops and compound statements (`For` and `Seq`) will have to perform list concatenation.

SVG

The output of the interpreter is a list of commands for the "path" construct in SVG files. The path construct accepts a list of commands, two kinds of which we use are:

- relative move-to $(\delta x, \delta y)$: add $(\delta x, \delta y)$ to the current position, without drawing
- relative line-to $(\delta x, \delta y)$: likewise, but also draws the line segment

This means the web browser that renders the SVG file will keep track of the current position, so we don't have to. But now you have to do polar-to-rectangular conversion: from "distance r, angle a" to $(\delta x, \delta y)$. Keep in mind that while Turbo uses degrees, Haskell's `sin` and `cos` use radians. Also you have to read the pen state to decide whether it's a move-to or line-to.

The type `SVGPathCmd` represents these two kinds of commands.

PlayTurbo.hs is provided to run your interpreter and convert `[SVGPathCmd]` to an SVG file, so you can view it in a web browser. It also contains sample Turbo programs that draw a square, a pentagon, and a spiral, respectively.

End of questions.

Appendix

Prove that $e :: \forall a. a \to (a \to a) \to a$ satisfies

$$\mathit{foldn}\, z_R\, s_R\, (e\, 0\, \mathit{succ}) = e\, z_R\, s_R$$

where *foldn* is this recursive function on the natural numbers:

```
foldn :: a -> (a -> a) -> Natural -> a
foldn z s 0 = z
foldn z s n = s (foldn z s (n-1))
```

For example $\mathit{foldn}\, z\, s\, 2 = s\, (s\, z)$.
The parametricity theorem after expansions:

```
for all AL, AR,
        h :: AL -> AR :
    for all zL :: AL
            sL :: AL -> AL
            zR :: AR
            sR :: AR -> AR :
        if h zL = zR
        and (for all x::AL, y::AR :
                 if h x = y then h (sL x) = sR y
            )
        then h (e zL sL) = e zR sR
```

Choose `AL = Natural`, `zL = 0`, `sL = succ`, `h = foldn zR sR`. Verify these two statements (left as an exercise):

- `h zL = zR`
- `for all x::AL, y::AR : if h x = y then h (sL x) = sR y`

Conclude:

```
for all AR:
    for all zR :: AR
            sR :: AR -> AR :
        foldn zR sR (e 0 succ) = e zR sR
```