Program zaliczeniowy z Haskella (8p)

Wprowadzenie

Dany (w pliku `Reg.hs`) typ danych reprezentujący wyrażenia regularne nad alfabetem c:

```
data Reg c
  = Lit c            -- jeden znak
  | Reg c :> Reg c   -- konkatenacja
  | Reg c :| Reg c   -- alternatywa (suma)

  | Many (Reg c)     -- gwiazdka
  | Eps              -- slowo puste
  | Empty            -- jezyk pusty
  deriving (Eq,Show)
```

Dla danego wyrażenia r, przez L(r) oznaczamy język tego wyrażenia, rozumiany w standardowy sposób (w razie wątpliwości prosze pytać). Mówimy, że r akceptuje słowo w gdy w należy do L(r). Podobnie mówimy, że wyrażenie akceptuje (albo reprezentuje) język. Wyrażenia regularne r_1 i r_2 nazywamy *równoważnymi* jeśli akceptują one ten sam język.

Uzupełnij moduł RegExtra (którego szkielet dany jest w pliku RegExtra0.hs) o definicje omówione poniżej (ewentualnie zastepując występujące w nim zaślepki).

Equiv

Zdefiniuj instancje klasy Equiv dla Reg:

```
infix 4 ===
class Equiv a where
  (===) :: a -> a -> Bool

instance (Eq c) => Equiv (Reg c) where
```

tak aby relacja `===` była relacją zwrotną, symetryczną i przechodnią a ponadto spełniony był warunek

```
equivCompatible c d = (Lit c) === (Lit d) ==> c == d
```

Intencją relacji `(===)` jest równoważność (tu wyrażeń regularnych). Wystarczy jednak zdefiniować relację mniej dokładną, byle tylko spełnione były warunki podane w pliku `TestReg.hs`. Można nawet zacząć od przyjęcia

```
(===) = (==)
```

a potem doszlifować ją tak, by spełniała podane warunki.

Monoid

Dana (Mon.hs) klasa reprezentującą monoidy

```
class Mon m where
  m1 :: m
  (<>) :: m -> m -> m
```

Uzupełnij instancję `instance Mon (Reg c)` tak, aby dla dowolnych `x y z` spełnione były własności

```
leftUnit x = m1 <> x === x
rightUnit x =  x <> m1 === x
assoc x y z = (x<>y)<>z === x<>(y<>z)
```

(mówimy że np. własność `assoc` jest spełniona jesli dla każdych `x y z` odpowiedniego typu `assoc x y z` daje wartość `True`)

Uwaga: udostępniamy program `TestReg.hs`, który po skompilowaniu i uruchomieniu testuje wymagane własności. Wymaga on zainstalowania pakietu `QuickCheck`.

Słowo puste i język pusty

Napisz funkcje

```
nullable, empty :: Reg c -> Bool
```

takie, że

- `nullable r == True` gdy słowo puste należy do języka
- `empty r == True` gdy język jest pusty

Testy:

```
nullableUnit = nullable m1
nullableOp x y = nullable x && nullable y ==> nullable (x <> y)
```

Upraszczanie

Dla danego wyrażenia regularnego nierzadko możemy podać prostsze wyrażenie równoważne, np `Eps :> (Lit 0 :| Empty)` jest równoważne `Lit 0`

Napisz funkcję

```
simpl :: Eq c => Reg c -> Reg c
```

Testy:

```
nullableSimpl x = nullable x `iff` nullable (simpl x)
emptySimpl x = empty x `iff` empty (simpl x)
```

dającą wyrażenie równoważne argumentowi a prostsze (w jakimś sensie). Niektóre potrzebne uproszczenia ujawnią się w późniejszych etapach

Pochodne

Pochodną języka L względem c jest język zawierający słowa w takie, że cw należy do L. Pochodna języka regularnego jest zawsze językiem regularnym.

Napisz funkcje

```
der :: Eq c => c -> Reg c -> Reg c
ders :: Eq c => [c] -> Reg c -> Reg c
```

dające pochodną wyrażenia regularnego względem (odpowiednio) jednego znaku i ciągu znaków.

Uwaga: nie od rzeczy może być tu wykorzystanie funkcji `simpl`. Jaki rozmiar ma `ders (replicate 1000 A) (Many (Lit A) :> Lit B)` ?

Dopasowania

Łatwo zauważyć, że słowo należy do języka wtw gdy pochodna języka względem tego słowa zawiera słowo puste. Wykorzystując ten fakt i funkcje opisane powyzej, napisz funkcje

```
accepts :: Eq c => Reg c -> [c] -> Bool
mayStart :: Eq c => c -> Reg c -> Bool
match :: Eq c => Reg c -> [c] -> Maybe [c]
search :: Eq c => Reg c -> [c] -> Maybe [c]
findall :: Eq c => Reg c -> [c] -> [[c]]
```

takie, że

- `accepts r w` daje `True` gdy w należy do $L(r)$.
- `mayStart c r` daje `True` gdy $L(r)$ zawiera słowo zaczynające się od c

- `match r w` daje `Just p`, gdzie p to najdłuższy prefiks w należacy do $L(r)$, `Nothing` gdy nie ma takiego.
- `search r w` daje `Just u` gdzie u to pierwsze (najdłuższe) podsłowo w akceptowane przez r, `Nothing` gdy nie ma takiego.
- `findall r w` daje listę wszystkich (lokalnie najdłuższych) podsłów w pasujących do r.

Oddawanie i ocena rozwiązań

Rozwiązania mają być samodzielne. Wszelkie zapożyczenia z internetu itp. należy wyraźnie zaznaczyć.

Należy oddać *wyłącznie* plik RegExtra.hs. Nie może on importować nic ponadto co jest już importowane w RegExtra0.hs

Rozwiązania będą oceniane pod kątem:

- spełnienia warunków zadania; rozwiązania nie przechodzące testów będą nisko oceniane, nawet na 0p; rozwiązania będą też poddawane dodatkowym testom
- właściwego wykorzystania mechanizmów paradygmatu funkcyjnego i języka Haskell, tudzież czytelności i stylu.
- rozwiązania skrajnie nieefektywne będą karane; przy porządnym rozwiązaniu testy przechodzą w ok 1s. Rozwiązanie, gdzie będzie to trwało ponad 2minuty uznamy za nieefektywne.