Kurs języka Haskell 2024/25

LISTA NR 6 (TERMIN: 02.12.2024, godz 5:00)

Uwaga: Wszystkie rozwiązania należy umieścić w jednym module o nazwie `Lista6` zachowując sygnatury zgodne z szablonem rozwiązań zamieszczonym w SKOS-ie.

Zadanie 1. Algorytm Euklidesa obliczający największy wspólny dzielnik dwóch dodatnich liczb naturalnych można zdefiniować przy użyciu następującego pseudokodu:

```
function gcd(a, b)
    while a <> b
        if a > b
            a := a - b
        else
            b := b - a
    return a
```

Łatwo zauważyć, że potrzebuje on dwóch komórek pamięci. Zdefiniuj wersję tego algorytmu w monadzie State:

```
gcdSub :: State (Int, Int) Int
```

Wskazówka: Pętla while to to samo, co funkcja ogonowa – nawet, gdy programujemy z monadami.

Zadanie 2. Rozważmy znaną optymalizcję algorytmu Euklidesa, w której odejmowanie można zastąpić dzieleniem:

```
function gcd(a, b)
    do
        t := b
        b := a mod b
        a := t
    while b <> 0
    return a
```

Zdefiniuj funkcję

```
doWhile :: (Monad m) ⇒ m Bool → m a → m a
```

która implementuje konstrukcję do-while dla dowolnej monady. Użyj jej, by zaimplementować powyższą wersję algorytmu Euklidesa:

```
gcdMod :: State (Int, Int) Int
```

Uwaga: Powyższy pseudokod używa trzech komórek pamięci, a stan naszego algorytmu to wciąż (**Int**, **Int**). Proszę pamiętać, że programujemy w języku deklaratywnym i nie musimy się obawiać, że wartości nam „znikną".

Zadanie 3. (2 pkt) Rozważ następującą reprezentację języka WHILE składającego się z wyrażeń arytmetycznych, wyrażeń boolowskich i instrukcji:

```
type Ident = String
type Op = String

data A = AOp Op A A
       | AConst Int
       | AVar Ident

data B = BCmp Op A A
       | BOp Op
       | BConst Bool
       | BVar Ident

data C = Assign Ident A
       | If B C C
       | While B C
       | Skip
       | Cmp C C
```

Zdefiniuj interpreter tego języka. W szczególności, zadbaj o reprezentację pamięci:

```
type Memory = ...
```

oraz funkcje obliczające elementy języka:

```
evalA :: A → State Memory Int
evalB :: B → State Memory Bool
evalC :: C → State Memory ()
```

Zdefiniuj funkcję uruchamiającą program:

```
run :: C → Memory → Int
```

która zwraca wartość znajdującą się po zakończeniu programu w komórce pamięci result.

Zareprezentuj w języku WHILE wybraną postać algorytmu Euklidesa.

Zadanie 4. Użyj monady listowej do implementacji funkcji rozwiązującej problem n hetmanów:

```
type Solution = [Int]
queens :: Int → [Solution]
```

Na przykład:

```
ghci> queens 3
[]
ghci> queens 4
[[3,1,4,2],[2,4,1,3]]
```

Ile jest możliwych rozwiązań dla szachownicy o rozmiarze $n \times x$ dla $n = 13, 14, 15$? (Wskazówka: Żeby to zmierzyć, dobrze najpierw skompilować program).

Wskazówka: Zdefiniuj pomocniczą funkcję, która sprawdza poprawność rozszerzenia częściowego rozwiązania:

```
extends :: Int → Solution → Bool
```

Następnie użyj funkcji **guard** z modułu Control.**Monad**, żeby w głównej funkcji rozszerzyć rozwiązanie tylko o te pozycje, które spełniają extends.

Zadanie 5. O obliczeniach w monadzie listowej można myśleć jako o obliczeniach niedeterministycznych, które zwracają pewną liczbę możliwych wartości. Wówczas o obliczeniu, które zwraca listę pustą myślimy, że to obliczenie, które się nie powiodło. Porażka ta jest jednak lokalna, np. wyrażenie

```
do
  x ← [1..5]
  y ← if even x then [] else "ab"
  return (x,y)
```

oblicza się do wartości

```
[(1,' a '),(1,' b '),(3,' a '),(3,' b '),(5,' a '),(5,' b ')]
```

Zdefiniuj alternatywną instancję monady dla typu danych listy, w której porażka jest *globalna*: jeśli którekolwiek z obliczeń składowych zwróci listę pustą, całe obliczenie w monadzie daje listę pustą. Czy równości, których wymagamy dla instancji klasy **Monad** są zachowane?

Zadanie 6. Zdefiniuj monadę (niekoniecznie na listach), która implementuje niedetrminizm w stylu monady listowej, ale zawierającą zrówno porażkę *lokalną*, jak i *globalną*.