

# Programowanie Funkcyjne 2020

Lista zadań nr 10 dla grup mabi, mbu, ppo i efes

Na zajęcia 5 i 6 stycznia 2021

**Zadanie 1 (3 pkt).** Chcemy zdefiniować funkcję `sprintf` znaną z języka C, tak by np.

```
sprintf "Ala ma %d kot%s" :: Integer -> String -> String
```

pozwalą zdefiniować funkcję

```
\ n -> sprintf "Ala ma %d kot%s." n
  (if n == 1 then "a" else if 1 < n && n < 5 then "y" else "ów").
```

Na pierwszy rzut oka wydaje się, że rozwiązanie tego zadania wymaga typów zależnych, ponieważ typ funkcji `sprintf` zależy od wartości pierwszego argumentu. Okazuje się jednak, że polimorfizm parametryczny wystarczy. Dla uproszczenia założmy, że format nie jest zadany przez wartość typu `String`, ale przez konkatenację następujących dyrektyw formatujących:

- `lit s` — stała napisowa `s`,
- `int` — liczba typu `Integer`,
- `str` — napis typu `String`.

Zakładając, że operatorem konkatenacji dyrektyw jest `^^`, powyższy przykład może być zapisany następująco.

```
sprintf (lit "Ala ma " ^^ int ^^ lit " kot" ^^ str ^^ lit ".")
```

Zdefiniuj funkcje `lit`, `int`, `str`, (`^^`) oraz `sprintf`. Nie należy używać klas typów!

**Wskazówka:** Dyrektywy powinny być funkcjami transformującymi kontynuacje, a operator `^^` zwykłym złożeniem takich funkcji. Na przykład `int` powinien mieć typ `(String -> a) -> String -> Integer -> a` (argumentem jest kontynuacja oczekująca napisu, ale o nieokreślonym typie odpowiedzi, natomiast wynikiem, ma być kontynuacja oczekująca napisu i produkująca funkcję która oczekuje na liczbę).

**Zadanie 2 (3 pkt).** Dyrektywy formatujące z poprzedniego zadania można opisać następującym typem.

```
type Format a b = (String -> a) -> String -> b
```

Natomiast sama funkcja `sprintf` mogła (ale nie musiała) być zdefiniowana przy pomocy innej funkcji `ksprintf`, która jawnie przyjmuje swoją kontynuację. Funkcja `ksprintf` może mieć sygnaturę:

```
ksprintf :: Format a b -> (String -> a) -> b
```

W tym zadaniu postaramy się oddzielić składnię od semantyki dyrektyw formatujących. W tym celu zdefiniuj typ danych `Format a b`, który ma cztery konstruktory: `Lit`, `Int`, `Str` oraz `(:~:)` odpowiadające podstawowym dyrektywom formatującym oraz operacji konkatenacji. Oczywiście, żeby typy tych konstruktorów były odpowiednie, należy użyć GADT. Następnie zaimplementuj interpreter dyrektyw formatujących w postaci funkcji `ksprintf` o podanej wyżej sygnaturze i użyj jej do napisania funkcji `sprintf`. Takie podejście ma sporo zalet: teraz dyrektywy formatujące są zwykłymi danymi, które można przetwarzać — w szczególności możemy dostarczyć alternatywny interpreter. Zdefiniuj funkcję `kprintf`, o sygnaturze

```
kprintf :: Format a b -> (IO () -> a) -> b
```

która od razu drukuje wynik na standardowe wyjście bez wcześniejszego konstruowania całego napisu. Użyj jej do zaimplementowania funkcji `printf`.

**Zadanie 3 (1 pkt).** Zdefiniuj funkcję `echoLower :: IO ()` która będzie wczytywać znaki ze standardowego wejścia, zamieniać wielkie litery na małe i wypisywać z powrotem na standardowe wyjście, aż do napotkania końca strumienia wejściowego.

**Zadanie 4 (3 pkt).** Funkcja z poprzedniego zadania jest przykładem transformatora strumieni: transformuje jeden strumień (standardowe wejście) na drugi (standardowe wyjście). Znów oddzielimy składnię od semantyki i zareprezentujemy transformatory strumieni za pomocą następującego algebraicznego typu danych.

```
data StreamTrans i o a
  = Return a
  | ReadS (Maybe i -> StreamTrans i o a)
  | WriteS o (StreamTrans i o a)
```

Typ ten reprezentuje całe obliczenia z użyciem dwóch strumieni: wejściowego i wyjściowego. Wartość `WriteS x c` oznacza wypisanie wartości `x` do strumienia wyjściowego i kontynuowanie obliczenia `c`. Wartość `ReadS f` oznacza próbę czytania ze strumienia wejściowego, gdzie funkcja `f` jest kontynuacją dla wyniku czytania. Jeśli zostanie przeczytana wartość `x`, to dalej zostanie wykonane obliczenie `f (Just x)`. Przeczytanie `Nothing` oznacza koniec strumienia wejściowego. W końcu `Return x` oznacza zakończenie obliczenia z wartością `x`.

Zdefiniuj transformator strumieni `toLower :: StreamTrans Char Char a` podobny to tego z poprzedniego zadania. Następnie zdefiniuj funkcję

```
runIOStreamTrans :: StreamTrans Char Char a -> IO a
```

która pozwoli na przetestowanie przykładowego transformatora strumieni.

**Zadanie 5 (1 pkt).** Zdefiniuj funkcję `listTrans :: StreamTrans i o a -> [i] -> ([o], a)` pozwalającą transformować strumienie reprezentowane jako listy. Twoja implementacja powinna się dobrze zachowywać również dla list nieskończonych, np.

```
take 3 $ fst $ listTrans toLower ['A'..]
```

powinno zwrócić napis "abc".

**Zadanie 6 (1 pkt).** Zdefiniuj funkcję `runCycle :: StreamTrans a a b -> b`, która pozwala uruchomić transformator strumieni poprzez skierowanie wyjścia na jego wejście.

**Zadanie 7 (1 pkt).** Zdefiniuj operator

```
(|>|) :: StreamTrans i m a -> StreamTrans m o b -> StreamTrans i o b
```

pozwalający na przekierowanie wyjścia jednego transformatora na wejście drugiego.

**Zadanie 8 (1 pkt).** Napisz funkcję

```
catchOutput :: StreamTrans i o a -> StreamTrans i b (a, [o])
```

która przekształca podany transformator strumieni w taki, który nic nie wypisuje na wyjściu, tylko zbiera wypisywane elementy na listę.

**Zadanie 9 (2 pkt).** *Brainfuck* jest ekstremalnie prostym, ezoterycznym językiem programowania, którego składnię abstrakcyjną można przedstawić następującym typem danych.

```
data BF
  = MoveR      -- >
  | MoveL      -- <
  | Inc        -- +
  | Dec        -- -
  | Output     -- .
  | Input      -- ,
  | While [BF] -- [ ]
```

Składnia konkretna jest równie prosta: każda z sześciu instrukcji prostych jest reprezentowana pojedynczym znakiem (>,<,+,-,.,, — tak jak podano w komentarzu), natomiast jedyna złożona instrukcja `While` jest reprezentowana poprzez parę nawiasów kwadratowych, pomiędzy którymi znajduje się dowolny ciąg instrukcji (prostych i złożonych). Wszystkie pozostałe znaki są ignorowane (można z nich budować komentarze). Napisz transformator strumieni

```
brainfuckParser :: StreamTrans Char BF ()
```

który zamienia reprezentację konkretną programu w języku *Brainfuck* na ciąg instrukcji w składni abstrakcyjnej.

**Wskazówka:** Może (ale nie musi) okazać się przydatne rozwiązanie poprzedniego zadania oraz zadania 11.

**Zadanie 10 (2 pkt).** *Brainfuck* ma nie tylko prostą składnię, ale również prostą semantykę. Programy operują na nieskończonej taśmie liczb<sup>1</sup>, początkowo wypełnionej zerami. Na taśmie znajduje się kursor, który można przesuwać instrukcjami `MoveR` oraz `MoveL` (o jedną pozycję odpowiednio w lewo i w prawo). Instrukcje `Inc` oraz `Dec` odpowiednio zwiększają i zmniejszają liczbę na pozycji wskazywanej przez kursor. Instrukcja `Output` wypisuje na wyjście znak, o kodzie ASCII znajdującym się na pozycji wskazywanej przez kursor, natomiast `Input` wczytuje znak i umieszcza pod kursorem na taśmie. W końcu instrukcja `While` jest podobna do znanej wam pętli *while*: wykonuje podany ciąg instrukcji tak długo, jak pod kursorem znajduje się niezerowa liczba. Napisz funkcję

```
runBF :: [BF] -> StreamTrans Char Char ()
```

interpretującą program w języku *Brainfuck*. Wygodnie będzie napisać pomocniczą funkcję, która jawnie przekazuje stan taśmy reprezentowanej jako dwie listy: przed i za kursorem.

```
type Tape = ([Integer], [Integer])
evalBF :: Tape -> BF -> StreamTrans Char Char Tape

evalBFBlcok :: Tape -> [BF] -> StreamTrans Char Char Tape
evalBFBlcok = foldM evalBF -- jeśli rozwiązałeś zadanie 11
```

Do konwersji pomiędzy typami `Char` oraz `Integer` może się przydać następująca funkcja.

```
coerceEnum :: (Enum a, Enum b) => a -> b
coerceEnum = toEnum . fromEnum
```

**Wskazówka:** Może (ale nie musi) okazać się przydatne rozwiązanie zadania 11.

**Zadanie 11 (2 pkt).** Odkryj strukturę monady w typie `StreamTrans` i o a i zainstaluj go w klasie `Monad`. Implementacja funkcji `(>>=)` powinna sekwencjonować obliczenia: `m >>= f` wykonuje najpierw wszystkie akcje z `m`, a dopiero na koniec wywołuje `f`.

---

<sup>1</sup>Oryginalnie, *Brainfuck* operował na bajtach, ale na potrzeby tego zadania możemy przyjąć, że na taśmie znajdują się liczby typu `Integer`.