Laboratorul 8

Analiza sintactică

În acest laborator vom utiliza analizaoarele sintactice definite în curs.

Ne amintim că am definit un parser generic prin:

```
newtype Parser a =  Parser { apply :: String -> [(a, String)] }
```

și funcția

```
parse :: Parser a -> String -> a
parse m s = head [ x | (x,t) <- apply m s, t == ""]
```

Fie `anychar` un parser simplu care recunoaște un caracter arbitrar:

```
anychar :: Parser Char
anychar = Parser f
   where
   f []     = []
   f (c:s) = [(c,s)]
```

Sesizați diferența dintre funcțiile `apply` și `parse`.

```
apply anychar "a123"
parse anychar "a123"
```

Exersați funcțiile `anychar`, `satisfy`, `char`, `string` definite la curs.

Exercițiul 1: parserul `three`

Definiți parserul `three` care citește primele trei caractere din șirul de la intrare, și întoarce perechea formată din primul și al treilea (dacă șirul de intrare are mai putin de 3 caractere parserul întoarce lista vidă.

Ce tip are `three`?

Scrieți două definiții: una directă și una care folosește `anychar`.

Exercițiul 2: Monada Parser

Amintiți-vă definiția monadei Parser.

Scrieți parserele `string` și `three` folosind notațiile monadice.

Exercițiul 3: Functor Parser

La curs am definit

```
instance Functor Parser where
  fmap f ma = pure f <*> ma
```

Dați o definție directă a funcției `fmap`.

Scrieți un parser `anycharord` care consumă un caracter si returnează codul caracterului consumat.

Scrieți două definiții: una directă și una care folosește `fmap`.

Exercițiul 4: Combinarea alternativelor

Amintiți-vă că `Parser` este instanță a clasei `MonadPlus` și că putem combina alternativele folosind `<|>`.

Scrieți două parsere simple si combinați-le folosind `<|>`.

Exercițiul 5: `howmany c`

Scrieți un parser

```
howmany :: Char -> Parser Int
```

care consuma prefixele nenule formate numai din caracterul dat ca argument și întoarce lungimea acestora:

```
apply (howmany 'a') "aaaaaal"
[(6,"l"),(5,"al"),(4,"aal"),(3,"aaal"),(2,"aaaal"),(1,"aaaaal")]
```

Exercițiul 6: `fmap` și `(<*>)`

Analizați tipul următoarelor funcții/expresii:

```
:t fmap
:t fmap (+)
:t (<*>)
:t fmap (+) (Just 2) <*> (Just 3)
:t fmap (+) ([1,2]) <*> ([4,5,6])
:t fmap (+) anycharord
```

Definiți un parser care consuma doua caractere si întoarce suma codurilor caracterelor consumate. Pentru șiruri cu mai puțin de două caractere, întoarce lista vidă.

Exercițiul 7: numere cu 4 cifre

Fiind dată funcția

```
no :: Int -> Int -> Int -> Int -> Int
no x y z v = x*1000+y*100+z*10 + v
```

definiți un parser care consumă un prefix format din 4 cifre și întoarce numărul asociat. Parserul întoarce lista vidă dacă șirul de intrarenu începe cu 4 cifre.

> apply fourdigit “123” []
>
> apply fourdigit “123456” [(1234,“56”)]