Paradigme de Programare
Tema 2 – Haskell

Deadline:

- 26.04, 23:55 (soft)
- 07.05, 23:55 (hard)

Responsabil temă:

- Dinu Adrian

Change log:

- 06.04 publicare
- amanat deadline soft cu 3 zile

1. Introducere

Se dă limbajul IMP++ definit de următoarea gramatică:

```
<symbol> :: [a-zA-Z]+
<value>  :: [1-9][0-9]* | 0
<op>     :: + | - | * | == | <
<expr>   :: <expr> <op> <expr> | <symbol> | <value>
<assign> :: <symbol> = <expr>
<prog>   :: <assign>;
          | <prog> <prog>
          | if (<expr>) then {<prog>} else {<prog>}
          | for (<assign>; <expr>; <assign>) {<prog>}
          | assert <expr>;
          | return <expr>;
```

Între token-urile unui program pot exista sau nu whitespace-uri (space, tab, newline). Excepții sunt **assert** și **return** după care trebuie să existe cel puțin un whitespace. Astfel, următoarele 2 programe sunt echivalente:

```
s=0;n=10;for(i=0;i<n;i=i+1){s=s+i;}return s;
```

```
s = 0;
n = 10;

for (i = 0; i < n; i = i + 1)
{
    s = s + i;
}

return s;
```

Programul de mai sus va întoarce valoarea 45.

2. Cerințe

Să se implementeze în Haskell un interpretor pentru limbajul IMP++.

Interpretorul constă din 2 părți principale:

- o funcție de parsare, care primește ca input un string și returnează un TDA, reprezentând un program IMP++, dacă string-ul respectă gramatica precizată.
- o funcție de evaluare, care primește ca input un TDA reprezentând un program IMP++ și returnează fie rezultatul acestuia, fie un string ce descrie o eroare.

2.1 Input

Există două tipuri de input: sub formă de TDA și sub formă neprelucrată.

Astfel, puteți opta să nu implementați parser-ul, caz în care veți primi doar punctajul pentru testele cu TDA-uri.

Următoarele 2 exemple sunt reprezentările unui program sub formă de TDA și sub formă neprelucrată, respectiv:

```
Seq (Eq "x" (Value 2)) (Seq (Eq "y" (Value 2)) (Seq (Eq
"z" (Value 4)) (Seq (Return (Equal (Add (Symbol "x")
(Symbol "y")) (Symbol "z"))) (Return (Value 0)))))
```

```
x = 2;
y = 2;
z = 4;
assert x + y == z;
return 0;
```

3. Tratarea erorilor

Interpretorul trebuie să detecteze erorile și să producă un mesaj descriptiv pentru acestea.

3.1. Erori sintactice

Erorile sintactice sunt cele care țin de parsare. Pentru orice string care nu respectă gramatica IMP++, interpretorul trebuie să întoarcă mesajul "*Syntax error*".

Exemplu: (sintaxa pentru **if** este greșită)

```
a = 1;
if (a < 2) return 1;
else return 0;
```

3.2. Erori semantice

Erorile semantice sunt cele care pot apărea în timpul evaluării. Ele sunt de 3 tipuri și interpretorul trebuie să întoarcă un mesaj pentru prima eroare pe care o întâlnește:

1. Folosirea unei variabile neinițializate => "*Uninitialized variable*"
Observație: după ce o variabilă este inițializată, scope-ul ei este global
Exemplu:

```
x = 0;
return y;
```

2. Atunci când o instrucțiune **assert** eșuează => "*Assert failed*"
Exemplu:

```
x = 0;
assert x == 1;
return 0;
```

3. Lipsa unei instrucțiuni de **return** la sfârșitul evaluării programului => "*Missing return*"
Exemplu:

```
x = 0;
assert x == 0;
```

Deoarece IMP++ este un limbaj interpretat, ne interesează doar ramura logică aleasă în timpul evaluării programului. Astfel, chiar dacă unele ramuri prezintă potențiale erori semantice, interpretorul nu trebuie să returneze mesaj de eroare pentru ele decât dacă intră pe ramurile respective.

Exemplu: programul întoarce 1337 chiar dacă are posibile erori pe unele ramuri

```
x = 7;
if (x == 0) then {
   x = y + 5;
} else {
   x = x + 30;
}
if (x == 37) then {
   x = x + 300;
   return x + 1000;
} else {
   assert x == 42;
}
```

4. Scheletul de cod

Vă este pus la dispoziție un schelet minimal de cod, sub forma fișierului **Interpreter.hs**, în care sunt deja definite TDA-urile **Expr**, **Asgn** și **Prog**, reprezentând expresii, atribuiri și programe.

Tot în schelet se găsesc și declarațiile celor două funcții pe care checker-ul le folosește:

```
evalAdt :: Prog -> Either String Int
evalRaw :: String -> Either String Int
```

Funcția **evalRaw** este deja definită ca fiind o compunere dintre **evalAdt** și **parse**:

```
parse :: String -> Maybe Prog
```

Astfel, tot ceea ce trebuie să faceți este să implementați funcțiile **evalAdt** și **parse**, acestea fiind cele 2 componente principale de care vorbeam la capitolul **Cerințe**. Este recomandat să începeți prin implementarea funcției **evalAdt** deoarece **evalRaw** depinde de ea.

Puteți modifica scheletul, cu excepția TDA-urilor **Expr**, **Asgn** și **Prog** și a declarațiilor funcțiilor **evalAdt** și **evalRaw**. Puteți de asemenea să adăugați orice alte module, funcții sau TDA-uri de care aveți nevoie (**Hoogle** vă stă la dispoziție).

5. Resurse

În arhiva **2-haskell.zip** se găsesc:

- **Interpreter.hs**, scheletul de cod
- **Checker.hs**, folosit pentru testare
- folderul **tests**, care conține testele publice (în format neprelucrat și în format TDA), precum și rezultatele de referință

5.1. Checker-ul

Pentru a vă testa tema, puteți rula **Checker.hs** cu unul din următorii parametrii:

- **raw**, rulează doar testele cu format neprelucrat
- **adt**, rulează doar testele cu TDA-uri
- **both**, rulează ambele categorii de teste

Checker-ul are nevoie de modulul **Data.List.Utils**, care se găsește în pachetul **MissingH**. Puteți să-l instalați folosind **cabal**, care vine instalat odată cu **platforma Haskell**:

```
student@laptop$ cabal update
student@laptop$ cabal install MissingH
```

Dacă rezolvați tema pe Windows, puteți să urmați aceiași pași în **cmd**.
Pentru a rula checker-ul, puteți folosi **runghc** sau **runhaskell**:

```
student@laptop$ runghc Checker.hs both
student@laptop$ runhaskell Checker.hs adt
```

6. Trimitere

Tema trebuie trimisă sub forma unei arhive zip, care să conțină:

- **Interpreter.hs**
- **README**
- orice alt fișier sursă Haskell de care aveți nevoie

7. Punctaj

Tema valorează în total **1.5 puncte** din nota finală:

- 1.0 puncte pentru evaluarea programelor sub formă de TDA
- 0.5 puncte pentru parser

Testarea se va face automat. Pe lângă testele incluse în arhivă va exista și un set de teste private.