Paradigme de Programare
Tema 2 - Haskell

Termen de predare: **25.04.2016** (soft)
02.05.2016 (hard)

Responsabili temă: Călin Cruceru
Mihai Dumitru

Data publicării: *04.04.2016*
Ultima actualizare: *04.04.2016*

1 Introducere

Se dă un limbaj de programare, numit "IMP", definit de următoarea gramatică.

$\langle expr \rangle$ ::= $\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$ | $\langle symbol \rangle$ | $\langle value \rangle$

$\langle symbol \rangle$ ::= [a-zA-Z]+

$\langle value \rangle$ ::= [1-9][0-9]* | 0

$\langle op \rangle$ ::= + | - | * | == | <

$\langle prog \rangle$ ::= $\langle symbol \rangle$ = $\langle expr \rangle$;
| $\langle prog \rangle$ $\langle prog \rangle$
| if ($\langle expr \rangle$) then {$\langle prog \rangle$} else {$\langle prog \rangle$}
| while ($\langle expr \rangle$) {$\langle prog \rangle$}
| return $\langle expr \rangle$;

Un program valid se termină mereu cu o instrucțiune de tip "`return <expr>;`".

Între cuvântul-cheie "`return`" și expresia care urmează trebuie să fie cel puțin un caracter de spațiere (space, tab, newline). În rest, aceste caractere pot fi ignorate.

Astfel, următoarele două programe se comportă identic:

```
x=4;if(x<4)then{return x;}else{return x+1;}
```

```
x = 4;

if (x < 4) then {
  return x;
} else {
  return x + 1;
}
```

2 Cerințe

Se cere să implementați un interpretor pentru limbajul "IMP", în limbajul Haskell.

Interpretorul constă, în principal, din două părți:

- o funcție de parsare, care primește ca input un șir de caractere și returnează un TDA, reprezentând un program IMP, dacă șirul este valid conform gramaticii precizate.
- o funcție de evaluare, care primește un TDA ce descrie un program IMP și returnează fie valoarea produsă de acesta, fie un șir de caractere care descrie o eroare în program.

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 aferent testelor cu TDA-uri.

Următoarele două exemple sunt reprezentări ale aceluiași program:

```
Seq (Eq "x" (IVal 123)) (Seq (Eq "y" (IVal 456)) (Seq (If (Smaller
(Var "x") (Var "y")) (Return (Var "x")) (Return (Var "y")))))
```

```
x = 123;
y = 456;

if (x < y) then {
  return x;
} else {
  return y;
}
```

3 Tratarea erorilor

Interpretorul trebuie să detecteze erori și să producă un șir descriptiv pentru acestea.

3.1 Erori sintactice

Erorile sintactice țin de parser. Pentru orice șir care nu respectă gramatica IMP, trebuie întors mesajul `"Syntax error"`.

Exemplu:

```
x <- 0;
if x == y
  return x;
return y
```

3.2 Erori semantice

Va trebui să tratați următoarele erori semantice:

1. Folosirea unei varaibile neinițializate, caz în care se va întoarce șirul `"Uninitialized variable"`.

Exemplu:

```
x = 0;

if (x < y) then {
  return x;
} else {
  return y;
}
```

`y` e neinițializat în expresiile `"x < y"` și `"return y;"`.

2. Lipsa unei instrucțiuni de return la sfârșitul firului de execuție, caz în care se va întoarce șirul `"Missing return"`.

Exemplu:

```
x = 0;
y = 1;

if (x == y) then {
  return x;
} else {
  x = y;
}
```

Atenție! Pentru că limbajul nostru este interpretat, ne interesează doar ramura logică aleasă în timpul evaluării propriu-zise. Astfel, evaluarea următorului program va returna valoarea 1, nu un șir descriind o eroare, chiar dacă ramura `else` prezintă ambele erori semantice:

```
if (0 < 1) then {
  return 1;
} else {
  x = 2;
}
```

Dacă schimbăm condiția, atunci evaluarea va întoarce șirul `"Uninitialized variable"`, adică **prima eroare pe care o întâlnește:**

```
if (0 == 1) then {
  return 1;
} else {
  x = 2;
}
```

4 Schelet de cod

Vi se pune la dispoziție un schelet minimal de cod, sub forma fișierului "Interpreter.hs", în care sunt definite TDA-urile `Expr` și `Prog`, reprezentând expresii, respectiv programe.

Tot acolo sunt declarațiile celor două funcții pe care checker-ul le folosește și pe care trebuie să le definiți:

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

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

Funcția `evalRaw` este deja definită ca fiind o compunere dintre `evalAdt` și `parse`, definită astfel:

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

(Puteți să modificați lucruri, cu excepția TDA-urilor `Expr` și `Prog` și a declarațiilor funcțiilor `evalAdt` și `evalRaw`.)

În afara celor menționate mai sus, puteți adăuga orice alte module, TDA-uri sau funcții de care aveți nevoie.

5 Resurse

În arhiva `"2-haskell.zip"`, se găsesc:

- fișierul "Interpreter.hs" menționat mai sus
- fișierul "Checker.hs" folosit pentru testare
- folderul "tests", care conține testele pulice (în format neprelucrat și în format TDA), precum și rezultatul de referință

5.1 Checker

Pentru a vă testa tema, puteți rula fișierul "Checker.hs" cu unul din parametrii: `raw`, `adt`, `both` (care vor aplica evaluatorul vostru pe setul de teste corespunzător).

Checker-ul are nevoie de modulul `Data.List.Utils`, care se găsește în pachetul `MissingH`. Puteți să-l instalați folosind "cabal".

```
cabal update
cabal install MissingH
```

Cabal vine instalat odată cu platforma Haskell. Dacă nu îl găsiți, căutați în repository-urile voastre pachetul `"cabal-install"`.

Pentru a rula checker-ul, puteți folosi `runghc` sau `runhaskell`. Exemplu:

```
runghc Checker.hs both
runhaskell Checker.hs adt
```

6 Trimitere

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

- fișierul "Interpreter.hs" (cu definiția funcției `evalAdt` și, eventual, a funcției `evalRaw`)
- fișier "README"
- orice alt fișier sursă Haskell de care aveți nevoie

7 Punctaj

Tema valorează în total **1.5 puncte** din nota finală, dintre care:

- 1.0 puncte pentru evaluarea programelor sub formă de TDA.
- 0.5 puncte pentru implementarea parserului.

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

După deadline-ul soft, se vor scădea 0.5 puncte pe zi, până la deadline-ul hard.

8 Referințe

[1] Context-free grammar
https://en.wikipedia.org/wiki/Context-free_grammar

[2] Backus-Naur Form
https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form

[3] Interpreter
https://en.wikipedia.org/wiki/Interpreter_(computing)

[4] Interpreted programming language
https://en.wikipedia.org/wiki/Interpreted_language

[5] Syntactic analysis
https://en.wikipedia.org/wiki/Parsing

[6] Semantic analysis
https://en.wikipedia.org/wiki/Semantic_analysis_(compilers)

[7] Maybe and Either (Learn You a Haskell for Great Good)
http://learnyouahaskell.com/making-our-own-types-and-typeclasses

[8] Error handling in Haskell (School of Haskell)
https://www.schoolofhaskell.com/school/starting-with-haskell/basics-of-haskell/10_Error_Handling

[9] Error handling in Haskell (Real World Haskell)
http://book.realworldhaskell.org/read/error-handling.html#errors.maybe