INTRODUCERE ÎN HASKELL 98 PRIN EXEMPLE

Lect. Drd. Dan Popa de la Universitatea din Bacău, folosind Haskell în cursul cercetărilor pentru finalizarea tezei de doctorat, a început să scrie un manual despre Haskell în limba română. Volumul se numeşte "Introducere în Haskell 98 prin exemple". Din primul său capitol vă prezentăm cele de mai jos, fără însă a avea pretenţia de a fi scris un articol exhaustiv despre Haskell şi Hugs.

Vom explica şi cum veţi rula mici exemple pe computerul dumneavoastră dotat cu sistem de operare Linux. La realizarea cărţii "Introducere în Haskell 98 prin exemple" am folosit Mandrake Linux 8.2 (dar puteţi folosi fără probleme Mandrake Linux 9 sau 10, cu condiţia să aveţi inclus în distribuţie interpretorul Hugs sau să-l instalaţi ulterior). Hugs este disponibil atât ca arhivă zip cât şi ca pachet rpm. Teoretic vorbind, aceste exemple pot fi rulate şi pe o maşină Windows pentru care deţineţi software-ul necesar, o implementare de Haskell aliniată la standardul Haskell 98. Utilizatorii unor versiuni mai vechi sunt invitaţi să treacă la Haskell 98, versiunile mai vechi fiind actualmente depăşite. Există şi compilatoare pentru Haskell. Unul dintre cele mai cunoscute este GHC-ul.

Începem aici prin a răspunde în acest articol la câteva întrebări pe care vi le puneţi atunci când abordaţi un nou limbaj. Haskell 98 este un limbaj funcţional pur, dar în care se poate programna şi în stil imperativ (vom vedea că se foloseşte o noţiune care a făcut multă vâlvă în lumea matematicienilor, monada - de fapt o anumită structură algebrică). În Haskell 98 există de asemenea o serie de noţiuni specifice programării orientate pe obiecte cum ar fi clasă, instanţă a unei clase etc. Funcţiile, vom vedea, sunt polimorfice. Se pot descrie uşor de tot, liste şi arbori, reuniuni de tipuri, ba chiar mulţimi ordonate şi liste infinite. (Interesant, nu-i aşa?) Şi totul se scrie rapid şi compact (un algoritm Quicksort începe în doar patru-cinci rânduri de program!) şi fără aglomerarea aceea de paranteze cu care ne-au forţat să ne obişnuim limbaje ca Lisp (poreclit de unii "Limbajul InSuportabilelor Paranteze") sau ML. O parte dintre aceste concepte le vom prezenta rând pe rând, răspunzând una câte una la nişte întrebări pe care şi le pune cel care învaţă un limbaj de programare. Începem cu cele mai simple lucruri, cum ar fi:

- Cum se scriu în Haskell tipurile de date (arhi)cunoscute din alte limbaje: întregii, numerele reale şi alte tipuri comune?
- Cum se scriu în Haskell cele mai simple tipuri compuse (liste şi perechi)?
- Cum se scriu în Haskell operatorii?
- Cum se scrie o funcţie simplă?
- Cum se rulează un program?

Ulterior vom trece la o serie de exemple, scriind mai întâi simple funcţii în Haskell, şi trecând gradat la exemple mai complicate. Printre altele exemplificăm:

- Cum se scrie o funcţie simplă?
- Cum rulaţi programele scrise în Haskell?

Să începem:

Cum se scriu în Haskell tipurile de date (arhi)cunoscute din alte limbaje: întregi, numerele reale şi alte tipuri comune ?

Numerele întregi se numesc Integer (dar veţi vedea că se foloseşte şi `Int`).

Numerele reale sunt de mai multe feluri, reţineţi deocamdată tipul Float.

Valorile logice, booleene, două la număr (`True` şi `False`) formează tipul `Bool`.

Caracterele tipografice sunt elemente din tipul (care include şi simbolurile), `Char`.

Stringurile sunt de fapt liste de caractere iar tipul string se notează în consecinţă ca un tip compus: [`Char`]. Parantezele pătrate avertizează că este vorba de listă de elemente de tipul descris în paranteză, acţionând deci ca un constructor de tipuri (cam cum era perechea `RECORD ... END` în Pascal şi Oberon numai că aici se declară liste nu structuri).

Reţineţi deci: Cele mai simple tipuri de sunt: `Integer` (sau `Int`), `Bool`, `Float`, `Char` iar stringurile formează tipul [`Char`] denumit cu aliasul său `String`.

Atenţie: În Haskell numele tipurilor de date încep întotdeauna cu majusculă. Regula este valabilă şi pentru tipurile utilizator, motiv pentru care veţi scrie tipul Arbore cu A majusculă, atunci când veţi avea nevoie de un asemenea tip. În schimb funcţiile se scriu cu minusculă, de exemplu o funcţie care incrementează o variabilă o veţi numi inc. Haskell 98 este un limbaj din categoria celor în care minusculele şi majusculele au sens diferit, fiind asemenea din acest punct de vedere cu limbajele C, Java şi Oberon.

Cum se scriu în Haskell cele mai simple tipuri compuse?

Pe lângă modurile de a construi tipuri definite de utilizator (ex.arbori), Haskell permite construirea de liste şi perechi. Aceleaşi semne (paranteza pătrată, respectiv cea rotundă şi virgula care separă elementele) se folosesc şi pentru a construi tipul compus din tipuri mai simple dar şi pentru a construi valoarea compusă din valori mai simple.

Exemple:

`[Integer]` - listă de întregi iar [1,2,3] e lista formată din 1, 2 şi 3

`(Char,Int)` - perechi de `Char` şi `Int`, deci `('b',4)` înseamnă că 'b' şi 4 sunt împerecheate

Atenţie: Caracterele se scriu între ghilimele simple, obţinute cu tasta de lângă Enter. Celălalt fel de ghilimele simple, inverse, ce se obţin cu tasta de sub Escape, sunt folosite în alt scop în Haskell (la definirea operatorilor infixaţi). Nu le confundaţi! Vom remarca şi faptul că stringurile admit o scriere dublă, atât ca liste de caractere cât şi ca texte între ghilimele duble. Ex. `['a','l','f','a']` este totuna cu `"alfa"`.

Observaţie: Elementele unei liste pot fi: oricât de multe dar de acelaşi tip, unic. Elementele unei perechi sunt exact două, dar pot fi din două tipuri diferite, dinainte precizate. Noţiunea de pereche se poate generaliza la cea de n-uplu. Haskell permite să folosiţi şi asemenea n-uple.

Cum se scriu în Haskell operatorii?

Operatorii sunt grupaţi în 10 clase de prioritate. Un operator, chiar şi unul definit de utilizator (Da există aşa o posibilitate în Haskell!) trebuie plasat într-una din cele 10 clase de prioritate. În acest fel, Haskell poate stabili corect ordinea operaţiilor dintr-o expresie, indiferent ce operatori sunt implicaţi. Unii operatori asociază implicit la stânga, alţii la dreapta. Operatorii infixaţi care au un nume format din litere apar în programe cu acest nume încadrat de ghilimele simple inverse (de pe tasta cu semnul tilda). Regula aceasta este valabilă şi pentru operatorii definiţi de programator. Dacă veţi deschide cu un editor de text fişierul `/usr/share/hugs/lib/Prelude.hs` (Standard Prelude - biblioteca Standard) veţi găsi acolo, deja declaraţi, o serie întreagă de operatori. Îi vom enumera imediat pe cei mai mulţi dintre ei, precizând pentru fiecare printr-o pereche literă - cifră dacă e asociativ la dreapta (R) sau stânga (L) şi ce nivel de prioritate are.

Iată o mică listă de operatori declaraţi în Standard Prelude :

```
  R9   .
  L9   !!
  R8 ^ , ^^ , **
  L7   *   ,   /   ,   `quot` ,   `rem` ,   `div`   ,   `mod` ,
:%   , %
  L6   + , -
  R5   :
    4   ==, /=, <=, <, >=, >, `elem`, `notElem`
  R3 &&
  R2 ||
  L1   >> , >>=
  R1 =<<
  R0 $, $! , `seq`
```

Pe care încercăm să-i explicăm pe scurt.

```
 .                          - este compunerea funcţiilor
 ^                          - este ridicarea la putere
`quot`              - similară ca nume cu o funcţie din Lisp
`div`                       - împărţirea întreagă
`mod`                       - restul dat de împărţirea întreagă
 + , -                      - adunarea şi scăderea, supraîncărcate
  :                         - cons, adăugarea unui elment în capul unei
liste
==, /=, <=, <, >=,>- relaţiile, comparaţiile, supraîncărcate
`elem`, `notElem` - apartenenţa şi neapartenenţa la o listă (mulţime)
&&                          - conjuncţia
 ||                         - disjuncţia, (negaţia se notează cu not )
>> , >>=                   - operatori pentru monade
                            (de obicei înlănţuiesc calculele fiind cumva
                            similari lui ";" din limbajele imperative. Pot fi
                            redefiniţi în instanţele claselor monadice pentru a
                            obţine diverse efecte)
=<<                         - la fel, cu operanzii invers
```

Cu ajutorul lor sunt definiţi şi alţi operatori, de exemplu folosindu-se operatorul ":" (pronunţat "cons" ca în Lisp) este definită concatenarea a două liste sau a două stringuri, notată cu ++. Veţi folosi frecvent acest operator.

Cum se scrie o funcţie simplă ?

Matematicianul care face o declaraţie de funcţie va scrie întâi semnătura acesteia (adică schema de tipuri căreia funcţia i se conformează). În Haskell ea se nume-şte chiar tipul funcţiei, deoarece "->" este şi el un constructor de tipuri, ceva capabil să îmbine două

tipuri şi să obţină altul. (Obţinându-se astfel o "funcţie de la... la...").

Exemplu: `f : Z -> Z` prin relaţia `f(x) = x + 1` se scrie în Haskell:

```
f       :: Integer -> Integer
f x     = x + 1
```

`Cap1Par3Ex1.hs`

Pentru a-l testa, scrieţi acest mic program cu editorul dumneavoastră favorit (`kwrite, gedit, emacs, xemacs, vi, pico`) şi salvaţi-l sub numele `Cap1Par3Ex1.hs`.

Câteva explicaţii sunt aici şi acum necesare:

Alinierea pe verticală, este extrem de importantă. În limbaje ca Pascal, C sau Oberon şi multe altele, un programator putea scrie tot programul pe un singur rând, fără ca aceasta să deranjeze compilatorul sau interpretorul. Acest lucru nu se poate face în Haskell deoarece Haskell, pentru a elimina acele paranteze care ne copleşesc în Lisp sau ML, foloseşte un concept revoluţionar: sintaxa bidimensională. Iată ce trebuie să ştiţi deocamdată despre ea:

Trecerea pe rândul următor, dar mai la stânga decât pe precedentul rând echivalează cu o închidere de structură sau paranteză. E ca şi cum s-ar pune automat în Lisp paranteză închisă. (Reţineţi că Haskell este un limbaj cu sintaxă bidimensională!). Această aranjare în pagină se numeşte cu termenul englezesc "layout".

Extensia fişierelor conţinând surse de program Haskell este .hs. Mai există o extensie `.lhs` pentru "literate - Haskell" acele texte de lucrări în care totul este comentariu cu excepţia liniilor care încep cu > şi formează ele însele programul.

Cum se rulează un program ?

Hugs este numele interpretorului Haskell 98 livrat împreună cu distribuţiile de Linux. Nu este însă inclus în toate distribuţiile oferite spre download. Eu l-am găsit inclus în distribuţiile Mandrake 8.2 şi Mandrake 10. Îl puteti lansa cu una din comenzile:

```
hugs
hugs <numefisier>
hugs "<numefisier>"
```

Ghilimele sunt necesare în cazul în care numele de fişier conţine spaţii. Observati în imaginea de mai jos că fiind lansat cu comanda

```
hugs Cap1Par3Ex1.hs
```

interpretorul deschide şi citeste programe din două fişiere: Standard Prelude şi fişierul dat de noi în linia de comandă.

De asemenea, există o variantă a interpretorului numită runhugs care permite să folosiţi Haskell cam ca un limbaj de scripting. Acesta încarcă programul scris în Haskell şi evaluează expresia main, care trebuie să fie dată în program. Runhugs poate fi folosit exact cum foloseaţi `/bin/bash` pentru a crea shell scripturi. Sau pentru a face cgi-scripting pentru crearea de pagini web dinamice.

Interpretorul răspunde încărcând definiţiile din Standard Prelude şi pe cele din fişierul meu, afişând apoi promptul. (Promptul va fi diferit de Main doar atunci când veţi lucra cu module deoarece acest prompt Haskell implicit indică de fapt numele modulului curent):

```
Main>
```

Acum puteţi întreba interpretorul ce valoare are, de exemplu, `f (100)`. În Haskell aşa ceva se scrie fără paranteze:

```
Main> f 100
```

Interpretorul Haskell va răspunde, cum era de aşteptat:

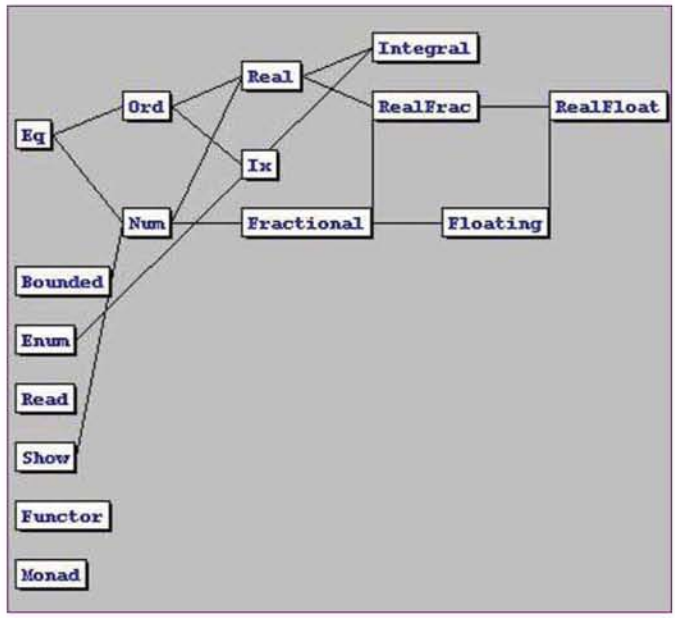
```
101
```

Veţi închide sesiunea de lucru cu interpretorul hugs apăsând `CTRL-D`.

Încheiere

Haskell 98 este un limbaj funcţional pur, bazat pe lazy-evaluation, elegant, mai uşor decât Lisp-ul dar mult mai flexibil. Spre deosebire de Lisp este un limbaj tipizat şi mult mai uşor de folosit. Oferă matematicienilor instrumente specifice (inclusiv mulţimi, liste infinite, functori şi monade) dar şi programatorilor noţiuni cu care s-au obişnuit din alte limbaje: structuri de date (arbori, liste etc), funcţii şi tipuri utilizator, clase, instanţe, module cu import şi export selectiv (ceea ce-l face potrivit pentru proiectele de mari dimensiuni) şi chiar o notaţie imperativă - the do-notation - pentru operaţiile de IO şi succesiunile de calculele dintr-o monadă.

Imaginea de mai jos reprezintă ierarhia claselor din Haskell 98.



Productivitatea muncii în Haskell este de 9-20 ori mai mare ca în alte limbaje. Nici nu e de mirare deoarece Haskell, fiind un limbaj funcţional, printre altele, poate încapsula şabloane întregi de programare sub forma unor funcţii, de nivel superior, funcţii care manipulează alte funcţii.

Acum puteţi merge la **http://www.haskell.org** - site-ul comunităţii şi consulta manualele de acolo. Sperăm că v-am deschis o cale. Un proverb chinezesc spune: "Călătoria de 10 000 de li începe cu un pas". Fie ca acest articol să fie începutul unei minunate călătorii. ■

Dan Popa
popavdan@yahoo.com