Dan Popa
Universitatea “Vasile Alecsandri” din Bacău

Combinatorii de analizoare sintactice: instrumente ale lingviştilor pentru învățarea asistată de calculator a limbilor străine

Abstract: A set of parser combinators implemented in the very high-level functional language *Haskell* is actually a way of programming linguistic exercises on a computer since they enable us to create modular syntax checkers and interpreters. Such programs represent a welcome addition to the manual of a written language, making it interactive in a different way. The tools / technical aspects have been displayed in the book *Practica Interpretării Monadice* / *The Practice of Monadic Interpretation*. Further debatable and similar topics are also approached.
Key-words: computer aided teaching, monadic parsing in Haskell

Obiective: Realizarea de produse software lingvistice prin construirea de parsere adaptabile, modulare.
Activitate: Implementarea tipică a unui exercițiu dintr-un manual de limba străină, care datorită proprietăților de adaptabilitate ale parserelor modulare, permite extinderea sa.
Rezultate anterioare: Am implementat un analizor sintactic modular, adaptabil, destinat să asiste la obținerea competenței de a formula fraze (de un tip dat) într-o limbă străină.
Ipoteză de lucru: Un set de combinatori de parsere implementat în limbajul funcțional Haskell (din categoria VHLL, “very high level language”) se poate folosi pentru implementarea de exerciții din domeniul lingvisticii, întrucât combinatorii de parsere permit crearea de verificatoare

modulare de sintaxă.
Suport tehnic pentru implementare: Toate componentele software necesare sunt conținute într-un software gratuit şi de domeniu public, numit ***The Haskell Platform***.

1.2.1 Introducere

Principalele etape ale proiectului sunt următoarele:
Pasul I: Prezentarea unei metodologii (care să fie, de asemenea, utilizabilă de către lingvişti) pentru transformarea regulilor unei gramatici într-un program scris în Haskell.
Pasul al II-lea: Formarea unei echipe compuse din lingvişti şi programatori în Haskell, cu scopul de a crea produse destinate pieței de instrumente lingvistice şi în special pieței de manuale electronice interactive.
Pasul al III-lea: Crearea unui corpus de reguli gramaticale implementate prin combinatori de parsere, acesta fiind un scop pe termen lung.

1.2.2 Scopul pe termen lung

Ne propunem să ajungem în posesia unui corpus de reguli care să reprezinte un întreg limbaj natural, descris cu ajutorul unei colecții de combinatori de parsere. O etapă intermediară este descrierea unei limbi străine la nivelul unui manual introductiv al acestei limbi şi producerea unui manual electronic bazat pe această tehnologie. O colecție de reguli gramaticale exprimate prin combinatori de parsere provenite de la mai multe exerciții din acest prim manual se transformă într-o gramatică de mari dimensiuni, deoarece combinatorii de persere sunt modulari şi adaptabili. Practic, ei se conformează ideii de proiect *software* adaptabil, aşa cum este el prezentat în [Arm 01].

12.3 Situația în prezent

Cu unele excepții, lingviştii/ filologii nu sunt familiarizați cu declarațiile (descrierile) arborilor sintactici în limbajul de programare Haskell. Lingviştii nu sunt, în general, familiarizați cu folosirea combinatorilor de parsere. Dar au două competențe foarte importante: cunoaşterea arborilor sintactici şi cunoaşterea gramaticilor.

1.2.4 Dezvoltarea domeniului pînă în prezent

Limbajul Haskell este el însuşi un limbaj oferit în domeniul public, cu o licență de tip BSD[1] şi este *free-software*. În plus, este disponibil pe toate sistemele de operare majoritare de pe piață: Windows ™, Linux ™ , Solaris ™, pentru Mac OS produs de Apple şi, de asemenea, pentru o serie - în creştere - de dispozitive mobile.

Combinatorii de parsere scrişi în Haskell au fost folosiți de o serie de autori, începând cu Graham Hutton şi Erik Meijer [Hut 96], [Hut 98]. În ultimul deceniu, au apărut o serie de alte biblioteci, inclusiv biblioteca de combinatori de parsere cu funcții de semnalizare a erorilor, *Parsec*, realizată de Daan Leijen [Lei-01]. Odată cu lansarea Haskell Platform SDK în anul 2009, putem găsi în bibliotecile acesteia toate componentele necesare proiectului nostru, inclusiv o versiune a bibliotecii Parsec.

La rândul lor, arborii sintactici sunt imediat implementabili în Haskell, deoarece limbajul oferă un instrument specific pentru declararea lor: declarația *data.*

Iar parserele modulare sunt componente software simple şi bine documentate de autorii bibliotecilor respective.

Structura algebrică pe care se bazează combinarea perserelor este *monada parserelor* – concept din teoria categoriilor –, dar lingviştii nu au nevoie să cunoască neaparat aceste detalii: *do notația* din Haskell face ca folosirea monadei parserelor să fie transparentă pentru utilizator, iar modulul Monad poate fi importat (inclus) în programe, fiind livrat ca o bibliotecă de sine stătătoare.

Remarcăm şi existența unor *idei preconcepute*, precum ideea că există o unică gramatică a unui limbaj. Aceasta este o idee falsă, lucru confirmat de teoria limbajelor formale.

Dimpotrivă, ideea că limbajul natural evoluează prin

---

1 Licența BSD permite includerea software-ului public în programe private, inclusiv comerciale.

adăugarea unor noi situații de comunicare se dovedeşte corectă. Construirea modulară a analizorului sintactic din combinatori de parsere se conformează acestei idei.

1.2.5 Arborele sintactic (aşa cum a fost el realizat pentru un exercițiu de manual)

Pentru o primă implementare, am ales un exercițiu dintr-un cunoscut (în România) manual de limbă japoneză, [Hon-91]. Pentru a face implementarea, se declară mai întâi tipul arborilor sintactici. Acest lucru este imediat, deoarece neterminalii gramaticii se transcriu în *constructorii de date* ai declarației de arbore *data* din Haskell.

```
data Arb = Propozitie Arb Arb
        | Adjdem String
        | Subst String
        | Adj String
        | Subiect Arb Arb
        | PredicatN Arb Arb
        | VerbCopulativ String
        deriving Show
```

Fiecare exercițiu din manual va avea propriul său tip de date, corespunzător arborelui său sintactic. Se poate obține însă şi un tip comun tuturor exercițiilor, declarând împreună toate variantele din instrucțiunile *data*.

Avantaje: Comparat cu programul scris în limbajul C – un limbaj de nivel mediu – cu ajutorul *pointerilor* şi structurilor, declarația (descrierea) de arbore sintactic în Haskell este, după cum se vede, mult mai simplă, mai scurtă. Lucru explicabil, Haskell fiind un VHLL (“very high level language”).

1.2.6 Aspecte privind ordinea cuvintelor în limba japoneză

Spre deosebire de limba română, limba japoneză foloseşte, de regulă, ordinea SOV (subiect, complement, verb). Combinatorii de parsere pot implementa rapid orice ordine.

Datorită posibilității lor de a se combina după necesități, ca

nişte piese de lego, combinatorii de parsere pot implementa orice ordine a părţilor de propoziţie. De exemplu, dacă în limba vizată, propoziţiile respectă ordinea SOV, regula gramaticală se va scrie ca mai jos:

-- <sentence> -> <subject> <object> <verb>

Iar analizorul sintactic monadic modular se scrie imediat în *do-notaţie*[2],sub forma:

```
sentence =
  do { x <- subject ;
       y <- object ;
       z <- verb ;
       return (Sentence x y z )
     }
```

În acest exemplu, am implementat SOV, ordinea din limba japoneză. Celelalte cazuri se implementează prin simpla permutare a neterminalilor, respectiv a rândurilor corespunzătoare din program.
Observaţie: aşa cum este scris codul sursă de mai sus, parserul modular *sentence* este scris cu intenţia de a se folosi un tip de arbore (sintactic) conţinând un *constructor de date*[3] numit *Sentence* în locul unuia numit *Propoziţie,* care apare într-unul din paragrafele anterioare.

1.2.7 Primele exerciţii utilizează o sintaxă simplă

Am constatat că primele exerciţii din manual, mai simple, utilizează la rândul lor o sintaxă simplă şi pot avea ocazional

2 *Do-notaţia* este o scriere a secvenţelor de operaţii în Haskell, care se transformă automat, transparent pentru utilizator, în apeluri de funcţii. Ca scriere, seamănă cu limbajul C, familiar multor programatori.

3 Constructorul de date este numele primului cuvânt de după "|" sau "=" din declaraţia arborelui sintactic. V. exemplele precedente (1.2.5).

nevoie de reguli mai simple decât cea de mai sus.

-- <sentence> -> <subject> <verb>

Corespunzător regulii, analizorul sintactic modular este mai simplu:

```
sentence =
  do { x <- subject ;
       z <- verb ;
       return (Sentence x  z )
     }
```

Exemplul de mai sus corespunde ordinii SOV, dar din care complementele (*Objects*) lipsesc.

1.2.8 O gramatică utilizată într-unul din exercițiile incluse în [Hon 91]

Unul dintre modelele de exerciții comune manualelor de limbi străine este cel numit în limba engleză “filling the blancks” sau “create sentences according to the model” (“completați spațiile libere sau formulați propoziții după model”.)

Având în vedere gramatica limbajului natural, un lingvist sau un cunoscător de limbaje formale poate indica uşor subsetul de reguli gramaticale cerute de un anumit exercițiu. Un asemenea set de reguli, creat pentru un exercițiu din lecția a doua, din manualul de limbă japoneză [Hon 91] (ex. B, punctul a, p. 8), este dat în continuare:

```
--  <subst>  -> "ie" | "kiku" | "hon" | "koe"
--  <adj>  -> "ookii" | "chiisai" | "omoshiroi" |  "tsumaranai"
--  <adjdem>   -> "kono" | "sono" | "ano"
-- <subiect>  -> <adjdem> <subst> "wa"
-- <predicat>   -> <adj> <vbcopulativ>
-- <vbcopulativ>  -> "desu" | "dewa arimasen"
```

```
-- <propozitie>  -> <subiect> <predicat>
```

1.2.9 Implementarea exemplului

Regulile gramaticale de mai sus sunt, în cele ce urmează, transformate în program Haskell, folosind combinatori de parsere (printre care *symbol* – care transformă un *string* în parserul care aşteaptă şi acceptă acel string), combinatori proveniți din biblioteca *ParseLib*. Alte biblioteci pot fi, de asemenea, folosite (ex: Parsec prezentată în [Lei 01]) .

```
-- <subst>  -> "ie" | "kiku" | "hon" | "koe"
subst    = do { symbol "ie" ;
                return (Subst "ie") ;
              }
           +++
           do { symbol "kiku" ;
                return (Subst "kiku") ;
              }
           +++
           do { symbol "hon" ;
                return (Subst "hon") ;
              }
           +++
           do { symbol "koe" ;
                return (Subst "koe") ;
              }

-- <adj>    -> "ookii" | "chiisai" | "omoshiroi" | "tsumaranai"
adj    = do { symbol "ookii" ;
              return (Adj "ookii") ;
            }
         +++
         do { symbol "chiisai" ;
              return (Adj "chiisai") ;
            }
         +++
         do { symbol "omoshiroi" ;
```

```
                  return (Adj "omoshiroi") ;
                }
             +++
            do { symbol "tsumaranai" ;
                  return (Adj "tsumaranai") ;
                }

--  <adjdem> -> "kono" | "sono" | "ano"
adjdem = do { symbol "kono" ;
                   return (Adjdem "kono") ;
                 }
             +++
            do { symbol "sono" ;
                  return (Adjdem "sono") ;
                 }
             +++
            do { symbol "ano" ;
                  return (Adjdem "ano") ;
                 }

-- <subiect> -> <adjdem> <subst> "wa"
subiect = do { a <- adjdem ;
                   s <- subst ;
                   symbol "wa" ;
                   return (Subiect a s) }

-- <predicat> -> <adj> <vbcopulativ>
predicat = do { a <- adj;
                    vbc <- vbcopulativ;
                    return (PredicatN  a vbc); }

-- <vbcopulativ> -> "desu" | "dewa arimasen"
vbcopulativ = do { symbol "desu" ;
                        return (VerbCopulativ "desu"); }
                  +++
                  do { symbol "dewa" ;
                        symbol "arimasen";
                        return (VerbCopulativ "desu"); }
```

```
-- <propozitie>  -> <subiect> <predicat>
propozitie =
  do { x <- subiect ;
       y <- predicat ;
       return (Propozitie x y )    }
```

După rescrierea regulilor gramaticale în Haskell (utilizând *do-notația* şi combinatorii de parsere) şi după compilare, codul obiect obținut este *link-editat* împreună cu restul bibliotecii de combinatori de parsere. În final, rezultă un program executabil binar. Am folosit compilatorul GHC, aşa cum se vede în imaginea următoare.

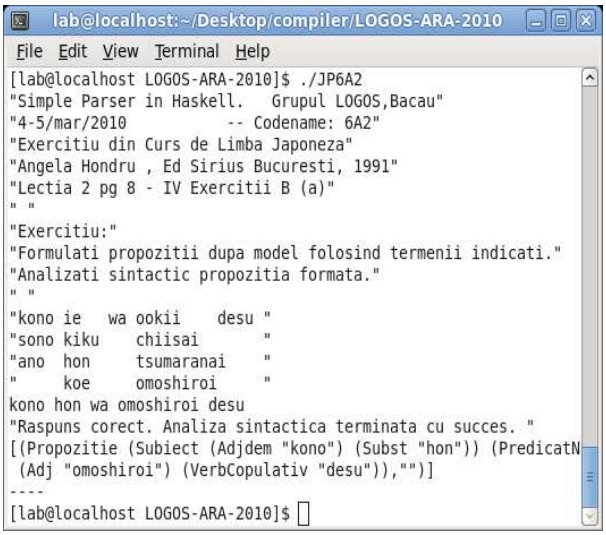

Fig. 16. Un exercițiu de manual, implementat ca parser modular adaptabil

1.2.10 Programul principal

Programul principal are următorul cod sursă:

```
module Main where
import Prelude hiding (read)
import Monad
import Data.Char

-- Portions  from "Practica interpretarii monadice"
-- Dan Popa, MatrixRom, 2009
-- Parser combinators :  Graham Hutton and Erik Meijer

-------------
main ::IO()
main =
  do { print "Simple Parser in Haskell.Grupul LOGOS,  Bacau"   ;
         print "4-5/mar/2010          -- Codename: 6A2" ;
         print "Exercitiu din Curs de Limba Japoneza";
         print "Angela Hondru, Ed Sirius Bucuresti, 1991" ;
         print "Lectia 2 pg 8 - IV Exercitii B (a)" ;
         print " ";
         print "Exercitiu:" ;
         print "Formulati propozitii dupa model folosind termenii indicati." ;
         print "Analizati sintactic propozitia formata.";
         print " ";
         print "kono ie   wa ookii    desu " ;
         print "sono kiku    chiisai       " ;
         print "ano  hon     tsumaranai    " ;
         print "       koe     omoshiroi     "  ;
         raspuns <- getLine ;
         let rezultat =  parse propozitie raspuns  in
         if (null rezultat)
          then do { print "Raspuns incorect in raport cu cerintele." ;   return ();    }
         else do { print "Raspuns corect. Analiza sintactica terminata cu succes. "  ;
                   print rezultat ;
                   putStrLn "----";
                   return ()
                  } ;
        return ()
        }
```

Iar la executarea sa, ca urmare a fazei de analiză sintactică, programul oferă:
a. reprezentarea arborelui sintactic al textului (propoziţiei) dat(e) ca răspuns (în cazul în care acest text trece integral de analiza sintactică)
sau

b. lista vidă (cu semnificația de răspuns incorect), în caz contrar.

1.2.11 Continuări posibile

Avem în vedere următoarele posibilități:

- constituirea unor echipe mixte, formate din lingvişti şi programatori în Haskell;
- transformarea unui set mai amplu de exerciții şi producerea analizoarelor sintactice modulare;
- producerea de cărți electronice interactive şi de compact discuri dedicate învățării limbilor străine;
- investigarea problemelor suplimentare care pot să apară;
- implementarea în produse a bibliotecilor de parsere mai noi, printre care Parsec, recent inclusă (din 2009) în *The Haskell Platform software development kit*.

1.2.12 Glosar

Pentru a face textele de mai sus accesibile cititorilor nefamiliarizați cu limba japoneză, includem un vocabular minimal (japonez-român), care cuprinde termenii folosiți în exemplul anterior :

```
-- <subst>
-- ie   = casa
-- kiku = crizantema
-- hon  = carte
-- koe  = voce

-- <adj>
-- ookiii = mare
-- chiisai = mic
-- omoshiroi = interesant
-- tsumaranai = neinteresant

--  <adjdem>
```

-- kono = acest/aceasta (de linga mine)
-- sono = acest/aceasta (de lina tine)
-- ano = acel/acea (de la distanta)

-- <vbcopulativ>
-- desu = a fi
-- dewa arimasen = a nu fi

1.2.13 Concluzii

Un set de combinatori de parsere implementați în limbajul funcțional Haskell constituie un bun instrument pentru a programa exerciții lingvistice. Pentru aceasta, propunem folosirea un computer dotat cu o implementare Haskell, deoarece oferă posibilitatea de a produce rapid analizoare sintactice şi interpretoare modulare, adaptabile prin inserarea de module. Metoda de lucru este accesibilă lingviştilor, aspectele categoriale ale implementării, monada parserelor, operatorii *bind* şi *return* fiind ascunşi de macrodefinițiile *do-notației*.

Bibliografie

[Arm-01] Armour Philip: *The business and software: Zeppelins and jet planes: a methaphor for modern software projects*. Comm. Of ACM, 44(10):13-15 Oct.2001

[Hud-99] Hudak Paul, Peterson John, Fasel Joseph – *A Gentle Introduction to Haskell 98*, Yale University, Los Alamos Laboratory, oct 1999

[Hut-96] Hutton, Graham; Meijer, Errik; *Monadic Parser Combinators* - "Technical report NOTTCS-TR-96-4" Dept. Comp. Sci. Univ. Nottingham - 1996
http://www.cs.nott.ac.uk/Department/Staff/gmh/monparsing.ps

[Hut-98] Hutton, Graham; Meijer,Erik; *Monadic Parsing in Haskell* Journal Of Functional Programming 8(4):437-444,

July 1998
http://www.cs.nott.ac.uk/Department/Staff/gmh/bib.html#pearl

[Hon 91] Hondru, Angela – Curs de limba japoneză - fără profesor – Sirius Publishing House, Buc. 1991

[Lei-01] Leijen, Daan; *Parsec a fast combinator parser*, Univ. of Utrecht, Dept. Of Computer Science, Utrecht, The Nederlands, 4 oct 2001

[Popa-2010] Metode şi tehnici de realizare a interpretoarelor adaptabile, Conducător ştiinţific Prof. Univ. Doctor. Habilitat. Dumitru Todoroi, septembrie 2010