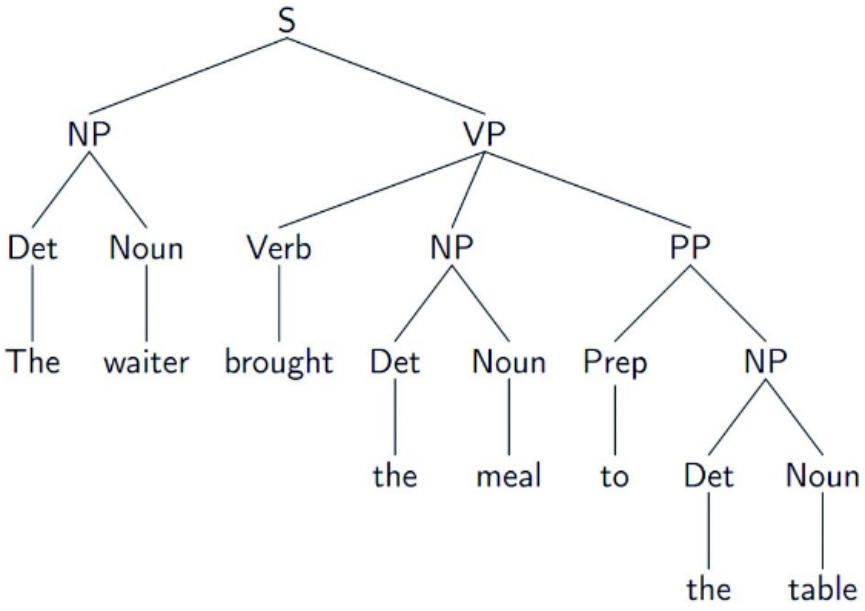
«Επεξεργασία Φυσικής Γλώσσας»

Απαλλακτική εργασία
Ιουνίου – Σεπτεμβρίου 2022

Μάθημα: Επεξεργασία Φυσικής Γλώσσας
Εξάμηνο: 6ο
Ον/μο: Ειρήνη Δικονιμάκη
ΑΜ: π19045
Επιβλέποντες καθηγητής: Θεμιστοκλής Παναγιωτόπουλος

Β.1 Λύση σε Prolog

Θέμα 1ο (20 μονάδες)
(α) Με ποιά γραμματική σε μορφή DCG μπορούμε να αναγνωρίσουμε την πρόταση : [the, waiter, brought, the, meal, to, the, table], σύμφωνα με το παρακάτω σχήμα;
(β) Με ποιά γραμματική σε μορφή DCG μπορούμε να παράγουμε σε μορφή functor το συντακτικό δένδρο για την αναγνώριση της πρότασης :
[the, waiter, brought, the, meal, to, the, table], σύμφωνα με το παρακάτω σχήμα;



(The waiter brought the meal to the table)

Λύση:

α) Για την λύση του ερωτήματος αυτού χρησιμοποίησα τον συμβολισμό '-->' για να ορίσω τους συντακτικούς κανόνες βάση το σχήμα.Όπου στο αριστερό μέρος του βέλους μπαίνει η πρόταση (πχ S) και στο δεξί μέρος πώς αναλύεται/ ή απο τι αποτελείται (πχ S-->NP,VP).Αυτο το έκανα για κάθε πρόταση και ο κώδικας έχει αυτή την μορφή:

```
s --> np, vp.

np --> det, noun.

pp --> prep, np.

vp --> verb, np, pp.

det --> [the].

verb --> [brought].

prep --> [to].

noun --> [waiter].

noun --> [meal].

noun -->[table].
```

Κανοντας το ερώτημα:
?- s([the, waiter, brought, the, meal,to,the,table],[]).
Πήραμε την απάντηση true.

```
?- s([the, waiter, brought, the, meal,to,the,table],[]).
true
```

?- s([the, waiter, brought, the, food],[]).
Πήραμε την απάντηση false.

```
?-  s([the, waiter, brought, the, food],[]).
false.
```

(β)Για να το πραγματοποιηθεί αυτό έπρεπε να διαμορφωθεί λιγο τον παραπάνω κώδικας.Με αυτή την γραμματική μπορούμε πλέον να

παράγουμε και το συντακτικό δένδρο της πρότασης στην μορφή ενός functor της Prolog, ο οποίος σαν δομή δέντρου αντιστοιχεί στο σχήμα. Ο κώδικας έχει αυτή την μορφή:

```
s(s(NP,VP))   --> np(NP), vp(VP).
np(np(D,N))   --> det(D), noun(N).
pp(pp(P,NP)  --> prep(P), np(NP).
vp(vp(V,NP,PP)) --> verb(V), np(NP), pp(PP).
det(det(the)) --> [the].
verb(verb(brought)) --> [brought].
prep(prep(to)) --> [to].
noun(noun(waiter)) --> [waiter].
noun(noun(meal)) --> [meal].
noun(noun(table)) --> [table]
```

Θέμα 2ο (20 μονάδες)

Το παρακάτω πρόγραμμα αναγνωρίζει και υπολογίζει αριθμητικές εκφράσεις όπως αναλύθηκε στο θεωρητικό μέρος. Να αναπτυχθεί ένα αντίστοιχο πρόγραμμα όπου οι αριθμοί είναι διαδικοί και οι αριθμητικές εκφράσεις είναι αντίστοιχα αριθμητικές εκφράσεις διαδικών αριθμών

```
expression(Value) --> number(Value).
expression(Value) --> number(X), [+], expression(V),
{Value is X+V}.
expression(Value) --> number(X), [-], expression(V),

{Value is X-V}.
expression(Value) --> number(X), [*], expression(V),
{Value is X*V}.
expression(Value) -->
number(X), [/], expression(V),
{V/=0, Value is X/V}.
expression(Value) --> left_parenthesis, expression(Value), right_parenthesis.
left_parenthesis --> ['('].
right_parenthesis --> [')'].
number(X) --> digit(X).
number(Value) --> digit(X), number(Y),{numberofdigits(Y,N), Value is X*10^N+Y}.
digit(0) --> [0].
digit(1) --> [1].
```

```
digit(2) --> [2].
digit(3) --> [3].
digit(4) --> [4].
digit(5) --> [5].
digit(6) --> [6].
digit(7) --> [7].
digit(8) --> [8].
digit(9) --> [9].
numberofdigits(Y,1) :- Z is Y/10, Z<1.
numberofdigits(Y,N) :-
  Z is (Y - mod(Y,10))/10,
  numberofdigits(Z,N1),
  N is N1+1.
```

Λύση:

Για να δημιουργήσουμε ένα πρόγραμμα αναγνώρισης αριθμητικών εκφράσεων δυαδικών αριθμών, πρέπει να ορίσουμε την γραμματική μας.

```
digit(Value)--> [Value).

digit(0000) --> [0000].

digit(0001) --> [0001].

digit(0010) --> [0010].

digit(0011) --> [0011].

digit(0100) --> [0100].

digit(0101) --> [0101].

digit(0110) --> [0110].

digit(0111) --> [0111].

digit(1000) --> [1000].

digit(1001) --> [1001].

number(Value) --> digit(Value).

expression(Value) --> number(Value).

expression(Value) --> number(X), [+], expression(V),
{Value is X+V}.

expression(Value) --> number(X), [-], expression(V),
{Value is X-V}.

expression(Value) --> number(X), [*], expression(V),
{Value is X*V}.

expression(Value) -->number(X), [/], expression(V),
{V/=0, Value is X/V}.
```

```
left_parenthesis --> ['('].
right_parenthesis --> [')'].
arithmetic_operator --> [+].
arithmetic_operator --> [-].
arithmetic_operator --> [*].
arithmetic_operator --> [/].
expression(Value) --> left_parenthesis, expression(Value), right_parenthesis.
expression(Value)  --> expression(Value),arithmetic_operator,expression(Value).
expression(V,[]).
```

Τρέχοντας 2 παραδείγματα βλέπουμε ότι η γραμματική μας τα αναγνωρίζει ως σωστά.

```
?- expression(0001+0010,[]).
true.
```

```
?- expression([0001*(0010/0011)+0011],[]).
true.
```

Στην συνέχεια θέλουμε να υπολογίζει τις αριθμητικές εκφράσεις αυτές. Για να το υλοποιήσουμε αυτό χρησιμοποιήσαμε τον εξής κώδικα:

```
number(X) --> digit(X).

number(A) --> digit(X), number(B),

{numberofdigits(B,N), A is X*2^N+B}.

numberofdigits(Y,1) :- Z is Y/2, Z<1.

numberofdigits(Y,N) :-

Z is (Y - mod(Y,2))/2,

numberofdigits(Z,N1),

N is N1+1.
```

number(X) --> digit(X). : Όταν ο αριθμός έχει ένα ψηφίο, έχει την αξία του ψηφίου.

Πχ η αξία του 0 ειναι το 0 και του 1 το 1.

number(A)-->digit(X), number(B),
{numberofdigits(B,N), Value is X*2^N+B}.

Οταν ο αριθμός Α έχει παραπάνω απο ένα ψηφία, το Χ και τον αριθμό Β οπου οΒ έχει Ν ψηφία τοτε η αξία του Α υπολογίζεται ως εξής:
X*2^N+B

ΠΧ 0001
0*2^3+001

numberofdigits(Y,1) :- Z is Y/2, Z<1.
Οταν Υ έχει ένα ψηφίο.

numberofdigits(Y,N) :-

Z is (Y - mod(Y,2))/2,
numberofdigits(Z,N1),
N is N1+1.
Οταν Υ έχει παραπάνω από 1 ψηφία.

Θέμα 3ο (60 μονάδες)
Χρησιμοποιείστε το σύνολο του κώδικα που σας δόθηκε σε Prolog για την «κατανόηση» μιας μικρής ιστορίας. Πρέπει να παράγετε τα περιεχόμενα της βάσης γνώσης και να μπορείτε να εισάγετε νέες πληροφορίες από το πληκτρολόγιο, να κάνετε ερωτήσεις στην βάση γνώσης, κ.λπ., παρόμοιες με αυτές της πρότυπης λύσης.

Αρχικά χρησιμοποίησα τον κώδικα σε Prolog για την κατανόηση μιας μικρής ιστορίας.
Εβαλα την ιστορία αυτή μέσα σε ένα αρχείο με όνομα test.txt.

Για να μπορέσουμε να εισάγουμε νέες πληροφορίες από το πληκτρολόγιο:
Αυτο γίνεται με τα εξής βήματα(κανόνες):

1) Update_knowledge_base,
2) Show_kb,
3) tell(Sentence)

Πιο αναλυτικά:

1)

```
/* update knowledge base                          */
/*-------------------------------------------------------------------------*/
update_knowledge_base([]) :- !.
update_knowledge_base([S|Sem]) :-
   assert(kb_fact(S)),
   write(kb_fact(S)),write(' asserted'),nl,
   update_knowledge_base(Sem), !.
```

Ανανεώνει την βάση γνώσης(κάνει update).

2)

```
/* all facts of knowledge base                          */
/*-----------------------------------------------------------------------*/
show_kb :- listing(kb_fact/1).

/*-----------------------------------------------------------------------*/
```

“Διαβάζει”ολες τις πληροφορίες απο την βάση γνώσης.

3)Τέλος, προσθέτει τις επιπλέον πληροφορίες (γνώση) στην βάση γνώσης.

```
tell(Sentence):-
  sem(_, Sem, Sentence, []),
  assert(kb_fact(Sem)),
  nl,write(kb_fact(Sem)), nl, write(' added to knowledge base.'),nl, !.

/*-----------------------------------------------------------------------*/
```

Παραδείγματα:
ask([who,shoots])].

```
?- tell([mary,shoots,slowly]).

kb_fact(shoots(mary,slowly))
 added to knowledge base.
true.

?-  tell([mary,is,short]).

kb_fact(short(mary))
 added to knowledge base.
true.

?-  tell([mary,walks]).

kb_fact(walks(mary))
 added to knowledge base.
true.

?- tell([mary,eats]).
false.
```

Για να κάνουμε ερωτήσεις στην βάση γνώσης:

Οι ερωτήσεις στην βάση γνώσης γινονται με τον κανόνα ask,παρολαυτα υπαρχουν οι ερωτήσεις που απαντώνται με ΝΑΙ-ΟΧΙ και οι άλλες ερωτήσεις ,που δεν απαντώνται με ένα απλο ναι ή οχι.

Yes-No questions

Οι ερωτήσεις που απαντώνται με ΝΑΙ-ΟΧΙ.
Για αυτό το είδος ερωτήσεων χρησιμοποιούμε τον κανόνα ask

```
ask(X):-  q( , tf,  Sem, X, []),
  if_then_else(kb_fact(Sem),  write('Yes.'), write('No.') ), !.
```

Και μπορουμε να κάνουμε ερώτηση:
Πχ ?- ask([is,mary,eating]).

Η απαντηση θα είναι της μορφής YES-NO.

```
?- ask([is,mary,eating]).
false.

?- ask([is,mary,running]).
No.
true.
```

ask([is,mary,eating]).

yes/no queries

```
q(1,tf,Sem) --> sem_av(does), sem_pn(N), sem_v(V,q), sem_n(N1),
{Sem=..[V,N,N1]}.
q(1,tf,Sem) --> sem_av(did), sem_pn(N), sem_v(V,q), sem_n(N1),
{Sem=..[V,N,N1]}.
q(2,tf,Sem) --> sem_av(is), sem_pn(N), sem_adj(A),
{Sem=..[A,N]}.
q(3,tf,Sem) --> sem_av(does), sem_pn(N), sem_v(V,q), sem_n(N1),
{Sem=..[V,N,N1]}.
q(4,tf,Sem) --> sem_av(is), sem_pn(N), sem_iv(V,q),
{Sem=..[V,N]}.
q(5,tf,Sem) --> sem_av(is), sem_pn(N), sem_iv(V,q), sem_adv(A),
{Sem=..[V,N,A]}.
q(6,tf,Sem) --> sem_av(does), sem_pn(N), sem_tv(V,q), sem_pn(N1),
sem_np(N2), {Sem=..[V,N,N1,N2]}.
q(6,tf,Sem) --> sem_av(does), sem_pn(N), sem_tv(V,q), sem_pn(N1),
sem_np(N2), {Sem=..[V,N,N1,N2]}.
```

other questions

Άλλες ερωτήσεις που δεν απαντώνται με ΝΑΙ-ΟΧΙ.

```
ask(X):- q(_,fact, Fact, X, []), write(Fact), !.
```

Παράδειγμα ερώτησης:
Πχ ?- ask([who,is,running]).
?- ask([who,is,walking]).

Η απαντηση θα είναι της μορφής κειμένου.

fact queries

```
q(1,fact,F)  -->  [who],  sem_v(V,s),sem_n(N1),                    {Sem=..
[V,F,N1], kb_fact(Sem)}.

q(1,fact,F)  -->  [what],  sem_av(does),sem_pn(N),sem_v(V,q),

      {Sem=..[V,N,F], kb_fact(Sem)}.

q(2,fact,F)  -->  [who],  sem_av(is),  sem_adj(A),
{Sem=..[A,F],kb_fact(Sem)}.

q(3,fact,F)  --> [who], sem_vp(1,V,N1),                            {Sem=..
[V,F,N1],kb_fact(Sem)}.

q(3,fact,F)  -->  [who],  sem_av(does),sem_pn(N),sem_v(V,q),
{Sem=..[V,N,F],kb_fact(Sem)}.

q(4,fact,F)  -->  [who],  sem_av(is),sem_iv(V,q),                  {Sem=..
[V,F],kb_fact(Sem)}.

q(5,fact,F)  -->  [how],  sem_av(does),sem_pn(N),sem_iv(V,s),
{Sem=..[V,N,F],kb_fact(Sem)}.

q(5,fact,F)  -->  [how],  sem_av(is),sem_pn(N),sem_iv(V,q),
{Sem=..[V,N,F],kb_fact(Sem)}.

q(5,fact,F)  --> [who], sem_iv(V,s),sem_adv(A),                    {Sem=..
[V,F,A],kb_fact(Sem)}.

q(6,fact,F)  -->  [who],  sem_tv(V,s),sem_pn(N1),sem_np(N2),
{Sem=..[V,F,N1,N2],kb_fact(Sem)}.

q(6,fact,F)  --> [who], sem_av(is),sem_pn(N),sem_tv(V,q2),sem_np(N2),[to],

      {Sem=..[V,N,F,N2] ,Sem}.

q(6,fact,F)  -->  [what],[is],sem_pn(N),sem_tv(V,q2),[to],sem_pn(N1),

      {Sem=..[V,N,N1,F] ,Sem}.
```