«**Επεξεργασία Φυσικής Γλώσσας**»

Απαλλακτική εργασία
Ιουνίου – Σεπτεμβρίου 2021

Ανάπτυξη & Τεκμηρίωση
Λεκτικού Αναλυτή
Συντακτικού Αναλυτή
Σημασιολογικού Αναλυτή
Διαχείριση Βάσης Γνώσης

Θεμιστοκλής Παναγιωτόπουλος
2021

Α. Θεωρία

1. Εισαγωγή

Η εργασία αυτή αφορά στην ανάπτυξη Λεκτικού, Συντακτικού και Σημασιολογικού Αναλυτή, σε πρώτο λόγο, στην ενημέρωση μιας σχετικής Βάσης Γνώσης σε δεύτερο λόγο, και στην παραγματοποίηση ερωταποκρίσεων στην Βάση Γνώσης σχετικά με το εν λόγω κείμενο.

Η λύση σε όλα αυτά τα ερωτήματα δίδεται εδώ σε Prolog και χρησιμοποιώντας τις Definite Clause Grammars, αλλά στο Teams έχουν αναρτηθεί και λύσεις σε άλλη γλώσσα (π.χ. Python).

2. Definite Clause Grammars (DCG)

Οι DCGs (Definite Clause Grammars) είναι μια ιδιαίτερη τεχνική που προσφέρεται στην Prolog, με την βοήθεια της οποίας μπορεί κάποιος εύκολα να ορίσει τους συντακτικούς κανόνες ενός Συντακτικού Αναλυτή (Parser). Αυτό βοηθάει ιδιαίτερα στην ανάλυση κειμένου αλλά και σε άλλες εφαρμογές.

Κατ'αρχήν χρησιμοποιείται ο συμβολισμός '**-->**' που μεταφράζεται με την έκφραση 'αναλύεται σε'. Έτσι αν γράψει κάποιος :

sentence --> noun_phrase, verb_phrase.

ερμηνεύεται ως εξής :
μια πρόταση (sentence) αναλύεται σε (ή αποτελείται από) μια φράση ουσιαστικού (noun_phrase) ακολουθούμενη από μια ρηματική φράση (verb_phrase).

2.1 Μια απλή Γραμματική για ανάλυση φυσικής γλώσσας

Μια Γραμματική, δηλαδή ένα σύνολο συντακτικών κανόνων, μπορεί εύκολα να διατυπωθεί σε DCGs. Για παράδειγμα έστω ότι έχουμε το παρακάτω σύνολο συντακτικών κανόνων ως ένα πρόγραμμα Prolog :

```
s --> np, vp.
np --> det, noun.
vp --> verb, np.
det --> [the].
verb --> [brought].
noun --> [waiter].
noun --> [meal].
```

2.1.1 Γεννήτορας προτάσεων

Αν δώσουμε την ερώτηση :
?- s(X,[]).

Η Prolog θα απαντήσει με τέσσερις απαντήσεις
X = [the, waiter, brought, the, waiter] ;
X = [the, waiter, brought, the, meal] ;
X = [the, meal, brought, the, waiter] ;
X = [the, meal, brought, the, meal].

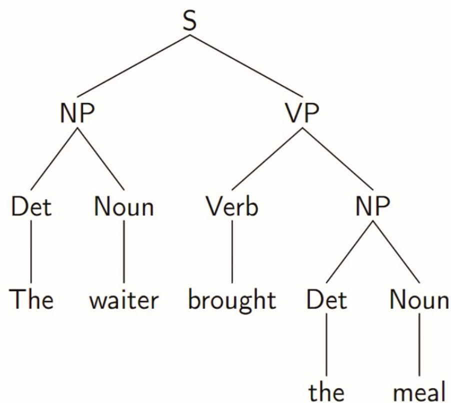
Δηλαδή, η γραμματική μας μπορεί να χρησιμοποιηθεί ως «**γεννήτορας προτάσεων**» που είναι **συντακτικά ορθές**, ανεξάρτητα αν βγάζουν νόημα, αυτό αφορά την σημασιολογία.

2.1.2 Αναγνώριση συντακτικής ορθότητας

Επίσης η ίδια γραμματική μπορεί να χρησιμοποιηθεί για την αναγνώριση της **συντακτικής ορθότητας** μιας πρότασης :

Αν κάνουμε την ερώτηση :
?- s([the, waiter, brought, the, meal],[]).
true.
?- s([the, waiter, brought, the, table],[]).
false.

Η πρώτη πρόταση προκύπτει από τους συντακτικούς κανόνες που έχουμε ορίσει, ενώ η δεύτερη πρόταση δεν προκύπτει από τους κανόνες αυτούς (γιατί δεν έχουμε δηλώσει το table ως noun). Σχηματικά, φαίνεται παρακάτω, το σχετικό συντακτικό δένδρο.



Το δένδρο αυτό σε Prolog θα αναπαριστάται με τον functor :
s(np(det(the), noun(waiter)), vp(verb(brought), np(det(the), noun(meal))))

2.1.3 Παραγωγή Συντακτικού Δένδρου

Αν επιθυμούμε η Prolog να παράγει το ίδιο το συντακτικό δένδρο σαν έξοδο, πρέπει να διαμορφώσουμε λίγο τους κανόνες της γραμματικής μας ως εξής :

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

Με αυτή την γραμματική μπορούμε πλέον να παράγουμε και το συντακτικό δένδρο της πρότασης στην μορφή ενός functor της Prolog, ο οποίος, όπως είδαμε, σαν δομή δένδρου αντιστοιχεί στο προηγούμενο σχήμα.

```
?- s(S,[the,waiter,brought,the,meal],[]).
S = s(np(det(the), noun(waiter)), vp(verb(brought), np(det(the), noun(meal)))).
```

2.2 Γραμματικές αναγνώρισης μαθηματικών εκφράσεων

Δεν είναι η φυσική γλώσσα το μόνο πεδίο εφαρμογής για την χρησιμοποίηση των DCGs. Στα επόμενα θα δούμε μια εφαρμογή για την αναγνώριση μαθηματικών εκφράσεων.

1.2.1 Μια Γραμματική για αναγνώριση απλών αριθμητικών εκφράσεων

Θέλουμε να αναπτύξουμε ένα συντακτικό αναλυτή που να αναγνωρίζει την συντακτική ορθότητα απλών μαθηματικών εκφράσεων που ακολουθούν τους παρακάτω κανόνες σε μορφη BNF :

BNF rules
<expr> :: = <num> | <num> + <expr> | <num> - <expr>
<num> :: = <digit> | <num> <digit>

Ο πρώτος κανόνας διαβάζεται ως εξής :

Μια έκφραση (<expr>) είναι (::=)
είτε ένας αριθμός (<num>),
είτε (|) ένας αριθμός ακολουθούμενος από το σημείο της πρόσθεσης (+) ακολουθούμενο από μια έκφραση (<expr>),
είτε (|) ένας αριθμός ακολουθούμενος από το σημείο της αφαίρεσης (-) ακολουθούμενο από μια έκφραση (<expr>).

Ο δεύτερος κανόνας διαβάζεται ως εξής : Ένας αριθμός (<num>) είναι (::=) είτε ένα ψηφίο (<digit>), είτε (|) ένας αριθμός (<num>) ακολουθούμενος από ένα ψηφίο (<digit>).

Οι κανόνες αυτοί σε μορφή DCG θα γραφτούν όπως φαίνεται παρακάτω. Για να αποφύγουμε το πρόβλημα της αριστερής αναδρομής (left recursion), που οδηγεί σε ατέρμονες βρόγχους, θα γράψουμε τον κανόνα :

```
number --> number, digit.
```

κάπως διαφορετικά :

```
number --> digit, number.
```

Έτσι καταλήγουμε στην γραμματική :

```
expression --> number.
expression --> number, arithmetic_operator, expression.
number --> digit.
number --> digit, number.
arithmetic_operator --> [+].
arithmetic_operator --> [-].
digit --> [0].
digit --> [1].
digit --> [2].
digit --> [3].
digit --> [4].
digit --> [5].
digit --> [6].
digit --> [7].
digit --> [8].
digit --> [9].
```

Πράγματι, κάνοντας την ερώτηση

```
?- expression([3,+,4,-,5],[]).
true .
```

Δηλαδή η μαθηματική έκφραση 3+4-5 αναγνωρίζεται από την γραμματική.

```
?- expression([+,4,-,5],[]).
false.
```

Η μαθηματική έκφραση +4-5 δεν αναγνωρίζεται από την γραμματική.

```
?- expression([4,+,-,5],[]).
false.
```

Η μαθηματική έκφραση 4+-5 δεν αναγνωρίζεται από την γραμματική.

2.2.2 Μια γραμματική λίγο πιό σύνθετων αριθμητικών εκφράσεων

Θέλουμε να επεκτείνουμε την γραμματική ώστε να δέχεται επιπρόσθετους τελεστές (*, /) και παρενθέσεις. Ας σημειωθεί ότι ο κανόνας :

```
expression --> expression, arithmetic_operator, expression.
```

Παρουσιάζει πρόβλημα αριστερής αναδρομής και δεν μπορεί να εφαρμοστεί.

```
expression --> number.
expression --> number, arithmetic_operator, expression.
expression --> left_parenthesis, expression, right_parenthesis.
number --> digit.
number --> digit, number.
arithmetic_operator --> [+].
arithmetic_operator --> [-].
arithmetic_operator --> [*].
arithmetic_operator --> [/].
left_parenthesis --> ['('].
right_parenthesis --> [')'].
digit --> [0].
digit --> [1].
digit --> [2].
digit --> [3].
digit --> [4].
digit --> [5].
digit --> [6].
digit --> [7].
digit --> [8].
digit --> [9].
```

Γράφουμε σε Prolog τις παρακάτω ερωτήσεις :

Ερώτηση 1

```
?- expression([1,+,'(',2,+,3,')'],[]).
true .
```

Ερώτηση 2

```
?- expression([1,*,'(',2,/,3,')'],[]).
true .
```

Ερώτηση 3

```
?- expression([1,*,'(',2,/,3,')',+,3],[]).
false.
```

Ερώτηση 4

```
?- expression(['(',55,*,'(',27,/,3,')',')'],[]).
```

false.

Ερώτηση 5
?- expression(['(',5,*,'(',2,/,3,')',')'],[]).
true .

Ερώτηση 6
?- expression(['(',5,+,'(',2,/,3,')',')'],[]).
true .

Ερώτηση 7
?- expression(['(',55,+,'(',2,/,3,')',')'],[]).
false.

Ερώτηση 8
?- expression(['(',5,5,+,'(',2,/,3,')',')'],[]).
true .

Εξηγούνται αυτά τα αποτελέσματα; Έχουμε κάνει κάποιο λάθος; Για παράδειγμα στις ερωτήσεις 4 και 7 θέλαμε να γράψουμε τον διψήφιο 55, όμως με βάση την γραμματική έπρεπε να γράψουμε 5,5 όπως στην ερώτηση 8. Η ερώτηση 3 γιατί βγάζει false;

2.2.3 Προσθέτοντας σημασιολογία

Η σημασιολογία είναι η ερμηνεία των όρων μιας πρότασης. Μπορούμε λοιπόν εμείς να θέλουμε όχι μόνο να αναγνωρίζουμε την ορθότητα του συντακτικού μιας αριθμητικής έκφρασης αλλά αν τα καταφέρουμε να βρίσκουμε και το αποτέλεσμα των πράξεων. Η σημασιολογία στους κανόνες DCG εισάγεται μέσα σε άγκιστρα, και γενικά αν θέλουμε να προσθέσουμε σε κανόνες DCG κάποιο κατηγόρημα Prolog αυτό πρέπει να γίνει μέσα σε άγκιστρα.

2.2.3.1 Η αριθμητική αξία ενός πολυψήφιου αριθμού

Ας δοκιμάσουμε λοιπόν με την τελευταία γραμματική, αλλά ας ξεκινήσουμε από την σημασιολογία για τους αριθμούς. Εδώ εμφανίζεται το πρόβλημα του υπολογισμού της τιμής ενός πολυψήφιου αριθμού. Ας υποθέσουμε ότι έχουμε ένα πολυψήφιο αριθμό, που αποτελείται από ψηφία που ανήκουν στο σύνολο {0,..,9} δηλαδή έχει σαν βάση το 10.

Αριθμούμε τα ψηφία ξεκινώντας από τα δεξιά, δηλαδή το τέλος του αριθμού, και κινούμενοι προς τα αριστερά, δηλαδή την αρχή του αριθμού δίνονταας την εξής αρίθμηση : Το τελευταίο ψηφίο 0, το προτελευταίο 1, κ.ο.κ. Έτσι για παράδειγμα για τον αριθμό 5789 έχουμε την εξής αρίθμηση :

```
5 7 8 9
3 2 1 0
```

Τότε η αξία του αριθμού είναι :
5*10^3 + 7*10^2 + 8*10^1 + 9 *10^0 =
5*1000 + 7*100 + 8*10 + 9*1 =
5789

Αυτό συμβαίνει επειδή η βάση μας είναι το 10 (ο αριθμός των ψηφίων που χρησιμοποιούμε για την αναπαράσταση του αριμού είναι 0,...,9 δηλαδή 10 ψηφία).

Στο δυαδικό σύστημα θα είχαμε δυνάμεις του 2 (έχουμε 2 ψηφία το 0 και το 1) :
Αριθμός : 1110001

Ψηφία αριθμού	1 1 1 0 1 0 1
Δυνάμεις	6 5 4 3 2 1 0
Αξία	1*2^6 + 1*2^5+1*2^4+0*2^3+1*2^2+0*2^1+1*2^0 =
	1*64 + 1*32 + 1*16 + 0*8 + 1*4 + 0*2 + 1*1 =
	64 + 32 + 16 + 0 + 4 + 0 + 1 = 117

2.2.3.1 Προσθέτοντας σημασιολογία στο μέρος του προγράμματος για αριθμούς

```
number(X) --> digit(X).
number(A) --> digit(X), number(B),
  {numberofdigits(B,N), A is X*10^N+B}.
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.
```

Εξήγηση του κώδικα :

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

(β) number(Value)-->digit(X), number(Y), {numberofdigits(Y,N), Value is X*10^N+Y}.

Όταν ο αριθμός A έχει ένα ψηφίο, έστω X, παραπάνω από ένα αριθμό B και ο αριθμός B έχει N ψηφία, τότε ο A έχει αξία : A = X*10^N + B (1)
για παράδειγμα ο 523 έχει ένα ψηφίο, το 5, παραπάνω από το 23, και ο 23 έχει 2 ψηφία, η αξία του 523 είναι 5*10^2+23.
Έτσι λοιπόν χρειαζόμαστε ως σημασιολογία μέσα στα άγκιστρα ένα κατηγόρημα που μας επιστρέφει τον αριθμό των ψηφίων ενός αριθμού B, και τον τύπο (1) :

{numberofdigits(B,N), A is X*10^N+B}.

(γ)

```
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.
```

Τον κώδικα αυτό τον αναπτύξαμε κατά την διάρκεια των παρουσιάσεων (όπως βέβαια και τα προηγούμενα προγράμματα). Την εξήγησή του αφήνουμε ως άσκηση.

Παράδειγμα :

```
?- number(V,[3,5,5],[]).
V = 355 .
```

2.2.3.2 Προσθέτοντας σημασιολογία στο υπόλοιπο μέρος του προγράμματος

Η σημασιολογία που προσθέτουμε βρίσκεται μέσα στα άγκιστρα. Το expression και το number έχουν αποκτήσει και ένα όρισμα στο οποίο υπολογίζεται η αριθμητική τους αξία.

```
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.
```

Παράδειγμα :

```
?- expression(V,['(',4,5,*,'(',6,0,+,4,0,')',')'],[]).
V = 4500 .
```

Συμπερασματικά, αρχικά αναπτύξαμε ένα πρόγραμμα αναγνώρισης αριθμητικών εκφράσεων, και στη συνέχεια ένα πρόγραμμα αναγνώρισης και υπολογισμού τους με βάση την σημασιολογία που προσθέσαμε στην τελευταία έκδοση του προγράμματος.

2.3 Βαθύτερη κατανόηση των DCGs

Ο φορμαλισμός των DCGs, μετατρέπεται στην Prolog, κατά την φάση μεταγλώττισης σε ένα ισοδύναμο πρόγραμμα Prolog. Για τα μη τερματικά σύμβολα (**Non-terminals**) προστίθενται δύο επιπλέον ορίσματα :

Το μέρος του κώδικα σε DCG :
sentence → noun_phrase, verb_phrase.
Γίνεται σε «normal» Prolog :
sentence(In, Out) :- noun_phrase(In, Temp), verb_phrase(Temp, Out).

Μια ακολουθία συμβόλων στο **In**, μπορούν να αναγνωριστούν σαν πρόταση (**sentence**), αφήνοντας την υπόλοιπη ακολουθία στο **Out** σαν «υπόλοιπο», αν μια φράση ουσιαστικού (**noun phrase**) μπορεί να αναγνωριστεί στην αρχή του **In**, αφήνοντας σαν υπόλοιπο το **Temp**, και μια ρηματική φράση (verb phrase) μπορεί να ανγνωριστεί στην αρχή του **Temp**, αφήνοντας το **Out** σαν υπόλοιπο.

Για τα τερματικά σύμβολα (**Terminal symbols**) όταν γράφουμε σε DCG :
noun --> [ball].
στην Prolog γίνεται :
noun(In, Out) :- recognise_terminal(In, ball, Out).
Το κατηγόρημα **recognise_terminal** ορίζεται ως :
recognise_terminal([Term|List], Term, List).

Αυτός είναι και ο λόγος που σε ένα κανόνα DCG δεν μπορούμε να προσθέσουμε ένα κατηγόρημα Prolog γιατί η Prolog θα του βάλλει άλλα δύο ορίσματα. Άν όμως προσθέσουμε το κατηγόρημα μέσα σε άγκιστρα τότε αυτό θα εκτελεστεί κανονικά.

3. Λεκτική Ανάλυση

Μέχρι τώρα δίναμε τις προτάσεις σε μορφή λίστας (π.χ. [the,waiter,brought,the,meal]). Το ερώτημα είναι αν μπορούμε να διαβάζουμε τις προτάσεις σε μια απλοποιημένη μορφή κανονικού κειμένου από ένα αρχείο και να τις μετατρέπουμε σε αυτή τη λίστα.

Στο αρχείο text.txt έχουμε ένα απλό παράδειγμα κειμένου :

‘the dog needs food. the cat has the food. the dog hates the cat. the dog chased the cat. the cat is scary.’.

Έχουμε φροντίσει να απλοποιήσουμε το κείμενο (δεν έχουμε κεφαλαία , αναφορικά, κ.λπ.). Στη συνέχεια παραθέτουμε τον κώδικα του λεκτικού αναλυτή που μπορείτε και εσείς να χρησιμοποιήσετε. Στον κώδικα υπάρχουν επεξηγηματικά σχόλια.

```
/*==========================================================*/
/*           LEXICAL ANALYSIS                               */
/*==========================================================*/

/*----------------------------------------------------------------------*/
/* Reads a text from a file and produces a list of sentences.           */
/* Each sentence is a list of words                                     */
/*----------------------------------------------------------------------*/

% First Read from File then analyse
lexical_analyse(Text, Result) :-
  % Text = 'text.txt'
  see(Text),                       /* open this file to read */
  read(X),                         /* read from File */
  seen,                            /* close the input File */
  analyse(X,Result),               /* Lexical Analysis */
  write_results(Result),           /* Write Result */
  !.                               /* stop now */

/*----------------------------------------------------------------------*/
/* analyse                                                              */
/* 1. turn imput into list of ascii codes                               */
/*2. group together ascii codes belonging to same word                  */
/*----------------------------------------------------------------------*/
% If we reach End of File then stop
analyse(end_of_file,[]) :- !.

% analyse
analyse(Input,All) :- input_to_sentences(Input,All).

/*----------------------------------------------------------------------*/
/* Input to ascii                                                       */
/* turn input into list of ASCII codes, pass list to tokeniser          */
/*----------------------------------------------------------------------*/
input_to_sentences(Input, List_of_Words):-
  name(Input,Ascii_List),
  tokenise(Ascii_List, List_of_Words).

/*----------------------------------------------------------------------*/
/*    TOKENISER                                                         */
/*----------------------------------------------------------------------*/

% If no ASCII codes left then stop
tokenise([],[]):-!.
```

```
% identify first sentence, move onto rest of sentences
tokenise(Ascii_List,All):-
  one_sentence(Ascii_List, Sentence, Rest, T),
  T=end_of_sentence,
  tokenise(Rest, List_of_Sentences),
  append([Sentence],List_of_Sentences,All).

/*--------------------------------------------------------------------*/
/*     ONE SENTENCE                                                    */
/* one_sentence(Ascii_List, Sentence, Rest, end_of_sentence)           */
/* From List of Ascii codes identify a sentence                        */
/*--------------------------------------------------------------------*/

% full-stop = end of sentence
one_sentence(Ascii_List, [Word], Rest_Ascii, end_of_sentence):-
   one_word(middle_of_word, Ascii_List, Ascii_Word, Rest_Ascii, T),
   T=end_of_sentence,
   name(Word,Ascii_Word), !.

% end of text = no codes left
one_sentence([], [], [], end_of_text):- !.

/*--------------------------------------------------------------------*/
/* if not the end of a sentence then add code to output list and       */
/* call recursively                                                    */
/*--------------------------------------------------------------------*/
one_sentence(Ascii_List, Sentence, Rest_Text, Type):-
   one_word(start_of_word, Ascii_List, Ascii_Word, Rest_Ascii, T),
   T=end_of_word,
   name(Word,Ascii_Word),
   one_sentence(Rest_Ascii, Rest_Words, Rest_Text, Type),
   append([Word], Rest_Words, Sentence).


/*--------------------------------------------------------------------*/
/* End of ONE SENTENCE                                                 */
/*--------------------------------------------------------------------*/

/*--------------------------------------------------------------------*/
/*     ONE WORD                                                        */
/*--------------------------------------------------------------------*/
% Terminate recursion :
% end of text = no codes left
one_word(middle_of_word, [], [], [], end_of_text):-  !.
% full-stop = end of sentence
one_word(middle_of_word, [46|T], [], T, end_of_sentence):-  !.
% space = end of word
one_word(middle_of_word, [32|T], [], T, end_of_word):-  !.

/*--------------------------------------------------------------------*/
/* if not the end of a word then add code to output list and           */
/* recurse                                                             */
/*--------------------------------------------------------------------*/
% ignore Carriage return (Ascii 13)
one_word(Any, [13|T], Word, Rest_Codes, Type):-
  one_word(Any, T,  Word,  Rest_Codes, Type).
% ignore Line feed (Ascii 10)
one_word(Any, [10|T], Word, Rest_Codes, Type):-
  one_word(Any, T,  Word,  Rest_Codes, Type).
% ignore leading space
one_word(start_of_word, [32|T], Word, Rest_Codes, Type):-
  one_word(start_of_word, T,  Word,  Rest_Codes, Type).
% We have moves to analysing the word
one_word(_, [Ascii_Code|T], [Ascii_Code|Word], Rest_Codes, Type):-
  Ascii_Code \= 32,
  one_word(middle_of_word, T, Word, Rest_Codes, Type).
```

```
/*-------------------------------------------------------------------------*/
/* End of ONE WORD                                                         */
/*-------------------------------------------------------------------------*/


/*=======================================================*/
/*     END OF LEXICAL ANALYSIS                           */
/*=======================================================*/
```

Φυσικά μπορεί κάποιος να βρει ένα έτοιμο λεκτικό αναλυτή στο διαδίκτυο τόσο σε Prolog όσο και σε άλλες γλώσσες. Η Python για παράδειγμα θεωρείται ότι έχει πολύ εξελιγμένο toolset/library για επεξεργασία φυσικής γλώσσας. Θα είναι χαρά μου να δοθεί και να εξηγηθεί ο κώδικας ενός λεκτικού αναλυτή σε οποιαδήποτε γλώσσα της αρεσκείας σας.

4. Σημαντικά σχόλια και ορισμοί

```
/*=============================================================================*/
/*     COMMENTS / DEFINITIONS                                                  */
/*=============================================================================*/

/*-----------------------------------------------------------------------------*/
/* Πρόσθετα επεξηγηματικά σχόλια                                               */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/* Proper Noun                                                                 */
/*-----------------------------------------------------------------------------*/
/* Proper Nouns (proper_noun): Αναφέρεται σε ονόματα και τοποθεσίες.           */
/* Μπορεί να είναι υποκείμενο αλλά και  αντικείμενο σε μια πρόταση και δεν     */
/* παίρνει άρθρο.                                                              */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/* Verb Phrase                                                                 */
/*-----------------------------------------------------------------------------*/
/* Verb Phrase: Φράσεις που περιέχουν το ρήμα και το αντικείμενο ή κατηγορούμενο */
/* (αν αυτό υπάρχει):                                                          */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/* Intransitive Verb                                                           */
/*-----------------------------------------------------------------------------*/
/* Intransitive Verbs (intransitive_verb): Ρήμα που δεν δέχεται αντικείμενο.   */
/* Αναφέρει πράξη του υποκειμένου  και μπορεί από μόνο του να αποτελεί φράση   */
/*(verb phrase).                                                               */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/* Transitive Verb                                                             */
/*-----------------------------------------------------------------------------*/
/* Transitive Verbs (transitive_verb): Μεταβατικά ρήματα, δέχονται υποκείμενο, */
/* και πάνω από ένα αντικείμενα. Ένα άμεσο και ένα απλό. Στο παράδειγμα        */
/* «Ο Κώστας δίνει ένα μήλο στην Μαρία» το μήλο είναι το αντικείμενο και η     */
/* Μαρία το άμεσο αντικείμενο. Χρησιμοποιείταιξεχωριστά καθώς στην βάση γνώσης */
/* έχει τρία γνωρίσματα  (1 υποκείμενο, 2 αντικείμενα) και χρησιμοποιούνται    */
/* τρείς τύποι του. Στην παρούσα έκδοση του προγράμματος υπάρχει μόνο ένα τέτοιο */
/* ρήμα το give.                                                               */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/* Verb                                                                        */
/*-----------------------------------------------------------------------------*/
/* Verbs (verb): Ο όρος χρησιμοποιείται για όλα τα ρήματα τα οποία δέχονται    */
/* αντικείμενο. Μεταξύ τους υπάρχουν πολλοί τύποι ρημάτων (emotion verbs,      */
/* action verbs, helping verbs). Δίνονται πάλι όπως στα iv δύο τύποι συνδεόμενοι */
/* για χρήση σε διαφορετικούς τύπους προτάσεων. Στην παραγωγή γνώσης παίρνουν  */
/* δυο ορίσματα, το υποκείμενο και το αντικείμενο                              */
/*-----------------------------------------------------------------------------*/
```

```
/*----------------------------------------------------------------------------*/
/* Auxiliary Verb                                                             */
/*----------------------------------------------------------------------------*/
/* Auxiliary Verbs (auxiliary_verb): Auxiliary verbs ή αλλιώς helping verbs   */
/* ονομάζονται τα ρήματα που λειτουργούν ως βοηθητικά στα ρήματα που τα ακολουθούν,*/
/*δίνοντας τους ή συμπληρώνοντας το νόημα τους.                               */
/*----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------*/
/* Noun Phrase                    `                                           */
/*----------------------------------------------------------------------------*/
/* Noun Phrase (noun_phrase): Is either a pronoun or any group of words that can */
/* be replaced by a pronoun. Φράση που δηλώνει το υποκείμενο σε μια πρόταση.  */
/* Μπορεί να αποτελείται και από ένα proper noun                              */
/*----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------*/
/* Noun                                                                       */
/*----------------------------------------------------------------------------*/
/* Noun (noun): Τα ουσιαστικά στο πρόγραμμα χρησιμοποιούνται ως αντικείμενα και */
/* μόνο.Τον ρόλο του υποκειμένου παίρνουν συνήθως αλλά όχι αποκλειστικά τα    */
/* proper nouns.                                                              */
/*----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------*/
/* Adverb                                                                     */
/*----------------------------------------------------------------------------*/
/* Adverb (adverb): Μέρος του λόγου που περιγράφει ένα επίρρημα,              */
/*----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------*/
/* Adjective                                                                  */
/*----------------------------------------------------------------------------*/
/* Adjectives (adjective): Στην ελληνική γλώσσα τα επίθετα, χρησιμοποιούνται  */
/* ως χαρακτηρισμός προσώπου. Όμοια και στο συγκεκριμένο πρόγραμμα.           */
/*----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------*/
/* Determiner                                                                 */
/*----------------------------------------------------------------------------*/
/* Determiner (det): Ονομάζονται οι λέξεις που είτε χρησιμοποιούνται για την  */
/* κλήση ενός ουσιαστικού (άρθρα όπως a, the)                                 */
/*----------------------------------------------------------------------------*/
```

5. Το σύνολο του προγράμματος

5.1. Εισαγωγικές δηλώσεις και δεδομένα

```
/*---------------------------------------------------------------------*/
/*           PROJECT      ON                    */
/*     NATUTAL LANGUAGE PROCESSING (NLP) */
/*---------------------------------------------------------------------*/
/* Author :Themis Panayiotopoulos              */
/* June 2020                                   */
/*---------------------------------------------------------------------*/
/* 'Natural Language Understanding :           */
/*                         from a story to a knowledge base'      */
/*---------------------------------------------------------------------*/
/*      1. Lexical Analysis                           */
/*      2. Syntactic analysis                         */
/*      3. Semantic analysis                          */
/*      4. Updating the Knowledge Base                */
/*      5. Inserting Info to knowledge base    using tell(Sentence)       */
/*      6. asking questions about the story using ask(Question)    */
/* Top query                                   */
/*     ?- understand('text.txt'),              */
/*     ?- understand('maria.txt'),                    */
/*---------------------------------------------------------------------*/
/* story in text.txt :                                */
/* the dog needs food. the cat has the food. the dog hates the     */
/* cat. the dog chased the cat. the cat is scary.           */
/*---------------------------------------------------------------------*/
/* story in maryt.txt :                        */
/* mary runs quickly. mary is tall. mary is slim. mary is blonde.  */
/* mary gives john a dog. mary gives tomy a book.           */
/* mary loves books                                   */
/*---------------------------------------------------------------------*/
:- discontiguous ask/1, q/5.
/*---------------------------------------------------------------------*/
/* vocabulary elements of the text.txt example       */
/*---------------------------------------------------------------------*/
:- dynamic chased/2, hates/2, has/2, needs/2, scary/1.

/*---------------------------------------------------------------------*/
/* aditional vocabulary elements  (also for mary.txt example)     */
/*---------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
/* Verbs (v)                           */
/*--------------------------------------------------------------------*/
:- dynamic loves/2, love/2, hate/2, have/2, kicks/2, jumps/2.

/*--------------------------------------------------------------------*/
/* Auxiliary Verbs (av)                        */
/*--------------------------------------------------------------------*/
:- dynamic does/2, are/2, do/2.

/*--------------------------------------------------------------------*/
/* Intransitive Verbs (iv)                            */
/*--------------------------------------------------------------------*/
:- dynamic runs/1,hurts/1,walks/1,jumps/1,shoots/1.
:- dynamic runs/2,hurts/2,walks/2.

/*--------------------------------------------------------------------*/
/* Transitive Verbs (tv)                       */
/*--------------------------------------------------------------------*/
:- dynamic gives/2, gave/2.

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

```
/* Adjectives (adj)                                              */
/*-------------------------------------------------------------------------*/
:- dynamic tall/1,short/1,blonde/1,slim/1,fat/1.
```

5.2. Το κυρίως πρόγραμμα

```
understand(Story) :-
lexical_analyse(Story, Sentences),
  write('the lexical analysis completed!'), nl,nl,nl,
syntax_analyse(Sentences, Syntax),
  write('the syntactic analysis completed!'),nl,nl,nl,
semantics_analyse(Sentences, Semantics),
  write('the semantic analysis completed!'), nl,nl,nl,
update_knowledge_base(Semantics),
  nl,write('Knowledge Base Updated!'),nl,nl,nl.
```

5.3. Λεκτική Ανάλυση

```
/*==========================================================*/
/*           LEXICAL ANALYSIS                               */
/*==========================================================*/

/*-------------------------------------------------------------------------*/
/* Reads a text from a file and produces a list of sentences.      */
/* Each sentence is a list of words                             */
/*-------------------------------------------------------------------------*/

% First Read from File then analyse
lexical_analyse(Text, Result) :-
  % Text = 'text.txt'
  see(Text),                       /* open this file to read */
  read(X),                 /* read from File */
  seen,                    /* close the input File */
  analyse(X,Result),       /* Lexical Analysis */
  write_results(Result),   /* Wite Result */
  !.                /* stop now */

/*-------------------------------------------------------------------------*/
/* analyse                                */
/* 1. turn imput into list of ascii codes                    */
/*2. group together ascii codes belonging to same word      */
/*-------------------------------------------------------------------------*/
% If we reach End of File then stop
analyse(end_of_file,[]) :- !.

% analyse
analyse(Input,All) :- input_to_sentences(Input,All).

/*-------------------------------------------------------------------------*/
/* Input to ascii                                */
/* turn input into list of ASCII codes, pass list to tokeniser     */
/*-------------------------------------------------------------------------*/
input_to_sentences(Input, List_of_Words):-
  name(Input,Ascii_List),
  tokenise(Ascii_List, List_of_Words).

/*-------------------------------------------------------------------------*/
/*    TOKENISER                           */
/*-------------------------------------------------------------------------*/

% If no ASCII codes left then stop
tokenise([],[]):-!.

% identify first sentence, move onto rest of sentences
tokenise(Ascii_List,All):-
  one_sentence(Ascii_List, Sentence, Rest, T),
  T=end_of_sentence,
```

```
  tokenise(Rest, List_of_Sentences),
  append([Sentence],List_of_Sentences,All).

/*-------------------------------------------------------------------*/
/*     ONE SENTENCE                         */
/* one_sentence(Ascii_List, Sentence, Rest, end_of_sentence)  */
/* From List of Ascii codes identify a sentence             */
/*-------------------------------------------------------------------*/

% full-stop = end of sentence
one_sentence(Ascii_List, [Word], Rest_Ascii, end_of_sentence):-
   one_word(middle_of_word, Ascii_List, Ascii_Word, Rest_Ascii, T),
   T=end_of_sentence,
   name(Word,Ascii_Word), !.

% end of text = no codes left
one_sentence([], [], [], end_of_text):- !.

/*-------------------------------------------------------------------*/
/* if not the end of a sentence then add code to output list and   */
/* call recursively                         */
/*-------------------------------------------------------------------*/
one_sentence(Ascii_List, Sentence, Rest_Text, Type):-
   one_word(start_of_word, Ascii_List, Ascii_Word, Rest_Ascii, T),
   T=end_of_word,
   name(Word,Ascii_Word),
   one_sentence(Rest_Ascii, Rest_Words, Rest_Text, Type),
   append([Word], Rest_Words, Sentence).

/*------------------------------------------------------------------*/
/* End of ONE SENTENCE                      */
/*------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/*     ONE WORD                             */
/*-------------------------------------------------------------------*/
% Terminate recursion :
% end of text = no codes left
one_word(middle_of_word, [], [], [], end_of_text):-  !.
% full-stop = end of sentence
one_word(middle_of_word, [46|T], [], T, end_of_sentence):-  !.
% space = end of word
one_word(middle_of_word, [32|T], [], T, end_of_word):-  !.

/*-------------------------------------------------------------------*/
/* if not the end of a word then add code to output list and      */
/* recurse                                  */
/*-------------------------------------------------------------------*/
% ignore Carriage return (Ascii 13)
one_word(Any, [13|T], Word, Rest_Codes, Type):-
  one_word(Any, T,  Word,  Rest_Codes, Type).
% ignore Line feed (Ascii 10)
one_word(Any, [10|T], Word, Rest_Codes, Type):-
  one_word(Any, T,  Word,  Rest_Codes, Type).
% ignore leading space
one_word(start_of_word, [32|T], Word, Rest_Codes, Type):-
  one_word(start_of_word, T,  Word,  Rest_Codes, Type).
% We have moves to analysing the word
one_word(_, [Ascii_Code|T], [Ascii_Code|Word], Rest_Codes, Type):-
  Ascii_Code \= 32,
  one_word(middle_of_word, T, Word, Rest_Codes, Type).
/*------------------------------------------------------------------*/
/* End of ONE WORD                          */
/*------------------------------------------------------------------*/
/*==========================================================*/
/*     END OF LEXICAL ANALYSIS                  */
/*==========================================================*/
```

5.4. Συντακτική Ανάλυση

```
/*=========================================================*/
/*           SYNTACTIC ANALYSIS        */
/*=========================================================*/

/*-------------------------------------------------------------------------*/
/*Takes as input a list of sentences and produces their     */
/* syntax trees                                  */
/*-------------------------------------------------------------------------*/
syntax_analyse(Sentences, Structures) :-
   syntactic_analysis(Sentences, Structures),  % Syntactic Analysis
   write_results(Structures),                  % Wite Result
   !.                             % stop now

syntactic_analysis([],[]) :- !.
syntactic_analysis([Sentence|Sentences], [Structure|Structures]) :-
   snt(Structure, Sentence, []),
   syntactic_analysis(Sentences, Structures), !.

/*=========================================================*/
/*           GRAMMAR RULES             */
/*=========================================================*/

/*------------------------------------------------------------------------*/
/*The s rules of Syntactic Analysis                    */
/*------------------------------------------------------------------------*/
/* To test:                         */
/*  ?- snt(Structure,[the,dog,chased,the,cat],[]).          */
/* Structure =                                   */
/*      s(np(d(the),n(dog)),vp(v(chased),np(det(the),n(cat))))     */
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/* Sentence (snt)                                */
/* Proper Nouns (pn)                             */
/* Intransitive Verbs (iv)                             */
/* Auxiliary Verbs (av)                          */
/* Verbs (v)                        */
/* Transitive Verbs (tv)                         */
/* Adverb (adv)                                  */
/* Adjectives (adj)                                    */
/* Determiner (det)                                    */
/* Noun (n)                         */
/* Noun Phrase (np)                                    */
/* Verb Phrase (vb)                                    */
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/* Sentence (snt)                                */
/*------------------------------------------------------------------------*/
snt(s(NP,VP))       --> np(NP), vp(VP).

/*------------------------------------------------------------------------*/
/* Noun Phrase (np)                                    */
/*------------------------------------------------------------------------*/
np(np(N))           --> pn(N).
np(np(D,N))   --> det(D), n(N).
np(np(N))           --> n(N).

/*------------------------------------------------------------------------*/
/* Verb Phrase (vb)                                    */
/*------------------------------------------------------------------------*/
% Intransitive verbs :
vp(vp(V))           --> iv(V).
vp(vp(V,ADV))       --> iv(V), adv(ADV).
% Auxiliary verbs
vp(vp(AV,A)) --> av(AV), adj(A).
```

```
% Transitive verbs :
vp(vp(TV, PN, NP))  --> tv(TV), np(PN), np(NP).
% verbs
vp(vp(V,NP))  --> v(V), np(NP).

/*============================================================*/
/*     VOCABULARY OF EXAMPLE                    */
/*============================================================*/
/* the dog needs food. the cat has the food. the dog hates the     */
/* cat. the dog chased the cat. the cat is scary.          */
/*-----------------------------------------------------------------------*/
/* det : the verbs : needs, has, hates, chased, is   */
/* adjectives : scary                   nouns : cat, dog    */
/*-----------------------------------------------------------------------*/

/*-----------------------------------------------------------------------*/
/* Intransitive Verbs (iv)                          */
/*-----------------------------------------------------------------------*/
% needed for example :

% extension of vocabulary :
iv(iv(runs))-->[runs].
iv(iv(run))-->[run].
iv(iv(running))-->[running].
iv(iv(hurts))-->[hurts].
iv(iv(hurt))-->[hurt].
iv(iv(hurting))-->[hurting].
iv(iv(walks))-->[walks].
iv(iv(walk))-->[walk].
iv(iv(walking))-->[walking].
iv(iv(jumps))-->[jumps].
iv(iv(jump))-->[jump].
iv(iv(jumping))-->[jumping].
iv(iv(shoots))-->[shoots].
iv(iv(shoot))-->[shoot].
iv(iv(shooting))-->[shooting].

/*-----------------------------------------------------------------------*/
/* Auxiliary Verbs (av)                       */
/*-----------------------------------------------------------------------*/
% needed for example :
av(av(is))-->[is].
% extension of vocabulary :
av(av(does))-->[does].
av(av(are))-->[are].
av(av(do))-->[do].

/*-----------------------------------------------------------------------*/
/* Transitive Verbs (tv)                      */
/*-----------------------------------------------------------------------*/
% needed for example :

% extension of vocabulary :
tv(tv(gives)) -->[gives].
tv(tv(give))  -->[give].
tv(tv(gave))  -->[gave].
tv(tv(giving))      -->[giving].

/*-----------------------------------------------------------------------*/
/* Verbs (v)                          */
/*-----------------------------------------------------------------------*/
% needed for example :
v(v(chased))-->[chased].
v(v(chase))-->[chase].
v(v(needs))-->[needs].
v(v(need))-->[need].
v(v(hates))-->[hates].
v(v(hate))-->[hate].
```

```
v(v(has))  -->[has].
v(v(have))  -->[have].
% extension of vocabulary :
v(v(loves))-->[loves].
v(v(love))-->[love].
v(v(kicks))-->[kicks].
v(v(kick))-->[kick].
v(v(jumps))-->[jumps].
v(v(jump))-->[jump].

/*-----------------------------------------------------------------------*/
/* Adjectives (adj)                                                      */
/*-----------------------------------------------------------------------*/
% needed for example :
adj(adj(scary))-->[scary].
% extension of vocabulary :
adj(adj(tall))-->[tall].
adj(adj(short))-->[short].
adj(adj(blonde))-->[blonde].
adj(adj(slim))-->[slim].
adj(adj(fat))-->[fat].

/*-----------------------------------------------------------------------*/
/* Adverb (adv)                                              */
/*-----------------------------------------------------------------------*/
% needed for example :

% extension of vocabulary :
adv(adv(quickly))-->[quickly].
adv(adv(slowly))-->[slowly].
adv(adv(independently))-->[independently].

/*-----------------------------------------------------------------------*/
/* Noun (n)                                         */
/*-----------------------------------------------------------------------*/
% needed for example
n(n(food))-->[food].
n(n(cat))-->[cat].
n(n(cats))-->[cats].
n(n(dog))-->[dog].
n(n(dogs))-->[dogs].
% extension of vocabulary
n(n(book))-->[book].
n(n(books))-->[books].
n(n(feather))-->[feather].
n(n(feathers))-->[feathers].
n(n(baby))-->[baby].
n(n(babies))-->[babies].
n(n(boy))-->[boy].
n(n(boys))-->[boys].
n(n(girl))-->[girl].
n(n(girls))-->[girls].
n(n(icecream))-->[icecream].
n(n(icecreams))-->[icecreams].

/*-----------------------------------------------------------------------*/
/* Proper Nouns (pn)                                         */
/*-----------------------------------------------------------------------*/
pn(pn(mary))-->[mary].
pn(pn(john))-->[john].
pn(pn(tomy))-->[tomy].

/*-----------------------------------------------------------------------*/
/* Determiner (det)                                                      */
/*-----------------------------------------------------------------------*/
% needed for example :
det(det(the)) -->[the].
% extension of vocabulary
```

```
det(det(a))   -->[a].
det(det(an))   -->[an].

/*==========================================================*/
/*     END OF SYNTACTIC ANALYSIS        */
/*==========================================================*/
```

5.5. Σημασιολογική Ανάλυση

```
/*==========================================================*/
/*            SEMANTIC ANALYSIS         */
/*==========================================================*/
/*----------------------------------------------------------------------*/
/*Takes as input a list of sentences and produces their     */
/* semantics - easier if done along with syntactic analysis */
/*----------------------------------------------------------------------*/
semantics_analyse(Sentences, AllSemantics) :-
   semantics_analysis(Sentences, AllSemantics),      % Semantic Analysis
   write_results(AllSemantics),                % Write Result
   !.                             % stop now

semantics_analysis([],[]) :- !.
semantics_analysis([Sentence|Sentences], [Sem|Semantics]) :-
   sem(_, Sem, Sentence, []),
   semantics_analysis(Sentences, Semantics), !.

/*==========================================================*/
/*     SEMANTICS  CREATION  RULES             */
/*==========================================================*/

sem(1,Sem) --> sem_np(N), sem_vp(1,V,N1),      {Sem=..[V,N,N1]}.
% example : [the,dog,hates,the,cat]
% Sem = hates(dog,cat)
% example : [mary,loves,the,cat]
% Sem = loves(mary,cat)

sem(2,Sem) --> sem_np(N), sem_vp(2,_,A),      {Sem=..[A,N]}.
% example : [the,cat,is,scary]
% Sem = scary(cat)
% example : [nikos,is,slim]
% Sem = slim(nikos)

sem(3,Sem) --> sem_np(N), sem_iv(V,s),        {Sem=..[V,N]}.
% example : [maria,runs]
% Sem = runs(maria)
% example : [the,gun,shoots]
% Sem = shoots(gun)

sem(4,Sem) --> sem_np(N), sem_iv(V,s), sem_adv(A),   {Sem=..[V,N,A]}.
% example : [george,runs,quickly]
% Sem = runs(george,quickly)

sem(5,Sem)     -->sem_np(N),     sem_tv(V,s),     sem_np(N1),     sem_np(N2),
{Sem=..[V,N,N1,N2]}.
% example : [george,gave,mary,a,book]
% Sem = gave(george,mary,book)

/* noun phrase */
sem_np(N)     --> sem_pn(N).
sem_np(N)     --> sem_det(_), sem_n(N).
sem_np(N)     --> sem_n(N).

/* verb phrase */
sem_vp(1,V,N) --> sem_v(V,s), sem_np(N).
sem_vp(2,is,A) --> sem_av(is), sem_adj(A).
```

```
/*=========================================================*/
/*     SEMANTICS  VOCABULARY                    */
/*=========================================================*/

/*--------------------------------------------------------------------*/
/* Intransitive Verbs (sem_iv)                          */
/*--------------------------------------------------------------------*/
% needed for example :

% extension of vocabulary :
sem_iv(runs,s)      -->[runs].
sem_iv(runs,q)      -->[running].
sem_iv(hurts,s)     -->[hurts].
sem_iv(hurts,q)     -->[hurting].
sem_iv(walks,s)     -->[walks].
sem_iv(walks,q)     -->[walking].
sem_iv(jumps,s)     -->[jumps].
sem_iv(jumps,q)     -->[jumping].
sem_iv(shoots,s)    -->[shoots].
sem_iv(shoots,q)    -->[shooting].

/*--------------------------------------------------------------------*/
/* Auxiliary Verbs (sem_av)                             */
/*--------------------------------------------------------------------*/
% needed for example :
sem_av(is)          -->[is].
% extension of vocabulary :
sem_av(does)  -->[does].
sem_av(do)    -->[do].
sem_av(does)  -->[did].
sem_av(are)   -->[are].

/*--------------------------------------------------------------------*/
/* Transitive Verbs (sem_tv)                            */
/*--------------------------------------------------------------------*/
% needed for example :

% extension of vocabulary :
sem_tv(gives,s)     -->[gives].
sem_tv(gives,q)     -->[give].
sem_tv(gave,s)      -->[gave].
sem_tv(giving,q2)   -->[giving].

/*--------------------------------------------------------------------*/
/* Verbs (sem_v)                                */
/*--------------------------------------------------------------------*/
% needed for example :
sem_v(chased,_)     -->[chased].
sem_v(chase,_)      -->[chase].
sem_v(needs,s)      -->[needs].
sem_v(need,q) -->[need].
sem_v(hates,s)      -->[hates].
sem_v(hate,q) -->[hate].
sem_v(has,s)        -->[has].
sem_v(have,q)       -->[have].
% extension of vocabulary :
sem_v(loves,s)      -->[loves].
sem_v(loves,q)      -->[love].
sem_v(hates,s)      -->[hates].
sem_v(hates,q)      -->[hate].
sem_v(has,s)        -->[has].
sem_v(has,q)        -->[have].
sem_v(kicks,s)      -->[kicks].
sem_v(kicks,q)      -->[kick].
sem_v(jumps,s)      -->[jumps].
sem_v(jumps,q)      -->[jump].
```

```
/*--------------------------------------------------------------------*/
/* Adjectives (sem_adj)                        */
/*--------------------------------------------------------------------*/
% needed for example :
sem_adj(scary)      -->[scary].
% extension of vocabulary :
sem_adj(tall)       -->[tall].
sem_adj(short)             -->[short].
sem_adj(blonde)            -->[blonde].
sem_adj(slim)       -->[slim].
sem_adj(fat)        -->[fat].

/*--------------------------------------------------------------------*/
/* Adverb (sem_adv)                            */
/*--------------------------------------------------------------------*/
% needed for example :

% extension of vocabulary :
sem_adv(quickly)           -->[quickly].
sem_adv(slowly)            -->[slowly].
sem_adv(independently)     -->[independently].

/*--------------------------------------------------------------------*/
/* Noun (sem_n)                                */
/*--------------------------------------------------------------------*/
% needed for example
sem_n(food)         -->[food].
sem_n(cat)          -->[cat].
sem_n(cats)         -->[cats].
sem_n(dog)          -->[dog].
sem_n(dogs)         -->[dogs].
% extension of vocabulary
sem_n(book)         -->[book].
sem_n(books)        -->[books].
sem_n(feather)             -->[feather].
sem_n(feathers)            -->[feathers].
sem_n(baby)         -->[baby].
sem_n(babies)       -->[babies].
sem_n(boy)          -->[boy].
sem_n(boys)         -->[boys].
sem_n(girl)         -->[girl].
sem_n(girls)        -->[girls].
sem_n(icecream)            -->[icecream].
sem_n(icecreams)           -->[icecreams].

sem_n(X)                   -->sem_pn(X). % a proper noun is also a noun
/*--------------------------------------------------------------------*/
/* Proper Nouns (sem_pn)                       */
/*--------------------------------------------------------------------*/
sem_pn(mary)        -->[mary].
sem_pn(john)        -->[john].
sem_pn(tomy)        -->[tomy].

/*--------------------------------------------------------------------*/
/* Determiner (det)                                   */
/*--------------------------------------------------------------------*/
% needed for example :
sem_det(the) -->[the].
% extension of vocabulary
sem_det(a)   -->[a].
sem_det(an)  -->[an].

/*=========================================================*/
/*    END OF SEMANTICS  VOCABULARY             */
/*=========================================================*/
```

5.6. Ενημέρωση και Ερωτοαποκρίσεις στη Βάση γνώσης

```
/*==========================================================*/
/*     KNOWLEDGE BASE SESSION                    */
/*==========================================================*/

/*------------------------------------------------------------------*/
/* 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), !.

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

/*------------------------------------------------------------------*/
/* insert additional information to knowledge base          */
/*------------------------------------------------------------------*/
/*------------------------------------------------------------------*/
/* examples :                             */
/* ?- tell([mary,shoots,slowly]).                      */
/* ?- tell([mary,is,short]).                           */
/* ?- tell([mary,walks]).                        */
/*------------------------------------------------------------------*/
tell(Sentence):-
  sem(_, Sem, Sentence, []),
  assert(kb_fact(Sem)),
  nl,write(kb_fact(Sem)), nl, write(' added to knowledge base.'),nl, !.

/*------------------------------------------------------------------*/
/* ask knowledge base                            */
/*------------------------------------------------------------------*/
% Yes-No questions
/*------------------------------------------------------------------*/
/* examples :                             */
/* ?- ask([does,mary,love,books]).                     */
/* ?- ask([is,mary,tall]).                       */
/* ?- ask([is,mary,running]).                          */
/*------------------------------------------------------------------*/
ask(X):-  q(_, tf,  Sem, X, []),
  if_then_else(kb_fact(Sem),  write('Yes.'), write('No.') ), !.

/*------------------------------------------------------------------*/
/* 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
/*-------------------------------------------------------------------*/
/* examples :                              */
/* ?- ask([who,loves,books]).                          */
/* ?- ask([what,does,mary,love]).                */
/* ?- ask([who,is,tall]).                        */
/* ?- ask([who,is,running]).                           */
/* ?- ask([who,gives,john,dog]).                       */
/* ?- ask([who,is,mary,giving,a,dog,to]).                    */
/* ?- ask([what,is,mary,giving,to,john]).                    */
/*-------------------------------------------------------------------*/
ask(X):-  q(_,fact, Fact, X, []), write(Fact), !.

/*-------------------------------------------------------------------*/
/* 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}.

/*=========================================================*/
/*     END OF KNOWLEDGE BASE SESSION             */
/*=========================================================*/
```

5.7. Βιβλιοθήκη Προγράμματος

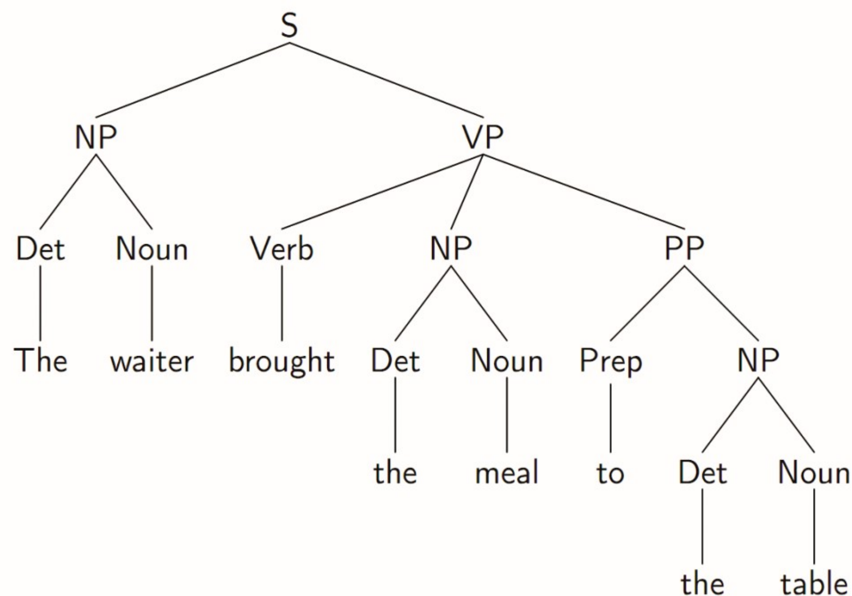
```
/*=========================================================*/
/*     GENERAL LIBRARY                           */
/*=========================================================*/
/*-------------------------------------------------------------------*/
/* Write Results                                 */
/*-------------------------------------------------------------------*/
write_results([]) :- nl, !.
write_results([H|T]) :-
  write(H), nl,
  write_results(T).

if_then_else(Condition,A,_):- call(Condition), call(A), !.
if_then_else(_,_,B):- call(B), !.
```

Β. Θέματα Εργασίας

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)

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.
```

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.

Β.1 Λύση σε άλλη γλώσσα προγραμματισμού

Εναλλακτικά μπορείτε να χρησιμοποιήσετε άλλες γλώσσες προγραμματισμού ή libraries τον κώδικα και την λειτουργικότητα των οποίων πρέπει να τεκμηριώσετε πειστικά και κατανοητά σύμφωνα με τα παρακάτω :

Θέμα 1ο (20 μονάδες)
Ανάπτυξη Λεκτικού Αναλυτή σε άλλη γλώσσα Προγραμματισμού. Αναζητείστε στο διαδίκτυο ή αναπτύξτε εσείς Λεκτικό Αναλυτή που διαβάζει μια μικρή ιστορία και μπορεί να παράγει μια λίστα από προτάσεις, κάθε μια από τις οποίες περιέχει μια λίστα από λέξεις. Τεκμηριώστε πειστικά τον κώδικά σας.

Θέμα 2ο (20 μονάδες)
Ανάπτυξη Συντακτικού Αναλυτή σε άλλη γλώσσα Προγραμματισμού. Αναζητείστε στο διαδίκτυο ή αναπτύξτε εσείς Συντακτικό Αναλυτή που με βάση τους κανόνες συντακτικής ανάλυσης της πρότυπης λύσης σε Prolog που σας δόθηκε παράγει το συντακτικό δένδρο της πρότασης. Τεκμηριώστε πειστικά τον κώδικά σας.

Θέμα 3ο (30 μονάδες)
Ανάπτυξη Σημασιολογικού Αναλυτή σε άλλη γλώσσα Προγραμματισμού. Αναζητείστε στο διαδίκτυο ή αναπτύξτε εσείς Σημασιολογικό Αναλυτή που με βάση τους κανόνες σημασιολογικής ανάλυσης της πρότυπης λύσης σε Prolog που σας δόθηκε παράγει τα σημαινόμενα της πρότασης (σχέσεις μεταξύ ρημάτων, ουσιαστικών, επιθέτων, κ.λπ). Τεκμηριώστε πειστικά τον κώδικά σας.

Θέμα 4ο (30 μονάδες)
Πρόγραμμα στην γλώσσα προγραμματισμού της επιλογής σας, για την ενημέρωση και πραγματοποίηση ερωταποκρίσεων σε Βάση Γνώσης που έχετε αναπτύξει γιαα αυτό τον σκοπό. Οι ερωτήσεις και απαντήσεις θα δίδονται σε φυσική γλώσσα.

Το σύνολο των μονάδων για κάθε μια από τις δύο εναλλακτικές απαλλακτικές είναι 100 μονάδες. Θα πρέπει να απαντήσετε είτε στην πρώτη εναλλακτική (Prolog), είτε στην δεύτερη αναλλακτική (άλλη γλώσσα προγραμματισμού). Την εργασία σας σε μορφή zip, με το κείμενό σας σε word, open office, ή pdf, μαζύ με αρχεία εκτελέσιμου κώδικα καταθέτετε στο gunet2.

Η εργασία είναι απαλλακτική 1 ατόμου. Το deadline παράδοσης για την εξεταστική Ιουνίου θα είναι θα είναι Δευτέρα 2 Αυγούστου, 11:59 μμ.