**Πανεπιστήμιο Πειραιώς**

Σχολή Πληροφορικής

Τμήμα Πληροφορικής

Εικόνα που περιέχει κείμενο, clipart

Περιγραφή που δημιουργήθηκε αυτόματα

Ιούλιος 2022

# Επεξεργασία Φυσικής Γλώσσας

|  |  |
| --- | --- |
| Γεωργιάδης Ευθύμιος Πάτροκλος | Π19031 |

Επιμέλεια: Παναγιωτόπουλος Θεμιστοκλής

Περιεχόμενα

**Θέμα 1 σελ. 3**

**Θέμα 2 σελ. 4**

**Θέμα 3 σελ.7**

**Το σύνολο του προγράμματος σελ. 8**

1. **Εισαγωγικές δηλώσεις και δεδομένα σελ. 8**
2. **Το κυρίως πρόγραμμα και τα κύρια κατηγορούμενα-μέθοδοι σελ. 9**
3. **Λεκτική Ανάλυση σελ. 10**
4. **Συντακτική Ανάλυση (Γραμματικοί κανόνες, Λεξιλόγιο) σελ. 13**
5. **Σημασιολογική Ανάλυση σελ. 17**
6. **Ενημέρωση και Ερωταποκρίσεις στη Βάση γνώσης σελ. 21**
7. **Βιβλιοθήκη Προγράμματος σελ.23**
8. **Σχόλια και ορισμοί σελ. 23**

**Επεξήγηση των προσθέσεων στο πρόγραμμα σελ. 25**

**Α. Λεκτική ανάλυση σελ. 26**

**Β. Συντακτική ανάλυση σελ.26**

**Γ. Σημασιολογική ανάλυση σελ.26**

**Δ. Κύριο πρόγραμμα σελ.26**

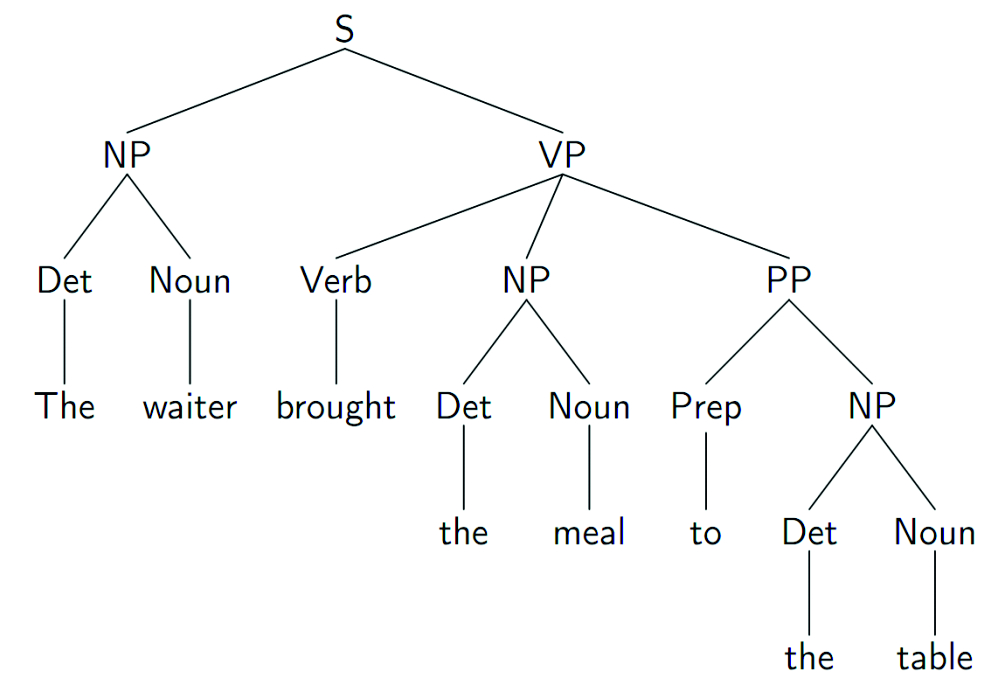
**Παραδείγματα εκτέλεσης σελ. 27**

Θέμα 1ο

**(α) Με ποια γραμματική σε μορφή DCG μπορούμε να αναγνωρίσουμε την πρόταση : [the, waiter, brought, the, meal, to, the, table], σύμφωνα με το παρακάτω σχήμα;**

**(β) Με ποια 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)

(α) Οι κανόνες DCG που μπορούμε να παράγουμε από το παραπάνω συντακτικό δέντρο είναι οι εξής:

s --> np, vp.

np --> det, noun.

vp --> verb, np, pp.

pp --> prep, np.

noun --> [waiter].

noun --> [meal].

noun --> [table].

verb --> [brought].

det --> [the].

prep --> [to].

Ταυτόχρονα, για να λειτουργήσουν αυτοί οι κανόνες DCG και να παράγουμε το συντακτικό δέντρο στην Prolog, είναι απαραίτητο να διαμορφωθούν ως εξής:

s(s(NP, VP)) --> np(NP), vp(VP).

np(np(D, N)) --> det(D), noun(N).

vp(vp(V, NP, PP)) --> verb(V), np(NP), pp(PP).

pp(pp(PR, NP)) --> prep(PR), np(NP).

noun(noun(waiter)) --> [waiter].

noun(noun(meal)) --> [meal].

noun(noun(table)) --> [table].

verb(verb(brought)) --> [brought].

det(det(the)) --> [the].

prep(prep(to)) --> [to].

(β) Χρησιμοποιώντας την παραπάνω γραμματική και το συντακτικό δέντρο μπορούμε να παράγουμε το συντακτικό δέντρο σε μορφή functor:

s(np(det(the), noun(waiter)), vp(verb(brought), np(det(the), noun(meal)), pp(prep(to), np(det(the), noun(table))))).

Θέμα 2ο

**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 δυαδικοί και οι αριθμητικές εκφράσεις είναι αντίστοιχα αριθμητικές εκφράσεις δυαδικών 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.

**Το σύνολο του προγράμματος**

**expression(Value) --> number(X), {list\_to\_number(X,Value,\_)}.**

**expression(Value) --> number(X), [+], expression(V),{list\_to\_number(X,A,\_), Value is A+V}.**

**expression(Value) --> number(X), [-], expression(V),{list\_to\_number(X,A,\_), Value is A-V}.**

**expression(Value) --> number(X), [\*], expression(V),{list\_to\_number(X,A,\_), Value is A\*V}.**

**expression(Value) --> number(X), [/], expression(V),{list\_to\_number(X,A,\_), V\=0, Value is A/V}.**

**expression(Value) --> left\_parenthesis,expression(Value),right\_parenthesis.**

**left\_parenthesis --> ['('].**

**right\_parenthesis --> [')'].**

**number(X) --> digit(I),{X=[I]}.**

**number(Value) --> digit(X),number(Y),{append([X],Y,Value)}.**

**digit(0) --> [0].**

**digit(1) --> [1].**

**list\_to\_number([],0,1).**

**list\_to\_number([H|T],N,C):-**

**list\_to\_number(T,N1,C1),C is C1\*2,N is N1+H\*C1.**

**decimal\_to\_binary(0,V,\_,V).**

**decimal\_to\_binary(D,V,C,N):-**

**Digit is D mod 2,D1 is D//2,V1 is V+Digit\*(10^C),**

**C1 is C+1,**

**decimal\_to\_binary(D1,V1,C1,N).**

**f\_result(Decimal,Binary):- decimal\_to\_binary(Decimal,0,0,Binary).**

**Για να εξηγήσουμε το νέο πρόγραμμα, θα συγκρίνουμε τους κανόνες και τις λειτουργίες του με το αντίστοιχο αρχικό πρόγραμμα που μας δίδεται ως παράδειγμα παραπάνω.**

* **Αρχικά, δεν θα χρησιμοποιήσουμε το κατηγόρημα number\_of\_digits/2, για να υπολογίσουμε το πλήθος των ψηφίων των αριθμών που μας δίνονται μέσα σε μία έκφραση, όμως χρησιμοποιούμε το κατηγόρημα list\_to\_number/3. Με αυτό το κατηγόρημα θα μετατρέψουμε τους αριθμούς των εκφράσεων που είναι σε μορφή λίστας ψηφίων, σε κανονικούς αριθμούς, δηλαδή σε μορφή ακέραιου αριθμού, για να πραγματοποιήσουμε τις απαιτούμενες πράξεις της αντίστοιχης έκφρασης.**

**Το κατηγόρημα list\_to\_number/3, περιέχει 3 ορίσματα: Μία λίστα (που περιέχει αριθμητικά ψηφία), και 2 μεταβλητές Ν, C (αριθμητικές). Η λίστα είναι η είσοδος του αριθμού που μας δίνεται σε μία έκφραση, η μεταβλητή Ν είναι ο κατασκευασμένος αριθμός από τα ψηφία που περιέχονται στην λίστα (μέχρι στιγμής), και η μεταβλητή C αντιπροσωπεύει την μονάδα μεγέθους του αριθμού Ν (χρησιμοποιείται για να μετρήσουμε το ποσό των μονάδων, δεκάδων, εκατοντάδων, κλπ, περιέχει ο αριθμός Ν και χρησιμοποιείται, για τον υπολογισμό του Ν). Επίσης, αποτελείται από 2 κανόνες: ο τερματικός κανόνας, με ορίσματα την κενή λίστα και τις μεταβλητές Ν και C ίσες με 0 και 1 αντίστοιχα, και τον κύριο κανόνα. Ο κύριος κανόνας καλεί αρχικά με αναδρομή το ίδιο κατηγόρημα, χρησιμοποιώντας την ουρά της δεδομένης λίστας ως το 1ο όρισμα και χρησιμοποιεί τα αποτελέσματα της αναδρομής μαζί με την κεφαλή της λίστας, για να υπολογίσει την μεταβλητή Ν.**

* **Μιλώντας, για τις αριθμητικές εκφράσεις, οι εκφράσεις expression/1 παραμένουν ίδιες με αυτές του αρχικού προγράμματος, με την εξαίρεση πώς κάθε εμφάνιση του κανόνα-έκφρασης number(X), συνοδεύεται από το κατηγόρημα list\_to\_number(X,A,\_) (ή list\_to\_number(X,A,\_)), για τον υπολογισμό των πραγματικών αριθμών και την κανονική εκπλήρωση των αντίστοιχων αριθμητικών πράξεων που αναπαριστά η κάθε αριθμητική έκφραση.**
* **Οι κανόνες digit/1, που λειτουργούν ως το λεξικό της γραμματικής μας, έχουν επίσης υποστεί αλλαγές. Καθώς επιθυμούμε να χρησιμοποιήσουμε το δυαδικό σύστημα για την δυαδική αναπαράσταση των αριθμών, φυσικά απαιτείται η χρήση μόνο δυαδικών ψηφίων. Επομένως, δεν θα χρειαστούμε τα ψηφία 2-9 στο λεξικό μας, καθώς αυτά τα ψηφία δεν ανήκουν δυαδικό σύστημα.**
* **Οι κανόνες number/1, καλούν με τον ίδιο τρόπο τους κανόνες digit/1 και τον number/1 αναδρομικά, με την διαφορά πως δεν επιστρέφουν αριθμούς αλλά λίστες από τα ψηφία αριθμών που δέχονται από τους κανόνες expression/1. Αποφεύγουμε την προηγούμενη υλοποίηση καθώς πολλές εκδόσεις τις prolog, (π.χ. SWI-Prolog multi-threaded) δεν επιστρέφουν τα μηδενικά ψηφία μπροστά από έναν αριθμό, αλλοιώνοντας έτσι τον δεδομένο αριθμό, κάτι που μπορεί να είναι ιδιαίτερα προβληματικό στην δυαδική αναπαράσταση των αριθμών, όπου χρησιμοποιεί μεγάλο πλήθος μηδενικών.**
* **Τέλος, για να επιστρέψουμε δυαδικές τιμές θα χρησιμοποιήσουμε το κατηγόρημα f\_result/2, μετατρέποντας με την χρήση του κατηγορήματος decimal\_to\_binary/4, τον δεκαδικό αριθμό του αποτελέσματος σε δυαδικό. Το πρώτο όρισμα του f\_result είναι ο δεκαδικός αριθμός που θέλουμε να μετατρέψουμε και το δεύτερο όρισμα είναι ο δυαδικός αριθμός που απαιτούμε ως έξοδο στην συγκεκριμένη περίπτωση.**

**Το decimal\_to\_binary, έχει 2 κανόνες με 4 ορίσματα ο καθένας. Το πρώτο όρισμα αντιπροσωπεύει τον δεκαδικό αριθμό που θέλουμε να μετατρέψουμε σε δυαδικό. Το τελευταίο όρισμα είναι ο τελικός δυαδικός αριθμός που επιθυμούμε ως έξοδο. Το δεύτερο όρισμα είναι επίσης ο δυαδικός αριθμός που σταδιακά παράγουμε με κάθε κλήση της μεθόδου decimal\_to\_binary/4 (αρχική τιμή 0). Το τρίτο όρισμα, αντιπροσωπεύει τις μονάδες, δεκάδες, κλπ του δεκαδικού αριθμού που εξετάζουμε εκείνη την στιγμή (αρχική τιμή 0).**

**Συνοπτικά, διαιρούμε σε κάθε κλήση του decimal\_to\_binary τον δεκαδικό αριθμό που μας δίνεται (το πρώτο όρισμα) διά 2. Πολλαπλασιάζουμε το υπόλοιπο της διαίρεσης με το τρίτο όρισμα που μας δίνεται και προσθέτουμε το αποτέλεσμα στο δεύτερο όρισμα. Έπειτα ξανακαλούμε την decimal\_to\_binary, με πρώτο όρισμα τo πηλίκο της διαίρεσης, δεύτερο όρισμα το αποτέλεσμα των πράξεων μας, τρίτο όρισμα το προηγούμενο τρίτο όρισμα αυξημένο κατά 1, και τελευταίο όρισμα το ίδιο με πριν. Ο αλγόριθμος θα σταματήσει όταν το πρώτο όρισμα γίνει ίσο με 0, όπου το τελευταίο όρισμα θα εξισωθεί με το δεύτερο όρισμα.**

***Παραδείγματα χρήσης:***

**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα**

Θέμα 3ο

**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.**

**Το σύνολο του προγράμματος**

**1. Εισαγωγικές δηλώσεις και δεδομένα**

**/\*----------------------------------------------------------------------\*/**

**/\* PROJECT ON \*/**

**/\* NATUTAL LANGUAGE PROCESSING (NLP) \*/**

**/\*----------------------------------------------------------------------\*/**

**/\* Author :Themis Panayiotopoulos \*/**

**/\* June 2020 \*/**

**/\* Contributor :Patroklos Georgiadis \*/**

**/\* July 2022 \*/**

**/\*----------------------------------------------------------------------\*/**

**/\* '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 mary.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 \*/**

**/\*----------------------------------------------------------------------\*/**

**/\*----------------------------------------------------------------------\*/**

**/\* story in hide.txt : \*/**

**/\* simon seeks. john hides. simon finds john. \*/**

**/\*----------------------------------------------------------------------\*/**

**:- 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.**

**2. Το κυρίως πρόγραμμα και τα κύρια κατηγορούμενα-μέθοδοι**

**/\*==========================================================\*/**

**/\* MAIN \*/**

**/\*==========================================================\*/**

**%Read story from file, analyse it and store it in the knowledge base**

**understand(Text):-**

**% First Read from File then analyse**

**% Text = 'text.txt'**

**see(Text), /\* open this file to read \*/**

**read(Story), /\* read from File \*/**

**seen, /\* close the input File \*/**

**parse(Story).**

**%Read story from input, analyse it and store it in the knowledge base**

**make\_statements():-**

**write("Make a statement that is true about your story."), nl,**

**read\_string(user\_input, "\n", "\r\t ", \_, Line), /\* get user's statement \*/**

**(lexical\_analyse(Line, [Sentence]),**

**tell(Sentence), confirm\_s()); /\* add entry to the KB \*/**

**(write("Try again"), nl,**

**make\_statements()).**

**%Analyse input and store it in the knowledge base.**

**parse(Story) :-**

**lexical\_analyse(Story, Sentences),**

**write('the lexical analysis completed!'), nl,nl,nl,**

**syntax\_analyse(Sentences, \_),**

**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.**

**%Read the users questions and answer using the knowledge base.**

**make\_questions():-**

**write("Go ahead and ask me something"), nl,**

**read\_string(user\_input, "\n", "\r\t ", \_, Line), /\* get user's question \*/**

**question\_to\_statement(Line, ClearText), /\* remove question marks \*/**

**(lexical\_analyse(ClearText, [Sentence]),**

**ask(Sentence),confirm\_q()); /\* ask question in the KB \*/**

**(write("Try again"),nl,**

**make\_questions()).**

**confirm\_s():-**

**write(' Any other statements? [Yes/No]'), nl,**

**read\_string(user\_input, "\n", "\r\t ", \_, Line),**

**(Line="Yes" -> make\_statements();**

**Line="No" -> write('Ok, bye!');**

**confirm\_s()).**

**confirm\_q():-**

**write(' Any other questions? [Yes/No]'), nl,**

**read\_string(user\_input, "\n", "\r\t ", \_, Line),**

**(Line="Yes" -> make\_questions();**

**Line="No" -> write('Ok, bye!');**

**confirm\_q()).**

**%Remove question marks**

**question\_to\_statement(Question, Statement):-**

**split\_string(Question, "?", "", Q1),**

**atomic\_list\_concat(Q1,".", Q2),**

**split\_string(Q2, ";", "", Q3),**

**atomic\_list\_concat(Q3,".", Statement).**

**3. Λεκτική Ανάλυση**

**/\*==========================================================\*/**

**/\* LEXICAL ANALYSIS \*/**

**/\*==========================================================\*/**

**/\*----------------------------------------------------------------------\*/**

**/\* Reads a text from a file and produces a list of sentences. \*/**

**/\* Each sentence is a list of words \*/**

**/\*----------------------------------------------------------------------\*/**

**lexical\_analyse(Text, Result) :-**

**string\_lower(Text, LowerText), /\* Make all Capital letters to lowercase\*/**

**analyse(LowerText, Result), /\* Lexical Analysis \*/**

**%write\_results(Result), /\* Write Result \*/**

**!. /\* stop now \*/**

**/\*----------------------------------------------------------------------\*/**

**/\* modify \*/**

**/\* 1. turn all capital letters to lowercase \*/**

**/\*----------------------------------------------------------------------\*/**

**/\*----------------------------------------------------------------------\*/**

**/\* analyse \*/**

**/\* 1. turn input 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 \*/**

**/\*==========================================================\*/**

**4. Συντακτική Ανάλυση (Γραμματικοί κανόνες, Λεξιλόγιο)**

**/\*==========================================================\*/**

**/\* 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), % Write 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) \*/**

**/\* Pronoun (p) \*/**

**/\* 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).**

**np(np(N)) --> p(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 new example :**

**iv(iv(seeks))-->[seeks].**

**iv(iv(seek))-->[seek].**

**iv(iv(seeking))-->[seeking].**

**iv(iv(hides))-->[hides].**

**iv(iv(hide))-->[hide].**

**iv(iv(hiding))-->[hiding].**

**% 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].**

**% needed for new example :**

**v(v(finds))-->[finds].**

**v(v(find))-->[find].**

**% 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].**

**pn(pn(simon))-->[simon].**

**/\*---------------------------------------------------------------------\*/**

**/\* Pronoun (p) \*/**

**/\*---------------------------------------------------------------------\*/**

**p(p(it))-->[it].**

**% possible extensions**

**p(p(they))-->[they].**

**p(p(he))-->[he].**

**p(p(she))-->[she].**

**/\*---------------------------------------------------------------------\*/**

**/\* 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. 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).**

**sem\_np(N) --> sem\_p(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 new example :**

**sem\_iv(seeks,s)-->[seeks].**

**sem\_iv(seeks,q)-->[seeking].**

**sem\_iv(hides,s)-->[hides].**

**sem\_iv(hides,q)-->[hiding].**

**% 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].**

**% needed for new example :**

**sem\_v(finds,s) -->[finds].**

**sem\_v(finds,q) -->[find].**

**% 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].**

**sem\_pn(simon) -->[simon].**

**/\*---------------------------------------------------------------------\*/**

**/\* Pronouns (sem\_p) \*/**

**/\*---------------------------------------------------------------------\*/**

**% needed**

**sem\_p(it) -->[it].**

**% possible extensions**

**sem\_p(they) -->[they].**

**sem\_p(he) -->[he].**

**sem\_p(she) -->[she].**

**/\*---------------------------------------------------------------------\*/**

**/\* Determiner (det) \*/**

**/\*---------------------------------------------------------------------\*/**

**% needed for example :**

**sem\_det(the) -->[the].**

**% extension of vocabulary**

**sem\_det(a) -->[a].**

**sem\_det(an) -->[an].**

**/\*==========================================================\*/**

**/\* END OF SEMANTICS VOCABULARY \*/**

**/\*==========================================================\*/**

**6. Ενημέρωση και Ερωταποκρίσεις 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(X) :- listing(kb\_fact(X)).**

**/\*---------------------------------------------------------------------\*/**

**/\* 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(did), 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 \*/**

**/\*==========================================================\*/**

**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), !.**

**8. 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 δε \*/**

**/\* 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 αντικείμενa. 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 γνώση \*/**

**/\* 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) \*/**

**/\*------------------------------------------------------------------------\*/**

**/\*------------------------------------------------------------------------\*/**

**/\* Pronoun \*/**

**/\*------------------------------------------------------------------------\*/**

**/\* Pronoun (pronoun): Οι αντωνυμίες, χρησιμοποιούνται με τον ίδιο τρόπο που \*/**

**/\* χρησιμοποιούνται τα Proper Nouns \*/**

**/\*------------------------------------------------------------------------\*/**

**Επεξήγηση των προσθέσεων στο πρόγραμμα**

Έχοντας χρησιμοποιήσει τον κώδικα που μας δόθηκε, προστέθηκαν κάποια νέα κομμάτια κώδικα για την εισαγωγή νέων πληροφοριών από το πληκτρολόγιο και για την εκπλήρωση ερωτήσεων στην βάση γνώσεων (πάλι από το πληκτρολόγιο). Θα εξηγήσω τα νέα κομμάτια κώδικα ανά τμήμα του προγράμματος:

**Α. Λεκτική ανάλυση**

Το τμήμα της λεκτικής ανάλυσης είναι σχεδόν ίδιο με το αρχικό με την εξαίρεση της προσθήκης του κανόνα string\_lower/2 της Prolog. Χρησιμοποιείται πλέον ως η αρχική ενέργεια του κανόνα lexical\_analyse/2, μετατρέποντας όλους τους χαρακτήρες του κειμένου που δίνεται ως το πρώτο όρισμα του lexical\_analyse, σε μικρούς (Lowercase) χαρακτήρες. Αυτό πραγματοποιείται για την διευκόλυνση της λεκτικής ανάλυσης αλλά και αργότερα για την συντακτική και σημασιολογική ανάλυση, καθώς δεν τίθεται πλέον αναγκαίο ο χρήστης να εισάγει όλες τις προτάσεις του μόνο με μικρά γράμματα.

**Β. Συντακτική ανάλυση**

Στο κομμάτι της συντακτικής ανάλυσης, οι περισσότερες αλλαγές αφορούν το λεξικό του προγράμματος, καθώς εισάγουμε καινούργια ουσιαστικά (πρόσωπα), ρήματα (συγκεκριμένα τύπου intransitive) και ένα καινούργιο μέρος του λόγου, αντωνυμίες (pronouns).

Αναλυτικότερα, τα νέα πρόσωπα είναι: «simon», ενώ τα νέα ρήματα (κανονικά και intransitive) είναι τα «find» και «hide», «seek» και οι αντίστοιχοι τύποι τους σε άλλα πρόσωπα ή/και στον ενεστώτα διαρκείας. Ύστερα, οι αντωνυμίες που προσθέσαμε είναι το «it», «they», «she» και «he» (π.χ p(p(it)) --> [it].).

Φυσικά για να μπορούμε να αναγνωρίσουμε αυτές τις αντωνυμίες, πρέπει να προστεθούν σε κάποιους γραμματικούς κανόνες. Συγκεκριμένα, έχει προστεθεί ένας νέος γραμματικός κανόνας για την αναγνώριση αντωνυμιών στους κανόνες που αφορούν τις ονοματικές προτάσεις, το np(np(N)) --> p(N).

**Γ. Σημασιολογική ανάλυση**

Παρόμοια με την συντακτική ανάλυση, έχουν προστεθεί σε αντίστοιχη μορφή κανόνων το νέο πρόσωπο «simon» και τα νέα ρήματα (κανονικά και intransitive) «find» σε μορφή ενεστώτα και «hide», «seek» σε μορφή ενεστώτα και ενεστώτα διαρκείας για την σημασιολογική ανάλυση των ιστοριών μας. Ταυτόχρονα, έχουν επίσης προστεθεί οι αντωνυμίες που αναφέραμε παραπάνω (it, they, she, he) με την μορφή κανόνων για την σημασιολογική ανάλυση (π.χ. sem\_p(it) --> [it].). Όσο αναφορά τους σημασιολογικούς κανόνες, έχει προστεθεί ένας νέος κανόνας όπου αφορά τις αντωνυμίες και είναι αντίστοιχος με αυτόν που χρησιμοποιήσαμε στην συντακτική ανάλυση: sem\_np(N) --> sem\_p(N).

**Δ. Κύριο πρόγραμμα**

Τέλος, ώρα να αναφερθούμε στις αλλαγές του κύριου προγράμματος και στα νέα κατηγορήματα που μας επιτρέπουν να εισάγουμε νέες πληροφορίες από το πληκτρολόγιο και να κάνουμε ερωτήσεις στην βάση γνώσης από το πληκτρολόγιο.

* understand/1: ίδια με προηγουμένως.
* parse/1: παρόμοια με την understand, αλλά αντί για αρχείο δέχεται ως είσοδο string.
* make\_statements/0: Το κατηγόρημα που χρησιμοποιούμε ώστε ο χρήστης να εισάγει νέες

πληροφορίες από το πληκτρολόγιο.

Αρχικά, παροτρύνουμε τον χρήστη να εισάγει μια πρόταση για την ιστορία του, δηλαδή να εισάγει μια νέα πληροφορία που επιθυμεί και διαβάζουμε την πρόταση που γράφει. Ελέγχουμε πως είναι γραμματικά σωστή και την εισάγουμε στην βάση γνώσης με το κατηγόρημα tell/1, αλλιώς ανακοινώνουμε το λάθος στον χρήστη και επαναλαμβάνουμε την διαδικασία. Έπειτα, ρωτάμε τον χρήστη εάν επιθυμεί να γράψει και άλλη πρόταση στην βάση γνώσεων με το κατηγόρημα confirm\_s/0. Αν απαντήσει ναι, επαναλαμβάνουμε την διαδικασία επιστρέφοντας στο make\_statements(). Αν όχι, τελειώνει o αλγόριθμος.

* make\_questions/0: Το κατηγόρημα που χρησιμοποιούμε ώστε ο χρήστης να

πραγματοποιήσει ερωτήσεις.

Όπως και στο make\_statements, παροτρύνουμε τον χρήστη να πραγματοποιήσει μια ερώτηση για την ιστορία του (να κάνει μια ερώτηση στην βάση γνώσεων). Διαβάζοντας την ερώτηση του, μετατρέπουμε όλα τα ερωτηματικά «?/;» σε τελείες «.», με το κατηγόρημα question\_to\_statement/2, για να πραγματοποιήσουμε την λεκτική ανάλυση της πρότασης. Αν είναι γραμματικά σωστή, παραθέτουμε το ερώτημα στην βάση γνώσης με το κατηγόρημα ask/1. Αν δεν είναι, ανακοινώνουμε το λάθος στον χρήστη και επαναλαμβάνουμε την διαδικασία. Ρωτάμε τον χρήστη εάν επιθυμεί να κάνει και άλλη ερώτηση στην βάση γνώσεων με το κατηγόρημα confirm\_q/0. Αν απαντήσει ναι, επαναλαμβάνουμε την διαδικασία επιστρέφοντας στο make\_questions(). Αν όχι, τελειώνει o αλγόριθμος.

**Παραδείγματα εκτέλεσης**

**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα**

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα