Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

«**Επεξεργασία Φυσικής Γλώσσας**»

Απαλλακτική εργασία

Ιουνίου – Σεπτεμβρίου 2022

ΙΩΑΝΝΗΣ-ΙΑΣΩΝ ΜΠΟΪΔΑΝΗΣ Π19217

**Καθηγητής Θεμιστοκλής Παναγιωτόπουλος**

Θέμα 1ο

Α,B) Η προσκείμενη πρόταση σε μορφή DCG μπορεί να αναγνωριστεί στα εξής γραμματικά χαρακτηριστικά:

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].

Για να παράξει η Prolog το ίδιο το συντακτικό δένδρο σαν έξοδο, διαμορφώνουμε τους κανόνες 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].

Με την παραπάνω γραμματική μπορεί να αναπαρασταθεί το συντακτικό δέντρο και 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

Λύση προβλήματος για πρόσθεση και αφαίρεση μόνο:

<https://stackoverflow.com/questions/29380105/prolog-binary-addition>

Μία δεύτερη εσφαλμένη λύση μεε βάση δύο μετατροπές:

<https://stackoverflow.com/questions/67839517/convert-binary-to-decimal-predicate>

<https://stackoverflow.com/questions/47906462/converting-base-10-to-base-2-in-prolog>

και η Τρίτη λύση με βάση μόνο:

<https://stackoverflow.com/questions/47906462/converting-base-10-to-base-2-in-prolog>

Ολοκληρωμένος κώδικας 1ης και 2ης λύσης:

Μετατρέπω τους δυαδικούς αριθμούς σε δεκαδικούς, κάνω τις πράξεις ,κι τους ξαναμετατρέπω σε δυαδικούς, έπειτα τους εμφανίζω στην οθόνη.

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.

dec2Bin(0,V,\_,V).

dec2Bin(N,V,Counter,Val):-

Reminder is N mod 2,

N1 is N//2,

V1 is V + Reminder\*(10^Counter),

Counter1 is Counter + 1,

dec2Bin(N1,V1,Counter1,Val).

convert(N,V):-

N > -1,

dec2Bin(N,0,0,V),

writeln(V).

tobin(0,0).

tobin(S,X) :- X > 0 ,

X1 is X // 2 ,

tobin(S1,X1),

S0 is X mod 2 ,

S is S0 + S1 \* 10 .

cbd(Binary, Decimal) :-

number\_codes(Binary, Codes),

number\_codes(Decimal, [0'0, 0'b| Codes]).

binary\_expression(X,+,Y,Ans):-cbd(X,A),cbd(Y,B),expression(V,[A,+,B],[]),convert(V,Ans).

binary\_expression(X,-,Y,Ans):-cbd(X,A),cbd(Y,B),expression(V,[A,-,B],[]),convert(V,Ans).

binary\_expression(X,\*,Y,Ans):-cbd(X,A),cbd(Y,B),expression(V,[A,\*,B],[]),convert(V,Ans).

binary\_expression(X,/,Y,Ans):-cbd(X,A),cbd(Y,B),expression(V,[A,/,B],[]),convert(V,Ans).

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Αλλά αυτή η λύση υστερεί σε αριθμούς μεγαλύτερους του 9,

άρα σαν μία τελική λύση έχω(thema2final\_p19217.pl):

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\*2^N+Y}.

digit(0) --> [0].

digit(1) --> [1].

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

numberofdigits(Y,N) :-

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

numberofdigits(Z,N1),

N is N1+1,!.

dec2Bin(0,V,\_,V).

dec2Bin(N,V,Counter,Val):-

Reminder is N mod 2,

N1 is N//2,

V1 is V + Reminder\*(10^Counter),

Counter1 is Counter + 1,

dec2Bin(N1,V1,Counter1,Val).

convert(N,V):-

N > -1,

dec2Bin(N,0,0,V).

To εκτελώ για κάποιες εκφράσεις:Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματατο 21 είναι το 10101, το 7 το 111, κ.ο.κ

Πράγματι

Μία τρίτη λύση για πρόσθεση και αφαίρεση:

μετατρέπω το αποτέλεσμα του κώδικα σε κανονικό, αυτό που διαβάζεται από αριστερά προς τα δεξιά και χρειάζεται να πάρω ως είσοδο τις ανάποδες λίστες για να διαβάζονται σωστά. Στην δική μου εκδοχή η αριθμητική πράξη θα γίνεται ανάμεσα 2 μόνο αριθμών, όπου οι δυαδικοί αριθμοί θα βρίσκονται σε λίστες.

:- use\_module(library(clpfd)). %for #=, which gives value on the right

%hand side which otherwise would be imposible because X and Y dont have

%a value before we run the program. simply using "is" doesnt work

expression (Xs, +, Ys,Ans):-reverse(Xs,[],X),reverse(Ys,[],Y),binary\_addition(X, Y, As),reverse(As,[],Ans).

binary\_addition(Xs, Ys, As):-

phrase(binary\_addition\_(Xs, Ys, 0), As).

/\*The SWI-Prolog implementation of phrase/3 verifies that the List and

Rest arguments are unbound, bound to the empty list or a list cons cell.

Other values raise a type error.\*/

binary\_addition\_([], [], 0) --> [0].

binary\_addition\_([], [], 1) --> [1].

binary\_addition\_([X|Xs], [], C) --> binary\_addition\_([X|Xs], [C], 0). %adds whatever is left with the carry

binary\_addition\_([], [Y|Ys], C) --> binary\_addition\_([C], [Y|Ys], 0).%bit zero

binary\_addition\_([X|Xs], [Y|Ys], C0) -->

{ [X,Y] ins 0..1,

Sum #= X + Y + C0 }, %at first time,the carry bit is 0

sum\_carry(Sum, C),

binary\_addition\_(Xs, Ys, C).

sum\_carry(0, 0) --> [0].

sum\_carry(1, 0) --> [1].

sum\_carry(2, 1) --> [0].

reverse([], Y, R) :-

R = Y.

reverse([H|T] , Y, R) :-

reverse(T, [H|Y], R).

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Πράγματι 9+7=16,4+7=11=1011,2+7=9,2+3=5

Στο 2ο παράδειγμα οι αριθμοί στο πρόγραμμα διαβάζονται σαν «001» κι έπειτα αναστρέφονται για να γίνει σωστά η πράξη όταν υπάρχει διαφορετικό πλήθος ψηφίων μεταξύ των αριθμών.

Για την αφαίρεση προϋποθέτω ότι ο πρώτος αριθμός είναι μεγαλύτερος από τον δεύτερο για αποφυγή λαθών στην κατανόηση των αρνητικών προσημασμένων bit καθώς του ότι το πρόγραμμα βγάζει σαν αποτέλεσμα το συμπλήρωμα του αριθμού.

Υλοποιώντας ουσιαστικά την xor, θέλω όταν η αφαίρεση των bit κάνει 0 να έχει την τιμή 0,και όταν 1 την τιμή 1, αλλά για να μην χρειάζεται να βάλω απόλυτες τιμές στα Χ και Υ, παίρνω το άθροισμα, όταν κάνει 1 να παίρνει την τιμή 1 και στις άλλες 2 περιπτώσεις την τιμή 0.

binary\_sub(Xs, Ys, As):-

phrase(binary\_sub\_(Xs, Ys), As).

binary\_sub\_([],[])-->[].

binary\_sub\_([X|Xs], []) --> [].

binary\_sub\_([], [Y|Ys]) --> [].

binary\_sub\_([X|Xs], [Y|Ys]) -->

{ [X,Y] ins 0..1,

Sub #= X + Y}, %για να μην βγαινει αρνητικο

sub\_carry(Sub),

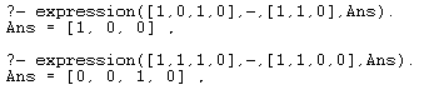
binary\_sub\_(Xs, Ys).

sub\_carry(0)--> [0].

sub\_carry(1)--> [1].

sub\_carry(2)--> [0].

expression(Xs,-, Ys,Ans):-reverse(Xs,[],X),reverse(Ys,[],Y),binary\_sub(X, Y, As),reverse(As,[],Ans).



Πράγματι 10-6=4,14-12-2

Θέμα 3ο

Κώδικας:

\*αλλαγές στις σελ 9,15-16,19

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

/\* 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.

/\* MAIN , the main program\*/

%Reading the file, analysing it and storing it in the knowledge base

understand(Text):-

%Text = 'voice.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\_sentence()); /\* 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('comlpeting lexical analysis....and done! lexical analysis completed!'), nl,nl,nl,

syntax\_analyse(Sentences, \_),

write('comlpeting syntactic analysis....and done! syntactic analysis completed!'),nl,nl,nl,

semantics\_analyse(Sentences, Semantics),

write('comlpeting sematic analysis....and done! 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("Ask a question"), nl,

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

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

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

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

(write("Try again"),nl,

make\_questions()).

confirm\_sentence():-

write(' Want to make another statement? [Yes/No]'), nl,

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

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

Line="No" -> write('See you later!');

confirm\_s()).

confirm\_q():-

write(' Want to make any other questions? [Yes/No]'), nl,

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

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

Line="No" -> write('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).

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

/\* 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 \*/

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

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

/\* 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],[]). \*/

/\* ?- qstn(Structure, [is, it, a, dog], []). \*/

/\* 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 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].

tv(tv(brings)) -->[brings].

tv(tv(brings)) -->[bring].

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

tv(tv(bringing)) -->[bringing].

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

/\* 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].

v(v(plays))-->[plays].

v(v(play))-->[play].

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

/\* 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].

n(n(basketball))-->[basketball].

n(n(football))-->[football].

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

/\* Proper Nouns (pn) \*/

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

pn(pn(mary))-->[mary].

pn(pn(john))-->[john].

pn(pn(tomy))-->[tomy].

pn(pn(electra))-->[electra].

pn(pn(iason))-->[iason].

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

/\* Determiner (det) \*/

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

% needed for example :

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

% extension of vocabulary

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

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

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

/\* END OF SYNTACTIC ANALYSIS \*/

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

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

/\* 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).

sem\_vp(3,V,N) --> sem\_v(V,q), sem\_np(N).

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

/\* 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].

sem\_tv(brings,s) -->[brings].

sem\_tv(brings,q) -->[bring].

sem\_tv(brought,s) -->[brought].

sem\_tv(bringing,q2) -->[bringing].

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

/\* 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].

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

sem\_v(play,q) -->[play].

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

/\* 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(basketball) -->[basketball].

sem\_n(football) -->[football].

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(electra)-->[electra].

sem\_pn(iason)-->[iason].

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

/\* 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 \*/

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

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

/\* 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 \*/

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

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

/\* 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), !.

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

/\* 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 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 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) \*/

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

Η βάση γνώσης ενημερώνεται για τις εξής προτάσεις:

Είναι συντακτικά λεκτικά και σημασιολογικά ορθές οπότε η βάση τις δέχεται και επιστρέφει true

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Κάνω κάποιες ερωτήσεις:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρώ ότι είναι όλες οι ερωτήσεις ερωτώνται σωστά, και όσες υπάρχουν στην βάση επιστρέφουν yes.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα



Βλέπουμε τι θα βγάλει αν ρωτήσουμε κάτι συντακτικά λάθος

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

