Λογικός Προγραμματισμός με Περιορισμούς
Τμ. Εφαρμοσμένης Πληροφορικής

Εργασία 1 (2019-2020)

ΠΑΠΑΓΕΩΡΓΙΟΥ ΓΕΩΡΓΙΟΣ
dai17233
dai17233@uom.edu.gr

1. List Processing

Κώδικας:

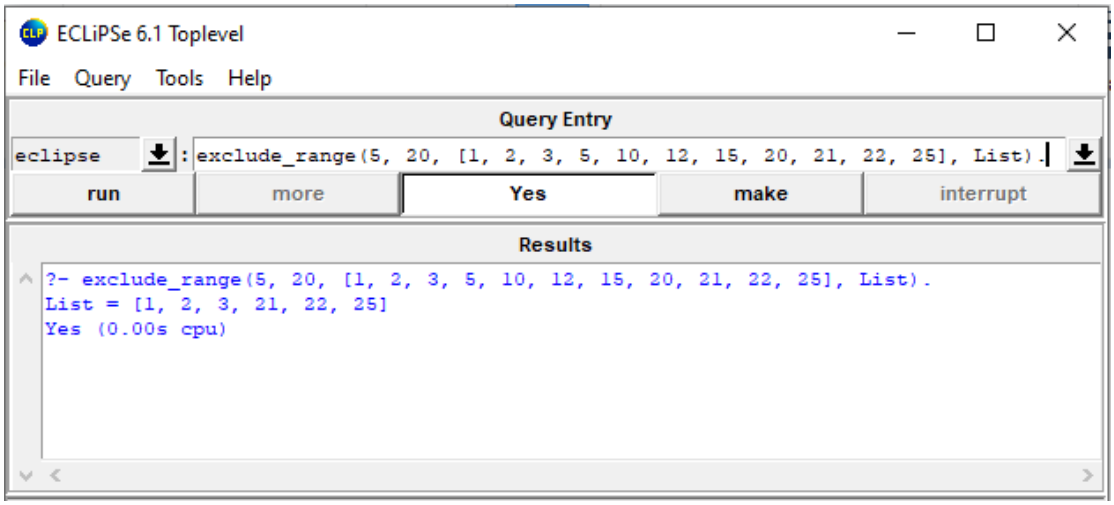
```
%%% Exec 1

%%% exclude_range/4
%%% exclude_range(Low, High, List, NewList)
%%% Sucess when given a list of integers (3rd argument),
%%% the NewList list contains all integers that do NOT belong
%%% to the closed space defined by the first two Low and High
arguments.exclude_range(_, _, [], []).
exclude_range(Low, High, [A|List], [A|NewList]) :-
        ( A<Low ; A>High ),
        !,
         exclude_range(Low, High, List, NewList).

exclude_range(Low, High, [A|List], NewList) :-

        ( A>=Low ; A=<High ),
        !,
        exclude_range(Low, High, List, NewList).
```

Σχολιασμός κώδικα:

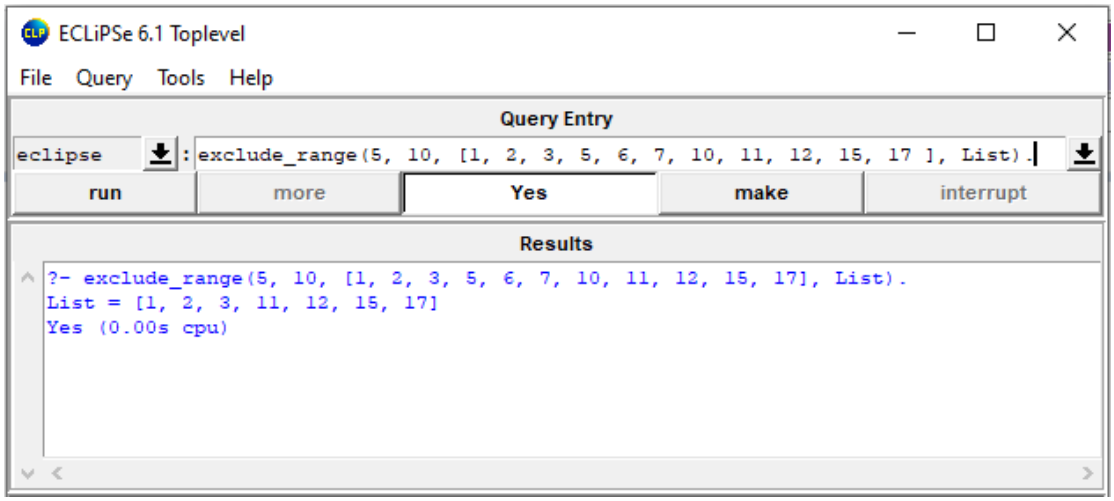
Ο κώδικας του παραπάνω Prolog κατηγορήματος **exclude_range(Low, High, List, NewList)** ελέγχει για κάθε στοιχείο(ακέραιος) της λίστας **List**, αν ΔΕΝ ανήκει στο κλειστό διάστημα **Low-High**.
Αν το αποτέλεσμα του ελέγχου **A<Low ή A>High** είναι **αληθές**, το στοιχείο προστίθεται στην λίστα **NewList**. **Αλλιώς**, εκτελείτε ο παρακάτω έλεγχος **A>=Low ή A=<High** για το τρέχον στοιχείο της λίστας και ανάλογα με το αποτέλεσμα, το στοιχείο προστίθεται ή όχι στην λίστα **NewList**. Χρησιμοποιώ **cut(!)** μετά τους ελέγχους για την αποφυγή εύρεσης επιπλέον λύσεων από την Prolog.
Στην περίπτωση που η είσοδος είναι κενή λίστα **List[ ]**, το κατηγόρημα, επιστρέφει επίσης κενή λίστα **NewList[ ].**

Παραδείγματα εκτέλεσης:

- ?- exclude_range(5, 20, [1, 2, 3, 5, 10, 12, 15, 20, 21, 22, 25], List).
  List = [1, 2, 3, 21, 22, 25]
  Yes (0.00s cpu)



- ?- exclude_range(5, 10, [1, 2, 3, 5, 6, 7, 10, 11, 12, 15, 17], List).
  List = [1, 2, 3, 11, 12, 15, 17]
  Yes (0.00s cpu)



Bugs και προβλήματα που έχει ο κώδικας:

Δεν έχω παρατηρήσει κάποιο πρόβλημα/bug στον κώδικα.

2. Matching Number Series

Κώδικας:

```
%%% Exec 2

%%% Relations Available for Exec 1
double(X,Y):-Y is X * 2.
inc(X,Y):-Y is X + 1.
square(X,Y):- Y is X*X.


%%% math_match/3
%%% math_match(List, C, Solution)
%%% Succeed when arguments satisfy a condition
math_match([H|T], C, Solution):-
        onePair([H,T|_], [X,Y]),
        maplist(C, X, Y),!,
        pair([H|T], Solution),!.

%%% onePair/2
%%% onePair(List, PairList)
%%% Return a consecutive pair(one at a time)
onePair([X,Y|_], [X,Y]).
onePair([_|Tail], XY):-
        onePair(Tail,XY).

%%% pair/2
%%% pair(List, ListConsPairs)
%%% Return a list of all consecutive pairs
pair([_],[]).
pair([X,Y|T],[[X,Y]|T1]):-
   pair([Y|T],T1).


%%% math_match_alt/3
%%% math_match_alt/3(List, C, Solution)
%%% Succeed when arguments satisfy a condition
math_match_alt([H|T], C, Solution):-
        onePair([H,T|_], [X,Y]),
        maplist(C, X, Y),!,
        findall((X,Y),append(_,[X,Y|_],[H|T]),Solution).
```

<u>Σχολιασμός κώδικα:</u>

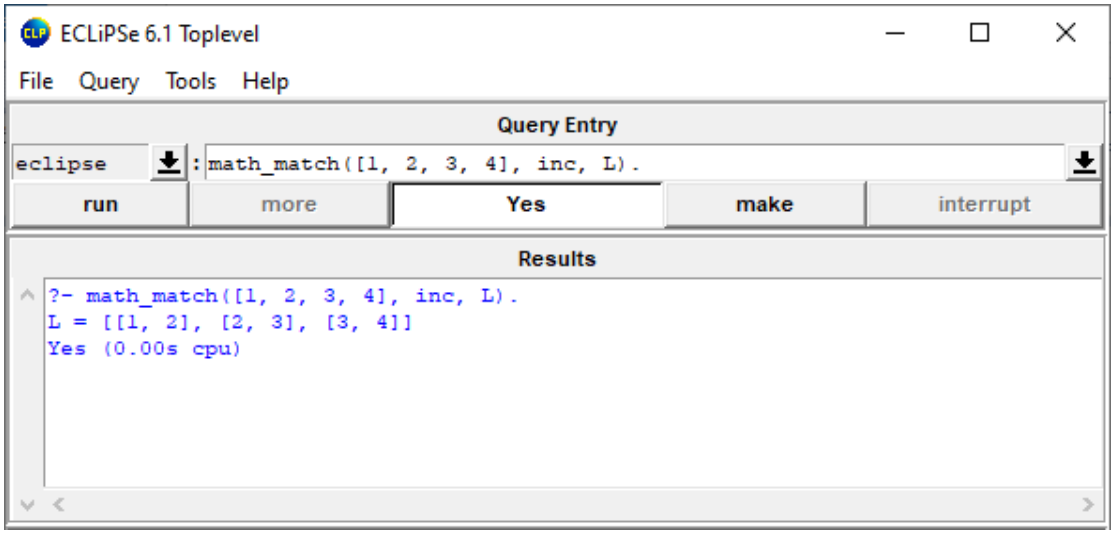
- <u>Αναδρομικός ορισμός (math_match/3):</u>
  Το κατηγόρημα **math_match/3** καλεί το κατηγόρημα **onePair/2** το οποίο δέχεται την λίστα **List** και επιστρέφει τα **ζεύγη** των **διαδοχικών στοιχείων** της λίστας **List(ένα την φορά**). Έπειτα, με την χρήση του **build-in** κατηγορήματος **maplist/3** καλείται οποιοδήποτε κατηγόρημα **arity/2 (C),** με ορίσματα τα **ζεύγη** των διαδοχικών στοιχείων της λίστας **List**.
  Εάν ο έλεγχος είναι **αληθής**, καλείται το κατηγόρημα **pair/2**, το οποίο δέχεται την λίστα **List** και επιστρέφει μια λίστα **Solution** με όλα τα **ζεύγη** των διαδοχικών στοιχείων της λίστας **List**. Αλλιώς, το **math_match/3** επιστρέφει μια κενή λίστα Solution.

- <u>Μη αναδρομικός ορισμός (math_match_alt/3):</u>
  Το κατηγόρημα **math_match_alt/3** καλεί το κατηγόρημα **onePair/2** το οποίο δέχεται την λίστα **List** και επιστρέφει τα **ζεύγη** των **διαδοχικών στοιχείων** της λίστας **List(ένα την φορά**). Έπειτα, με την χρήση του **build-in** κατηγορήματος **maplist/3** καλείται οποιοδήποτε κατηγόρημα **arity/2 (C),** με ορίσματα τα **ζεύγη** των διαδοχικών στοιχείων της λίστας **List**.
  Εάν ο έλεγχος είναι **αληθής**, καλείται το **build-in** κατηγόρημα **findall/3,** το οποίο λίστες(μια τη φορά) με τα **ζεύγη** των **διαδοχικών στοιχείων** της λίστας **List,** καλεί το **build-in** κατηγόρημα **append/3** και με την χρήση **backtracking, ενοποιεί** το αποτέλεσμα στο όρισμα **Solution**. Αλλιώς, το **math_match_alt/3** επιστρέφει μια κενή λίστα Solution.

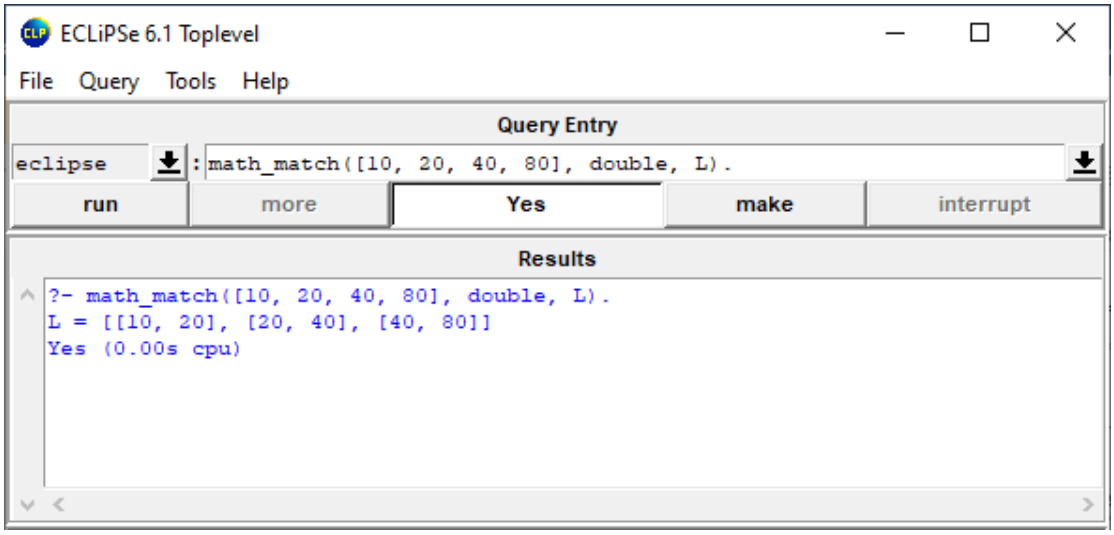
Παραδείγματα εκτέλεσης:

1. Αναδρομικός ορισμός (math_match/3):

- ?- math_match([1, 2, 3, 4], inc, L).
  L = [[1, 2], [2, 3], [3, 4]]
  Yes (0.00s cpu)

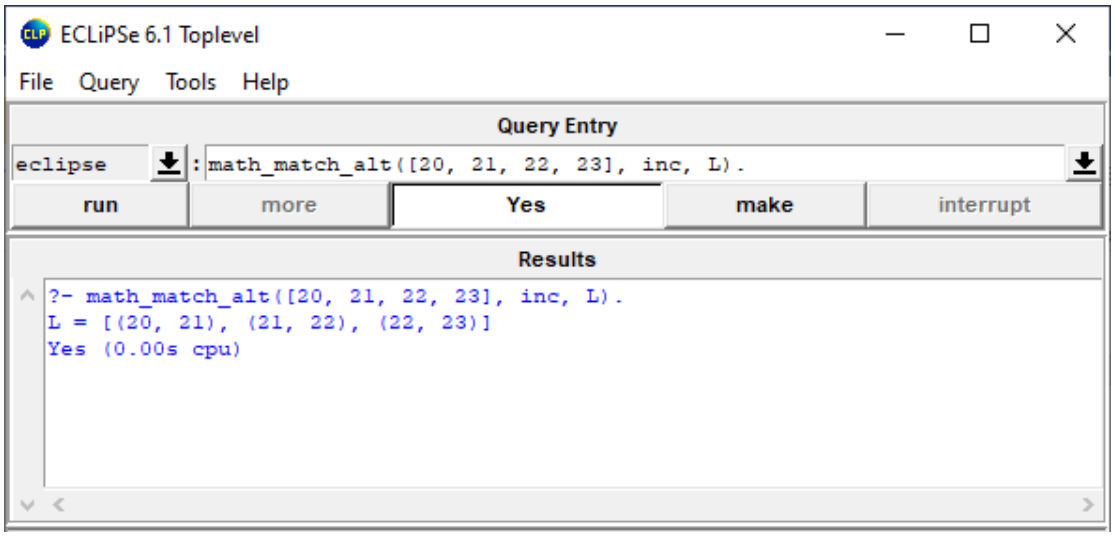


- ?- math_match([10, 20, 40, 80], double, L).
  L = [[10, 20], [20, 40], [40, 80]]
  Yes (0.00s cpu)

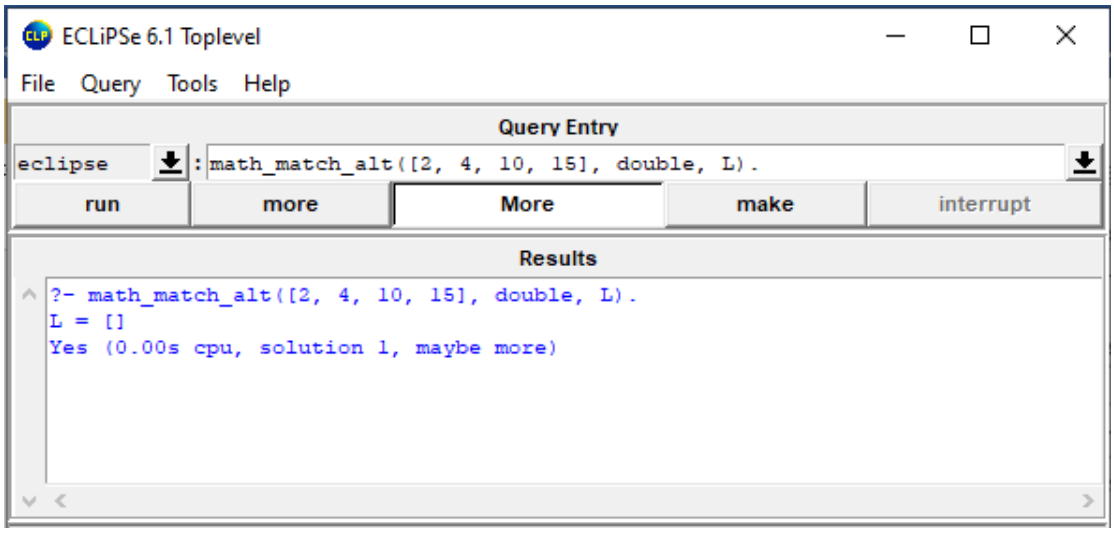


2. Μη αναδρομικός ορισμός (math_match_alt/3):

- ?- math_match_alt([20, 21, 22, 23], inc, L).
  L = [(20, 21), (21, 22), (22, 23)]
  Yes (0.00s cpu)



- ?- math_match_alt([2, 4, 10, 15], double, L).
  L = []
  Yes (0.00s cpu, solution 1, maybe more)



Bugs και προβλήματα που έχει ο κώδικας:

- Όταν το **κατηγόρημα math_match_alt/3** επιστρέφει **κενή λίστα**, μου επιτρέπει να πατήσω το πλήκτρο **more** για **επιπλέον λύσεις**, ενώ στα ενδεικτικά παραδείγματα επιστρέφει μόνο μια λύση. (Λογικά το πρόβλημα λύνεται με την χρήση **cut(!),** αλλά δεν κατάφερα να το κάνω να δουλέψει σωστά, οπότε το άφησα ως έχει.)