Λογικός Προγραμματισμός με Περιορισμούς
Coursework 1

Απόστολος Τσέλιος
dai17115

Μάιος 2, 2020

Περιεχόμενα

Άσκηση 1 - List Processing

Ορισμός κατηγόρηματος exclude_range/4

Ο παρακάτω κώδικας υλοποιεί αναδρομικά το κατηγόρημα exclude_range/4 με ορίσματα Low, High τα οποία ορίζουν το διάστημα, List η δοθείσα λίστα ακεραίων και New List μια λίστα που περιέχει όλους τους ακεραίους που δεν ανήκουν στο κλειστό διάστημα [Low, High].

```
%%% exclude_range/4
exclude_range(_Low, _High, [], []).

exclude_range(Low, High, [Element|List], [Element|NewList]) :-
    Element < Low,
    !,
    exclude_range(Low, High, List, NewList).

exclude_range(Low, High, [Element|List], [Element|NewList]) :-
    Element > High,
    !,
    exclude_range(Low, High, List, NewList).

exclude_range(Low, High, [_Element|List], NewList) :-
    exclude_range(Low, High, List, NewList).
```

Επεξήγηση υλοποίησης

Η βασική περίπτωση της αναδρομής είναι όταν το 3ο όρισμα List είναι η κενή λίστα:

```
exclude_range(_Low, _High, [], []).
```

Σε αυτό το σημείο αρχικοποιείται και το 4ο όρισμα NewList, η λίστα που περιέχει το αποτέλεσμα. Αυτό ήταν το μη-αναδρομικό μέρος του ορισμού.

Στο αναδρομικό μέρος του ορισμού αποδομείται η δοθέισα λίστα στοιχείο-στοιχείο. Υπάρχουν 3 περιπτώσεις. Πρώτον το στοιχείο να είναι μικρότερο του Low και δεύτερον να είναι μεγαλύτερο του High επομένως να ανήκει στο διάστημα και να προστείθεται στη NewList.

```
exclude_range(Low, High, [Element|List], [Element|NewList]) :-
    Element < Low,
    !,
    exclude_range(Low, High, List, NewList).

exclude_range(Low, High, [Element|List], [Element|NewList]) :-
    Element > High,
    !,
    exclude_range(Low, High, List, NewList).
```

Το Cut (!) φροντίζει να διαγράψει τις άλλες αναλλακτικές λύσεις όταν κάποιο στοιχείο ικανοποιεί έναν περιορισμό. Δηλαδή αν ο περιορισμός:

```
Element < Low
```

ικανοποιείται θα εκτελεστεί το Cut οπότε δεν θα ελεχθεί αν

```
Element > High
```

Τέλος ο εναλλακτικός ορισμός του κατηγορήματος:

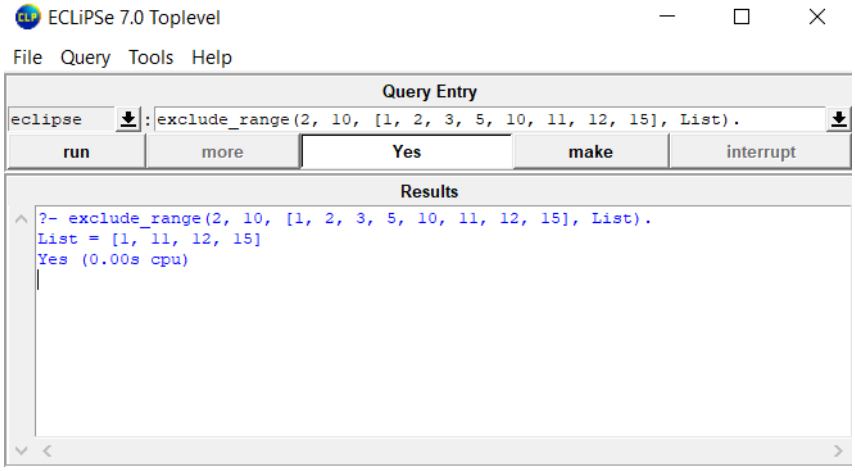
```
exclude_range(Low, High, [_Element|List], NewList) :-
    exclude_range(Low, High, List, NewList).
```

εκτελείται όταν το στοιχείο (Element) δεν ανήκει μέσα στο διάστημα [Low, High] οπότε το προσπερνάει.

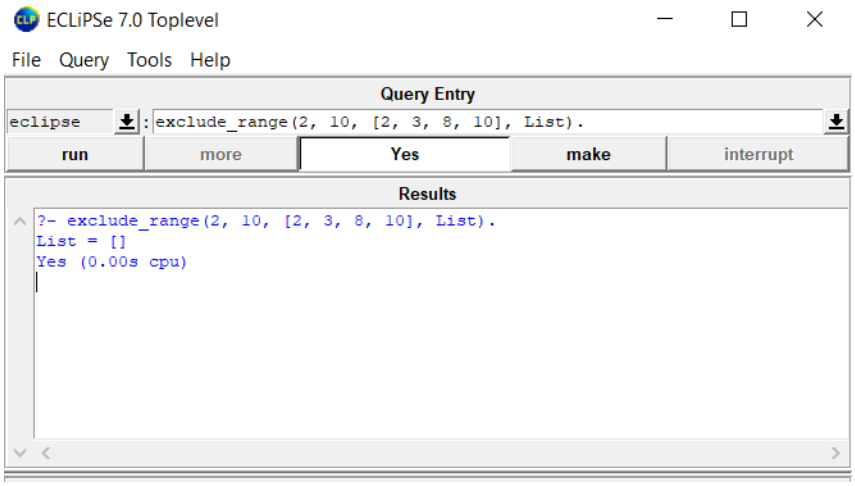
Αριθμός Ελέγχων

Ο αριθμός των ελέχνω που θα κάνει το κατηγόρημα στην χειρότερη περίπτωση ειναι N^2 καθώς θα χρειαστεί να ελέγξει αν το στοιχείο είναι μικρότερο του Low και αν είναι μικρότερο του High. Αυτή η περίπτωση συμβαίνει αν όλα τα στοιχεία της δοθείσας λίστας δεν ανήκουν στο διάστημα [Low, High].

Στιγμιότυπα εκτέλεσης exclude_range/4



Σχήμα 1: Στιγμιότυπο εκτέλεσης του κατηγορήματος exclude_range/4.



Σχήμα 2: Στιγμιότυπο εκτέλεσης του κατηγορήματος exclude_range/4.

Άσκηση 2 - Matching Number Series

Ορισμός κατηγόρηματος math_match/3

Ο παρακάτω κώδικας υλοποιεί αναδρομικά το κατηγόρημα math_match/3 με 1ο όρισμα μια δοθείσα λίστα, 2ο όρισμα το όνομα ενός κατηγορήματος τάξης 2 και 3ό όρισμα μία λίστα που περιέχει ζεύγη διαδοχικών στοιχείων της δοθείσας λίστας (1ο όρισμα) που ικανοποιούν το κατηγόρημα ταξης 2 (2ο όρισμα). Τα κατηγορήματα double/2, inc/2, square/2 είναι κάποια παραδείγματα κατηγορημάτων τάξης 2 για να δωθούν ως 2ο όρισμα στο math_match/3.

```
%%% double/2
double(X, Y):-Y is X * 2.

%%% inc/2
inc(X, Y):-Y is X + 1.

%%% square/2
square(X,Y):- Y is X * X.

%%% math_match/3
math_match([_X], _C, []).

math_match([X, Y|Rest], C, [(X,Y)|Solution]) :-
    Predicate =.. [C, X, Y],
    call(Predicate),
    math_match([Y|Rest], C, Solution), !.

math_match([_X, Y|Rest], C, Solution) :-
    math_match([Y|Rest], C, Solution).
```

Επεξήγηση υλοποίησης

Η βασική περίπτωση της αναδρομής για το math_match/3 είναι η δοθείσα λίστα να περιέχει μόνο ένα στοιχείο καθώς δεν θα υπάρχει επόμενο για να υπολογιστεί ζευγός. Σε αυτή την περίπτωση αρχικοποιείται το 3ο ορισμά, η λίστα των αποτελεσμάτων, με την κενή λίστα.

```
math_match([_X], _C, []).
```

Στο μη αναδρομικό μέρος του κατηγορήματος υπάρχουν δύο περιπτώσεις. Στην πρώτη περίπτωση δυο διαδοχικά στοιχεία της λίστας ικανοποιούν το κατηγόρημα τάξης 2, C (2ο όρισμα), οπότε το ζεύγος προστίθεται στην λίστα Solution.

```
math_match([X, Y|Rest], C, [(X,Y)|Solution]) :-
    Predicate =.. [C, X, Y],
    call(Predicate),
    math_match([Y|Rest], C, Solution), !.
```

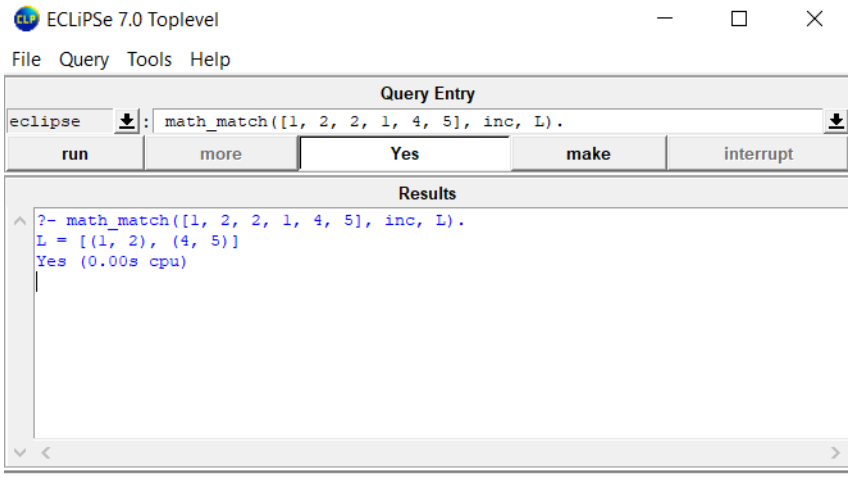
Το Cut (!) στο τέλος του κατηγορήματος διαγράφει τις εναλλακτικές λύσεις ώστε να μας επιστρέφει μία και μοναδική.
Στην δεύτερη περίπτωση το διαδοχικό ζεύγος δεν ικανοποιεί το C (2ο όρισμα), οπότε δεν προστίθεται στην λίστα Solution.

```
math_match([_X, Y|Rest], C, Solution) :-
    math_match([Y|Rest], C, Solution).
```

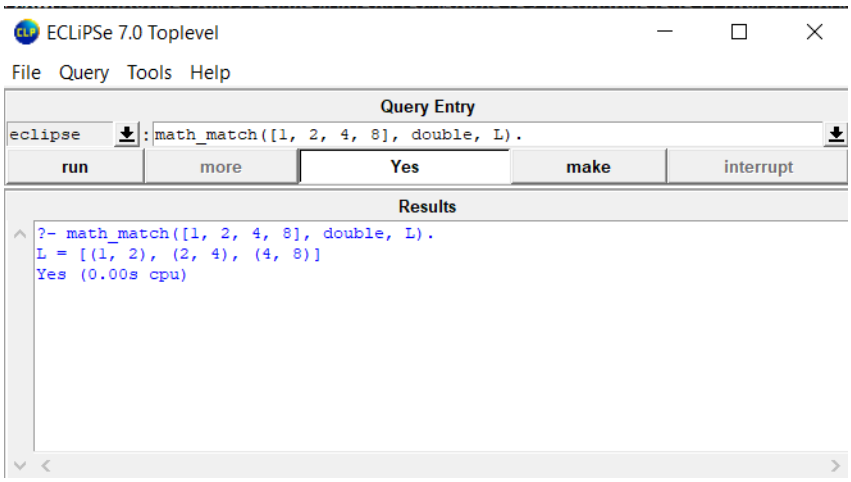
Αριθμός Ελέγχων

Ο αριθμός των ελέγχων που κάνει το κατηγόρημα math_match/3 και στην χειρότερη και στην καλύτερη περίπτωση είναι $N-1$ καθώς από μία λίστα με N στοιχεία μπορούν να δημιουργηθούν $N-1$ διαδοχικά ζεύγη και πρέπει να ελεγχούν όλα αν ικανοποιούν το κατηγόρημα C (2ο όρισμα).

Στιγμιότυπα εκτέλεσης math_match/3



Σχήμα 3: Στιγμιότυπο εκτέλεσης του κατηγορήματος math_match/3.



Σχήμα 4: Στιγμιότυπο εκτέλεσης του κατηγορήματος math_match/3.

Ορισμός κατηγόρηματος math_match_alt/3

Το παρακάτω κατηγόρημα math_match_alt/3 είναι ένας μη-αναδρομικός ορισμός του κατηγορήματος math_match/3 που χρησιμοποιεί τα built-in κατηγορήματα συλλογής λύσεων της Prolog.

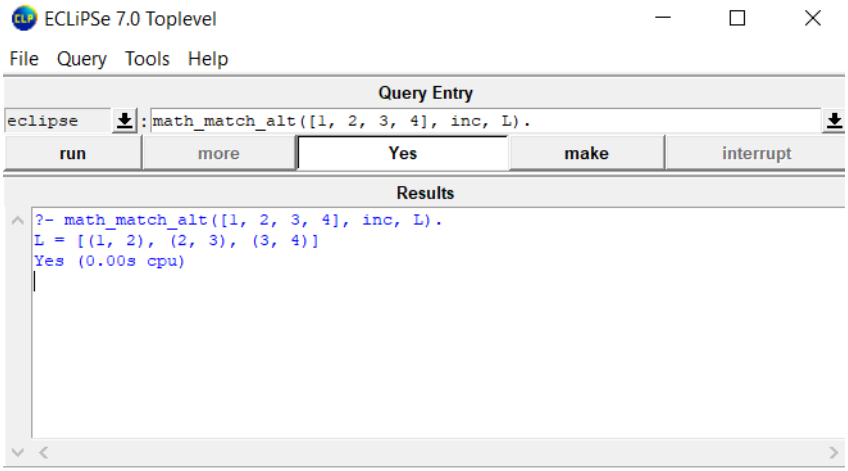
```
%%% sublist/2
sublist(L1, L2):-
        append(S1, _, L2),
        append(_, L1, S1).

%%% math_match_alt/3
math_match_alt(List, C, Solution) :-
    Predicate =.. [C, X, Y],
    findall((X,Y), sublist([X,Y],List), Tuples),
    findall((X,Y), (member((X,Y), Tuples),call(Predicate)), Solution).
```

Επεξήγηση υλοποίησης

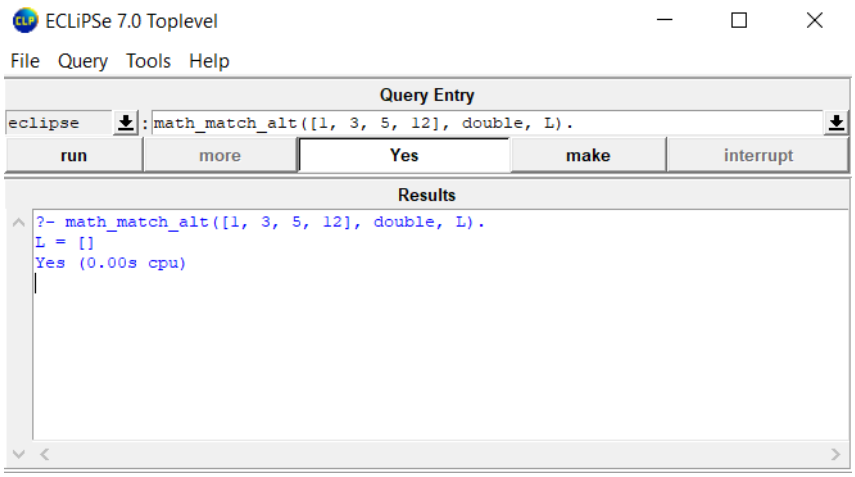
Αρχικά ορίζεται ένα βοηθητικό κατηγόρημα, το sublist/2 που πετυχαίνει αν η δοθείσα λίστα L1 είναι υπολίστα της λίστας L2.

Στην συνέχεια με την χρήση του findall/3 βρίσκονται όλα τα διαδοχικά ζεύγη του ορίσματος List και τοποθετούνται στην λίστα Tuples. Τέλος ελέγχεται κάθε διαδοχικό ζεύγος που τοποθετήθηκε στη λίστα Tuples αν ικανοποιεί το κατηγόρημα τάξης 2 C. Αν ναι τοποθετείται στη λίστα Solution.

Στιγμιότυπα εκτέλεσης math_match_alt/3



Σχήμα 5: Στιγμιότυπο εκτέλεσης του κατηγορήματος math_match_alt/3.



Σχήμα 6: Στιγμιότυπο εκτέλεσης του κατηγορήματος math_match_alt/3.

Άσκηση 3 - State of Blocks

Ορισμός κατηγορήματος stack_blocks/3

Το κατηγόρημα stack_blocks/3 δημιουργεί δύο στοίβες τριών Block η κάθε μία, έτσι ώστε όταν ένα Block μπαίνει πάνω από ένα άλλο, να έχει ίδιο ή μικρότερο πλάτος, οι δύο στοίβες να έχουν το ίδιο ύψος και να μην χρησιμοποιείται ένα Block δύο φορές. Δέχεται τρεία ορίσματα, 1ο ορισμα και 2ο όρισμα είναι οι δύο στοίβες και 3ο όρισμα είναι το ύψος των στοίβων .Οι πληροφορίες των Block δίνεται από το κατηγόρημα a_block/3 ως εξής :

a_block('κωδικός στοιχείου','ύψος', 'πλάτος')

Τα κατηγορήματα all_different/3, width_constraint/3, compute_height/2 ορίστηκαν για την διευκόλυνση του ορισμού του stack_block/3.

```
%%% a_block/3
a_block(b1,2,4).
a_block(b2,1,3).
a_block(b3,3,3).
a_block(b4,1,2).
a_block(b5,4,1).
a_block(b6,2,1).
a_block(b7,5,3).
a_block(b8,5,2).
a_block(b9,4,4).
a_block(b10,2,3).

%%% all_different/1
all_different([]).

all_different([_, []]).

all_different([Element|Tail]) :-
    not(member(Element, Tail)),
    all_different(Tail).

%%% width_constraint/3
width_constraint(Width1, Width2, Width3) :-
    Width1 >= Width2,
```

```
    Width2 >= Width3.

%%% compute_height/2.
compute_height([], 0).

compute_height([Height|RestHeights], HeightSum) :-
    compute_height(RestHeights, RestHeightSum),
    HeightSum is RestHeightSum + Height.

%%% stack_blocks/3
stack_blocks([B_A1, B_A2, B_A3], [B_B1, B_B2, B_B3], Height) :-
    % Get the information about the blocks.
    % Stack A Blocks
    a_block(B_A1, Height_A1, Width_A1),
    a_block(B_A2, Height_A2, Width_A2),
    a_block(B_A3, Height_A3, Width_A3),
    % Stack B Blocks
    a_block(B_B1, Height_B1, Width_B1),
    a_block(B_B2, Height_B2, Width_B2),
    a_block(B_B3, Height_B3, Width_B3),
    % Check if all the elements are different.
    append([B_A1, B_A2, B_A3], [B_B1, B_B2, B_B3], Stacks),
    all_different(Stacks),
    % Check the constraint about the width.
    width_constraint(Width_A1, Width_A2, Width_A3),
    width_constraint(Width_B1, Width_B2, Width_B3),
    % Check if the height of stacks is the same.
    HeightsA = [Height_A1, Height_A2, Height_A3],
    HeightsB = [Height_B1, Height_B2, Height_B3],
    compute_height(HeightsA, Height),
    compute_height(HeightsB, Height).
```

Επεξήγηση υλοποίησης

Ας ξεκινήσουμε από τα βοηθητικά κατηγορήματα. Το αναδρομικό κατηγόρημα all_different/3 πετυχαίνει όταν όλα τα στοιχεία μίας δοθείσας λίστας είναι διαφορετικά.

```
%%% all_different/1
all_different([]).
```

```
all_different([_, []]).

all_different([Element|Tail]) :-
    not(member(Element, Tail)),
    all_different(Tail).
```

Το κατηγόρημα width_constraint/3 ορίζει τον περιορισμό περί πλάτους στις λίστες. Δέχεται 3 ακεραίους (πλάτη) και πετυχαίνει τα Block έχουν τοποθετηθεί σωστά έτσι ώστε όταν ένα μπαίνει πάνω από ένα άλλο, να έχει ίδιο ή μικρότερο πλάτος.

```
%%% width_constraint/3
width_constraint(Width1, Width2, Width3) :-
    Width1 >= Width2,
    Width2 >= Width3
```

Το αναδρομικό κατηγόρημα compute_height/3 υπολογίζει αναδρομικά το άθροισμα της δοθείσας λίστας. Στην περίπτωσή μας τα στοιχεία της λίστας αναπαριστούν τα ύψη τον Block.

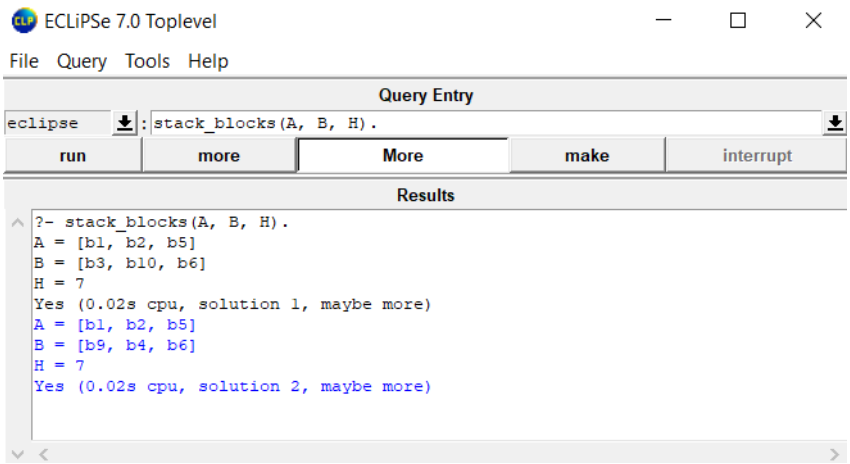
```
%%% compute_height/2.
compute_height([], 0).

compute_height([Height|RestHeights], HeightSum) :-
    compute_height(RestHeights, RestHeightSum),
    HeightSum is RestHeightSum + Height.
```

Τέλος το βασικό κατηγόρημα της άσκησης, το stack_blocks/3. Το κατηγόρημα αυτό δημιουργεί τις δύο στοίβες. Αρχικά μαζεύονται όλες οι πληροφορίες για τα Blocks χρησιμοποιώντας το κατηγόρημα a_block/3. Στη συνέχεια με την βοήθεια του κατηγορήματος all_different/1 σιγουρεύεται ότι κανένα Block δεν έχει χρησιμοποιηθεί δύο φορές. Το width_constraint/3 ελέγχει τα πλάτη και τέλος με την χρήση compute_height/2 ελέγχεται ότι οι στοίβες έχουν το ίδιο ύψος.

```
%%% stack_blocks/3
stack_blocks([B_A1, B_A2, B_A3], [B_B1, B_B2, B_B3], Height) :-
    % Get the information about the blocks.
    % Stack A Blocks
    a_block(B_A1, Height_A1, Width_A1),
    a_block(B_A2, Height_A2, Width_A2),
    a_block(B_A3, Height_A3, Width_A3),
```

```
% Stack B Blocks
a_block(B_B1, Height_B1, Width_B1),
a_block(B_B2, Height_B2, Width_B2),
a_block(B_B3, Height_B3, Width_B3),
% Check if all the elements are different.
append([B_A1, B_A2, B_A3], [B_B1, B_B2, B_B3], Stacks),
all_different(Stacks),
% Check the constraint about the width.
width_constraint(Width_A1, Width_A2, Width_A3),
width_constraint(Width_B1, Width_B2, Width_B3),
% Check if the height of stacks is the same.
HeightsA = [Height_A1, Height_A2, Height_A3],
HeightsB = [Height_B1, Height_B2, Height_B3],
compute_height(HeightsA, Height),
compute_height(HeightsB, Height).
```

Στιγμιότυπα εκτέλεσης stack_blocks/3



Σχήμα 7: Στιγμιότυπο εκτέλεσης του κατηγορήματος stack_blocks/3.

Ορισμός κατηγορήματος find_lowest_stack/4

Το κατηγόρημα find_lowest_stack/4 εντοπίζει τις στοίβες με το χαμηλότερο ύψος και το πόσες εναλλακτικές λύσεις υπολογίστηκαν για την εύρεση του. Δέχεται 4 ορίσματα.

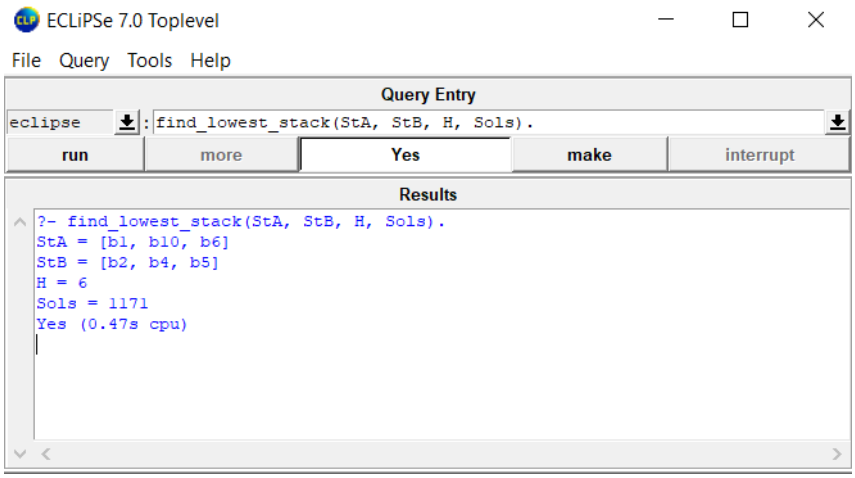
Το 1ο και 2ο όρισμα είναι οι δύο στοίβες, το 3ο όρισμα είναι το ύψος τους και το 4ο οι εναλλακτικές λύσεις που υπολογίστηκαν.

```
%%% find_lowest_stack/4
find_lowest_stack(StackA, StackB, Height, Solutions) :-
    findall((H, StackA, StackB), stack_blocks(StackA, StackB, H), SolutionList),
    setof(H, member(H, SolutionList), [(Height, StackA, StackB)|_]),
    Solutions is length(SolutionList) - 1.
```

Επεξήγηση υλοποίησης

Για την υλοποίηση του κατηγορήματος χρησιμοποιήθηκαν τα built-in κατηγορηματα συλλογής λύσεων της Prolog. Με την χρήση του findall/3 εντοπίζονται όλες οι λύσεις του stack_blocks/3 και τοποθετούνται στην λίστα SolutionList. Με το setof/3 ταξινομείται η λίστα με βάση το ύψος των στοιβών. Το πρώτο στοιχείο αυτής της λίστας είναι η λύση με το χαμηλότερο ύψος. Τέλος, το πλήθος των εναλλακτικών λύσεων είναι το πλήθος της λίστας SolutionList - 1, την λύση που θέλουμε.

Στιγμιότυπα εκτέλεσης find_lowest_stack/4



Σχήμα 8: Στιγμιότυπο εκτέλεσης του κατηγορήματος find_lowest_stack/4.