Λογικός Προγραμματισμός με Περιορισμούς

Τμ. Εφαρμοσμένης Πληροφορικής

Εργασία 1 2019-2020

1. List Processing (35/100)

Έστω το Prolog κατηγόρημα **`exclude_range(Low,Hight,List,NewList)`** το οποίο επιτυγχάνει όταν δοθείσας μιας λίστας ακεραίων **`List`** (3ο όρισμα), η λίστα **`NewList`** περιέχει όλους τους ακεραίους που ΔΕΝ ανήκουν στο κλειστό διάστημα που ορίζουν τα πρώτα δύο ορίσματα **`Low`** και **`High`**. Για παράδειγμα:

```
?- exclude_range(2, 10, [1, 2, 3, 5, 10, 11, 12, 15], List).
List = [1, 11, 12, 15]
?- exclude_range(2, 10, [2, 3, 8, 10], List).
List = []
```

Να δώσετε τον αναδρομικό ορισμό του κατηγορήματος κάνοντας όσο το δυνατό λιγότερους ελέγχους.

2. Matching Number Series (40/100)

Έστω ότι έχετε μια σειρά από κατηγορήματα με τάξη (arity) 2, τα οποία πετυχαίνουν όταν τα όρισμά τους ικανοποιούν κάποια συνθήκη. Για παράδειγμα, ανάμεσα στα κατηγορήματα που έχετε είναι τα ακόλουθα:

```
double(X,Y):-Y is X * 2.
inc(X,Y):-Y is X + 1.
```

Έστω το Prolog κατηγόρημα **`math_match(List, C, Solution)`** το οποίο επιτυγχάνει όταν η λίστα **`Solution`** περιέχει μόνο τα ζεύγη των διαδοχικών στοιχείων της λίστας **`List`** τα οποία ικανοποιούν το κατηγόρημα (τάξης 2) **`C`**. Στη μεταβλητή **`C`**, δίνεται μόνο το όνομα του κατηγορήματος. Προφανώς το κατηγόρημα **`math_match/3`** είναι *γενικό*, δηλαδή μπορεί να δουλέψει με οποιοδήποτε όνομα κατηγορήματος τάξης 2 και όχι μόνο με τα παραπάνω. Για παράδειγμα:

```
?- math_match([1, 2, 3], inc, L).
L = [(1, 2), (2, 3)]
Yes
```

Στην παραπάνω ερώτηση, το ζεύγος (1,2) ικανοποιεί την συνθήκη **`inc`** καθώς το **`inc(1,2)`** είναι αληθές. Η σειρά των ορισμάτων στο κατηγόρημα είναι εκείνη με την οποία εμφανίζονται στη λίστα. Άλλα παραδείγματα φαίνονται παρακάτω:

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

?- math_match([1, 2, 2, 1, 4, 5], inc, L).
L = [(1, 2), (4, 5)]
Yes
```

```
?- math_match([1, 2, 4, 8], double, L).
L = [(1, 2), (2, 4), (4, 8)]
Yes

?- math_match([1, 3, 5, 12], double, L).
L = []
Yes
```

α) Να δώσετε τον **αναδρομικό ορισμό** (**math_match/3**) του κατηγορήματος. Να χρησιμοποιήσετε όσο το δυνατό λιγότερους ελέγχους.

β) Να δώσετε ένα **μη αναδρομικό ορισμό** (**math_match_alt/3**), βασιζόμενοι σε κάποιο κατηγόρημα συλλογής λύσεων. Το κατηγόρημα θα έχει την ίδια συμπεριφορά με το προηγούμενο.

3. Stacks of Blocks (25/100)

Έχετε ένα σύνολο από δομικά στοιχεία (blocks), που το καθένα έχει συγκεκριμένο ύψος και συγκεκριμένο πλάτος. Η πληροφορία για τα στοιχεία δίνεται από γεγονότα της μορφής:

a_block(<κωδικός στοιχείου>,<ύψος>, <πλάτος>)

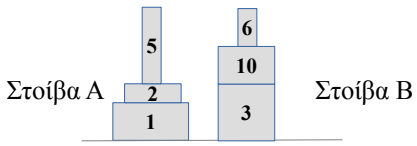
Για παράδειγμα:

```
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).
```

Το πρόβλημα που έχετε να λύσετε, είναι ότι θέλετε να φτιάξετε δύο στοίβες *με το ίδιο ύψος*, *τριών* στοιχείων η κάθε μια, έτσι ώστε όταν ένα στοιχείο μπαίνει πάνω από ένα άλλο, να έχει *ίδιο ή μικρότερο πλάτος*. Δεν μπορείτε να χρησιμοποιήσετε το ίδιο στοιχείο δύο φορές.

α) Να υλοποιήσετε ένα κατηγόρημα **stack_blocks(StackA,StackB,H)**, όπου **StackA** είναι μια λίστα με τους αριθμούς των δομικών στοιχείων της πρώτης στοίβας, **StacksB** η αντίστοιχη λίστα για τη δεύτερη στοίβα και **H** το ύψος των δύο στοιβών. Παράδειγμα εκτέλεσης (υπάρχουν πολλές περισσότερες λύσεις):

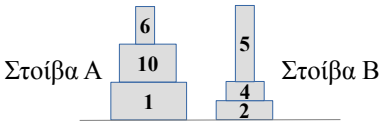
```
?- stack_blocks(A, B, H).
A = [b1, b2, b5]
B = [b3, b10, b6]
H = 7
```



Η παραπάνω λύση για την στοίβα Α, σημαίνει ότι θα μπουν τα στοιχεία 1, 2 και 5, με το στοιχείο 1 χαμηλότερα στην στοίβα.

β) Να υλοποιήσετε το κατηγόρημα **`find_lowest_stack(StA, StB ,H ,Sols)`**, το οποίο επιτυγχάνει αν οι στοίβες **`StA`** και **`StB`** έχουν *το ελάχιστο δυνατό* ύψος **`H`**. Το όρισμα **`Sols`** επιστρέφει το σύνολο των εναλλακτικών λύσεων που εξετάστηκαν για να βρεθεί εκείνη με το ελάχιστο ύψος. Παράδειγμα εκτέλεσης:

```
?-find_lowest_stack(StA, StB, H, Sols).
StA = [b1, b10, b6]
StB = [b2, b4, b5]
H = 6
Solutions = 1171
```



Σημείωση 1: Υπάρχουν περισσότερες από μια λύσεις με ύψος 6. Μπορείτε να επιστρέψετε οποιαδήποτε.
Σημείωση 2: Θα πρέπει να χρησιμοποιήσετε **απλή** Prolog και **όχι** Constraint Programming.
Σημείωση 3: Ο κώδικάς σας θα πρέπει να είναι *γενικός*, δηλαδή να λειτουργεί για οποιοδήποτε σύνολο γεγονότων **`a_block/3`** και όχι μόνο των παραπάνω.

ΠΑΡΑΔΟΣΗ

Θα παραδώσετε εντός της ημερομηνίας που αναφέρεται στο compus τα ακόλουθα:

- Ένα αρχείο με το όνομα **exec1.ecl** το οποίο θα περιέχει τις λύσεις (κατηγορήματα) **και των τριών ασκήσεων.**
- Το απαραίτητο αρχείο βρίσκονται στο compus στην ενότητα **Έγγραφα -> Coursework**
- Ένα αρχείο **report.pdf** (***σε μορφή pdf***) το οποίο θα περιέχει:
  - **Στην πρώτη σελίδα το Όνομά σας, τον Αριθμό μητρώου σας και το email σας.**
  - Για κάθε μια από τις τρείς ασκήσεις:
    - τον **κώδικα** (ασχέτως αν βρίσκεται και στο αρχείο exec1.ecl) και σχολιασμό σχετικά με αυτόν.
    - Παραδείγματα εκτέλεσης (2 για κάθε κατηγόρημα)
    - Bugs και προβλήματα που έχει ο κώδικάς σας.

ΠΡΟΣΟΧΗ: ΝΑ ΑΚΟΛΟΥΘΗΣΕΤΕ ΑΥΣΤΗΡΑ ΤΑ ΟΝΟΜΑΤΑ ΚΑΙ ΤΗ ΣΕΙΡΑ ΤΩΝ ΟΡΙΣΜΑΤΩΝ ΠΟΥ ΔΙΝΕΤΑΙ ΠΑΡΑΠΑΝΩ (ΑΥΤΟΜΑΤΟΣ ΕΛΕΓΧΟΣ ΚΩΔΙΚΑ)

Καλή επιτυχία (*και have fun with Prolog!*)