Λογικός Προγραμματισμός με Περιορισμούς

Τμ. Εφαρμοσμένης Πληροφορικής

Εργασία 1 2017-2018

1. The JVM Arithmetic Expressions

Η μηχανή JVM είναι μια μηχανή στοίβας, δηλαδή κάθε αριθμητική εντολή παίρνει τα ορίσματα από την στοίβα και αποθέτει το αποτέλεσμα και πάλι στην κορυφή της στοίβας. Έτσι για να εκτελεστεί το άθροισμα 3 + 5, θα πρέπει

1. να τοποθετηθεί το 3 στην στοίβα,
2. να τοποθετηθεί το 5 στην στοίβα,και έπειτα
3. να κληθεί η εντολή της πρόσθεσης (iadd) η οποία θα πάρει (διαγράψει/αφαιρέσει) τα **δύο ορίσματα από την στοίβα**, και θα τοποθετήσει στην στοίβα το άθροισμά τους.

Καθένα από τα παραπάνω βήματα είναι μια εντολή της JVM.

Οι διαθέσιμες εντολές για τις βασικές αριθμητικές πράξεις των ακεραίων είναι **iadd** (πρόσθεση), **isub** (αφαίρεση), **imul** (πολλαπλασιασμός), **idiv** (διαίρεση).

Οι διαθέσιμες εντολές (σε σημειογραφία Prolog) για την τοποθέτηση ακεραίων (*θα θεωρήσουμε μόνο μικρούς σχετικά θετικούς ακέραιους*):

- **iconst(X):** τοποθέτηση του X στην στοίβα όταν το X είναι μεγαλύτερο ή ίσο με 0 και μικρότερο ή ίσο του 5.
- **bipush(X):** τοποθέτηση του X στην στοίβα όταν το X είναι μεγαλύτερο του 5 και μικρότερο ή ίσο του 127.
- **sipush(X):** τοποθέτηση του X στην στοίβα όταν το X είναι μεγαλύτερο του 127 και μικρότερο ή ίσο του 32767.

Δεδομένων των εντολών αυτών, η αριθμητική πράξη 3 + 5, είναι η ακολουθία εντολών **[iconst(3), iconst(5), iadd]**.

Όταν οι πράξεις είναι πιο σύνθετες, ακολουθείται η καθιερωμένη από αριστερά προς τα δεξιά αποτίμηση των πράξεων. Για παράδειγμα:

3 + 5 + 8 = (3 + 5) + 8 = [iconst(3), iconst(5), iadd, bipush(8), iadd]

α) evaluate/3

Να ορίσετε ένα Prolog κατηγόρημα **`evaluate(Stack,ListCommands,Res)`** **`(evaluate/3)`**, όπου stack η στοίβα της JVM, **`ListCommands`** μια ακολουθία εντολών της JVM, και **`Res`** *το κορυφαίο στοιχείο της στοίβας όταν τελειώσει η εκτέλεση των εντολών*. Για παράδειγμα:

```
?- evaluate([], [iconst(3)], Res).
Res = 3

?- evaluate([], [iconst(3), bipush(30), iadd], Res).
Res = 33
```

```
?- evaluate([], [iconst(30), bipush(10), isub], Res).
Res = 20

?- evaluate([], [bipush(10), isub], Res).
No (0.00s cpu)

?- evaluate([], [bipush(10)], Res).
Res = 10

?- evaluate([20], [bipush(10), isub], Res).
Res = 10

?- evaluate([], [bipush(6), bipush(10), iadd, iconst(3), imul], Res).
Res = 48
```

β) jvm_expressions/2

Να ορίσετε ένα κατηγόρημα **jvm_expression(Exp,ListofCommands) (jvm_expression/2)** το οποίο δέχεται μια αριθμητική έκφραση θετικών ακεραίων και ενοποιεί την λίστα **ListofCommands** με την ακολουθία εντολών της JVM που υλοποιούν την έκφραση. Για παράδειγμα:

```
?- jvm_expression(3 + 5, L).
L = [iconst(3), iconst(5), iadd]

?- jvm_expression(3 + 5 - 6, L).
L = [iconst(3), iconst(5), iadd, bipush(6), isub]

?- jvm_expression(3 + 5 + 8, L).
L = [iconst(3), iconst(5), iadd, bipush(8), iadd]

?- jvm_expression(7 - 5 + 6 * 10, L).
L = [bipush(7), iconst(5), isub, bipush(6), bipush(10), imul, iadd]
```

Προσοχή: Το κατηγόρημα πρέπει να επιστρέφει μόνο μια λύση.

Σημείωση:

- *Δεν απαιτείται να βρείτε την ανάστροφη Πολωνική γραφή της έκφρασης*. Χρησιμοποιείστε αναδρομή. Θα σας φανεί χρήσιμο το παράδειγμα με την εύρεση της πρώτης παραγώγου.
- Δεν απαιτείται να ορίσετε την σειρά (προτεραιότητα) των τελεστών (*,+,/,-). Είναι ήδη ορισμένα σε Prolog.
- Προφανώς τα δύο κατηγορήματα μπορούν να λειτουργήσουν σε συνδυασμό μεταξύ τους. Για παράδειγμα

```
?- jvm_expression(7 - 5 + 6 * 10, L), evaluate([], L, Res).
L = [bipush(7), iconst(5), isub, bipush(6), bipush(10), imul, iadd]
Res = 62
```

2. FacebookFriends

Ένα δίκτυο "φίλων" στο facebook δίνεται σαν ένα σύνολο γεγονότων **friend/2**, όπου τα

ορίσματα είναι οι κωδικοί των αντίστοιχων χρηστών που είναι φίλοι. Για παράδειγμα το

```
friend(236,186).
```

δηλώνει απλά ότι οι χρήστες με κωδικούς 236 και 186 είναι φίλοι. Δεν έχει σημασία η σειρά των ορισμάτων, δηλαδή η σχέση **friend/2** είναι αμφίδρομη. Για κάθε ζεύγος φίλων υπάρχει μόνο ένα γεγονός. Το σύνολο των γεγονότων σας δίνεται στο αρχείο friends.ecl.

(α) Να υλοποιήσετε ένα κατηγόρημα **is_a_friend(X,Y)** το οποίο πετυχαίνει όταν τα **X** και **Y** είναι φίλοι. Στην οπισθοδρόμηση το κατηγόρημα θα επιστρέφει όλες τις λύσεις. Για παράδειγμα:

```
?- is_a_friend(113, 56).
Yes (0.00s cpu, solution 1, maybe more)

?- is_a_friend(113, 116).
No (0.00s cpu)

?- is_a_friend(113, Y).
Y = 104
Yes (0.00s cpu, solution 1, maybe more)
Y = 261
Yes (0.03s cpu, solution 2, maybe more)
Y = 142
Yes (0.03s cpu, solution 3, maybe more)
Y = 66
Yes (0.03s cpu, solution 4, maybe more)
Y = 132
Yes (0.03s cpu, solution 5, maybe more)
Y = 342
Yes (0.05s cpu, solution 6, maybe more)
Y = 119
Yes (0.05s cpu, solution 7, maybe more)
Y = 271
Yes (0.06s cpu, solution 8, maybe more)
Y = 26
Yes (0.06s cpu, solution 9, maybe more)
(υπάρχουν ακόμη πολλές λύσεις...)
```

(β) Να ορίσετε ένα κατηγόρημα **all_people(List)** το οποίο επιστρέφει στην λίστα **List** όλους τους χρήστες facebook για τους οποίους υπάρχει πληροφορία στα γεγονότα **friends/2**. Κάθε χρήστης πρέπει να εμφανίζεται στην λίστα *μόνο μια φορά*. Για παράδειγμα:

```
?- all_people(People).
People = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 16, 17, 19, ...]
Yes (0.00s cpu)
```

(γ) Να ορίσετε το κατηγόρημα **my_friends(Person,Friends,Number)** το οποίο πετυχαίνει όταν η λίστα **Friends** περιέχει τους φίλους του **Person**. Στο όρισμα **number** επιστρέφει το συνολικό αριθμό των φίλων της λίστας **Friends**. Παράδειγμα εκτέλεσης:

```
?- my_friends(33, Friends, Number).
Friends = [42]
Number = 1
Yes (0.00s cpu)
```

```
?- my_friends(45, Friends, Number).
Friends = [67, 104, 122, 132, 142, 186, 200, 221, 303, 322, 345]
Number = 11
Yes (0.00s cpu)
```

(δ) Να ορίσετε ένα κατηγόρημα **covered_from(Group,Person,Covered)**, το οποίο από μια λίστα χρηστών **Group**, και για ένα χρήστη **Person** που ανήκει στην **Group** επιστρέφει στην λίστα **Covered** τους χρήστες οι οποίοι είναι φίλοι του **Person και** ανήκουν στη λίστα **Group**. Στην λίστα **Covered** θα πρέπει να επιστρέφεται και ο χρήστης **Person**. Στην οπισθοδρόμηση επιστρέφονται όλες οι εναλλακτικές λύσεις. Για παράδειγμα

```
?- covered_from([21, 30, 40, 66, 56, 78, 10], P, C).
P = 21
C = [21, 40, 66, 56]
Yes (0.00s cpu, solution 1, maybe more)

P = 30
C = [30, 56]
Yes (0.00s cpu, solution 2, maybe more)

P = 40
C = [40, 21, 56]
Yes (0.02s cpu, solution 3, maybe more)

P = 66
C = [66, 21, 56]
Yes (0.02s cpu, solution 4, maybe more)

P = 56
C = [56, 21, 30, 40, 66]
Yes (0.02s cpu, solution 5, maybe more)

P = 78
C = [78]
Yes (0.02s cpu, solution 6, maybe more)

P = 10
C = [10]
Yes (0.02s cpu, solution 7, maybe more)
No (0.02s cpu)
```

Προσοχή: Σε μερικές περιπτώσεις η λύση θα απαιτήσει πολύ χρόνο να βρεθεί.

(ε) Να ορίσετε ένα κατηγόρημα **covers_most(Group,Person,Covered)**, το οποίο πετυχαίνει όταν το όρισμα **Person** είναι ο χρήστης από την λίστα χρηστών **Group**, ο οποίος έχει **τους περισσότερους φίλους** ανάμεσα στους χρήστες του **Group**. **Covered** είναι οι φίλοι του **Person** που εμφανίζονται μέσα στο **Group**. Για παράδειγμα

```
?- covers_most([21, 30, 40, 66, 56, 78, 10], P, C).
P = 56
C = [56, 21, 30, 40, 66]
Yes
```

(ζ) Αν υποθέσουμε ότι μια ανακοίνωση διαβάζεται από όλους τους φίλους του χρήστη, το ζητούμενο είναι ποιος είναι ο αριθμός χρηστών οι οποίοι πρέπει να αναρτήσουν μια ανακοίνωση έτσι ώστε να αναγνωσθεί από όλους του χρήστες. Θα πρέπει να προσπαθήσετε να δώσετε τον μικρότερο αριθμό χρηστών που θα βγάλουν την ανακοίνωση, χωρίς απαραίτητα να δώσετε τον ελάχιστο αριθμό. Να ορίσετε ένα κατηγόρημα **`cover(Group,Posts)`**, το οποίο πετυχαίνει όταν το όρισμα **`Posts`** είναι η λίστα των χρηστών οι οποίοι απαιτείται να αναρτήσουν ένα μήνυμα για να αναγνωσθεί από όλους τους χρήστες της λίστας χρηστών **`Group`**. Για παράδειγμα:

```
?- cover([21, 30, 40, 66, 56, 78, 10], Posts).
Posts = [56, 78, 10]
Yes
```

(η) Σε πόσους χρήστες πρέπει να αναρτηθεί μια ανακοίνωση για να αναγνωσθεί από όλους; Να δώσετε την αντίστοιχη Prolog ερώτηση που απαντά στο 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 δύο ασκήσεων.**
- 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 τρεις 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!*)