**ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ-PROLOG**

**ΠΑΡΑ∆ΕΙΓΜΑ 1**

∆ίνονται τα εξής γεγονότα: «Ο Παύλος είναι πατέρας του Γιάννη και της Γεωργίας» και «Η Ελένη είναι µητέρα της Μαρίας και του Πέτρου». Επίσης, µας δίνεται και η εξής γνώση τύπου κανόνα, που αφορά το πότε δύο άνθρωποι είναι αδέλφια:

∆ύο άνθρωποι Χ, Υ είναι αδέλφια

1. Αν έχουν τον ίδιο πατέρα
2. Αν έχουν την ίδια µητέρα.

Ζητείται να αναπαρασταθεί η παραπάνω γνώση στην Prolog, ώστε να δημιουργηθεί αντίστοιχο πρόγραμμα.

# ΑΠΑΝΤΗΣΗ

Το πρόγραμμα θα έχει ως εξής:

is\_father(paul, john). is\_father(paul, georgia). is\_mother(helen, maria). is\_mother(helen, peter).

is\_sibling(X, Y) :- is\_father(Z, X), is\_father(Z, Y). is\_sibling(X, Y) :- is\_mother(Z, X), is\_mother(Z, Y).

Η ερµηνεία αυτών των κανόνων είναι αυτονόητη.

Αν θέσουµε το ερώτηµα:

?- is\_sibling(georgia, john).

τότε θα εκτυπωθεί

Yes

Αν θέσουµε τώρα το ερώτηµα:

?- is\_sibling(Χ, john).

τότε θα εκτυπωθεί

X = john ;

X = georgia ;

No

(Εδώ το No σημαίνει ότι δεν υπάρχουν άλλες απαντήσεις-λύσεις).

Όπως είναι φανερό, το πρόγραµµα εξάγει τοσυµπέρασµα ότι ο Γιάννης είναι αδελφός του εαυτού του, το οποίο δεν θα θέλαµε να εξάγεται, παρ’ ότι κατά µια

έννοια δεν είναι λανθασµένο. Για να αποτρέψουµε τέτοιου είδους συµπεράσµατα, θα πρέπει να κάνουµε τις εξής προσθήκες:

is\_father(paul, john). is\_father(paul, georgia). is\_mother(helen, maria). is\_mother(helen, peter).

equal(john, john). equal(georgia, georgia). equal(maria, maria). equal(peter, peter).

is\_sibling(X, Y) :- is\_father(Z, X), is\_father(Z, Y), not(equal(X, Y)).

is\_sibling(X, Y) :- is\_mother(Z, X), is\_mother(Z, Y), not(equal(X, Y)).

Έχουµε προσθέσειτο κατηγόρηµα equal/2, το οποίο χρησιµοποιούµε για να αποκλείσουµε τα τις παραπάνω περιπτώσεις. Τώρα,

αν θέσουµε το ερώτηµα:

?- is\_sibling(Χ, john).

τότε θα εκτυπωθεί

X = georgia ; No

Μια ευκολότερη λύση, που διαθέτει η Prolog, είναι η χρησιµοποίηση του κατηγορήµατος ‘=’. Ουσιαστικά γίνεται ό,τι και παραπάνω, αλλά ευκολότερα. Το πρόγραµµα θα γίνει τώρα ως εξής:

is\_father(paul, john). is\_father(paul, georgia). is\_mother(helen, maria). is\_mother(helen, peter).

is\_sibling(X, Y) :- is\_father(Z, X), is\_father(Z, Y), not(X=Y). is\_sibling(X, Y) :- is\_mother(Z, X), is\_mother(Z, Y) , not(X=Y).

Τώρα, αν ξαναθέσουµε το ερώτηµα:

?- is\_sibling(Χ, john).

τότε πάλι θα εκτυπωθεί

X = georgia ; No

# ΠΑΡΑΔΕΙΓΜΑ 2

Δίνεται το παρακάτω δυαδικό δέντρο:

Giorgos

Kyriakos

Petros

Giannis

Katerina

Anna

Eleni

(α) Να γράψετε πρόγραμμα Prolog που να αναπαριστά το παραπάνω δέντρο με μια σειρά από γεγονότα, χρησιμοποιώντας το κατηγόρημα tree/3.

(β) Να συμπληρώσετε το πρόγραμμα με κανόνα(ες) έτσι ώστε να διαπερνά το δέντρο κατά βάθος με ταυτόχρονη εκτύπωση των ονομάτων των κόμβων (δηλ. Giorgos- Giannis-Petros-Kyriakos-Eleni-Anna-Katerina).

(γ) Το ίδιο με το (β) για διαπέραση κατά πλάτος (δηλ. Giorgos-Giannis-Eleni-Petros- Kyriakos-Anna-Katerina).

Σημείωση: Τα προγράμματα στα (β) και (γ) να είναι τέτοια που να μπορούν να εφαρμοστούν σε δέντρα οποιουδήποτε βάθους. Επίσης, η εκκίνηση κάθε προγράμματος να γίνεται με το κατηγόρημα go (χωρίς ορίσματα).

# ΕΠΙΛΥΣΗ

tree(giorgos, giannis, eleni). tree(giannis, petros, kyriakos). tree(eleni, anna, katerina). tree(petros,nil,nil). tree(kyriakos,nil,nil). tree(anna,nil,nil). tree(katerina,nil,nil).

%Diaperasi kata vathos

print\_tree(X):-

tree(X,Y,Z),

write(X), nl, print\_tree(Y), print\_tree(Z).

print\_tree(X):-tree(X,nil,nil).

go:- write('Give Root: '), read(T), nl, print\_tree(T).

%Diaperasi kata platos

print\_br(X):- write(X), nl, print\_tree(X).

print\_tree(X):- tree(X,nil,nil). print\_tree(X):- tree(X,Y,Z),

write(Y), nl,

write(Z), nl, print\_tree(Y), print\_tree(Z).

go:- write('Give Root: '), read(T), nl, print\_br(T).

# ΠΑΡΑΔΕΙΓΜΑ 3

Το παρακάτω κύκλωμα CIRCUIT παρουσιάζει έναν πλήρη αθροιστή 1-bit, ο οποίος έχει τρεις εισόδους και δύο εξόδους. Οι είσοδοι X1 και X2 αναπαριστούν τα δύο bits που θα αθροιστούν και η είσοδος C αναπαριστά ένα carry bit το οποίο μεταφέρεται από κάποια άλλη πράξη. Η έξοδος OUTPUT είναι το άθροισμα που παράγει το κύκλωμα, ενώ η έξοδος CARRY είναι το carry bit που προκύπτει από την πρόσθεση.

X1

CIRCUIT

XOR1

XOR2

AND2

OR

AND1

X2 OUTPUT

C

CARRY

(α) Υλοποιήστε το παραπάνω κύκλωμα χρησιμοποιώντας την δηλωτική γλώσσα προγραμματισμού Prolog.

# ΑΠΑΝΤΗΣΗ

and(0, 0, 0).

and(0, 1, 0).

and(1, 0, 0).

and(1, 1, 1).

or(0, 0, 0).

or(0, 1, 1).

or(1, 0, 1).

or(1, 1, 1).

xor(0, 0, 0).

xor(0, 1, 1).

xor(1, 0, 1).

xor(1, 1, 0).

circuit(X1, X2, C, Out, Carry) :- xor(X1, X2, XOR1),

and(X1, X2, AND1),

xor(XOR1, C, Out), and(C, XOR1, AND2), or(AND2, AND1, Carry).

(β) Συμπληρώστε το πρόγραμμα ώστε να ξεκινά με το κατηγόρημα go (χωρίς ορίσματα) και να διαχειρίζεται την είσοδο και να παράγει την έξοδο μέσω ενός στοιχειώδους user interface. Ως οδηγός δίνεται η παρακάτω εκτέλεση:

1 ?- go.

Insert 1st bit: 1. Insert 2nd bit: 1.

Insert carry bit: 0.

The output of the circuit is: 0 The carry bit is: 1

# ΑΠΑΝΤΗΣΗ

go:-

write('Insert 1st bit: '), read(X1),nl,

write('Insert 2nd bit: '), read(X2),nl,

write('Insert carry bit: '), read(C),

circuit(X1, X2, C, Out, Carry),

write(' '),

nl,

write('The output of the circuit is: '), write(Out),

nl,

write('The carry bit is: '), write(Carry),

nl,

write(' '),

nl.

# ΠΑΡΑ∆ΕΙΓΜΑ 4

Έστω ότι µας ζητούν να γράψουµε ένα πρόγραµµα Prolog που να εκτυπώνει την οποιαδήποτε υπο-λίστα της παρακάτω λίστας:



gray

yellow

green

blue

red

χρησιµοποιώντας το κατηγόρηµα next/2 (που παίρνει σαν ορίσµατα δύο διπλανά στοιχεία).

# ΑΠΑΝΤΗΣΗ

Με βάση το κατηγόρημα next/2 αποτυπώνουμε τη λίστα ως γεγονότα διπλανών στοιχείων:

next(red, blue). next(blue, green). next(green, yellow). next(yellow, gray). next(gray, nil).

Στη συνέχεια, πρέπει να γράψουµε ένα κανόνα (ή περισσότερους), που να εκτυπώνει την οποιαδήποτε υπό-λίστα. Προφανώς, το κατηγόρηµα στο οποίο θα αναφέρεται ο κανόνας θα πρέπει να έχει σαν όρισµα το στοιχείο το οποίο θα είναι κεφαλή της υπο- λίστας, έστω το print\_list/1. Ο κανόνας αυτός θα είναι ο εξής:

print\_list(X):- next(X, Y), write(X), nl, print\_list(Y).

Ας προσπαθήσουµε να ερµηνεύσουµε αυτόν τον κανόνα. Ο κανόνας λοιπόν αυτός λέγει το εξής: Για να εκτυπώσεις την υπο-λίστα µε κεφαλή κάποιο στοιχείο X (print\_list(X)), θα πρέπει (:-) (α) να βρεις το διπλανό στοιχείο του X, έστω Υ (next(X, Y)), (β) να εκτυπώσεις το Χ (write(X)), (γ) να πας στην επόµενη γραµµή και (δ) να εκτυπώσεις την υπο-λίστα µε κεφαλή το Υ.

Όπως καταλαβαίνετε, πρόκειται για αναδρομικό ορισμό: η εκτύπωση µιας υπο-λίστας ορίζεται σαν η εκτύπωση της κεφαλής της συν την εκτύπωση της υπο-λίστας µε κεφαλή το επόµενο (διπλανό) στοιχείο της. Τα δύο ουσιαστικά στοιχεία στο δεξί

µέρος του κανόνα, δηλ. αυτά που έχουν σχέση µε τη διαδικασία συλλογισµού της Prolog από τη βάση γνώσης του προβλήµατος, είναι τα next(X,Y) και print\_list(Y), τα υπόλοιπα παρεµβάλλονται για λόγους εκτύπωσης.

Ας δούµε τώρα πώς εκτελεί αυτόν τον κανόνα η Prolog. Αν κάνουµε το ερώτηµα:

?- print\_list(red).

δηλ. ζητήσουµε να εκτυπωθεί όλη η λίστα, τότε θα συµβούν τα παρακάτω.

Η Prolog αναζητεί να βρει γεγονός ή κανόνα που να ταιριάζει µε το ερώτηµα, παίρνοντάς τα µε τη σειρά. Όταν λέµε ότι ένα ερώτηµα, όπως το παραπάνω,

«ταιριάζει» µ’ ένα γεγονός ή µε την κεφαλή ενός κανόνα, σηµαίνει ότι έχουν το ίδιο κατηγόρηµα και τα ίδια ή ενοποιούµενα ορίσµατα. ∆ύο ορίσµατα είναι ενοποιούµενα. αν το ένα είναι σταθερά και το άλλο µεταβλητή ή αντίστροφα. Στην περίπτωσή µας δεν υπάρχει γεγονός που να ταιριάζει. Ταιριάζει µόνο ο κανόνας (δηλ. η κεφαλή του), οπότε και η µεταβλητή Χ (του κανόνα) παίρνει την τιµή ‘red’.

Στη συνέχεια, εξετάζει αν εκπληρώνονται οι συνθήκες του κανόνα, δηλ. οι ατοµικές εκφράσεις που βρίσκονται στα δεξιά του συµβόλου ‘:-’. Ξεκινά µε την ‘next(X, Y)’. Επειδή η Χ έχει πάρει την τιµή ‘red’, η έκφραση γίνεται ‘next(red, Y)’. Η έκφραση αυτή ταιριάζει µε το γεγονός ‘next(red, blue)’, οπότε η Υ παίρνει την τιµή ‘blue’.

Μετά εξετάζει την write(X) (δηλ. write(red)), η οποία ταιριάζει µε τον ενσωµατωµένο (στην Prolog) ορισµό του κατηγορήµατος write και έχει σαν αποτέλεσµα την εκτύπωση της τιµής της Χ, δηλ. εκτυπώνεται το red. Στη συνέχεια εξετάζει την nl, η οποία έχει σαν αποτέλεσµα την αλλαγή γραµµής για την επόµενη εκτύπωση.

Τέλος, εξετάζει την ‘print\_list(Y)’ (δηλ. ‘print\_list(blue)’). Η ατοµική αυτή έκφραση δεν ταιριάζει µε κανένα γεγονός, αλλά ταιριάζει µε τον κανόνα. Οπότε ξεκινά ο ίδιος πάλι κύκλος, όπως περιγράφτηκε παραπάνω. Σε κάθε κύκλο, όπως φαίνεται, εκτυπώνεται και ένα στοιχείο της λίστας, αυτό που είναι κεφαλή της τρέχουσας υπο-λίστας.

Στο τέλος του τελευταίου κύκλου αποµένει για ταίριασµα το ‘print\_list(nil)’, το οποίο ταιριάζει µε τον κανόνα. Η πρώτη συνθήκη που πρέπει να εκπληρωθεί τώρα είναι η

‘next(nil, Y)’. Επειδή δεν υπάρχει γεγονός ή κανόνας που να ταιριάζει, σταµατά η διαδικασία και επιστρέφει ‘Νο’.

Για να αποφύγουµε την επιστροφή του ‘Νο’, που φαίνεται σαν να έχει αποτύχει η διαδικασία, µπορούµε να προσθέσουµε τον παρακάτω κανόνα:

print\_list(X) :- next(X, nil), write(X), nl.

ο οποίος θα µπει πριν από τον ήδη υπάρχοντα κανόνα. Ο κανόνας αυτός διαπραγµατεύεται την οριακή περίπτωση του τέλους της λίστας. Λέγει δηλ. ότι αν φτάσουµε στο τελευταίο στοιχείο της λίστας, οπότε δεν υπάρχει διπλανό στοιχείο (next(X, nil)), τότε εκτύπωση της υπο-λίστας σηµαίνει εκτύπωση του τελευταίου

στοιχείου (write(X)) και µετάβαση στην επόµενη γραµµή (nl), µιας και επόµενη υπο-λίστα δεν υπάρχει.

Εποµένως η απάντηση στο ερώτηµα:

?- print\_list(red).

Θα είναι

red blue

green yellow gray

Yes

Αν θέσουµε τώρα το ερώτηµα:

?- print\_list(green).

τότε θα εκτυπωθεί

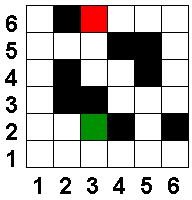
green yellow gray

Yes

# ΠΑΡΑΔΕΙΓΜΑ 5

Ένα ρομπότ κινείται σε ένα πλέγμα το οποίο περιλαμβάνει εμπόδια. Το ρομπότ μπορεί να κινηθεί ένα τετράγωνο τη φορά, προς τα επάνω, δεξιά, κάτω ή αριστερά. Το ρομπότ δεν μπορεί να κινηθεί σε μία θέση-εμπόδιο.

Στο σχήμα φαίνεται μια αρχική κατάσταση του προβλήματος. Οι θέσεις-εμπόδια έχουν μαύρο χρώμα, η αρχική θέση του ρομπότ έχει κόκκινο χρώμα και η επιθυμητή τελική θέση του ρομπότ είναι με πράσινο χρώμα.



Θεωρούμε ότι μία κατάσταση του προβλήματος ορίζεται από τη θέση του ρομπότ στο πλέγμα και μπορεί να αναπαρασταθεί με το ζεύγος (x,y), όπου x είναι η στήλη και y η γραμμή της θέσης του ρομπότ, 1 ≤ x ≤ 6, 1 ≤ y ≤ 6, και στη θέση (x,y) δεν βρίσκεται εμπόδιο. Συνεπώς, η αρχική κατάσταση στο πρόβλημα αυτό είναι η (3,6) και η τελική η (3,2).

**Α. Αναπαράσταση χώρου**

Αναπαραστήστε το πλέγμα με τα εμπόδια και την εκάστοτε θέση του ρομπότ με γεγονότα μέσω ενός κατηγορήματος Prolog (προτεινόμενο κατηγόρημα: pos/3, με ορίσματα τις συντεταγμένες της θέσης και ένα προσδιορισμό για το αν περιέχει το ρομπότ ή όχι).

**Απάντηση**

pos(1,1,empty).

pos(1,2,empty).

pos(1,3,empty).

pos(1,4,empty).

pos(1,5,empty).

pos(1,6,empty).

pos(2,1,empty).

pos(2,2,empty). pos(2,3,blocked). pos(2,4,blocked). pos(2,5,empty). pos(2,6,blocked).

pos(3,1,empty).

pos(3,2,empty). pos(3,3,blocked). pos(3,4,empty).

pos(3,5,empty).

pos(3,6,robot).

pos(4,1,empty). pos(4,2,blocked). pos(4,3,empty).

pos(4,4,empty). pos(4,5,blocked). pos(4,6,empty).

pos(5,1,empty).

pos(5,2,empty).

pos(5,3,empty). pos(5,4,blocked). pos(5,5,blocked). pos(5,6,empty).

pos(6,1,empty). pos(6,2,blocked). pos(6,3,empty).

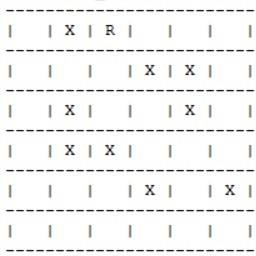
pos(6,4,empty).

pos(6,5,empty).

pos(6,6,empty).

**Β**. **Εκτύπωση χώρου**

Με βάση την αναπαράσταση στο Α, γράψτε πρόγραμμα Prolog που να εκτυπώνει στην οθόνη μια απεικόνιση του πλέγματος στην αρχική κατάσταση. Μια τέτοια απεικόνιση είναι η παρακάτω (όπου ‘Χ’ αναπαριστά θέση-εμπόδιο και το ‘R’ τη θέση του ρομπότ):



**Απάντηση**

make\_line :- write(' ').

show\_box(N,M):- pos(N,M,Z), (Z==empty, write('|'), tab(3); Z==blocked, write('| X '); Z==robot, write('| R ')).

show\_row(N):- make\_line, nl, show\_box(1,N), show\_box(2,N), show\_box(3,N), show\_box(4,N), show\_box(5,N), show\_box(6,N), write('|'), nl.

show\_grid:- show\_row(6), show\_row(5), show\_row(4), show\_row(3), show\_row(2), show\_row(1),make\_line, nl, nl.

*Το κατηγόρημα show\_grid σχεδιάζει τις γραμμές του πλέγματος μία-μία με τη βοήθεια του κατηγορήματος show\_row(Ν), το οποίο καλείται 6 φορές, μία για κάθε γραμμή, θέτοντας διαδοχικά Ν=6, 5, …, 1. Το show\_row(Ν) εκτυπώνει το πάνω μέρος της Ν-οστής γραμμής του πλέγματος (μέσω του make\_line), μετακινεί τον δείκτη στην επόμενη γραμμή στην οθόνη (μέσω το nl) και στη συνέχεια εκτυπώνει τα 6 κελιά-θέσεις της Ν-οστής γραμμής του πλέγματος, μέσω του κατηγορήματος show\_box(N,M), καλώντας το διαδοχικά για Μ=1,2,…,6. Το show\_box(N,M) εξετάζει αν το κελί-θέση (Ν,Μ) είναι άδεια ή έχει εμπόδιο ή είναι η τρέχουσα θέση του ρομπότ (μέσω της μεταβλητής Ζ) και ανάλογα τυπώνει το αντίστοιχο γράφημα. Επισημαίνεται η χρήση του τελεστή «;», που αντιπροσωπεύει το λογικό ‘Η (OR).*

**Γ**. **Τελεστές κίνησης**

Υλοποιήστε σε Prolog τις δυνατές κινήσεις του ρομπότ, ως κανόνες Prolog. Χρησιμοποιήστε ένα κατηγόρημα για κάθε τελεστή (προτείνονται: move\_up/0, move\_down/0, move\_left/0, move\_right/0).

**Απάντηση**

move\_up:- pos(X,Y,robot), Y=<5, Y1 is Y+1, pos(X,Y1,Z), Z==empty, retract(pos(X,Y,robot)), assert(pos(X,Y,empty)), retract(pos(X,Y1,empty)), assert(pos(X,Y1,robot).

*Στον ορισμό του τελεστή «μετακίνηση προς τα πάνω» (move\_up) εντοπίζεται η θέση του ρομπότ* (pos(X,Y,robot)) *και στη συνέχεια εξετάζεται αν ισχύουν-αληθεύουν οι προϋποθέσεις του τελεστή Ε. Πρώτα εξετάζεται αν η τρέχουσα θέση του ρομπότ είναι πιο κάτω από το πάνω όριο (γραμμή 6) του πλέγματος, δηλ. αν αληθεύει ότι* Y=<5. *Στη συνέχεια εξετάζεται αν η νέα θέση για μετακίνηση προς τα πάνω είναι κενή* (Y1 is Y+1, pos(X,Y1,Z), Z==empty)*. Αν είναι, τότε διαγράφεται το γεγονός της τρέχουσας θέσης του ρομπότ* (retract(pos(X,Y,robot))) *και αντικαθίσταται από το γεγονός άδειας θέσης* (assert(pos(X,Y,empty)). *Επίσης, εισάγεται το γεγονός της νέας θέσης του ρομπότ* (assert(pos(X,Y1,robot))) *και διαγράφεται το (παλαιό) γεγονός του ότι είναι κενή, ως μη ισχύον πλέον.*

*Παραπλήσιες επεξηγήσεις ισχύουν και για τους παρακάτω κανόνες-τελεστές.*

move\_down:- pos(X,Y,robot), Y>=2, Y1 is Y-1, pos(X,Y1,Z), Z==empty, retract(pos(X,Y,robot)), assert(pos(X,Y,empty)), retract(pos(X,Y1,empty)), assert(pos(X,Y1,robot)),

move\_right:- pos(X,Y,robot), X=<5, X1 is X+1, pos(X1,Y,Z), Z==empty, retract(pos(X,Y,robot)),assert(pos(X,Y,empty)), retract(pos(X1,Y,empty)), assert(pos(X1,Y,robot)),

move\_left:- pos(X,Y,robot), X>=2, X1 is X-1, pos(X1,Y,Z), Z==empty, retract(pos(X,Y,robot)), assert(pos(X,Y,empty)), retract(pos(X1,Y,empty)), assert(pos(X1,Y,robot)),