ΠΑΡΑΔΕΙΓΜΑΤΑ ΣΤΗ ΓΛΩΣΣΑ PROLOG

ΠΑΡΑΔΕΙΓΜΑ 1

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

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

ΑΠΑΝΤΗΣΗ-ΕΠΕΞΗΓΗΣΕΙΣ

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 Y (`next(X, Y)`), (β) να εκτυπώσεις το X (`write(X)`), (γ) να πας στην επόμενη γραμμή και (δ) να εκτυπώσεις την υπο-λίστα με κεφαλή το Y.

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. Δύο 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 X (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 X έχει πάρει την τιμή 'red', η έκφραση γίνεται 'next(red, Y)'. Η έκφραση αυτή ταιριάζει με το γεγονός 'next(red, blue)', οπότε η Y παίρνει την τιμή 'blue'.

Μετά εξετάζει την write(X), η οποία ταιριάζει με τον ενσωματωμένο (στην Prolog) ορισμό του κατηγορήματος write και έχει σαν αποτέλεσμα την εκτύπωση της τιμής της X, 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)'. 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 'No'.

Για να αποφύγουμε την επιστροφή του 'No', 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 2

Δίνονται 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:

Δύο άνθρωποι X, Y 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(X, 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 το συμπέρασμα 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 προσθέσει το κατηγόρημα equal/2, το οποίο χρησιμοποιούμε για να αποκλείσουμε τα τις παραπάνω περιπτώσεις. Τώρα,

αν θέσουμε το ερώτημα:

```
?- is_sibling(X, john).
```

τότε θα εκτυπωθεί

```
X = georgia ;
No
```