ΠΑΡΑΡΤΗΜΑ 1

PROLOG και Τεχνητή Νοημοσύνη

Στο παράρτημα αυτό παρουσιάζεται η υλοποίηση διαφόρων θεμάτων ΤΝ που παρουσιάστηκαν θεωρητικά στο βιβλίο, όπως αλγορίθμων αναζήτησης, αναπαράστασης γνώσης με πλαίσια, ενός απλού συστήματος σχεδιασμού ενεργειών καθώς και μερικών εφαρμογών πρακτόρων. Για την υλοποίηση υιοθετήθηκε η γλώσσα λογικού προγραμματισμού PROLOG.

Για την καλύτερη κατανόηση των παραδειγμάτων, παρατίθεται στην αρχή ένα συνοπτικό εγχειρίδιο της γλώσσας. Το παράρτημα δεν έχει σκοπό την πλήρη διδασκαλία της PROLOG αλλά την παρουσίαση μόνο βασικών της. Η πλήρης μελέτη και κατανόηση όλων των παραδειγμάτων προϋποθέτει καλή γνώση της γλώσσας. Αν και θα μπορούσε να είχε χρησιμοποιηθεί άλλη γλώσσα προγραμματισμού, η συγκεκριμένη γλώσσα ενδείκνυται για εφαρμογές ΤΝ λόγω της εκφραστικότητας αλλά και απλότητας που παρέχει στην περιγράφη της γνώσης ενός προβλήματος, καθώς και της εξειδίκευσής της στο χειρισμό συμβόλων.

Σε πολλές περιπτώσεις παρουσιάζεται μόνο μέρος του κώδικα. Ο πλήρης κώδικας και όλα τα απαραίτητα αρχεία μαζί με οδηγίες για την εκτέλεση των προγραμμάτων είναι διαθέσιμα από την ιστοσελίδα του βιβλίου: `http://aibook.csd.auth.gr`. Στην ίδια ιστοσελίδα παρέχονται και σύνδεσμοι για διάφορες υλοποιήσεις της PROLOG, όπως SICSTUS PROLOG, LPA PROLOG, και SWI-PROLOG. Τα προγράμματα έχουν αναπτυχθεί χρησιμοποιώντας τα σταθερά χαρακτηριστικά της γλώσσας, γνωστά και ως *συντακτικό του Εδιμβούργου*, ώστε να είναι αποδεκτά από όλες τις εκδόσεις της γλώσσας που υποστηρίζουν αυτό το συντακτικό.

Π1.1 Εισαγωγή στη Γλώσσα PROLOG

Π1.1.1 Ιστορική Αναδρομή

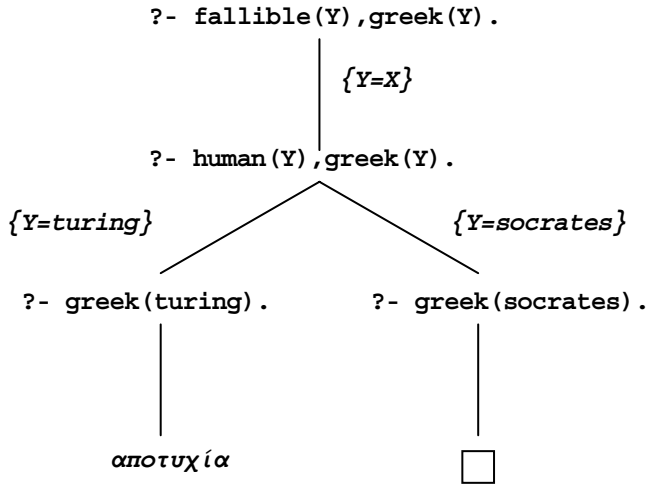
Η PROLOG (PROgramming in LOGic) είναι μια συμβολική γλώσσα προγραμματισμού που βασίζεται στην κατηγορηματική λογική. Η ανάπτυξή της ξεκίνησε στη

Η πρώτη από αριστερά κλήση ενοποιείται με την πρόταση 3. Η αντικατάσταση που προκύπτει από την ενοποίηση είναι {**Y=socrates**}, και η ερώτηση γίνεται:

```
?- greek(socrates).
```

Η κλήση αυτή ενοποιείται με την πρόταση 1 και επειδή δεν υπάρχουν προτάσεις που να μην έχουν εξετασθεί, δεν υπάρχουν εναλλακτικές λύσεις. Η μοναδική, λοιπόν, απάντηση στην αρχική ερώτηση είναι η **Y=socrates**.

Η διαδικασία που περιγράφηκε παραπάνω μπορεί να παρασταθεί με το δένδρο στο Σχήμα Π1.1 το οποίο ονομάζεται *δένδρο υπολογισμού* (*computation tree*).



Σχήμα Π1.1: Παράδειγμα δένδρου υπολογισμού.

Κάθε κόμβος του δένδρου αντιστοιχεί στην τρέχουσα ερώτηση. Οι συνδέσεις (κλαδιά) μεταξύ των κόμβων αντιπροσωπεύουν ένα υπολογιστικό βήμα και χαρακτηρίζονται από τις αντικαταστάσεις των μεταβλητών που γίνονται σε αυτό. Κάθε διαδρομή από τον αρχικό κόμβο μέχρι κάποιο φύλλο του δένδρου (τελικός κόμβος) καταλήγει είτε σε επιτυχία είτε σε αποτυχία. Οι διαδρομές που καταλήγουν σε επιτυχία αντιπροσωπεύουν τις εναλλακτικές απαντήσεις στην ερώτηση του χρήστη. Ο τελικός κόμβος στις διαδρομές αυτές είναι η κενή πρόταση, που συμβολίζεται με το □. Σε περίπτωση που μια κλήση ενοποιείται με περισσότερες από μια προτάσεις του προγράμματος, τότε από τον κόμβο αυτόν ξεκινούν περισσότερα του ενός κλαδιά. Το σημείο αυτό είναι ένα σημείο οπισθοδρόμησης.

Π1.1.6 Αναδρομή και Αναδρομικές Δομές

Η αναδρομή ως τεχνική προγραμματισμού

Η *αναδρομή* (*recursion*) αποτελεί βασικό στοιχείο στον προγραμματισμό με PROLOG. Με τον όρο αναδρομή εννοείται η δυνατότητα ένας κανόνας να περιέχει στο σώμα του μια κλήση προς τον εαυτό του. Οι κανόνες που χρησιμοποιούν αναδρομή χαρακτηρίζονται σαν αναδρομικοί κανόνες. Η χρήση της αναδρομής οδηγεί σε μικρότερα προγράμματα.

Άρνηση ως αποτυχία

Μία από τις σημαντικότερες διαφορές της PROLOG από την κατηγορηματική λογική είναι η σημασία της *άρνησης* (*negation*). Στην κατηγορηματική λογική μπορεί να υπάρξουν αρνητικά γεγονότα, δηλαδή ατομικοί τύποι για τους οποίους δηλώνεται ρητά ότι είναι ψευδείς. Στην PROLOG δεν υπάρχει η δυνατότητα αναπαράστασης τέτοιας γνώσης, παρά μόνο θετικής γνώσης. Η ύπαρξη κάποιου γεγονότος δηλώνει την αλήθεια του, ενώ θεωρείται πως ό,τι δεν αναφέρεται ρητά στη μνήμη της PROLOG, τότε δεν ισχύει. Αυτό ονομάζεται "*υπόθεση κλειστού κόσμου*" (*closed-world assumption*) και δίνει τη δυνατότητα ύπαρξης ενός είδους περιορισμένης άρνησης, της "*άρνησης ως αποτυχίας*" (*negation-as-failure*).

Σύμφωνα με τα παραπάνω, οποιαδήποτε σχέση δεν μπορεί να αποδειχθεί από το σύστημα θεωρείται ως ψευδής. Η υλοποίηση αυτή της άρνησης διαφέρει σαφώς από την κλασική έννοια της άρνησης στη λογική, αλλά χρησιμοποιήθηκε καθώς απαλλάσσει τον προγραμματιστή από την υποχρέωση του ορισμού όλης την αρνητικής πληροφορίας για κάποια εφαρμογή.

Το κατηγόρημα **not** δέχεται ως όρισμα μια οποιαδήποτε κλήση της PROLOG. Αν η κλήση μπορεί να αποδειχθεί τότε το κατηγόρημα αποτυγχάνει. Αν όχι τότε πετυχαίνει.

```
?- not(member(a,[a,b,c])).
no
?- not(member(d,[a,b,c])).
yes
```

Πολλές εκδόσεις της PROLOG παρέχουν το κατηγόρημα **\+** ως ισοδύναμο του **not** για να αποφευχθεί η σύγχυση της άρνησης της PROLOG με την άρνηση της λογικής.

Π1.2 Η PROLOG στην Τεχνητή Νοημοσύνη

Η PROLOG, λόγω της δηλωτικότητάς της και της ευκολίας χειρισμού συμβόλων που παρέχει, ενδείκνυται για την υλοποίηση εφαρμογών ΤΝ. Στη συνέχεια, παρουσιάζεται ενδεικτικά η υλοποίηση σε γλώσσα PROLOG κάποιων από τις τεχνικές και τους αλγορίθμους ΤΝ που αναπτύχθηκαν θεωρητικά στα προηγούμενα κεφάλαια του βιβλίου.

Π1.2.1 Αναπαράσταση Προβλημάτων

Στην ενότητα αυτή παρουσιάζεται η κωδικοποίηση διαφόρων προβλημάτων που αναφέρθηκαν στο ΜΕΡΟΣ Α του βιβλίου. Ουσιαστικά, πρόκειται για την κωδικοποίηση της αρχικής κατάστασης (**initial_state**), των τελικών καταστάσεων (**goal**) και των τελεστών μετάβασης (**operator**), έτσι ώστε να είναι δυνατή η επίλυσή του από τους αλγόριθμους αναζήτησης που θα περιγραφούν παρακάτω. Σε ορισμένα προβλήματα χρειάζεται και ο ορισμός της ευριστικής συνάρτησης (**heuristic**), όταν φυσικά για την επίλυσή τους εφαρμοστούν ευριστικοί αλγόριθμοι αναζήτησης.

```
    evaluate(R,RF,V).
evaluate([(X,Y,T)|R],[(X,Y,AT)|RF],V):-     % T σε κάποια άλλη θέση
    T \=AT,
    evaluate(R,RF,RV),
    V is RV+1.                               % αύξησε ευριστική τιμή κατά 1
```

Π1.2.2 Αλγόριθμοι Αναζήτησης

Στην ενότητα αυτή παρατίθεται η υλοποίηση των πιο αντιπροσωπευτικών αλγορίθμων αναζήτησης. Όλα τα προγράμματα υποθέτουν την ύπαρξη των **initial_state**, **goal** και **operator** που εξαρτώνται από το πρόβλημα. Οι υλοποιήσεις που παρουσιάζονται δεν είναι οι πιο αποδοτικές, γιατί το κριτήριο για την υλοποίηση ήταν η ευκολία κατανόησης του κώδικα από τον αναγνώστη.

Αναζήτηση πρώτα σε βάθος (DFS)

Η αναζήτηση πρώτα σε βάθος (DFS) υλοποιείται εύκολα στην PROLOG, αφού μπορεί να εκμεταλλευτεί τον ενσωματωμένο μηχανισμό εκτέλεσής της, ο οποίος ακολουθεί αυτή τη στρατηγική. Η έναρξη εκτέλεσης του προγράμματος γίνεται με την ερώτηση **?-godfs(Solution)**, όπου στη μεταβλητή **Solution** επιστρέφεται η λύση του προβλήματος, σε μορφή λίστας. Στην υλοποίηση που ακολουθεί πραγματοποιείται έλεγχος για βρόχους μόνο στις καταστάσεις του τρέχοντος μονοπατιού αναζήτησης και όχι στο σύνολο των καταστάσεων που έχει επισκεφθεί ο αλγόριθμος. Επίσης, επειδή η λύση από τον κανόνα **dfs** επιστρέφεται με ανάστροφη σειρά, χρησιμοποιείται το ενσωματωμένο κατηγόρημα **reverse** της PROLOG, το οποίο αντιστρέφει τη σειρά των στοιχείων μιας λίστας.

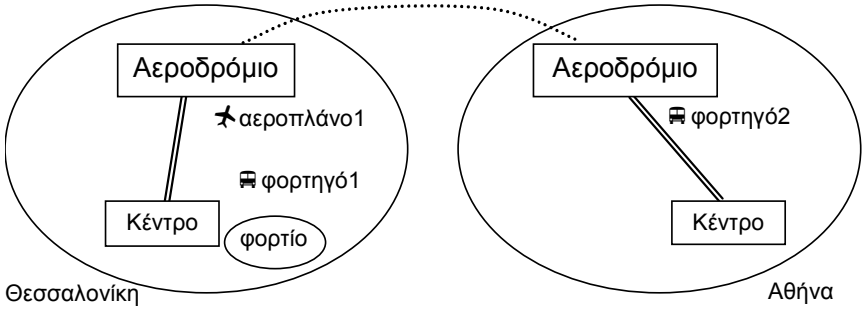
```
godfs(Solution):-
    initial_state(IS),
    dfs(IS, [IS], Solution1),
    reverse(Solution1,Solution).

dfs(State, Solution, Solution):-
    goal(State).
dfs(State, PathSoFar, Solution):-
    operator(State,Child),
    not(member(Child, PathSoFar)),
    dfs(Child, [Child|PathSoFar], Solution).
```

Αναζήτηση πρώτα σε πλάτος (BFS)

Η έναρξη εκτέλεσης του προγράμματος γίνεται με την κλήση **?-gobfs(Solution)**, όπου στη μεταβλητή **Solution** επιστρέφεται η λύση του προβλήματος. Στην υλοποίηση που ακολουθεί γίνεται έλεγχος για βρόχους στις καταστάσεις που επισκέπτεται ο αλγόριθμος.

Π1.2.4 Σχεδιασμός Ενεργειών

Έστω το πρόβλημα μεταφοράς φορτίων (transportation logistics) που φαίνεται στο Σχήμα Π1.3. Συγκεκριμένα, έστω δύο πόλεις, η Αθήνα και η Θεσσαλονίκη. Κάθε πόλη έχει δύο τοποθεσίες, το αεροδρόμιο και το κέντρο της. Κάθε πόλη διαθέτει ένα φορτηγό, το οποίο μπορεί να μετακινείται μεταξύ των δύο τοποθεσιών της πόλης, αλλά όχι από τη μία πόλη στην άλλη. Επιπλέον, υπάρχει ένα αεροπλάνο που μπορεί να μετακινείται μεταξύ των δύο αεροδρομίων. Τέλος, υπάρχει ένα φορτίο, το οποίο αρχικά βρίσκεται στο κέντρο της Θεσσαλονίκης. Το φορτίο μπορεί να φορτωθεί/ξεφορτωθεί στα φορτηγά των δύο πόλεων καθώς και στο αεροπλάνο.



Σχήμα Π1.3: Ένα πρόβλημα μεταφοράς φορτίων.

Αρχικά το φορτίο δεν είναι φορτωμένο πουθενά και βρίσκεται στο κέντρο της Θεσσαλονίκης. Έστω επίσης ότι αρχικά το φορτηγό της Θεσσαλονίκης βρίσκεται στο κέντρο της Θεσσαλονίκης, το αεροπλάνο βρίσκεται στο αεροδρόμιο της Θεσσαλονίκης και το φορτηγό της Αθήνας βρίσκεται στο αεροδρόμιο της Αθήνας, όπως φαίνεται στο Σχήμα Π1.3. Το ζητούμενο είναι να μεταφερθεί το φορτίο στο κέντρο της Αθήνας.

Το σύστημα σχεδιασμού ενεργειών που ακολουθεί δέχεται περιγραφές προβλημάτων κατά STRIPS και χρησιμοποιεί μια απλή αναζήτηση κατά πλάτος για την επίλυσή τους. Για τη λειτουργία του απαιτεί τον ορισμό των στατικών γεγονότων του προβλήματος σαν απλά γεγονότα PROLOG, ενώ τα δυναμικά γεγονότα της αρχικής κατάστασης πρέπει να περιλαμβάνονται σε ένα γεγονός της μορφής:

```
initial( [fact1, fact2, …]).
```

όπου οι όροι με πλάγια γράμματα πρέπει να αντικατασταθούν από τα δυναμικά γεγονότα που αληθεύουν στην αρχική κατάσταση. Παρόμοια, οι στόχοι του προβλήματος πρέπει να περιλαμβάνονται σε ένα γεγονός της μορφής:

```
goal([fact1, fact2, …]).
```

Τέλος τα σχήματα των ενεργειών πρέπει να οριστούν σαν κανόνες της μορφής:

```
operator(name(V1, V2, …)          % το όνομα και οι παράμετροι της ενέργειας
    [prec1, prec2, …],            % η λίστα προϋποθέσεων
    [del1, del2, …],              % η λίστα διαγραφής
    [add1, add2, …]):-            % η λίστα προσθήκης
    fact1, fact2, …. .            % Στατικές προϋποθέσεις εφαρμογής.
```