Παραδείγματα σε PROLOG

Στο παράρτημα αυτό παρουσιάζεται η υλοποίηση διαφόρων θεμάτων της ΤΝ, 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, συνεπώς η μελέτη των παραδειγμάτων προϋποθέτει καλή γνώση της γλώσσας. Φυσικά θα μπορούσε να είχε χρησιμοποιηθεί και άλλη γλώσσα προγραμματισμού, ωστόσο η συγκεκριμένη γλώσσα ενδείκνυται για εφαρμογές ΤΝ, λόγω της απλότητας με την οποία περιγράφεται η γνώση ενός προβλήματος.

Το παράρτημα αυτό δεν έχει σκοπό να μάθει στον αναγνώστη τη γλώσσα PROLOG. Αντίθετα, θεωρεί ότι υπάρχει ένα βασικό επίπεδο κατανόησής της και στόχος του είναι η επίδειξη της χρήσης της για επίλυση προβλημάτων που παρουσιάστηκαν στο βιβλίο.

Π1.1 Αλγόριθμοι αναζήτησης

Στην ενότητα αυτή παρατίθεται η υλοποίηση των πιο αντιπροσωπευτικών αλγορίθμων αναζήτησης. Όλα τα προγράμματα υποθέτουν ότι υπάρχουν δύο γεγονότα **`initial_state(State)`** και **`goal(State)`**, τα οποία δηλώνουν την αρχική και την τελική κατάσταση του προβλήματος αντίστοιχα (τα ορίσματα **`State`** πρέπει να είναι λίστες), καθώς και ένας ή περισσότεροι κανόνες με κεφαλή της μορφής **`operator(State,Child)`**, οι οποίοι αντιπροσωπεύουν τους τελεστές μετάβασης. Οι υλοποιήσεις που παρουσιάζονται δεν είναι οι πιο αποδοτικές, μιας και το κριτήριο επιλογής ήταν η ευκολία κατανόησης του κώδικα.

Π1.1.1 Αλγόριθμοι Τυφλής Αναζήτησης

Παρουσιάζεται στη συνέχεια η υλοποίηση διαφόρων αλγορίθμων τυφλής αναζήτησης σε PROLOG, καθώς και η εφαρμογή τους στην επίλυση του προβλήματος των ιεραποστόλων.

Αναζήτηση πρώτα σε βάθος

Η αναζήτηση πρώτα σε βάθος (DFS) υλοποιείται εύκολα στην PROLOG, αφού μπορεί να εκμεταλλευτεί τον ενσωματωμένο μηχανισμό αναζήτησης/εκτέλεσής της, ο οποίος επίσης ακολουθεί αυτή τη στρατηγική. Η έναρξη εκτέλεσης του προγράμματος γίνεται με την κλήση **`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.

Να σημειωθεί τέλος ότι το παρακάτω πρόγραμμα υποθέτει την δήλωση των τελεστών του προβλήματος, μέσω κανόνων του κατηγορήματος **operator/2**. Παράδειγμα ορισμού τελεστών ακολουθεί παρακάτω, στην ενότητα για το πρόβλημα των ιεραποστόλων.

```
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

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

```
gobfs(Solution):-
   initial_state(IS),
   bfs([[IS]],Solution1),
   reverse(Solution1,Solution).

bfs([[State|Path]|_], [State|Path]):-
   goal(State).
bfs([[State|Path]|RestFrontierSet], Solution):-
   expand(State,Path,ChildrenStates),
   append(RestFrontierSet,ChildrenStates,NewFrontierSet),
   bfs(NewFrontierSet, Solution).

expand(State,Path,Children):-
   findall([Child,State|Path],operator(State,Child),Children).
```

Να σημειωθεί ότι το παραπάνω πρόγραμμα μπορεί πολύ εύκολα να τροποποιηθεί ώστε να υλοποιεί τον αλγόριθμο αναζήτησης πρώτα σε βάθος, αλλάζοντας την κλήση στην **append/3** στο δεύτερο κανόνα **bfs/2** σε:

```
   append(ChildrenStates, RestFrontierSet, NewFrontierSet)
```

Επαναληπτική εκβάθυνση

Η έναρξη εκτέλεσης του προγράμματος γίνεται με την κλήση **goid(Solution)**, όπου στη μεταβλητή **Solution** επιστρέφεται η λύση του προβλήματος. Ο αλγόριθμος υλοποιήθηκε με επανάληψη (κατηγόρημα **repeat**) και όχι με αναδρομή, μιας και η επανάληψη είναι πιο αποδοτική. Στο παρακάτω πρόγραμμα το τρέχον βάθος δίνεται από

το δυναμικό γεγονός **depth/1**, ενώ οι κανόνες **bdfs/4** υλοποιούν την περιορισμένη αναζήτηση πρώτα σε βάθος (bounded depth first search).

```
goid(Solution):-
    initial_state(IS),
    id(IS,Solution).

id(IS,Solution):-
    abolish(depth/1),
    assert(depth(0)),
    repeat,
    retract(depth(PreviousD)),
    NextD is PreviousD + 1,
    assert(depth(NextD)),write(NextD),nl,
    bdfs(NextD,IS,[IS],Solution).

bdfs(_,State,_,[State]):-
    goal(State).
bdfs(Depth,State,PathSoFar,[State|RestSolution]):-
    Depth>0,
    operator(State,Child),
    not member(Child,PathSoFar),
    NewDepth is Depth-1,
    bdfs(NewDepth,Child,[State|PathSoFar],RestSolution).
```

Επέκταση και οριοθέτηση

Η έναρξη εκτέλεσης του προγράμματος γίνεται με την κλήση **go_bb**. Ο αλγόριθμος δεν τερματίζει μόλις βρει την πρώτη λύση, αλλά συνεχίζει για να βρει και άλλες καλύτερες. Οι λύσεις τυπώνονται στην οθόνη με εντολές **write**. Κάθε λύση που βρίσκεται είναι καλύτερη από όλες τις προηγούμενες που έχουν βρεθεί.

Να σημειωθεί τέλος ότι η υλοποίηση του κανόνα **evaluate_cost/3**, ο οποίος επιστρέφει το κόστος μετάβασης από μια κατάσταση σε μια γειτονική της, μπορεί να διαφέρει ανάλογα με το πρόβλημα. Στην παρακάτω υλοποίηση καταχρηστικά θεωρήθηκε ότι το κόστος αυτό ισούται με τη μονάδα.

```
go_bb:-
    abolish(bestsofar/2),
    assert(bestsofar(_,9999)),
    initial_state(InitialState),
    bandb(InitialState,[],Solution,0,TotalCost),
    update(Solution,TotalCost),
    write(TotalCost),nl,
    fail.

go_bb:-
    bestsofar(Solution,Cost),
    write(Solution),nl,
```

```
    write(Cost),nl.

update(Solution,TotalCost):-
    retract(bestsofar(_,_)),
    assert(bestsofar(Solution,TotalCost)),!.

bandb(State,_,[],Cost,Cost):-
    goal(State).
bandb(State,PathSoFar,[State|RestSolution],CostSoFar,TotalCost):-
    operator(State,Next),
    not member(Next,PathSoFar),
    evaluate_cost(State,Next,C),
    NewCostSoFar is CostSoFar+C,
    bestsofar(_,BestCost),
    NewCostSoFar < BestCost,
    bandb(Next,[Next|PathSoFar],RestSolution,NewCostSoFar,TotalCost).

evaluate_cost(State,Next,1).
```

Παράδειγμα: Το πρόβλημα των ιεραποστόλων

Το πρόβλημα των ιεραποστόλων έχει περιγραφεί στο 2ο κεφάλαιο. Εδώ παρουσιάζεται η κωδικοποίηση του προβλήματος, δηλαδή της αρχικής κατάστασης και των τελεστών μετάβασης, έτσι ώστε να είναι δυνατή η επίλυσή του από τους αλγόριθμους που περιγράφηκαν παραπάνω.

```
% Αρχική και τελική κατάσταση
initial_state(state(left(3,3),right(0,0), boat_left)).
goal(state(left(0,0),right(3,3), boat_right)).

% Τελεστής μετακίνησης της βάρκας από την αριστερή (αφετηρία) στη
% δεξιά όχθη (προορισμός)
operator(state(left(ML,CL),right(MR,CR),boat_left),
     state(left(NML,NCL),right(NMR,NCR),boat_right)) :-
    move(M, C),
    M =< ML,
    C =< CL,
    NML is ML-M,
    NCL is CL-C,
    NMR is MR+M,
    NCR is CR+C,
    valid(NML, NCL),
    valid(NMR, NCR).

% Τελεστής επιστροφής της βάρκας από τη δεξιά όχθη (προορισμός)
% στην αριστερή (αφετηρία).
operator(state(left(ML,CL),right(MR,CR),boat_right),
     state(left(NML,NCL),right(NMR,NCR),boat_left)) :-
    move(M, C),
    M =< MR,
```

```
    C =< CR,
    NML is ML+M,
    NCL is CL+C,
    NMR is MR-M,
    NCR is CR-C,
    valid(NML, NCL),
    valid(NMR, NCR).

% Βοηθητικοί κανόνες
valid(M,C) :- M >= C, !.
valid(0,_).
```

Π1.1.2 Αλγόριθμοι Ευριστικής Αναζήτησης

Οι αλγόριθμοι ευριστικής αναζήτησης απαιτούν την ύπαρξη μιας ευριστικής συνάρτησης, η οποία θα εκτιμά την απόσταση κάθε κατάστασης του χώρου αναζήτησης από την τελική κατάσταση. Παρακάτω θεωρείται ότι η ευριστική συνάρτηση δίνεται μαζί με την περιγραφή του προβλήματος, μέσω ενός κανόνα με κεφαλή **heuristic(S1,S2,V)**, όπου **S1** και **S2** οι καταστάσεις των οποίων θέλουμε να βρούμε την απόσταση και **V** η ζητούμενη απόσταση.

Ο αλγόριθμος αναρρίχησης λόφου

Πρώτα παρουσιάζεται ο αλγόριθμος αναρρίχησης λόφου. Ο αλγόριθμος καλείται με την κλήση **gohc**(**Solution**), όπου στη μεταβλητή **Solution** επιστρέφεται η λύση του προβλήματος.

```
gohc(Solution):-
    initial_state(IS),
    goal(FS),
    heuristic(IS, FS, V),
    hc(IS, [V-IS], Solution1, FS),
    reverse(Solution1, Solution).

hc(FS, Solution, Solution, FS):-!.
hs(State, PathSoFar, Solution, FS):-
    % βρες όλες τις καταστάσεις-παιδιά
    next_states(State, Children, FS),
    % ταξινόμηση των παιδιών και επιλογή του καλύτερου
    keysort(Children, [BestChild|_]),
    % Έλεγχος για βρόχο
    not member(Children, PathSoFar),
    hc(BestChild, [BestChild|PathSoFar], Solution, FS).

% Βρες όλες τις επόμενες καταστάσεις
% με την ευριστική τους τιμή
next_states(V-State, Children, FS):-
    findall(HV-Child,
          (operator(State, Child) , heuristic(Child, FS, HV)),
          Children).
```

*Ο Αλγόριθμος A**

Η έναρξη εκτέλεσης του προγράμματος γίνεται με την κλήση **goastar(Solution)**., όπου στη μεταβλητή **Solution** επιστρέφεται η λύση του προβλήματος. Αφήνεται σαν άσκηση στον αναγνώστη να τροποποιηθεί ο αλγόριθμος, ώστε αυτός να λειτουργεί σαν αναζήτηση πρώτα στο καλύτερο (best-first).

```
goastar(Solution):-
    initial_state(IS),
    goal(FS),
    heuristic(IS,FS,V),
    astar([V-[IS]],FS, Solution).

astar([Value-[FinalState|Path]|_],FinalState, [FinalState|Path]):-!.

astar([BestPath|RestPaths],FP, Solution):-
    next_states(BestPath,NewPaths,FP),
    append(NewPaths,RestPaths,Frontier),
    keysort(Frontier,OrderedFrontier),
    astar(OrderedFrontier,FP, Solution).

next_states(V-[State|Path],NewPaths,FP):-
    findall( V1-[NewState,State|Path],
            (
            operator(State,NewState),
            heuristic(NewState,FP,HV),
            length(Path,L),
            V1 is L+1+HV
            ),
        NewPaths).
```

Παράδειγμα: Ένα 3x3 παζλ

Στη συνέχεια παρουσιάζεται η περιγραφή του προβλήματος 3x3 παζλ, δηλαδή της αρχικής κατάστασης, των τελεστών μετάβασης και της ευριστικής συνάρτησης, έτσι ώστε να είναι δυνατή η επίλυσή του από κάποιον αλγόριθμο ευριστικής αναζήτησης. Οι καταστάσεις παριστάνονται με λίστες 9 στοιχείων, όπως ακριβώς διαβάζονται τα πλακάκια κατά γραμμές από πάνω αριστερά μέχρι κάτω δεξιά. Το σύμβολο **e** υποδηλώνει το κενό.

```
% Αρχική κατάσταση
initial_state([2,3,6,1,5,4,7,e,8]).

% Τελική κατάσταση
goal([1,2,3,4,5,6,7,8,e]).

% Τελεστής μετάβασης
operator(P,NP):-
    trans(TempP,P),
    member((X,Y,e),TempP),
    move(X,Y,X1,Y1),
```

```
    member((X1,Y1,T),TempP),
    change(X,Y,X1,Y1,T,TempP,NewTempP),
    trans(NewTempP,NP).

% Βοηθητικοί κανόνες
trans([(1,1,T1),(1,2,T2),(1,3,T3),
    (2,1,T4),(2,2,T5),(2,3,T6),
    (3,1,T7),(3,2,T8),(3,3,T9)],
    [T1,T2,T3,T4,T5,T6,T7,T8,T9]).

move(X,Y,X,NY):-NY is Y-1,NY>0.
move(X,Y,X,NY):-NY is Y+1,NY<4.
move(X,Y,NX,Y):-NX is X-1,NX>0.
move(X,Y,NX,Y):-NX is X+1,NX<4.

change(X,Y,X1,Y1,T,[],[]).
change(X,Y,X1,Y1,T,[(X,Y,e)|R],[(X,Y,T)|CR]):-
    change(X,Y,X1,Y1,T,R,CR).
change(X,Y,X1,Y1,T,[(X1,Y1,T)|R],[(X1,Y1,e)|CR]):-
    change(X,Y,X1,Y1,T,R,CR).
change(X,Y,X1,Y1,T,[(AX,AY,AT)|R],[(AX,AY,AT)|CR]):-
    AT\=e, AT\=T,
    change(X,Y,X1,Y1,T,R,CR).

% Ευριστική συνάρτηση
heuristic(S1,S2,V):-
    trans(TS1,S1),
    trans(TS2,S2),
    evaluate(TS1,TS2,V).

evaluate([],[],0).
evaluate([(X,Y,T)|R],[(X,Y,T)|RF],V):-
    evaluate(R,RF,V).
evaluate([(X,Y,T)|R],[(X,Y,AT)|RF],V):-
    T \=AT,
    evaluate(R,RF,RV),
    V is RV+1.
```

Π1.1.3 Αναπαράσταση Γνώσης με Πλαίσια

Στην ενότητα αυτή παρουσιάζεται η αναπαράσταση της γνώση για μια ιεραρχία αυτοκινήτων χρησιμοποιώντας πλαίσια. Παράλληλα επιδεικνύεται η δυνατότητα αναζήτησης τιμών μέσα στην ιεραρχία αυτή.

Αρχικά ορίζονται ένα γενικό πλαίσιο "αυτοκίνητο" και τρεις ειδικεύσεις του που αφορούν τρία συγκεκριμένα αυτοκίνητα. Στο γενικό πλαίσιο ορίζονται γενικές ιδιότητες και δίνονται προκαθορισμένες τιμές σε αυτές, ενώ στα ειδικότερα πλαίσια είτε ορίζονται νέες ιδιότητες ή δίνονται διαφορετικές τιμές για τις ιδιότητες του γενικού πλαισίου.

```
object(car).
```

```
slot(car,ako,thing).
slot(car,country_of_manufacture,range([britain,france,germany,italy])).
slot(car,miles_per_gallon,range([1,100])).
slot(car,reliability,range([low,medium,high])).
slot(car,fuel_consumption,compute(fuel_consumption)).

object(bmw).
slot(bmw,isa,car).
slot(bmw,country_of_manufacture,germany).
slot(bmw,reliability,high).

object(bmw_320).
slot(bmw_320,isa,bmw).
slot(bmw_320,miles_per_gallon,28).

object(my_bmw).
slot(my_bmw,isa,bmw_320).
slot(my_bmw,miles_per_gallon,20).
slot(my_bmw,reliablity,medium).

compute(Obj,fuel_consumption,V):-
   getprop(Obj,miles_per_gallon,X),
   V is 1/X.
```

Συλλογιστική

Το πρόγραμμα που ακολουθεί μπορεί να απαντά σε ερωτήσεις του τύπου: "Ποια η τιμή της ιδιότητας X του αντικειμένου Y". Το πρόγραμμα ψάχνει πρώτα να βρει εάν το αντικείμενο Y έχει την ιδιότητα X. Εάν ναι, επιστρέφει την τιμή της ιδιότητας. Εάν όχι, ψάχνει να βρει εάν η ιδιότητα X εμφανίζεται σε κάποιο από τα γονικά αντικείμενα του Y, αρχίζοντας από το πλησιέστερο. Μόλις βρει την ιδιότητα X, επιστρέφει την τιμή της. Εάν τελικά η ιδιότητα X δε βρεθεί σε κανένα από τα γονικά αντικείμενα, εμφανίζονται στην οθόνη σχετικά μηνύματα λάθους.

Για να πάρουμε την τιμή μιας ιδιότητας ενός πλαισίου, πρέπει να γίνει η κλήση **getv(Obj,Prop,V)**, όπου **Obj** και **Prop** το αντικείμενο και η ιδιότητα που μας ενδιαφέρουν (μεταβλητές εισόδου), και **V** η ζητούμενη τιμή.

```
getv(Obj,Prop,V):-
   getprop(Obj,Prop,Property),
   match(Obj,Property,V).

match(Obj,compute(Prop),V):-compute(Obj,Prop,V),!.
match(Obj,range(Constraints),V):- nonvar(V),test(Constraints,V).
match(_,V,V).
test([N1,N2],V):- integer(N1),integer(N2),V>=N1,V=<N2,!.
test(C,V):- member(V,C),!.
test(_,V):-write('This is not a valid value property'),nl.

getprop(Obj,Prop,range(Constraints)):-
      slot(Obj,Prop,range(Constraints)),!.
```

```
getprop(Obj,Prop,compute(Prop)):-
       slot(Obj,Prop,compute(Prop)),!.
getprop(Obj,Prop,V):-
       slot(Obj,Prop,V),!.

getprop(Obj,Prop,V):-
       slot(Obj,isa,Class),
       getprop(Class,Prop,V),!.
getprop(Obj,Prop,V):-
       write('i do not know value of property '),
       write(Prop), write(' of '), write(Obj), nl,
       !,fail.
```