Επίλυση – Resolution

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Η Μέθοδος της Επίλυσης στον Λογικό Προγραμματισμό

Λογικός Προγραμματισμός

- Εξαγωγή συμπερασμάτων από ένα σύνολο λογικών εκφράσεων.
- Χρησιμοποιεί μια περιορισμένη γλώσσα λογικής η οποία επιτρέπει την αποδοτική απόδειξη θεωρημάτων μέσω της μεθόδου της επίλυσης.
- Η λογική αυτή περιέχει προτάσεις της μορφής:

  $\forall x_1 \ldots \forall x_k \, (B_1 \wedge \ldots \wedge B_n \rightarrow A)$

  οι οποίες μπορούν να γραφτούν ως διαζεύξεις με μόνο ένα θετικό όρο

  $\forall x_1 \ldots \forall x_k \, (\neg B_1 \vee \ldots \vee \neg B_n \vee A)$

  και έχουν προτασιακή μορφή $\{\neg B_1, \ldots, \neg B_n , A\}$
- Μια τέτοια πρόταση στον λογικό προγραμματισμό γράφεται ως

  $\forall x_1 \ldots \forall x_k \, (A \leftarrow B_1, \ldots, B_n)$

  και μπορεί να ερμηνευθεί ως η διαδικασία: *για να υπολογίσεις το A υπολόγισε τα* $B_1, \ldots, B_n$.

Παράδειγμα

- Αξιώματα για συμβολοσειρές
  1. $\forall x\ substr(x,x)$
  2. $\forall x \forall y \forall z\ (substr(x,z) \leftarrow (suffix(y,z) \wedge substr(x,y))$
  3. $\forall x \forall y\ suffix(x,yx)$
  4. $\forall x \forall y \forall z\ (substr(x,z) \leftarrow (prefix(y,z) \wedge substr(x,y))$
  5. $\forall x \forall y\ prefix(x,xy)$

  Μπορούν να τύχουν ερμηνείας ως:
  1. Κάθε συμβολοσειρά είναι υποσυμβολοσειρά του εαυτού της
  2. Για να αποφασίσουμε αν το x είναι υποσυμβολοσειρά του z τότε εντοπίζουμε συμβολοσειρά y τέτοια ώστε η y να είναι επίθημα της z και η x υποσυμβολοσειρά της y.
  3. Η x είναι επίθημα της yx
  4. Για να αποφασίσουμε αν το x είναι υποσυμβολοσειρά του z τότε εντοπίζουμε συμβολοσειρά y τέτοια ώστε η y να είναι πρόθημα της z και η x υποσυμβολοσειρά της y.
  5. Η x είναι πρόθημα της xy

Βασική Ιδέα

- Πιθανά ερωτήματα (στόχοι):
  - substr(ααβ,γβαβααγβααββ) ;
  - ∃x (substr(x,αβγααγ) ∧ suffix (δαβ, x)) ;

- Βασική Ιδέα Διαδικασίας Επίλυσης στον ΛΠ
  - Θεώρησε την άρνηση του στόχου,
    π.χ. ¬ ∃x (substr(x,αβγααγ) ∧ suffix (δαβ, x))
  - Μετάτρεψε σε προτασιακή μορφή:
    {{¬substr(x,αβγααγ) , ¬suffix (δαβ, x)}}
  - Εφάρμοσε επίλυση στο σύνολο των αξιωμάτων που απαρτίζουν το πρόγραμμα και την προτασιακή μορφή του ερωτήματος.

Ροή Προγράμματος

- Μη ντετερμινισμός: Δυνατόν να υπάρχουν πολλά αξιώματα τα οποία μπορούν να συνδεθούν με τον στόχο του προγράμματος. Ποιο από αυτά θα επιλεχθεί σε κάθε βήμα;
- Ροή Προγράμματος:
  - Στον διαδικαστικό προγραμματισμό η ροή του προγράμματος καθορίζεται πλήρως από τον προγραμματιστή μέσω της σειράς παράταξης των εντολών που το απαρτίζουν.
  - Στον Λογικό Προγραμματισμό ο προγραμματιστής γράφει δηλώσεις που συλλαμβάνουν τη σχέση που έχουν οι οντότητες του προγράμματος. Εναπόκειται στον compiler να επιλύσει τον μη ντετερμινισμό που εμπεριέχεται στο πρόγραμμα.
- Κανόνας Υπολογισμού: Επιλογή στοιχείου της πρότασης-στόχος στο οποίο θα εφαρμοστεί επίλυση.
- Κανόνας Εύρεσης: Επιλογή "αξιώματος" στο οποίο να εφαρμοστεί επίλυση έναντι του στοιχείου της πρόταση-στόχος που έχουμε επιλέξει.

Ορισμοί

- Σε μια πρόταση $A \leftarrow B_1,\ldots,B_n$ (όρος Horn) ονομάζουμε
  - Τον θετικό όρο A κεφαλή (head) και
  - Τους αρνητικούς όρους $B_1,\ldots,B_n$ κορμό (body).
- Αν n = 0, τότε δεν υπάρχει κορμός και η πρόταση έχει τη μορφή $A \leftarrow$ και ονομάζεται γεγονός (fact).
- Αν η πρόταση δεν έχει κεφαλή, $\leftarrow B_1,\ldots,B_n$, ονομάζεται στόχος (goal).
- Ένα σύνολο από όρους Horn που δεν αποτελούν στόχους και των οποίων η κεφαλή χρησιμοποιεί το ίδιο κατηγορηματικό σύμβολο ονομάζεται διαδικασία (procedure).
- Μία διαδικασία η οποία δεν περιέχει μεταβλητές ονομάζεται βάση δεδομένων (database).
- Ένα σύνολο από διαδικασίες ονομάζεται λογικό πρόγραμμα.

Παράδειγμα

- Το πιο κάτω πρόγραμμα περιέχει δύο διαδικασίες, μία από τις οποίες αποτελεί βάση δεδομένων.

1. $q(x,y) \leftarrow p(x,y)$
2. $q(x,y) \leftarrow p(x,z),q(z,y)$

3. $p(b,a) \leftarrow$
4. $p(c,a) \leftarrow$
5. $p(d,b) \leftarrow$
6. $p(e,b) \leftarrow$
7. $p(f,b)$
8. $p(h,g)$
9. $p(i,h)$
10. $p(j,h)$

Αντικαταστάσεις ορθής απάντησης

- Έστω P ένα πρόγραμμα και G ένας στόχος. Μία αντικατάσταση θ ονομάζεται αντικατάσταση ορθής απάντησης αν $P \models \forall(\neg G\theta)$ όπου ο ποσοδείκτης $\forall$ συμβολίζει την εφαρμογή του καθολικού ποσοδείκτη σε όλες τις μεταβλητές που εμφανίζονται ελεύθερες στην G.

- Παράδειγμα:

  Έστω P το σύνολο των αξιωμάτων της αριθμητικής.

  Έστω G η πρόταση $\neg(6+y=13)$. $\neg G$ είναι η πρόταση $6+y=13$ και αντικατάσταση ορθής απάντησης για τη G αποτελεί η $\theta = \{7/y\}$.

  Έστω G η πρόταση $\neg(x = y+13)$. Αντικατάσταση ορθής απάντησης για τη G αποτελεί η $\theta = \{x - 13/y\}$.

Παράδειγμα

ancestor(X,Y) ← parent(X,Y)

ancestor(X,Y) ← parent(X,Z), ancestor(Z,Y)

parent(bob,alan) ←	parent(fred,dave)
parent(catherine,alan) ←	parent(harry,george)
parent(dave,bob)	parent(ida,harry)
parent(ellen,bob)	parent(john,harry)

Ο στόχος

← ancestor(Y, bob), ancestor(bob,Z)

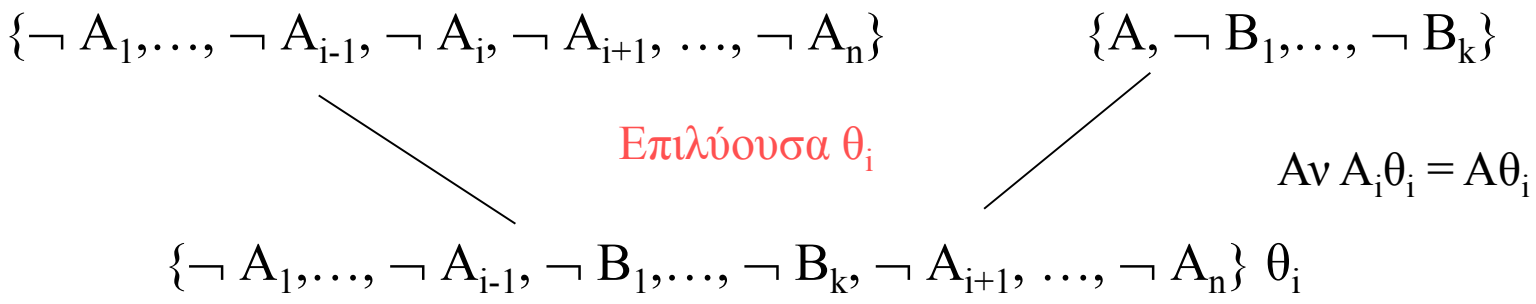
θα επιφέρει το αποτέλεσμα true και θα επιστρέψει την αντικατάσταση ορθής απάντησης Y = dave , Z = alan

Επίλυση SLD

- Η απόδειξη ενός στόχου γίνεται με τη χρήση του κανόνα SLD-Επίλυση ο οποίος αποτελεί ειδική περίπτωση της μεθόδου της επίλυσης εφόσον αναφέρεται σε ζεύγη προτάσεων που περιέχουν μόνο ένα θετικό όρο.

$$\frac{\leftarrow A_1,\ldots,A_{i-1}, A_i, A_{i+1}, \ldots, A_n, \quad A \leftarrow B_1,\ldots,B_k, A_i\theta_i = A\theta_i}{\leftarrow \ (A_1,\ldots,A_{i-1}, B_1,\ldots,B_k, A_{i+1}, \ldots, A_n)\ \theta_i}$$

- Αιτιολόγηση Κανόνα:



Παράδειγμα

- Θεωρήστε το πρόγραμμα P από τη διαφάνεια 5-7. Θέλουμε να αποδείξουμε ότι η πρόταση $\exists y \exists z\ (q(y,b) \wedge q(b,z))$ προκύπτει ως λογικό επακόλουθο του προγράμματος.

- Τότε, αφού $\neg\ (\exists y \exists z\ (q(y,b) \wedge q(b,z))) = \forall y \forall z\ \neg(q(y,b) \wedge q(b,z)) = \forall y\ \forall z\ (\neg q(y,b) \vee \neg\ q(b,z)) = \forall y\ \forall z\ (q(y,b) \wedge q(b,z) \rightarrow \text{False})$ ο στόχος της απόδειξης είναι ο

  $\leftarrow q(y,b), q(b,z)$

- Έχουμε ότι:
  - Από το (1) και το q(y,b), $\leftarrow p(y,b), q(b,z)$
  - Από το (5) και το p(y,b), $\leftarrow q(b,z)$ μέσω της αντικατάστασης {d/y}
  - Από το (1) και το q(b,z), $\leftarrow p(b,z)$
  - Και τέλος, από το (3) και το p(b,z), $\perp$ μέσω της αντικατάστασης {a/z}

- Μπορούμε να επιβεβαιώσουμε ότι πράγματι η {d/y, a/z} είναι αντικατάσταση ορθής απάντησης της πρότασης.

Ορθότητα και Πληρότητα

- Θεώρημα (Ορθότητα): Έστω P ένα σύνολο προτάσεων προγράμματος, και G ένας στόχος. Έστω η ύπαρξη μιας SLD-διάψευσης του G. Αν $\theta = \theta_1\theta_2\ldots\theta_n$ η ακολουθία των ενοποιητών που χρησιμοποιήθηκαν στη διάψευση και σ ο περιορισμός του θ στις μεταβλητές του G, τότε σ είναι μια αντικατάσταση ορθής απάντησης για τον στόχο G.

- Θεώρημα (Πληρότητα): Έστω P ένα σύνολο προτάσεων προγράμματος, και G ένας στόχος. Έστω σ μια αντικατάσταση ορθής απάντησης για τον στόχο G. Τότε υπάρχει μια SLD-διάψευση του G από το P τέτοια ώστε σ είναι ο περιορισμός της ακολουθίας των ενοποιητών που χρησιμοποιήθηκαν στη διάψευση $\theta = \theta_1\theta_2\ldots\theta_n$ στις μεταβλητές του G.

SLD-Επίλυση και μη-ντετερμινισμος (1)

- Θεωρήστε ξανά το πρόγραμμα P από τη διαφάνεια 5-7, την πρόταση $\exists y \exists z\ (q(y,b) \wedge q(b,z))$ και τον σχετικό στόχο $\leftarrow q(y,b), q(b,z)$
- Έχουμε επίσης ότι:
  - Από το (1) και το q(y,b), $\leftarrow p(y,b), q(b,z)$
  - Από το (6) και το p(e,b), $\leftarrow q(b,z)$ μέσω της αντικατάστασης {e/y}
  - Από το (1) και το q(b,z), $\leftarrow p(b,z)$
  - Και τέλος, από το (3) και το p(b,z), $\bot$ μέσω της αντικατάστασης {a/z}
- Επομένως και η {e/y, a/z} είναι αντικατάσταση ορθής απάντησης της πρότασης: μπορεί ο ίδιος στόχος να διαθέτει περισσότερες από μια αντικαταστάσεις ορθής απάντησης.

SLD-Επίλυση και μη-ντετερμινισμος (2)

- Αν, για το ίδιο πρόγραμμα, χρησιμοποιήσουμε ως κανόνα υπολογισμού τον:

  Να διαλέγεις πάντα το τελευταίο στοιχείο του στόχου.

  και ως κανόνα εύρεσης τον:

  Να διαλέγεις πάντα το τελευταίο συμβατό αξίωμα του προγράμματος.

  τότε θα είχαμε:

  ← q(y,b), q(b,z)
  ← q(y,b), p(b,z'), q(z',z)
  ← q(y,b), p(b,z'), p(z,z''), q(z'',z')
  …

- Αυτό δείχνει ότι υπάρχουν μέθοδοι υπολογισμού οι οποίοι δεν μας οδηγούν στην κατασκευή διάψευσης ακόμη και αν υπάρχουν διαψεύσεις.

SLD-Επίλυση και μη-ντετερμινισμος (3)

- Αν, για το ίδιο πρόγραμμα, χρησιμοποιήσουμε ως κανόνα υπολογισμού τον:

  Να διαλέγεις πάντα το πρώτο στοιχείο του στόχου

  τότε θα είχαμε:

← q(y,b), q(b,z)	
← p(y,z'), q(z',b), q(b,z)	(Από το (2))
← q(b,b), q(b,z)	(Από το (5) μέσω της αντικατάστασης {d/y,b/z'})
← p(b,b), q(b,z)	(Από το (1))

- Από τώρα και στο εξής, κανένας όρος δεν ενοποιείται με τον p(b,b). Επομένως, ενώ υπάρχει διάψευση στο πρόγραμμα, η εκτέλεση τερματίζει χωρίς να την εντοπίσει.

Δένδρα SLD

- Η διαδικασία της Επίλυσης μπορεί να αναπαρασταθεί μέσω δένδρων τα οποία έχουν την πρόταση υπό μελέτη ως ρίζα και ακμές ανάμεσα σε δύο κόμβους υπάρχουν αν υπάρχει εφαρμογή του κανόνα SLD-επίλυση που οδηγά από τον κόμβο-πατέρα προς τον κόμβο παιδί.

- Μια κατά πλάτος διερεύνηση σε ένα δένδρο SLD οδηγεί πάντα σε εύρεση διάψευσης αν υπάρχει ενώ η κατά βάθος διερεύνηση δεν συνοδεύεται από τέτοιες εγγυήσεις.

- Εντούτοις, το μέγεθος του δένδρο αυξάνεται εκθετικά από το ένα επίπεδο του δένδρο στο επόμενο και έτσι αποτελεί δαπανηρή διαδικασία.

Εργασία

(α) Θεωρήστε το πιο κάτω πρόγραμμα λογικού προγραμματισμού και χρησιμοποιήστε τη μέθοδο της SLD επίλυσης για να φθάσετε σε διάψευση του στόχου.

add(X, 0, X)

add(X, succ(Y), succ(Z)) ← add(X,Y,Z)

← add(succ(succ(0)), succ(succ(0)), U)

(β) Θεωρήστε το πιο κάτω πρόγραμμα λογικού προγραμματισμού και επιδείξτε τις δυνατές εκτελέσεις του μέσω ενός SLD-δένδρου.

add(X, 0, X)

add(X, succ(Y), succ(Z)) ← add(X,Y,Z)

← add(U, V, succ(succ(succ(0))))

Prolog – Γενικά

- PROgramming in LOGic
- Prolog: Alain Colmerauer 1973, R. Kowaksi Αρχή Επίλυσης
- Prolog compiler: David Warren, 1977
- Δηλωτική Γλώσσα: διάκριση μεταξύ λογικής και ελέγχου
- Εξίσωση Kowalski

  Πρόγραμμα = Λογική + Έλεγχος
- Κανόνας Υπολογισμού Prolog:
  - Επέλεξε το αριστερότερο στοιχείο από τον στόχο
- Κανόνας Εύρεσης Prolog:
  - Επέλεξε τον πρώτο κατάλληλο κανόνα διασχίζοντας το πρόγραμμα από την αρχή και προχωρώντας προς το τέλος.

Prolog – Οντότητες

- Ένα πρόγραμμα στην Prolog αφορά οντότητες του προβλήματος με το οποίο ασχολείται, τις ιδιότητές τους και τις σχέσεις τους. Μπορεί να περιλαμβάνει
  - Γεγονότα: παριστάνουν ιδιότητες και σχέσεις που έχουν αντικείμενα του προβλήματος μεταξύ τους – τα δεδομένα του προβλήματος
  - Κανόνες: παριστάνουν γενικευμένες σχέσεις που συνδέουν τα αντικείμενα του προβλήματος – οι συλλογισμοί μέσω των οποίων μπορούμε να καταλήξουμε σε καινούρια συμπεράσματα/γεγονότα
  - Ερωτήματα: παριστάνουν τα ζητούμενα από το πρόγραμμα
  - Σχόλια
- Ο συμβολισμός της Prolog διαφέρει από αυτόν που χρησιμοποιούμε μέχρι τώρα:
  - Οι μεταβλητές ξεκινούν με κεφαλαία γράμματα
  - Τα σύμβολα και οι σταθερές ξεκινούν με μικρά γράμματα
  - Το σύμβολο :- χρησιμοποιείται αντί του ←

Παράδειγμα

ancestor(X,Y) :- parent(X,Y)

ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y)

parent(bob,alan)

p(catherine,alan)

p(dave,bob)

p(ellen,bob)

p(fred,dave)

p(harry,george)

p(ida,harry)

p(john,harry)

Ο στόχος

:- ancestor(Y, bob), ancestor(bob,Z)

Θα επιφέρει το αποτέλεσμα true και θα επιστρέψει την αντικατάσταση ορθής απάντησης Y = dave , Z = alan

Εκτέλεση Προγράμματος

- Η αναζήτηση διάψευσης για την επίτευξη ενός στόχου μέσω των κανόνων υπολογισμού και εύρεσης της Prolog πιθανόν να οδηγήσει σε μη τερματισμό ακόμη και αν υπάρχει λύση.

- Επαφίεται στον προγραμματιστή να διατάξει τους κανόνες του προγράμματος με τέτοιο τρόπο έτσι ώστε να αποφευχθεί ο μη τερματισμός.

- Αποδοτική μέθοδος εύρεσης όρων προς ενοποίηση.

- Περιορισμένος αριθμός δομών δεδομένων προς διατήρηση

- Ανάγκη για οπισθοδρόμηση

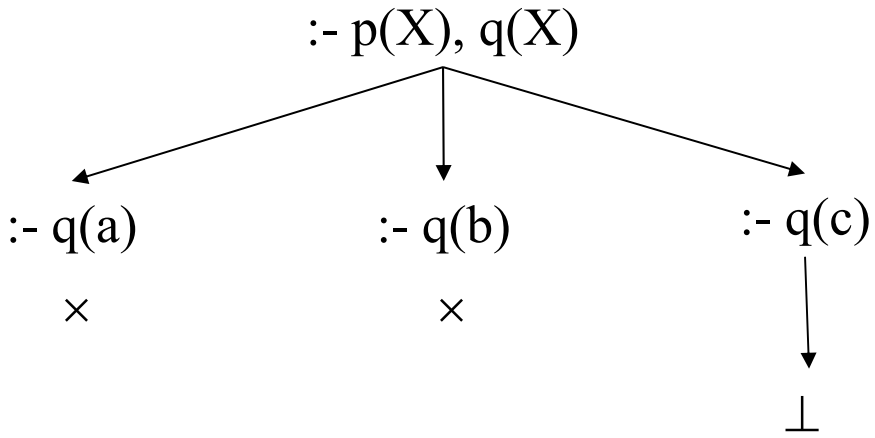
Παράδειγμα

- Θεωρήστε το πρόγραμμα με τα 4 γεγονότα και τον στόχο που εμφανίζονται πιο κάτω:

  p(a) p(b) p(c) q(c)

  :- p(X), q(X)

- Το δένδρο που θα κτιστεί κατά τη διαδικασία επίλυσης είναι το



Μη λογικά Κατηγορήματα

- Η Prolog περιέχει μη-λογικά κατηγορήματα τα οποία διευρύνουν την εκφραστικότητα και αποδοτικότητα της γλώσσας όπως:
  - Κατηγορήματα για Input/output: get, put
  - Αριθμητικές Εκφράσεις και σχετικές διαδικασίες υπολογισμού

    Result is Expression

- Παράδειγμα

```
selling_price(Item, Price) :- list_price(Item, List),
                    discount_percent(Item, Discount),
                    Price is List - List*Discount/100
```

  Η τιμή της έκφρασης List – List*Discount/100 θα τύχει υπολογισμού και θα ενοποιηθεί με τη μεταβλητή Price

- *Από τη στιγμή που μια μεταβλητή πάρει κάποια τιμή μέσω μιας εντολής is οποιαδήποτε επιπρόσθετη προσπάθεια να ενοποιηθεί με μια άλλη τιμή θα οδηγήσει σε λάθος.*

Cuts

- Μια αποκοπή (cut) αποτελεί μια σήμανση για διακοπή της μεθόδου της επίλυσης σε συγκεκριμένο σημείο.
- Θεωρήστε το πιο κάτω πρόγραμμα για υπολογισμό παραγοντικών αριθμών (fact) και έλεγχο κατά πόσο ο N-ιοστός παραγοντικός αριθμός είναι άρτιος .

```
fact(0,1).
fact(N,F) :-
   N1 is N-1,
   fact(N1,F1),
   F is N*F1.

check(N):- fact(N,F),
           even(F).
```

- Έστω το ερώτημα check(0). Αφού fact(0,1) και όχι even(1), η διαδικασία της επίλυσης θα οπισθοδρομήσει και θα επιχειρήσει να χρησιμοποιήσει τον δεύτερο κανόνα fact. Και η κλήση fact(-1,1) θα ξεκινήσει ένα ατέρμονο υπολογισμό.

Cuts

- Αυτό μπορεί να αποφευχθεί μέσω μιας αποκοπής.
- Γράφουμε

```
fact(0,1):-! .
```

απαγορεύοντας οπισθοδρόμηση μετά από αυτό το σημείο στη διαδικασία.

- Οι αποκοπές επεμβαίνουν στη φιλοσοφία του Λογικού Προγραμματισμού και είναι καλύτερο να αποφεύγονται. Στο παράδειγμα θα μπορούσαμε να γράψουμε

```
fact(N,F) :-
        N > 0,
        N1 is N-1,
        fact(N1,F1),
        F is N*F1 .
```