PARADIGME DE PROGRAMARE

Curs 11

Metapredicate. Probleme de căutare în spațiul stărilor. Probleme de satisfacere a constrângerilor.

Metapredicate

Metapredicat = predicat care primește scopuri ca argumente

- corespondentul funcționalelor din programarea funcțională

Metapredicate pentru colectarea soluțiilor (de satisfacere a unui scop)

- findall
- bagof
- setof

Metapredicate de tip for

- forall

Metapredicatul **findall**

```
findall(+Template, :Goal, -Bag)
```

- Pentru fiecare variantă de satisfacere a scopului Goal, instanțierea corespunzătoare a lui Template este depusă în Bag

Exemplu

scop compus

```
?- findall(X-Y, (numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
```

Metapredicatul **findall**

```
findall(+Template, :Goal, -Bag)
```

- Pentru fiecare variantă de satisfacere a scopului Goal, instanțierea corespunzătoare a lui Template este depusă în Bag

Exemplu

scop compus

```
?- findall(X-Y, (numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
L = [6-3, 8-4, 9-3, 10-5, 12-3, 12-4, 12-6].
```

Metapredicatul **bagof**

```
bagof(+Template, :Goal, -Bag)
```

- La fel ca findall, dar se construiește câte un Bag pentru fiecare instanțiere diferită a variabilelor libere din Goal (libere = care nu apar în Template)

Exemple

```
?- bagof(X-Y, (numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
```

```
?- bagof(X, (numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
```

Metapredicatul **bagof**

`bagof(+Template, :Goal, -Bag)`

- La fel ca findall, dar se construiește câte un Bag pentru fiecare instanțiere diferită a variabilelor libere din Goal (libere = care nu apar în Template)

Exemple

```
?- bagof(X-Y, (numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
NL = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12], L = [6-3, 8-4, 9-3, 10-5, 12-3, 12-4, 12-6].
?- bagof(X, (numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
NL = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12], Y = 3, L = [6, 9, 12] ;
NL = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12], Y = 4, L = [8, 12] ;
NL = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12], Y = 5, L = [10] ;
NL = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12], Y = 6, L = [12].
```

Metapredicatul **bagof**

```
bagof(+Template, :Goal, -Bag)
```

- La fel ca findall, dar se construiește câte un Bag pentru fiecare instanțiere diferită a variabilelor libere din Goal (libere = care nu apar în Template)
- Dacă doresc să nu se țină cont de instanțierile diferite ale unei variabile libere Y folosesc notația **Y^Goal**

Exemplu

```
?- bagof(X, Y^(numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
```

Metapredicatul **bagof**

```
bagof(+Template, :Goal, -Bag)
```

- La fel ca findall, dar se construiește câte un Bag pentru fiecare instanțiere diferită a variabilelor libere din Goal (libere = care nu apar în Template)
- Dacă doresc să nu se țină cont de instanțierile diferite ale unei variabile libere Y folosesc notația **Y^Goal**

Exemplu

```
?- bagof(X, Y^(numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
NL = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12], L = [6, 8, 9, 10, 12, 12, 12].
```

Metapredicatul **setof**

```
setof(+Template, :Goal, -Bag)
```

- La fel ca bagof, dar fiecare Bag este o mulțime (nu conține duplicate)
- Dacă doresc să nu se țină cont de instanțierile diferite ale unei variabile libere Y folosesc notația **Y^Goal** (ca la bagof)

Exemplu

```
?- setof(X, Y^(numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
```

Metapredicatul **setof**

```
setof(+Template, :Goal, -Bag)
```

- La fel ca bagof, dar fiecare Bag este o mulțime (nu conține duplicate)
- Dacă doresc să nu se țină cont de instanțierile diferite ale unei variabile libere Y folosesc notația **Y^Goal** (ca la bagof)

Exemplu

```
?- setof(X, Y^(numlist(3,12,NL),member(X,NL),member(Y,NL),X>Y,0 =:= mod(X,Y)), L).
NL = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12], L = [6, 8, 9, 10, 12].
```

Metapredicatul **forall**

```
forall(:Cond, :Action)
```

- Reușește dacă pentru fiecare satisfacere a lui Cond se poate demonstra Action

Exemple

```
?- forall(member(X,[2,4,6]), X mod 2 =:= 0).

?- forall(member(X,[2,4,3,6]), X mod 2 =:= 0).
```

Metapredicatul **forall**

`forall(:Cond, :Action)`

- Reușește dacă pentru fiecare satisfacere a lui Cond se poate demonstra Action

Exemple

```
?- forall(member(X,[2,4,6]), X mod 2 =:= 0).
true.
?- forall(member(X,[2,4,3,6]), X mod 2 =:= 0).
false.
```

Căutare în spațiul stărilor – Cuprins

- Spațiul stărilor
- Strategii de căutare
- Reprezentarea datelor
- Abstractizare – o bibliotecă pentru căutare

Spațiul stărilor

- Modelat ca **graf orientat**
  - **Noduri** = stări intermediare (soluții parțiale)
    - **Stare inițială** = soluție neexpandată (configurație de pornire)
    - **Stări finale** = soluții complete (configurații care respectă toate cerințele problemei)
  - **Arce** = tranziții legale între stări (de la o soluție parțială la o soluție parțială extinsă cu un pas)

Căutare în spațiul stărilor = explorarea spațiului, pornind de la o stare inițială, cu scopul de a găsi una sau mai multe stări finale

Spațiul stărilor – Exemplu

Fie problema săriturii calului pe o tablă de șah 5x5.

- Soluția presupune să plecăm din poziția 1/1 și să acoperim tabla prin sărituri de cal

Exemple (de elemente în graf)

- **Stare inițială I:** [1/1] (calul încă nu a efectuat nicio săritură)
- **Nod intermediar N1:** [1/1, 2/3] (din linia 1 și coloana 1 s-a sărit pe linia 2 și coloana 3)
- **Nod intermediar N2:** [1/1, 2/3, 1/5]
- **Stare finală F:** [1/1, 2/3, 1/5, 3/4, 1/3, 2/1, 4/2, 5/4, 3/5, 1/4, 2/2, 4/1, 3/3, 2/5, 4/4, 5/2, 3/1, 1/2, 2/4, 4/5, 5/3, 3/2, 5/1, 4/3, 5/5]
- **Arc:** (N1, N2) (tranziție legală, spre deosebire de (N1, F) care nu e arc în graf)

Căutare în spațiul stărilor – Cuprins

- Spațiul stărilor
- Strategii de căutare
- Reprezentarea datelor
- Abstractizare – o bibliotecă pentru căutare

Strategii de căutare

- **Generare și testare**
  - foarte ineficient (presupune generarea tuturor permutărilor)
- **Backtracking**
  - foarte adecvat Prolog-ului (care are backtracking încorporat în motorul de inferență)
- **DFS** / **BFS** / **Iterative deepening** / **A***
  - ușor de modelat folosind
    - liste (pe post de stive și cozi)
    - metapredicate (pentru a colecta vecinii unui anumit nod în spațiul stărilor)

Observație: Ne vom concentra pe mecanismul de backtracking.

Căutare în spațiul stărilor – Cuprins

- Spațiul stărilor
- Strategii de căutare
- Reprezentarea datelor
- Abstractizare – o bibliotecă pentru căutare

Reprezentarea datelor

Avem de reprezentat

- Noduri
  - Care este (fapt) sau cum se generează (regulă) o **stare inițială**
  - Care este (fapt) sau cum se recunoaște (regulă) o **stare finală**
- Arce
  - Ce reprezintă o **tranziție legală** dintr-o stare în alta (regulă)

Reprezentări tipice

- Noduri = căi între configurația inițială și cea curentă
  = liste (ex: lista ordonată a pozițiilor pe care calul a sărit deja)
  - Stare inițială = listă vidă (sau conținând o situație inițială)
- Arce = scopuri arc(+Sursă, -Destinație) unde Destinația e o expandare a Sursei

Reprezentarea stării finale

- **Template** (ex pt săritura calului: [1/1, _])
  - Adecvat cazurilor în care cunoaștem diverse detalii despre forma soluției finale
    - Lungimea acesteia
    - Ce se află pe anumite poziții (de exemplu, la soluția unui Sudoku)
  - Câmpurile deja instanțiate opresc automat expandările care nu se potrivesc cu template-ul
- **Regulă de recunoaștere** (ex pt săritura calului: length(Sol, 25))
  - O stare finală este o stare care îndeplinește un set de condiții
  - Înainte de orice expandare, verificăm dacă am ajuns deja într-o stare finală

Observație

- Cu sau fără template, se vor expanda multe căi inutile dacă programatorul nu filtrează încă din predicatul arc sau imediat după ce efectuează o tranziție folosind predicatul arc

Căutare în spațiul stărilor – Cuprins

- Spațiul stărilor
- Strategii de căutare
- Reprezentarea datelor
- Abstractizare – o bibliotecă pentru căutare

Exemplu – săritura calului

Rezolvări (la calculator)

- Varianta cu template
- Varianta fără template

Discuție

- Ce putem abstractiza astfel încât să obținem o bibliotecă utilă și în rezolvarea altor probleme?

Abstractizarea procesului de căutare

```
%% rezolvare = explorare pornind de la starea inițială
solve(Sol) :- initial_state(State), search(State, Sol).

%% explorarea se oprește când atingem o stare finală
search(State, Sol) :- final_state(State), !, reverse(State, Sol).

%% altfel:
search([S | State], Sol) :-
      arc(S, Next),                    %% alege o configurație următoare
      \+member(Next, State),           %% evitând ciclurile
      search([Next, S | State], Sol).  %% și continuă explorarea
```

Satisfacerea constrângerilor – Cuprins

- Descrierea problemei
- Strategii de rezolvare
- Puzzle-uri logice
- Abstractizare

Problemă de satisfacere a constrângerilor

Date de intrare

- O mulțime de variabile
- Domeniile din care variabilele pot lua valori
- Constrângeri asupra variabilelor

Ieșire

- O mulțime de asocieri variabilă-valoare care să satisfacă toate condițiile de mai sus

Exemplu

Intrare

- $X_1, X_2, X_3, X_4, Y_1, Y_2, Y_3, Y_4 \in [2 .. 9]$
- $X_1*2 = Y_3$, $Y_4*2 = X_2$, $Y_1 = X_1+X_3$, $X_1+X_2+X_3+X_4 > Y_1+Y_2+Y_3+Y_4$
- Toate variabilele au valori diferite între ele

Ieșire

X_i: 2 6 7 8

Y_i: 9 5 4 3

Satisfacerea constrângerilor – Cuprins

- Descrierea problemei
- Strategii de rezolvare
- Puzzle-uri logice
- Abstractizare

Strategii de rezolvare

- **Strategiile de căutare deja menționate**
- **Backtracking + Algoritm de arc-consistență**

Algoritm de arc-consistență (într-o rețea de constrângeri)

- Rețea de constrângeri = graf orientat
- Nod = variabilă
- Arc = constrângere care implică cele 2 variabile
  - (X,Y) arc-consistent dacă pentru orice valoare din domeniul lui X există o valoare în domeniul lui Y astfel încât constrângerea este respectată
- Funcționare: se examinează toate arcele pentru arc-consistență, eliminând valori din domeniile variabilelor atunci când este necesar (ceea ce duce la un număr finit de reexaminări)

Satisfacerea constrângerilor – Cuprins

- Descrierea problemei
- Strategii de rezolvare
- Puzzle-uri logice
- Abstractizare

Puzzle-uri logice

Puzzle logic = problemă de deducție, adecvată rezolvării în paradigma logică

- Strategia de rezolvare = **generare și testare + optimizări** (într-un sens larg, asta sunt toate strategiile menționate anterior)

- Posibilități
  - **Generare ușoară** (soluția se alege dintre puțini candidați, care pot fi enumerați și apoi testați)
    - Se pornește cu setul complet de candidați
    - Pentru fiecare constrângere, se scrie o regulă care produce un set actualizat **pe baza celui anterior**
  - **Generare dificilă** (foarte mulți candidați, durează foarte mult să fie generați toți)
    - Se pornește cu un template care descrie soluția
    - Se instanțiază acest template începând cu alegerile implicate în cât mai multe constrângeri, astfel încât să se reducă mult și rapid spațiul candidaților
    - Nu există o regulă clară despre ordinea care va da cele mai bune rezultate, și de la un punct încolo recurgem tot la generare de permutări (însă intuiția este să o facem cât mai „târziu”)

Generare ușoară – leul și unicornul

Leul și unicornul: Ce zi e azi?

Leul minte în fiecare luni, marți și miercuri și în rest spune adevărul.

Unicornul minte în fiecare joi, vineri și sâmbătă și în rest spune adevărul.

Leul spune: Ieri mințeam.

Unicornul spune: Și eu.

- Tipul problemei
  - **generare ușoară** (avem doar 7 candidați – cele 7 zile ale săptămânii)
- Rezolvare (la calculator)
  - Scrie câte o regulă de filtrare a setului de candidați pentru fiecare constrângere
    - Cea impusă de afirmația leului
    - Cea impusă de afirmația unicornului

Generare ușoară – ziua lui Cheryl

- Enunț și rezolvare – la calculator

Generare dificilă – devoratoarele de Pizza

În jurul unei pizza cu 6 felii stau:

- 6 prietene: mara, moira, maya, marla, myra, marva
- născute în orașele: austin, dallas, san_antonio, galveston, woodlands, houston
- și au ca topping: mushrooms, sausage, pepperoni, pepper, meatballs, broccoli

Va trebui sa aflăm cum sunt așezate ele, unde e născută și ce mănâncă fiecare, ținând cont de o serie de indicii (constrângeri).

- Tipul problemei
  - **generare dificilă** (foarte multe combinații posibile)
- Rezolvare (la calculator, alături de restul enunțului)
  - Pornim de la un template care descrie soluția
  - Generăm și testăm toate variantele, apoi schimbăm ordinea pentru a constata efectul asupra performanței

Satisfacerea constrângerilor – Cuprins

- Descrierea problemei
- Strategii de rezolvare
- Puzzle-uri logice
- Abstractizare

Biblioteca clpfd

- **CLP(FD):** Constraint Logic Programming over Finite Domains
  - :- use_module(library(clpfd)).
- Algoritm de arc-consistență încorporat – pentru constrângeri asupra numerelor întregi
  - Pentru cei interesați: https://github.com/triska/clpfd