Prolog: CSP

Obiective

- înțelegerea **problemelor de satisfacere a restricțiilor** și a mecanismelor de rezolvare a acestora
- înțelegerea conceptului de **arc-consistență** și implementarea unui algoritm de propagare a constrângerilor: AC3
- utilizarea limbajului **Prolog** pentru a implementa un mecanism general de rezolvare a problemelor de satisfacere a restricțiilor

Probleme de satisfacere a restricțiilor

O problemă de satisfacere a restricțiilor este descrisă printr-un set de variabile **X**, o mulțime de domenii finite de valori pentru acestea **D** și un set de constrângeri **C**. Vom nota *D*(*x*) domeniul variabilei $x \in$ **X**. O constrângere *c* va fi reprezentată printr-o relație între una sau mai multe variabile din **X**. O soluție pentru o astfel de problemă este reprezentată printr-o instanțiere a variabilelor din **X** cu valori ce satisfac toate restricțiile din **C**.

Algoritmul GAC3

Arc-consistența reprezintă o metodă pentru propagarea restricțiilor (eliminarea din domeniile variabilelor a acelor valori care nu pot face parte dintr-o soluție a problemei). Arc-consistența este obținută atunci când pentru fiecare valoare din domeniul unei variabile și pentru orice restricție care implică acea variabilă există o instanțiere a tuturor variabilelor implicate care conține acea valoare astfel încât restricția să fie satisfăcută.

AC3 este un algoritm pentru impunerea arc-consistenței asupra domeniilor de valori ale variabilelor unei probleme descrise prin restricții. AC3 a fost ulterior generalizat pentru hiperarce, descrise pentru relații între mai mult de 2 variabile, această variantă fiind numită GAC3. Pseudocodul GAC3 este scris în Algoritmul 1.

Algoritmul 1 GAC3

Intrări: setul de variabile $\mathbf{X}$, domeniile de valori $\mathbf{D}$, hiperarcele de verificat $\mathbf{H}$, setul de constrângeri $\mathbf{C}$

Ieșire: domeniile de valori ce respectă constrângerile $\mathbf{D}$

cât timp $\mathbf{H} \neq \varnothing$ **execută**
    $(x, c) \longleftarrow first(\mathbf{H})$
    $\mathbf{H} \longleftarrow \mathbf{H} \backslash (x, c)$
    $\mathbf{D}_x^* \longleftarrow \texttt{Revise}(x, \mathbf{D}, c)$
    dacă $\mathbf{D}(x) \neq \mathbf{D}_x^*$ **atunci**
        $\mathbf{D}(x) \longleftarrow \mathbf{D}_x^*$
        $\mathbf{H} \longleftarrow H \cup \{(y, c') | c' \in \mathbf{C} \wedge c' \neq c \wedge x \in vars(c')\}$
    termină dacă
termină cât timp

Algoritmul GAC primește variabilele problemei **X**, domeniile acestora **D**, un set de hiperarce ce trebuie verificate și mulțimea tuturor constrângerilor problemei **C**. GAC3 consideră la fiecare pas un hiperarc care corespunde unei restricții *c* și unei variabile *x*. Ceea ce se urmărește la un ciclu este eliminarea tuturor valorilor din domeniul lui *x* pentru care restricția *c* nu poate fi satisfăcută. Verificarea se face cu ajutorul funcției `Revise` (Algoritmul 2) care pentru fiecare valoare *v* din domeniul variabilei *x* caută un set de valori de suport *τ* pentru care este satisfăcută restricția. Dacă un astfel de set suport nu este găsit, valoarea *v* este eliminată din domeniul lui *x*.

Algoritmul 2 Revise

Intrări: variabila x, domeniile de valori $\mathbf{D}$, restricția c

Ieșire: domeniul de valori ale lui x care au suport D_x

$D_x \longleftarrow \mathbf{D}(x)$

pentru toate $v \in D_x$ **execută**

    dacă $\neg\exists\tau, \tau \in \bigtimes_{x_i \in Vars(c)} \mathbf{D}(x_i), \tau$ satisface $c \wedge \tau(x) = v$ **atunci**

        $D_x \longleftarrow D_x \backslash \{v\}$

    termină dacă

termină ciclu

Mulțimea suport *τ* conține o instanțiere a tuturor variabilelor implicate în restricția *c* în care *x* are valoarea *v*. De aceea, pentru hiperarce cu un număr mare de variabile implicate, căutarea acestei mulțimi poate reprezenta o operație foarte costisitoare.

Dacă, după aplicarea funcției `Revise` pentru un hiperarc domeniul variabilei *x* este redus, atunci se verifică domeniile tuturor variabilelor care apar împreună cu *x* într-o constrângere (alta decât *c*). Drept urmare, pentru orice constrângere *c'* care implică variabila *x* se adaugă câte un hiperarc pentru fiecare altă variabil*ăy* ∈ *Vars(c')*, *y≠x*.

Algoritmul se oprește atunci când nu mai există hiperarce de verificat sau când cel puțin unul dintre domeniile de valori este vid (în acest caz nu există soluție).

Algoritmul MAC

Înainte de începerea căutării se aplică un algoritm pentru obținerea arc-consistenței verificându-se toate hiperarcele posibile. Apoi, la fiecare nod al arborelui de căutare, se impune arc-consistența pentru acele restricții corespunzătoare variabilei instanțiate în acel nod. Mai precis, dacă variabila x este cea insanțiată la pasul curent, se va impune arc-consistența pentru toate hiperarcele (y, c), unde c ∈ **C** este o constrângere cu x ∈ Vars(c), iar y ∈ Vars(c)\{x}.

Pentru reducerea spațiului de căutare (a domeniilor variabilelor) la rularea algoritmului backtracking se pot impune restricții locale de consistență mai puternice decât arc-consistența, dar, în general, MAC reprezintă un compromis bun între costul propagării restricțiilor și dimensiunea spațiului efectiv explorat.

Cerințe

Cerința 1 (0.7p): Algoritmul GAC3

Să se scrie un predicat `gac3/5` pentru revizuirea domeniilor de valori ale unor variabile folosind algoritmul**GAC3** descris mai sus.

```
gac3(+Vars, +Domains, +Constraints, +HyperArcs, -RevisedDomains)
```

`Vars` reprezintă mulțimea tuturor variabilelor problemei (o listă de variabile Prolog).
    De exemplu: `[X, Y, Z]`.
`Domains` reprezintă lista domeniilor de valori pentru variabilele `Vars`.
    De exemplu: `[[1,2,3], [3,4], [1,4,7]]`.

`Constraints` reprezintă lista tuturor restricțiilor problemei. O constrângere va fi reprezentată printr-o structură `constraint(CVars, Expression)` unde

- `CVars` reprezintă lista variabilelor implicate de acea constrângere (o listă de variabile Prolog)
- `Expression` reprezintă un scop Prolog ce va putea fi verificat după instanțierea variabilelor din `CVars`

De exemplu: `[constraint([X], X > 0), constraint([X, Y, Z], X =:= Y + Z)]`.

`HyperArcs` reprezintă lista hiperarcelor ce trebuie verificate. Un hiperarc va fi reprezentat astfel:`hypearc(X, Ys, Constraint)`, unde

- `X` reprezintă variabila al cărei domeniu trebuie verificat pentru consistență
- `Ys` reprezintă lista celorlalte variabile implicate în restricție
- `Constraint` reprezintă un scop Prolog ce va putea fi verificat după instanțierea lui `X` și a celorlalte variabile `Ys`

De exemplu: `[hyperarc(X,[], X>0), hyperarc(Z, [X, Y], X =:= Y + Z)]`.

`RevisedDomains` reprezintă lista domeniilor variabilelor problemei după impunerea arc-consistenței.

Observație: listele `Vars`, `Domains` și `RevisedDomains` vor avea același număr de elemente.

Cerința 2 (0.3p): Algoritmul MAC

Să se scrie un predicat `solve_csp/4` pentru rezolvarea problmelor de satisfacere a restricțiilor:

```
solve_csp(+Vars, +Domains, +Constraints, -Solution)
```

Argumentele `Vars`, `Domains` și `Constraints` reprezintă aceleași lucruri precum în cazul predicatului `gac3/5`.

Prin satisfacerea unui scop `solve_csp`, variabila `Solution` va fi legată la o listă de valori pentru variabilele`Vars` care compun o soluție a problemei.

De exemplu:

```
?- solve_csp([X,Y,Z],
             [[1,2,3], [1,2,3], [1,2,3]],
             [constraint([X], X > 2), constraint([X, Y, Z], X < Y + Z)],
             Solution).

Solution = [3, 1, 3];
Solution = [3, 2, 2];
Solution = [3, 3, 1].
```

Cerința 3 (BONUS 0.2p): Reprezentarea unei probleme cu ajutorul restricțiilor

Asemeni exemplelor oferite, se cere reprezentarea Problemei lui Einstein folosind restricții (după modelul celor date ca exemplu în fișierul de test). Trebuie identificate variabilele, domeniile acestora, precum și constrângerile problemei.

Scrieți un predicat `einstein(-Vars, -FishNationality, -Domains, -Constraints)` prin sastisfacerea căruia `Vars` devine lista variabilelor, `FishNationality` va fi variabila ce va conține răspunsul ghicitorii,`Domains` va fi lista cu domeniile de valori, iar `Constraints` va fi lista tuturor constrângerilor problemei.

```
?- einstein(Vars, FishNationality, Domains, Constraints).
```