Paradigme de Programare
Tema 4: **CSP**

Responsabil: Tudor Berariu
Facultatea de Automatică și Calculatoare

4 mai 2012

1 Introducere

Scopul acestei teme îl reprezintă familiarizarea cu probleme de satisfacere a restricțiilor (CSP - *constraint satisfaction problem*), arc-consistență, algoritmi de rezolvare a acestor probleme și cu limbajul de programare Prolog.

2 Algoritmi pentru rezolvarea CSP

2.1 Problema satisfacerii restricțiilor

O problemă de satisfacere a restricțiilor este descrisă printr-un set de variabile $\mathbf{X}$, o mulțime de domenii finite de valori pentru acestea $\mathbf{D}$ (vom nota domeniul variabilei $x \in \mathbf{X}$ astfel: $\mathbf{D}(x)$) și un set de constrângeri $\mathbf{C}$. O constrângere $c \in \mathbf{C}$ descrie o relație între o mulțime de variabile $Vars(c)$ (desigur, $Vars(c) \subseteq \mathbf{X}$).

O soluție pentru o astfel de problemă este reprezentată de o instațiere a variabilelor din $\mathbf{X}$ cu valori ce satisfac toate restricțiile $\mathbf{C}$.

2.2 GAC3

Arc-consistența reprezintă o metodă pentru propagarea restricțiilor (eliminarea din domeniile variabilelor a unor 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, restricții descrise prin relații între mai mult de 2 variabile, această variantă fiind numită GAC3. Pseudocodul GAC3 este dat 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 $\mathbf{D}$

1: **cât timp** $\mathbf{H} \neq \varnothing$ **execută**
2: select (x, c) from $\mathbf{H}$
3: $\mathbf{H} \longleftarrow \mathbf{H} \backslash \{(x, c)\}$
4: $\mathbf{D}_x^* \longleftarrow$ `Revise`$(x, \mathbf{D}, c)$
5: **dacă** $\mathbf{D}(x) \neq \mathbf{D}_x^*$ **atunci**
6: $\mathbf{D}(x) \longleftarrow \mathbf{D}_x^*$
7: $\mathbf{H} \longleftarrow \mathbf{H} \cup \{(y, c') | c' \in \mathbf{C} \wedge c' \neq c \wedge x \in Vars(c') \wedge y \in Vars(c') \wedge y \neq x\}$
8: **termină dacă**
9: **termină cât timp**

Algoritmul GAC3 [2] primește variabilele problemei $\mathbf{X}$, domeniile acestora $\mathbf{D}$ (prin convenție domeniul variabilei x este $\mathbf{D}(x)$), un set de hiperarce ce trebuie verificate $\mathbf{H}$ (un hiperarc este o pereche (x, c): x este o variabilă al cărei domeniu trebuie verificat în raport cu constrângerea c) și mulțimea tuturor constrângerilor problemei $\mathbf{C}$. GAC3 consideră la fiecare pas un hiperarc (care corespunde unei restricții) și o variabilă x. Ceea ce se urmărește la un ciclu este eliminarea tuturor valorilor din domeniul lui x pentru care restricția corespunzătoare hiperarcului nu poate fi satisfăcută. Verificarea se realizează cu ajutorul funcției `Revise` (Algoritmul 2) care pentru fiecare valoare v din domeniul variabilei x caută un set de valori suport τ pentru care să fie satisfăcută restricția. Dacă un astfel de set suport nu este găsit, valoarea v este eliminată din domeniul lui x.

Mulțimea suport τ conține o instanțiere a tuturor variabilelor implicate în restricția c ($Vars(c)$) 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.

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

1: $D_x \longleftarrow \mathbf{D}(x)$
2: **pentru toate** $v \in D_x$ **execută**
3: **dacă** $\neg\exists\tau, \tau \in \underset{x_i \in Vars(c)}{\times} \mathbf{D}(x_i), \tau$ satisface $c \wedge \tau(x) = v$ **atunci**
4: $D_x \longleftarrow D_x \backslash \{v\}$
5: **termină dacă**
6: **termină ciclu**

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 (diferită de c). Drept urmare, pentru orice constrângere $c^{'}$ care implică variabila x se adaugă câte un hiperarc pentru fiecare altă variabilă $y \in Vars(c^{'}), y \neq x$.

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).

2.3 MAC

Una dintre tehnicile clasice de rezolvare a problemelor de satisfacere a restricțiilor o reprezintă algoritmul backtracking, algoritm complet (găsește soluțiile problemei, dacă acestea există). De cele mai multe ori, se dorește o optimizare a algoritmului backtracking prin reducerea spațiului de căutare chiar în timpul rulării algoritmului. O metodă de optimizare o reprezintă impunerea și menținerea arc-consistenței la fiecare pas al algoritmului (atât timp cât există variabile neinstanțiate). Metoda a fost propusă de Gaschnig [1], iar algoritmul este cunoscut după acronimul MAC (Maintaining Arc Consistency).

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 instanțiată la pasul curent, se va impune arc-consistența pentru

toate hiperarcele (y, c) unde $c \in \mathbf{C}$ este o constrângere cu $x \in Vars(c)$, iar $y \in Vars(c) \backslash \{x\}$.

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 spațiul efectiv explorat.

3 Cerințe

În această temă se cere implementarea unui program în Prolog care să rezolve probleme de satisfacere a restricțiilor folosind algoritmul MAC. Tema este împărțită în trei părți:

- implementarea algoritmului GAC3 pentru obținerea arc-consistenței;
- implementarea algoritmului MAC pentru găsirea tuturor soluțiilor unei probleme de satisfacere a restricțiilor;
- codificarea unei probleme în CSP și în programul implementat.

3.1 Cerința 1 (0.7p): Algoritmul GAC3

Să se scrie un predicat în Prolog `gac3/5` pentru impunerea arc-consistenței asupra domeniilor de valori ale variabilelor folosind algoritmul GAC3:

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

unde:

Vars reprezintă mulțimea tuturor variabilelor problemei și este o listă de variabile Prolog;

Domains reprezintă lista domeniilor de valori ale variabilelor din `Vars`;

RevDomains reprezintă lista domeniilor de valori după impunerea arc-consistenței;

- Toate cele trei liste: **Vars**, **Domains**, **RevDomains** vor avea același lungime, iar corelarea se va face prin poziția în cadrul listei.

HyperArcs reprezintă lista hiperarcelor ale căror restricții trebuie verificate (mulțimea **H** din Algoritmul 1);

- Un hiperarc va fi reprezentat astfel:

  `hyperarc(X,Ys,Constraint)`

  unde `X` este variabila al cărei domeniu este verificat pentru consistență, `Ys` reprezintă lista celorlalte variabile implicate de constrângerea respectivă, iar `Constraint` reprezintă o expresie validă în Prolog ce conține variabilele din `[X|Ys]` a cărei satisfacere poate fi verificată după instanțierea acestora.

Constraints reprezintă lista tuturor restricțiilor problemei (necesară pentru obținerea hiperarcelor)

- o constrângere va fi reprezentată astfel:

  `constraint(Vars,Constraint)`

  unde `Vars` este o listă de variabile (implicate de constrângerea respectivă), iar `Constraint` reprezintă o expresie validă în Prolog ce conține variabilele din `Vars` a cărei satisfacere poate fi verificată după instanțierea acestora.

Atenție, lista inițială de hiperarce (**H** în Algoritmul 1) este reprezentată de variabila `HyperArcs`, nu trebuie construită!

3.2 Cerința 2 (0.3p): Algoritmul MAC

Să se scrie un predicat în Prolog pentru rezolvarea de probleme descrise prin restricții. Predicatul `solveCSP/4` va fi construit astfel:

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

unde:

Vars reprezintă variabilele problemei și este o listă de variabile Prolog;

Domains reprezintă domeniile de valori și este o listă de liste de constante;

Constraints reprezintă constrângerile problemei și se reprezintă astfel:

`constraint(CVars,Relation)`

unde `CVars` reprezintă o listă de variabile Prolog (neinstanțiate), iar `Relation` este o expresie Prolog ce se poate evalua după instanțierea tuturor variabilelor din `CVars`.

Programul trebuie să construiască și să instanțieze pe rând variabila `Solution` cu toate soluțiile problemei. O soluție va fi reprezentată de o listă de valori pentru variabilele problemei (în exact aceeași ordine).

3.3 Cerința 3 (Bonus): Codificarea unei probleme utilizând constrângeri

Asemeni exemplelor oferite, se cere reprezentarea *Problemei lui Einstein* folosind restricții și rezolvarea acesteia cu programul scris anterior. Trebuie identificate variabilele, domeniile acestora, precum și constrângerile problemei.

Problema spune că în cinci case așezate de-a lungul unui drum locuiesc cinci bărbați de naționalități diferite care fumează cinci mărci diferite de țigări, au cinci băuturi preferate diferite și cinci animale de companie diferite. În plus, se știu următoarele:

- Englezul locuiește în casa roșie.
- Suedezul are câini.
- Danezul bea ceai.
- Casa verde este în stânga celei albe.
- Stăpânul casei verzi bea cafea.
- Fumătorul de Pall Mall crește păsări.
- Stăpânul casei galbene fumează Dunhills.
- Omul din casa din mijloc bea lapte.
- Norvegianul locuiește în prima casă.
- Fumătorul de Blend are un nebun ce ține pisici.

- Fumătorul de Blue Masters bea bere.
- Bărbatul care are cai locuiește lângă fumătorul de Dunhill.
- Germanul fumează Prince.
- Norvegianul locuiește lângă casa albastră.
- Fumătorul de Blend are un vecin a cărui băutură favorită este apa.

Scrieți o regulă care să afle naționalitatea celui care crește pești.

```
einstein(Nationality).
```

3.4 Detalii despre trimiterea temei

Toate predicatele corespunzătoare cerințelor de mai sus se vor scrie într-un singur fișier Prolog denumit `tema4.pl`. În acest fișier veți adăuga și următoarele trei fapte (adaptate la datele personale):

```
nume('Messi').
prenume('Lionel').
grupa('FC Barcelona').
```

Comentați codul și dați nume sugestive predicatelor auxiliare pe care le mai scrieți.

Se încurajează adăugarea unui fișier README, mai ales în cazul temelor incomplete.

Toate fișierele se vor pune într-o arhivă cu numele
`NumePrenume_SeriaGrupa_Tema3PP.zip`

4 Exemplu

În această secțiune se va detalia un exemplu de impunere a arc-consistenței folosind algoritmul AC3.

Problema este cea reprezentată în Figura 1. Vom considera că domeniile de valori sunt $\{r, g, b\}$ pentru toate cele cinci variabile $\{A, B, C, D, E\}$.

Pentru a limita numărul soluțiilor, întrucat nu contează care este valoarea culoarii unei țări, ci doar ca aceasta să fie diferită de cea a țărilor vecine, vom impune printr-o restricție ca țara A să primească valoarea a, iar țara B culoarea g. Restricțiile problemei, vor fi, deci:

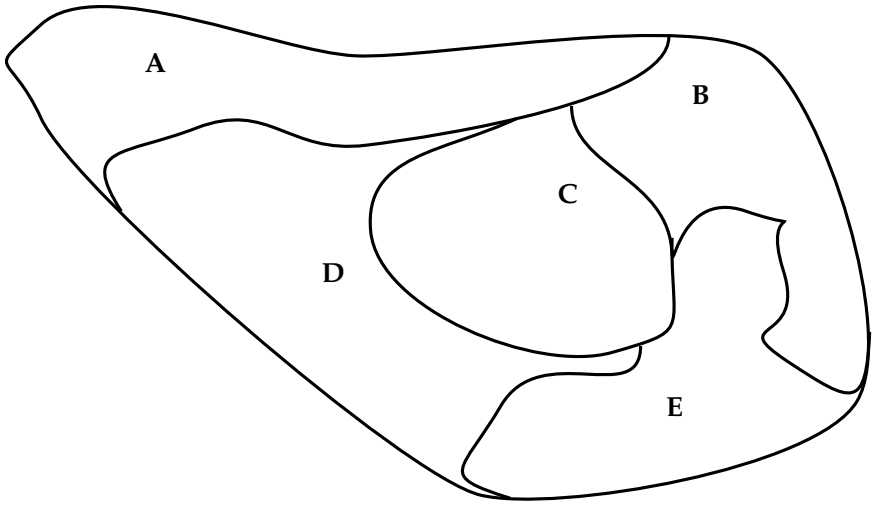

Figura 1: Harta pentru exemplul din secțiunea 4

- $constraint([A], A = r)$
- $constraint([B], B = g)$
- $constraint([A, B], A \neq B)$
- $constraint([A, C], A \neq C)$
- $constraint([A, D], A \neq D)$
- $constraint([B, C], B \neq C)$
- $constraint([B, E], B \neq E)$
- $constraint([C, D], C \neq D)$
- $constraint([C, E], C \neq E)$
- $constraint([D, E], D \neq E)$

Toate hiperarcele ce se pot construi din acest set de restricții sunt:

- $hyperarc(A, [], A = r)$

- $hyperarc(B, [], B = g)$
- $hyperarc(A, [B], A \neq B)$, $hyperarc(B, [A], A \neq B)$
- $hyperarc(A, [C], A \neq C)$, $hyperarc(C, [A], A \neq C)$
- $hyperarc(A, [D], A \neq D)$, $hyperarc(D, [A], A \neq D)$
- $hyperarc(B, [C], B \neq C)$, $hyperarc(C, [B], B \neq C)$
- $hyperarc(B, [E], B \neq E)$, $hyperarc(E, [B], B \neq E)$
- $hyperarc(C, [D], C \neq D)$, $hyperarc(D, [C], C \neq D)$
- $hyperarc(C, [E], C \neq E)$, $hyperarc(E, [C], C \neq E)$
- $hyperarc(D, [E], D \neq E)$, $hyperarc(E, [D], D \neq E)$

Acestea vor forma mulțimea **H**.

Algoritmul AC3 nu specifică ordinea în care hiperarcele ce trebuie verificate sunt prelucrate. Se observă că doar primele hiperarce ar putea produce modificări domeniilor de valori.

Putem presupune, pentru claritatea demonstrației, că hiperarcele cu 2 variabile au fost prelucrate fără a produce schimbări domeniilor de valori.

Ciclul 1

$$\mathbf{D} = \Big\{D_A = \{r, g, b\}, D_B = \{r, g, b\}, D_C = \{r, g, b\}, D_D = \{r, g, b\}, D_E = \{r, g, b\}\Big\}$$

$$\mathbf{H} = \Big\{hyperarc(A, [], A = r), hyperarc(B, [], B = g)\Big\}$$

Mulțimea hiperarcelor conține doar 2 hiperarce. Se verifică primul și efectul este reducerea domeniului de valori al lui a la $\{r\}$. Se vor adăuga în **H** următoarele hiperarce care leagă A de alte variabile:

- $hyperarc(B, [A], A \neq B)$,
- $hyperarc(C, [A], A \neq C)$,
- $hyperarc(D, [A], A \neq D)$.

Ciclul 2

$$\mathbf{D} = \Big\{D_A = \{r\}, D_B = \{r,g,b\}, D_C = \{r,g,b\}, D_D = \{r,g,b\}, D_E = \{r,g,b\}\Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{c} hyperarc(B, [], B = g), hyperarc(B, [A], A \neq B), \\ hyperarc(C, [A], A \neq C), hyperarc(D, [A], A \neq D) \end{array} \Big\}$$

Se prelucrează hiperarcul $hyperarc(B, [], B = g)$, iar domeniul de valori al lui B este redus la $\{g\}$. Drept urmare, se vor adăuga în $\mathbf{H}$:

- $hyperarc(A, [B], A \neq B)$,
- $hyperarc(C, [B], B \neq C)$,
- $hyperarc(E, [B], B \neq E)$.

Ciclul 3

$$\mathbf{D} = \Big\{D_A = \{r\}, D_B = \{g\}, D_C = \{r,g,b\}, D_D = \{r,g,b\}, D_E = \{r,g,b\}\Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{c} hyperarc(B, [A], A \neq B), hyperarc(C, [A], A \neq C), \\ hyperarc(D, [A], A \neq D), hyperarc(A, [B], A \neq B), \\ hyperarc(C, [B], B \neq C), hyperarc(E, [B], B \neq E) \end{array} \Big\}$$

Se alege $hyperarc(B, [A], A \neq B)$. Cum domeniul lui B conține doar valoarea g, iar domeniul lui A conține r, $\langle r, g \rangle$ reprezintă un suport pentru valoarea g. Domeniul lui B este consistent, deci nu se adaugă nimic în mulțimea $\mathbf{H}$.

Ciclul 4

$$\mathbf{D} = \Big\{D_A = \{r\}, D_B = \{g\}, D_C = \{r,g,b\}, D_D = \{r,g,b\}, D_E = \{r,g,b\}\Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{c} hyperarc(C, [A], A \neq C), hyperarc(D, [A], A \neq D), \\ hyperarc(A, [B], A \neq B), hyperarc(C, [B], B \neq C), \\ hyperarc(E, [B], B \neq E) \end{array} \Big\}$$

Se alege pentru verificare $hyperarc(C, [A], A \neq C)$. Domeniul lui C înaintea acestui ciclu era $\{r, g, b\}$. Singura valoarea care nu are suport este r, care va fi eliminată. Domeniul lui C devine, deci, $\{g, b\}$, iar în $\mathbf{H}$ se adaugă toate hiperarcele către c (mai puțin cel aparținând relației $A \neq C$) :

- $hyperarc(B, [C], B \neq C)$
- $hyperarc(D, [C], C \neq D)$
- $hyperarc(E, [C], C \neq E)$

Ciclul 5

$$\mathbf{D} = \Big\{D_A = \{r\}, D_B = \{g\}, D_C = \{g, b\}, D_D = \{r, g, b\}, D_E = \{r, g, b\}\Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{c} hyperarc(D, [A], A \neq D), hyperarc(A, [B], A \neq B) \\ hyperarc(C, [B], B \neq C), hyperarc(E, [B], B \neq E) \\ hyperarc(B, [C], B \neq C), hyperarc(D, [C], C \neq D) \\ hyperarc(E, [C], C \neq E) \end{array} \Big\}$$

Se prelucrează $hyperarc(D, [A], A \neq D)$ și se elimină din D_D valoarea r. Se vor adăuga toate hiperarcele către D, mai puțin inversul celui abia prelucrat.

Ciclul 6

$$\mathbf{D} = \Big\{D_A = \{r\}, D_B = \{g\}, D_C = \{g, b\}, D_D = \{g, b\}, D_E = \{r, g, b\}\Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{c} hyperarc(A, [B], A \neq B), hyperarc(C, [B], B \neq C), \\ hyperarc(E, [B], B \neq E), hyperarc(B, [C], B \neq C), \\ hyperarc(D, [C], C \neq D), hyperarc(E, [C], C \neq E), \\ hyperarc(C, [D], C \neq D), hyperarc(E, [D], D \neq E) \end{array} \Big\}$$

Verificarea hiperarcului $hyperarc(A, [B], A \neq B)$ nu va aduce modificări domeniului lui A

Ciclul 7

$$\mathbf{D} = \Big\{D_A = \{r\}, D_B = \{g\}, D_C = \{g, b\}, D_D = \{g, b\}, D_E = \{r, g, b\}\Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{c} hyperarc(C, [B], B \neq C), hyperarc(E, [B], B \neq E), \\ hyperarc(B, [C], B \neq C), hyperarc(D, [C], C \neq D), \\ hyperarc(E, [C], C \neq E), hyperarc(C, [D], C \neq D), \\ hyperarc(E, [D], D \neq E) \end{array} \Big\}$$

În ciclul al șaptelea se va impune arc-consistența pentru $hyperarc(C, [B], B \neq C)$, fiind eliminate acele valori din D_C care nu au suport în D_B. Cum domeniul lui B are doar valoarea g, aceasta va fi eliminată din D_c. Se va adăuga un singur hiperarc în **H**:

- $hyperarc(A, [C], A \neq C)$.

Celelalte se găseau deja în **H**.

Pentru optimizare, se poate șterge din **H** hiperarcul invers: $hyperarc(B, [C], B \neq C)$.

Ciclul 8

$$\mathbf{D} = \Big\{D_A = \{r\}, D_B = \{g\}, D_C = \{b\}, D_D = \{g, b\}, D_E = \{r, g, b\}\Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{c} hyperarc(E, [B], B \neq E), hyperarc(D, [C], C \neq D), \\ hyperarc(E, [C], C \neq E), hyperarc(C, [D], C \neq D), \\ hyperarc(E, [D], D \neq E), hyperarc(A, [C], A \neq C) \end{array} \Big\}$$

Impunerea arc-consistenței pentru hiperarcul $hyperarc(E, [B], B \neq E)$ va duce la eliminarea valorii g din domeniul lui E.

Ciclul 9

$$\mathbf{D} = \Big\{D_A = \{r\}, D_B = \{g\}, D_C = \{b\}, D_D = \{g, b\}, D_E = \{r, b\}\Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{c} hyperarc(D, [C], C \neq D), hyperarc(E, [C], C \neq E), \\ hyperarc(C, [D], C \neq D), hyperarc(E, [D], D \neq E), \\ hyperarc(A, [C], A \neq C), hyperarc(C, [E], C \neq E), \\ hyperarc(D, [E], D \neq E) \end{array} \Big\}$$

Impunerea arc-consistenței pentru hiperarcul $hyperarc(D, [C], C \neq D)$ va duce la eliminarea valorii b din domeniul lui D.

Ciclul 10

$$\mathbf{D} = \Big\{ D_A = \{r\}, D_B = \{g\}, D_C = \{b\}, D_D = \{g\}, D_E = \{r, b\} \Big\}$$

$$\mathbf{H} = \Big\{ \begin{array}{l} hyperarc(E, [C], C \neq E), hyperarc(E, [D], D \neq E), \\ hyperarc(A, [C], A \neq C), hyperarc(C, [E], C \neq E), \\ hyperarc(D, [E], D \neq E), hyperarc(A, [D], A \neq D) \end{array} \Big\}$$

Impunerea arc-consistenței pentru hiperarcul $hyperarc(E, [C], C \neq E)$ va duce la eliminarea valorii b din domeniul lui E.

Ciclurile 11-n

$$\mathbf{D} = \Big\{ D_A = \{r\}, D_B = \{g\}, D_C = \{b\}, D_D = \{g\}, D_E = \{r\} \Big\}$$

De aici, algoritmul va verifica pe rând fiecare hiperarc din **H**, fără a mai elimina valori din **D**.

5 Testare

Alături de acest document se găsește fișierul Prolog `tests.pl`, unde sunt reprezentate mai multe probleme folosind restricții și sunt construite o serie de teste ce pot fi folosite pentru verificarea implementărilor.

Cele 4 probleme sunt:

Colorarea hărților Exemplele 0 și 1. Scopul este acela de colora o hartă fără a folosi aceeași culoare pentru două țări vecine. Restricțiile se pun asupra perechilor de variabile corespunzătoare perechilor de state vecine.

În testul 0b, din codul de mai jos s-a codificat exemplul din secțiunea anterioară. Regula `test0b` confruntă soluțiile construite de `solveCSP` cu cele 2 soluții corecte.

Listing 1: Testul 0b pentru colorarea hărților

```
solve0(S):- Vars=[A,B,C,D,E],
Domains=[[r,g,b],[r,g,b],[r,g,b],[r,g,b],[r,g,b]],
Constraints=[constraint([A],A==r),
constraint([A,B],A\==B),constraint([A,C],A\==C),
constraint([A,D],A\==D),constraint([B,C],B\==C),
constraint([B,E],B\==E),constraint([C,D],C\==D),
constraint([C,E],C\==E),constraint([D,E],D\==E)],
solveCSP(Vars,Domains,Constraints,S).

test0b:-bagof(S,solve0(S),L), ok(L,[[r,g,b,g,r],[r,b,g,b,r]]),!.
```

Pătratul fermecat Exemplul 2. Se dă un pătrat de dimensiune 3x3. Să se completeze cu cifrele de la 1 la 9 astfel încât sumele liniilor, coloanelor și diagonalelor să fie 15.
Problema are 9 variabile: $\{X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}, X_{31}, X_{32}, X_{33}\}$, toate având același domeniu: $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Constrângerile problemei includ relațiile aritmetice dintre celulele pătratului:

- $X_{11} + X_{12} + X_{13} = 15$
- $X_{21} + X_{22} + X_{23} = 15$
- $X_{31} + X_{32} + X_{33} = 15$
- $X_{11} + X_{21} + X_{31} = 15$
- $X_{12} + X_{22} + X_{32} = 15$
- $X_{13} + X_{23} + X_{33} = 15$
- $X_{11} + X_{22} + X_{33} = 15$
- $X_{13} + X_{22} + X_{31} = 15$

, dar și faptul că cele 9 celule trebuie să aibă valori diferite. Această ultimă constrângere poate fi reprezentată printr-o singură relație:

- $allDifferent(X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}, X_{31}, X_{32}, X_{33})$

sau prin constrângeri separate pentru toate perechile de variabile:

- $X_{11} \neq X_{12}$

- ...
- $X_{23} \neq X_{33}$

În fișierul `tests.pl` se găsește reprezentarea în Prolog a acestei probleme.

N-Queens Exemplul 3 și 4. Să se așeze pe o tablă de șah de dimensiune $N \times N$ N regine astfel încât acestea să nu se atace între ele. Exemplul 3 codifică problema 6-Queens, în timp ce Exemplul 4 construiește problema N-Queens pentru un N dat.

`SEND+MORE=MONEY` Exemplul 5. Se cere rezolvarea acestui puzzle criptoaritmetic prin găsirea cifrelor care corespund literelor și fac relația aritmetică adevărată.

Există șase teste pentru arc-consistență (predicatul `gac3`) - testele cu sufixul `a` - și șase teste pentru găsirea tuturor soluțiilor problemelor descrise mai sus - testele cu sufixul `b`.

Pentru rularea unui test individual folosiți regulile: `test0a`, `test0b`, ..., `test5b`.

Pentru a verifica toate testele folosiți `testall`.

Bibliografie

[1] J. Gaschnig. A constraint satisfaction method for inference making. In *Proceedings of the Twelfth Annual Allerton Conference on Circuit Systems Theory*, pages 866–874, 1974.

[2] A.K. Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118, 1977.