UNIVERSITATEA TEHNICĂ A MOLDOVEI
Facultatea Calculatoare, Informatică şi Microelectronică
Catedra Informatică Aplicată

PROGRAMAREA LOGICĂ ŞI INTELIGENȚA ARTIFICIALĂ

Îndrumar de laborator

Chişinău
Editura „Tehnica-UTM”
2014

Îndrumarul de laborator include noțiuni introductive şi sarcini practice corespunzătoare menite să inițieze studentul, atât în domeniul programării logice, cât şi în domeniul inteligenței artificiale. Tipul sarcinilor este unul generic, care urmăreşte reiterarea materialului studiat prin implementarea metodelor descrise şi demonstrate în lucrare. O caracteristică definitorie a lucrărilor este studiul şi analiza detaliată a îndeplinirii sarcinilor utilizând limbajul de programare Prolog.

În lucrare sunt elucidate teme care au ca subiect sintaxa unui limbaj de programare logic şi structurile lui de date fundamentale, baze de cunoştințe, sisteme expert, rețele neuronale şi aplicarea lor practică în recunoaşterea imaginilor şi a vorbirii.

Îndrumarul este destinat studenților specialităților *Calculatoare 526.1* şi *Tehnologii Informaționale 526.2* pentru însuşirea disciplinei *Programarea Logică şi Inteligența Artificială*.

Autori: conf. univ., dr. L. Luchianova
lect.sup. V. Lazu

Redactor responsabil: lect. sup. V. Lazu

Recenzent: conf.univ. L. Carcea

Cuprins

CAPITOLUL I
LUCRAREA DE LABORATOR nr. 1 5
Introducere în limbajul Prolog

LUCRAREA DE LABORATOR nr. 2 15
Mecanisme de control al procesului de backtracking

LUCRAREA DE LABORATOR nr. 3 23
Structuri de date în Prolog

CAPITOLUL II
Exemple şi probleme în limbajul Prolog 30

LUCRAREA DE LABORATOR nr. 4 30
Sisteme expert

LUCRAREA DE LABORATOR nr. 5 39
Prelucrarea limbajului natural

LUCRAREA DE LABORATOR nr. 6 57
Algoritmi de recunoaştere

LUCRAREA DE LABORATOR nr. 7 63
Algoritmi de recunoaştere a imaginilor

LUCRAREA DE LABORATOR nr. 8 68
Rețelele neuronale Hamming

LUCRAREA DE LABORATOR nr. 9 79
Rețelele neuronale Hebb

BIBLIOGRAFIE 92

ANEXE 93
Anexa 1 93
Mediul GNU Prolog

Anexa 2 100
Codul sursă al programului care realizează o rețea neuronală Hamming în limbajul de programare C++

Anexa 3 104
Codul sursă al programului care realizează rețeaua neuronală Hamming, în limbajul de programare C#

CAPITOLUL I

LUCRAREA DE LABORATOR nr. 1

Introducere în limbajul Prolog

Scopul: Însuşirea principiilor fundamentale de programare a limbajului Prolog

PARTEA 1. Noțiuni teoretice

1.1. Entitățile limbajului Prolog

Prolog este un limbaj logic, descriptiv care permite specificarea spațiului problemei şi a soluției acesteia, operând în termeni de fapte cunoscute despre obiectele universului problemei şi ale relațiilor existente între aceste obiecte. Execuția unui program Prolog constă în deducerea implicațiilor dintre aceste fapte şi relații, programul definind astfel o mulțime de consecințe ce reprezintă înțelesul sau semnificația declarativă a programului.

Un program Prolog conține următoarele entități:

- *fapte* despre obiecte şi *relațiile* existente între aceste obiecte;
- *reguli* despre obiecte şi relațiile dintre acestea care permit deducerea (inferarea) unor fapte noi în baza celor cunoscute;
- întrebări, numite şi *scopuri*, despre obiecte şi relațiile acestora, la care programul răspunde în baza faptelor şi regulilor existente.

1.1.1. Fapte

Faptele sunt predicate de ordinul întâi de aritate *n* considerate adevărate (reprezintă cea mai simplă forma de predicat din Prolog). Ele stabilesc relațiile dintre obiectele universului problemei. Numărul de

argumente ale faptelor este reprezentat prin aritatea corespunzătoare a predicatelor.
Exemple:

Fapt:	***Aritate:***
câine(grivei).	1
place(ion, ioana).	2
place(ion, ana).	2
frumoasă(ana).	1
bun(daniel).	1
deplasează(cub, camera1, camera2).	3

Interpretarea particulară a predicatului şi a argumentelor acestuia depinde de programator. Ordinea argumentelor este importantă şi odată fixată trebuie păstrată pentru orice utilizare ulterioară a faptului cu aceeaşi semnificație. Mulțimea faptelor şi regulilor unui program Prolog formează *baza de cunoştințe* Prolog.

1.1.2. Scopuri

Obținerea consecințelor sau a rezultatului unui program Prolog se face prin stabilirea unor *scopuri* care în funcție de conținutul bazei de cunoştințe pot fi adevărate sau false. Scopurile sunt *predicatele* pentru care se doreşte aflarea valorii de adevăr în contextul faptelor existente ale unei baze de cunoştințe. Rezultatul unui program Prolog este răspunsul la o întrebare (sau la o îmbinare de întrebări). Acest răspuns poate fi afirmativ - **yes**, sau negativ - **no**. În cazul unui răspuns afirmativ la o întrebare, un program Prolog poate furniza în continuare şi alte informații din baza de cunoştințe.
Exemplu:
Considerând baza de cunoştințe specificată anterior, se pot stabili diverse întrebări, cum ar fi:

```
?- place(ion, ioana).
Yes     % deoarece acest fapt există în baza de cunoştințe
?- papagal(ion).
No      % deoarece acest fapt nu există în baza de cunoştințe
```

În exemplele prezentate de până acum, argumentele faptelor şi întrebărilor au fost obiecte particulare, numite şi constante sau atomi simbolici. Predicatele Prolog (ca şi orice predicate în logica cu predicate de ordinul I), permit argumente în formă de obiecte generice, numite variabile. În mod convenţional, în Prolog numele argumentelor variabile încep cu majusculă iar numele constantelor simbolice încep cu minusculă. O variabilă poate fi instanţiată (legată) dacă există un obiect asociat acestei variabile, sau neinstanţiată (liberă) dacă nu se ştie încă ce obiect va desemna variabila.

Iniţial, la fixarea unui scop Prolog care conţine variabile, acestea sunt neinstanţiate, iar sistemul încearcă satisfacerea acestui scop căutând în baza de cunoştinţe un fapt care se poate identifica cu scopul printr-o instanţiere adecvată a variabilelor din scopul dat. Eventual, se derulează un proces de unificare a predicatului - scop cu unul din predicatele - fapte existente în baza de cunoştinţe. La încercarea satisfacerii scopului, căutarea porneşte întotdeauna de la începutul bazei de cunoştinţe. Dacă se stabileşte un fapt cu un simbol predicativ identic cu cel al scopului, variabilele din scop se instanţiază conform algoritmului de unificare, iar valorile variabilelor astfel obţinute sunt afişate ca răspuns la satisfacerea acestui scop.

Exemple:

```
?- ciine(CineEste).
CineEste = bobi
?- deplaseaza(Ce, DeUnde, Unde).
Ce = cub, DeUnde = camera1, Unde = camera2
?- deplaseaza(Ce, Aici, Aici).
No
```

În cazul în care există mai multe fapte în baza de cunoştinţe care se unifică cu întrebarea pusă, deci există mai multe răspunsuri la întrebare, ce corespund mai multor soluţii ale scopului fixat, limbajul Prolog procedează în felul următor: prima soluţie rezultă din prima unificare şi există atâtea soluţii câte combinaţii de unificări există. La realizarea primei unificări este marcat faptul care a unificat şi care reprezintă prima soluţie. La obţinerea următoarei soluţii, căutarea este reluată de la marcaj, descendent în baza de cunoştinţe. Obţinerea primei soluţii este, de regulă numită satisfacerea scopului iar obţinerea celorlalte soluţii, resatisfacerea scopului. La satisfacerea unui scop căutarea se face întotdeauna de la

începutul bazei de cunoştinţe. La resatisfacerea unui scop, căutarea se face începând de la marcajul stabilit de satisfacerea anterioară a aceluiaşi scop.

Sistemul Prolog, fiind un sistem interactiv, permite utilizatorului obținerea fie a primului răspuns, fie a tuturor răspunsurilor. În cazul în care, după afişarea tuturor răspunsurilor, un scop nu mai poate fi re-satisfăcut, sistemul răspunde **no**.

Exemple:

```
?- place(ion, X).
X = ioana;
X = ana;
no
```

1.1.3. Reguli

O *regulă* Prolog exprimă un fapt care depinde de alte fapte şi are forma:

S :- S1, S2, …Sn.

Fiecare **Si, i =1,n** şi **S** au forma faptelor Prolog, deci sunt predicate cu argumente constante, variabile sau structuri. Faptul **S** care defineşte regula se numeşte *antet de regulă*, iar **S1, S2, …Sn** formează *corpul regulii* şi reprezintă conjuncția (unificarea) scopurilor care trebuie satisfăcute pentru satisfacerea ulterioară a antetului regulii.

Fie următoarea bază de cunoştinţe Prolog:

• Ana este frumoasă. **frumoasa(ana).** % Ipoteza 1
• Ion este bun. **bun(ion).** % Ipoteza 2
• Ion o cunoaşte pe Maria**. cunoaste(ion, maria).** % Ipoteza 3
• Ion o cunoaşte pe Ana**. cunoaste(ion, ana).** % Ipoteza 4
• Mihai o place pe Maria. **place(mihai, maria).** % Ipoteza 5
• Dacă două persoane X şi Y se cunosc şi persoana X este bună şi persoana Y este frumoasă, atunci cele două persoane, X şi Y, se plac. **place(X, Y) :- bun(X), cunoaste(X, Y), frumoasa(Y).** % Ipoteza 6

Enunțul de la ipoteza 6 defineşte o regulă Prolog. Relația **place(Cine, PeCine)** este definită atât printr-un fapt (Ipoteza 5), cât şi printr-o regulă (Ipoteza 6). În condițiile existenței regulilor în baza de cunoştinţe Prolog, satisfacerea unui scop se face printr-un procedeu similar cu cel prezentat în secțiunea b), iar unificarea scopului se încearcă atât cu fapte din baza de cunoştinţe, cât şi cu antetul regulilor de bază. La

unificarea unui scop cu antetul unei reguli, pentru satisfacerea acest scop trebuie satisfăcută şi regula. Aceasta, se reduce la satisfacerea tuturor faptelor din corpul regulii, deci conjuncția scopurilor. Scopurile din corpul regulii devin subscopuri a căror satisfacere se va încerca printr-un mecanism similar cu cel al satisfacerii scopului inițial.

Pentru baza de cunoştințe descrisă mai sus, satisfacerea scopului **?- place(ion, ana).** se va produce astfel: scopul se unifică cu antetul regulii (6) şi duce la instanțierea variabilelor din regula (6): **X = ion şi Y = ana.** Pentru ca acest scop să fie îndeplinit, trebuie îndeplinită regula, deci fiecare subscop din corpul acesteia. Aceasta revine la îndeplinirea scopurilor **bun(ion)**, care este satisfăcut prin unificare cu faptul (2) **cunoaste(ion, ana)**, satisfăcut prin unificare cu faptul (4) şi a scopului **frumoasa(ana)**, care este satisfăcut prin unificare cu faptul (1). În consecință, regula a fost îndeplinită, deci şi întrebarea inițială este satisfăcută (adevărată), iar sistemul răspunde **yes**.

În continuare vom observa ce se întâmplă dacă se pune întrebarea: **?- place(X, Y).**

Prima soluție a acestui scop este dată de unificarea cu faptul (5), iar răspunsul este:

X = mihai, Y = maria

Sistemul Prolog va pune un marcaj în dreptul faptului (5) care a satisfăcut scopul. Următoarea soluție a scopului **place(X, Y)** se obține începând căutarea de la acest marcaj în continuare în baza de cunoştințe. Scopul se unifică cu antetul regulii (6) şi se vor fixa trei noi subscopuri de îndeplinit, **bun(X)**, **cunoaste(X, Y)** şi **frumoasa(Y)**. Scopul **bun(X)** este satisfăcut de faptul (2) şi variabila **X** este instanțiată cu valoarea **ion**, **X=ion**. Se încearcă apoi satisfacerea scopului **cunoaste(ion, Y)**, care este satisfăcut de faptul (3) şi determină instanțierea **Y = maria**. Se introduce în baza de cunoştințe un marcaj asociat scopului **cunoaste(ion, Y)**, care a fost satisfăcut de faptul (3). Se încearcă apoi satisfacerea scopului **frumoasa(maria)**. Acesta eşuează. În acest moment sistemul intră într-un proces de backtracking în care se încearcă resatisfacerea scopului anterior satisfăcut, **cunoaste(ion, Y)**, în speranța că o noua soluție a acestui scop va putea satisface şi scopul curent care a eşuat, **frumoasa(Y)**. Re-satisfacerea scopului **cunoaste(ion, Y)** se face pornind căutarea de la marcajul asociat scopului în jos, deci de la faptul (3) în jos. O nouă soluție (resatisfacere) a scopului **cunoaste(ion, Y)** este dată de faptul (4) care determină instanțierea **Y = ana**. În acest moment se încearcă satisfacerea

scopului **frumoasa(ana)**. Când este vorba de un scop , căutarea se face de la începutul bazei de cunoştinţe şi scopul **frumoasa(ana)** este satisfăcut de faptul (1). În consecinţă este obţinută a doua soluţie a scopului **place(X, Y)** şi sistemul răspunde: **`X = ion, Y = ana,`** urmând un mecanism de backtracking, descris intuitiv în Figura 1.1 prin prezentarea arborilor de deducţie construiţi de sistemul Prolog.

La încercarea de resatisfacere a scopului **place(X, Y)**, printr-un mecanism similar, se observă că nu mai există alte soluţii. În concluzie, fiind dată baza de cunoştinţe Prolog anterioară, răspunsul sistemului Prolog la întrebarea **place(X, Y**) este:

```
?- place(X, Y).
X = mihai, Y = maria;
X = ion, Y = ana;
No
```

Observaţii:

• La satisfacerea unei conjuncţii de scopuri în Prolog, se încearcă satisfacerea fiecărui scop pe rând, de la stânga la dreapta. Prima satisfacere a unui scop determină plasarea unui marcaj în baza de cunoştinţe în dreptul faptului sau regulii care a determinat satisfacerea scopului.

• Dacă un scop nu poate fi satisfăcut (eşuează), sistemul Prolog reiterează şi încearcă resatisfacerea scopului din stânga, pornind căutarea în baza de cunoştinţe de la marcaj în mod descendent. Înainte de resatisfacerea unui scop se elimină toate instanţierile de variabile determinate de ultima satisfacere a acestuia. Dacă cel mai din stânga scop din conjuncţia de scopuri nu poate fi satisfăcut, întreaga conjuncţie de scopuri eşuează.

• Această comportare a sistemului Prolog în care se încearcă în mod repetat satisfacerea şi resatisfacerea scopurilor din conjuncţiile de scopuri se numeşte backtracking.

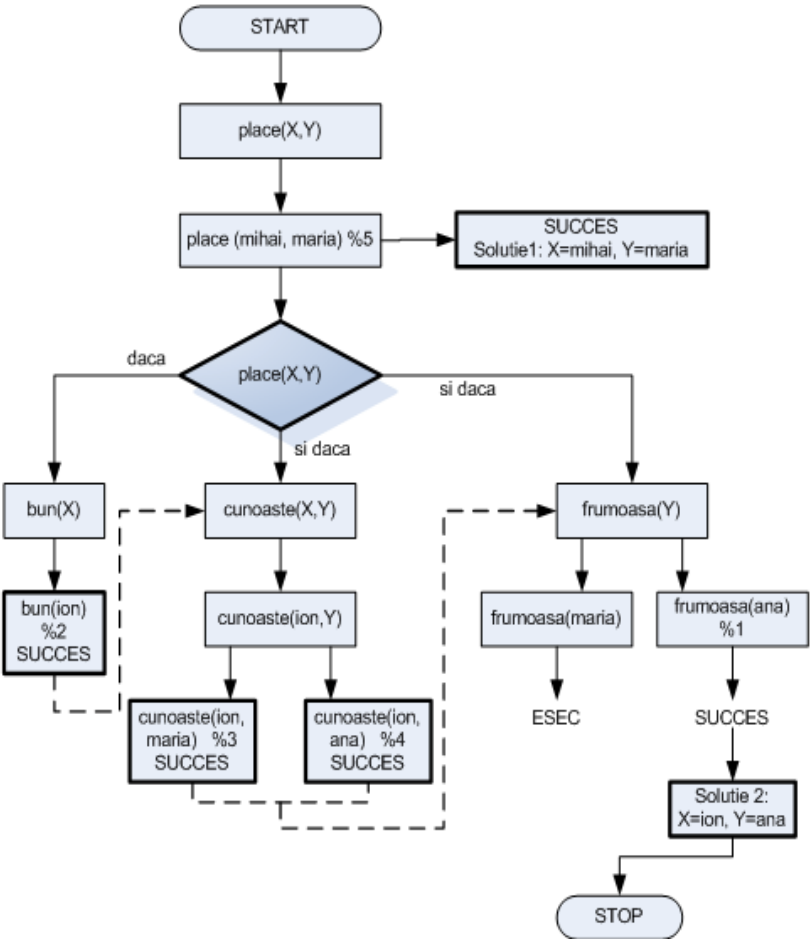


Figura 1.1. Algoritmul de satisfacere a scopurilor în Prolog

La general, o regulă are următoarea sintaxă:
nume_rel(arg1, …, argN) :- nume_rel_1(…), …, nume_rel_M(…).

Exemplu:
Putem defini predicatul soț astfel – Bărbat este soț dacă este bărbat şi este căsătorit cu o femeie (Femeie). Această definiție va fi codificată prin următoarea regulă:
sot(Barbat, Femeie):-

barbat(Barbat), cuplu_casatorit(Barbat, Femeie).
Similar,
sotie(Femeie, Barbat):-
femeie(Femeie), cuplu_casatorit(Barbat, Femeie).

Partea stângă a operatorului **:-** este numită *capul regulii* (head), iar cea din dreapta *corpul regulii* (body).

Observații:
• Un fapt poate fi văzut ca o regulă fără corp (fără condiție).
• Regulile şi faptele care definesc aceeaşi relație (au acelaşi nume de relație la secțiunea capului) trebuie grupate în sursa programului.
• În Prolog, relațiile sunt numite de obicei predicate.

După adăugarea celor două reguli, devine posibilă introducerea interogării:
?- sot(Barbat,Femeie), writeln(Barbat),fail.

Ca rezultat va apărea un răspuns în care se vor conține toate numele de bărbați din baza de cunoştințe (de exemplu: ion, andrei, gheorghe, gabriel, mihai.)

Pentru a putea urmări modul în care este rezolvată interogarea, poate fi utilizată comanda **trace**. Pe ecran vor apărea diferite subscopuri care trebuie rezolvate de programul Prolog pentru a obține răspunsul la interogare.

Următoarea regulă defineşte faptul că persoană este părintele unui copil.

parinte(Parinte,Copil):-tata(Parinte,Copil); mama(Parinte,Copil).

În această regulă, ”punctul şi virgula” (**;**) desemnează operatorul logic SAU.

Mai departe putem utiliza regula **parinte** pentru a crea o nouă regulă ce verifică dacă o persoană este copilul unei persoane anume.

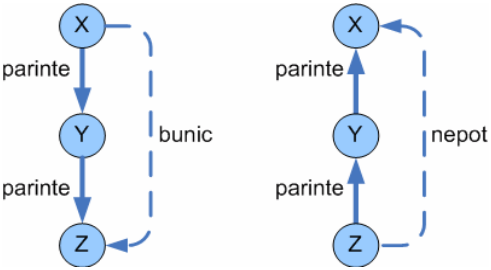


Figura 1.2. Graful pentru relațiile bunic- nepot utilizând relații deja definite

copil(Copil,Parinte):-parinte(Parinte,Copil).

Până la acest moment, am definit reguli bazate pe relațiile directe dintre persoanele ce alcătuiesc baza de cunoştințe. În continuare, vom defini două reguli care reprezintă relațiile indirecte dintre persoanele din baza de cunoştințe. Vom defini regula bunic şi regula nepot utilizând relațiile prezentate în Figura 1.2..

Pentru a afla bunicul **Z** al unei persoane **X,** procedăm astfel:

• Cine este părintele lui **X**? (presupunem că este **Y**).
• Cine este părintele lui **Y**? (presupunem că este **Z**).
• Deci, **Z** va fi bunicul lui **X**.

Relația **sora**, de exemplu, poate fi descrisă de următorul algoritm:

Pentru orice X şi Y,
X este sora lui Y dacă
(1) X şi Y au acelaşi părinte, şi
(2) X este femeie.

O posibilă implementare în Prolog a algoritmului dat mai sus este următoarea:

sora(X,Y):-parinte(Z,X), parinte(Z,Y), femeie(X), X <>Y.

Menționăm că modalitatea în care X şi Y au acelaşi părinte a fost prezentată anterior. În Prolog, operatorul <> are sensul de diferit.

PARTEA 2. Desfăşurarea lucrării

1. Se va citi breviarul teoretic. Se atrage atenția asupra faptului că toate cunoştințele din această lucrare vor fi necesare şi la efectuarea celorlalte lucrări.

2. Se vor studia exemplele propuse, încercând găsirea altor posibilități de soluționare a acestora. Se vor utiliza şi alte scopuri (interogări) pentru a testa definițiile predicatelor introduse

3. Se va elabora un arbore genealogic şi o bază de cunoştințe Prolog care ar descrie relațiile existente în familia dumneavoastră proprie care ar permite cercetarea acestor relații prin utilizarea scopurilor externe. Arborele genealogic elaborat trebuie să conțină cel puțin trei niveluri. Pentru cercetarea relațiilor existente în familie se vor utiliza nu mai puțin de şase scopuri.

4. Se va prezenta darea de seamă.

LUCRAREA DE LABORATOR nr. 2

Mecanisme de control al procesului de backtracking

Scopul: Însuşirea noţiunilor privind mecanismele specifice limbajului Prolog pentru controlul procesului de *backtracking*: *cut* şi *fail*

PARTEA 1. Noţiuni teoretice

Sistemul Prolog se implică automat într-un proces de *backtracking* atunci când acest lucru este necesar pentru satisfacerea unui scop. În unele situaţii acest comportament poate fi deosebit de util, pe când în altele poate deveni foarte ineficient. Se consideră următorul exemplu de program în care se definesc valorile unei funcţii:

```
f(X, 0) :- X < 3.          % 1
f(X, 2) :- 3 =< X, X < 6.  % 2
f(X, 4) :- 6 =< X.         % 3
```

La întrebarea:

?- f(1, Y).

Y = 0

răspunsul sistemului indică faptul că valoarea funcţiei pentru **X=1** este **Y=0**. Dacă se pune întrebarea formată din conjuncţia de scopuri:

?- f(1, Y), 2 < Y.

no

sistemul semnalează un eşec. Arborii de deducţie corespunzători sunt reprezentaţi în Figura 2.1.

Se observă că se încearcă resatisfacerea primului scop cu regulile 2 şi 3, iar acest lucru este inutil datorită semanticii acestor reguli. Astfel, dacă o valoare mai mică decât 3 pentru **X** duce la eşecul scopului **S** din conjuncţia de scopuri **f(X, Y), S**, este inutil să se încerce resatisfacerea scopului **f**, deoarece aceasta nu este posibilă. Încercarea de resatisfacere a scopului **f(X, Y)** poate fi împiedicată prin introducerea predicatului **cut**.

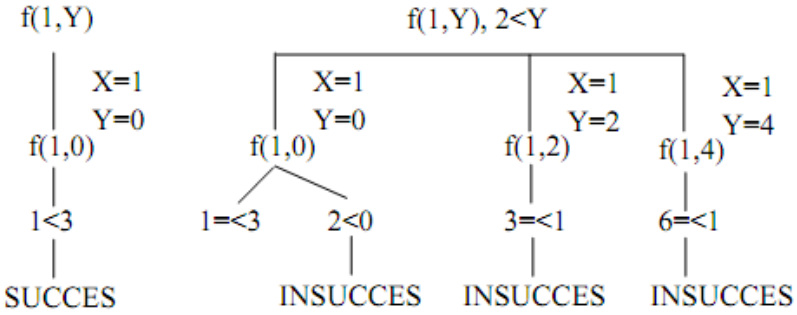


Figura 2.1. Arbori de deducție a scopurilor **f(1,Y) şi f(1,Y),2<Y**

Predicatul **cut**, notat cu atomul special **!**, este un predicat standard, fără argumente, care poate fi îndeplinit (este adevărat) întotdeauna şi nu poate fi resatisfăcut.

Predicatul **cut** are următoarele *efecte laterale*:

• La apariția predicatului **cut,** toate selecțiile făcute între scopul antet al regulii şi **cut** sunt "înghețate", deci marcajele de satisfacere a scopurilor sunt eliminate, ceea ce duce la eliminarea oricăror altor soluții alternative pentru această porțiune. O încercare de resatisfacere a unui scop între scopul antet de regula şi scopul curent va eşua.

• Dacă clauza în care s-a introdus predicatul cut reuşeşte, toate clauzele cu acelaşi antet, succesive clauzei în care a apărut cut, vor fi ignorate.

• Printr-o descriere succintă, comportamentul predicatului **cut** este următorul:

(C1) **H :- D**1, **D**2, …, **D**m, **!**, **D**m+1, …, **D**n.

(C2) **H :- A**1, …,**A**p.

(C3) **H**.

Dacă **D**1, **D**2, …, **D**m sunt satisfăcute, ele nu mai pot fi re-satisfăcute datorită lui **cut**. Dacă **D**1, …, **D**m sunt satisfăcute, C2 şi C3 nu vor mai fi utilizate pentru re-satisfacerea lui **H**. Resatisfacerea lui **H** se poate face numai prin resatisfacerea unuia din scopurile **D**m+1, …, **D**n, dacă acestea au mai multe soluții.

Utilizând predicatul **cut**, definiția funcției **f(X, Y)** poate fi rescrisă mult mai eficient astfel:

```
f(X, 0) :- X < 3, !.
f(X, 2) :- 3 =< X, X < 6, !.
f(X, 4) :- 6 =< X.
```

Predicatul **cut** poate fi util în cazul în care se dorește eliminarea din deducție a unor pași care nu conțin soluții sau eliminarea unor căi de căutare care, la fel, nu conțin soluții. El permite exprimarea în Prolog a unor structuri de control de tipul:

dacă condiție **atunci** acțiune1
altfel acțiune2

astfel

```
daca_atunci_altfel(Cond, Act1, Act2) :- Cond, !, Act1.
daca_atunci_altfel(Cond, Act1, Act2) :- Act2.
```

Se observă însă că există două contexte diferite în care se poate utiliza predicatul **cut**: într-un context predicatul **cut** se introduce numai pentru creşterea eficienței programului, caz în care el se numeşte *cut verde*; în alt context utilizarea lui **cut** modifică semnificația procedurală a programului, caz în care el se numeşte *cut roşu*.

Exemplul de definire a funcției **f(X, Y)** cu ajutorul lui **cut** este un exemplu de **cut** verde. Adăugarea lui **cut** nu face decât să crească eficiența programului, dar semnificația procedurală este aceeaşi, indiferent de ordinea în care se scriu cele trei clauze.

Utilizarea predicatului **cut** în definirea predicatului asociat structurii de control **daca_atunci_altfel** introduce un **cut** roşu, deoarece efectul programului este total diferit în cazul schimbării ordinii clauzelor. Introducerea unui **cut** roşu modifică corespondența dintre semnificația declarativă şi semnificația procedurală a programelor Prolog.

Considerăm exemplul de definire a predicatului de aflare a minimului dintre două numere în următoarele două variante:

```
min1(X, Y, X) :- X =< Y, !.      % cut verde
min1(X, Y, Y) :- X > Y.
min2(X, Y, X) :- X =< Y, !.      % cut roşu
min2(X, Y, Y).
```

În definiția predicatului **min**1 se utilizează un **cut** verde; acesta se foloseşte pentru creşterea eficienței programului, dar ordinea clauzelor de definire a lui **min**1 poate fi schimbată fără nici un efect asupra rezultatului programului. În cazul predicatului **min**2 se utilizează un **cut** roşu, asemănător structurii **daca_atunci_altfel**. Dacă se schimbă ordinea clauzelor de definire a predicatului **min**2, rezultatul programului va fi incorect pentru valorile **X < Y**.

Oportunitatea utilizării unui **cut** roşu sau a unui **cut** verde este, dintr-un anumit punct de vedere, similară cu cea a utilizării sau nu a salturilor în limbajele de programare clasică. Dacă s-ar rescrie predicatul **daca_atunci_altfel** folosind un **cut** verde, definiția lui ar fi:

```
daca_atunci_altfel(Cond, Act1, Act2) :- Cond, !, Act1.
daca_atunci_altfel(Cond, Act1, Act2) :- not (Cond),!,Act2
```

unde predicatul **not(Cond)** este satisfăcut dacă scopul **Cond** nu este satisfăcut. O astfel de definiție implică evaluarea lui **Cond** de două ori şi, dacă **Cond** se defineşte ca o conjuncție de scopuri, posibil sofisticate, atunci ineficiența utilizării unui **cut** verde în acest caz este evidentă. De la caz la caz, programatorul în Prolog trebuie să aleagă între claritate, adică păstrarea corespondenței semnificației declarative cu cea procedurală, şi eficiență.

Limbajul Prolog permite exprimarea directă a eşecului unui scop cu ajutorul predicatului **fail**. Predicatul **fail** este un predicat standard, fără argumente, care eşuează întotdeauna**.** Datorită acestui lucru, introducerea unui predicat **fail** într-o conjuncție de scopuri (de obicei la sfârşit) determină intrarea în procesul de backtracking.

Când **fail** este plasat după predicatul **cut**, nu se apelează backtracking-ul. Enunțul ***"Un individ este rău dacă nu este bun."*** se poate exprima astfel:

```
bun(gelu).
bun(vlad).
bun(mihai.
rau(X) :- bun(X), !,
fail. rau(X).
```

?-rau(gelu).
no
?- rau(petru).
yes

În cazul în care predicatul **fail** este folosit pentru a determina un eşec, cum şi în cazul exemplului de mai sus, acesta este de obicei precedat de predicatul **cut**, deoarece procesul de backtracking pe scopurile care îl preced este inutil, scopul eşuând oricum datorită lui **fail**.

Există cazuri în care predicatul **fail** este introdus intenționat pentru a genera procesul de backtracking pentru scopurile care îl preced, proces interesant nu atât din punctul de vedere al posibilității de re-satisfacere a scopului ce conține **fail,** cât din punctul de vedere al efectului lateral al acestuia.

```
rosu(mar).
 rosu(cub).
 rosu(soare.
afisare(X) :- rosu(X),
write(X),fail.
 afisare( _ ).
```

Scopul **afisare(X)** va afişa toate obiectele roşii cunoscute de programul Prolog datorită procesului de backtracking generat de **fail;** astfel, cu ajutorul lui **fail** se realizează o iterație pentru faptele rosu().

Clauza **afisare(_)** se adăugă pentru ca răspunsul final la satisfacerea scopului să fie afirmativ. În acest caz, introducerea unui **cut** înainte de **fail** ar fi determinat numai afişarea primului obiect roşu din program.

Analizând combinația **!,fail**, se consideră în continuare implementarea în Prolog a afirmației: ***"Mihai iubeşte toate sporturile cu excepția boxului".*** Această afirmație poate fi exprimată în pseudocod în forma:

dacă X este sport **şi** X este box
    atunci Mihai iubeşte X este fals
altfel dacă X este sport
    atunci Mihai iubeşte X este adevărat

şi tradusă în Prolog astfel:

iubeste(mihai, X) :- sport(X), box(X), !,
fail. iubeste(mihai, X) :- sport(X)

Predicatul **cut** utilizat aici este un **cut** roşu. Combinația **!, fail** este deseori utilizată în Prolog şi are rolul de negație. Se mai spune că limbajul Prolog modelează negația ca eşec al satisfacerii unui scop (negația ca insucces), aceasta fiind de fapt o particularizare a ipotezei lumii închise. Combinația **!, fail** este echivalentă cu un predicat standard existent în Prolog - predicatul **not**. Predicatul **not** permite drept argument un predicat Prolog şi reuşeşte dacă predicatul argument eşuează. Utilizând acest predicat, ultimul exemplu dat se poate exprima în Prolog astfel:

iubeste(mihai, X) :-sport(X),not(box(X)).
iubeste(mihai, X) :- sport(X).

Un alt predicat standard este predicatul **call**, care permite drept argument un predicat Prolog şi are ca efect încercarea de satisfacere a predicatului argument. Predicatul **call** reuşeşte dacă predicatul argument reuşeşte şi eşuează în caz contrar. Utilizând acest predicat, se poate explicita efectul general al predicatului standard **not** în următorul mod:

not(P) :- call(P), !,fail.
not(P).

Atât predicatul **not** cât şi predicatul **call** sunt predicate de ordinul II în Prolog (metapredicate), deoarece admit ca argumente alte predicate.

PARTEA 2. Desfăşurarea lucrării

1. Citiți breviarul teoretic. Se atrage atenția asupra faptului că toate cunoştințele din această lucrare vor fi necesare şi la efectuarea celorlalte lucrări.

2. Lansați la executare programul elaborat în lucrarea 1 şi cercetați schimbările semanticii procedurale:

- prin schimbarea ordinii propozițiilor - fapte;
- prin schimbarea ordinii propozițiilor - reguli; (două variante)
- prin schimbarea subscopurilor în reguli; (două variante)
- trageți concluzii.

3. Rezolvați următoarele probleme propuse şi urmăriți execuția lor corectă.

3.1. Elaborați şi testați un program pentru determinarea unei valori minime din două numere (**X** şi **Y**), fără utilizarea predicatului cut.

3.2. Elaborați şi testați un program pentru determinarea unei valori minime din două numere (**X** şi **Y**), utilizând predicatul cut roşu şi cut verde.

3.3. Care vor fi răspunsurile programului

```
p( 1).
p( 2) :- !.
p( 3).
```

Efectuați o analiză comparativă între utilizarea predicatelor cut în spațiul bazei de cunoştințe şi spațiul scopurilor pentru întrebările formulate în lista de scopuri ce urmează:

```
p( X).
p( X), p(Y).
p( X), !, p(Y).
```

3.4. Doi copii pot juca un meci într-un turneu de tenis dacă au aceeaşi vârstă. Fie următorii copii şi vârstele lor:

copil(peter,9). copil(paul,10). copil(chris,9). copil(susan,9).

Definiți un predicat din care rezultă toate perechile de copii care pot juca un meci într-un turneu de tenis.

4. Introduceți schimbările corespunzătoare în programul din

punctul 2., utilizând cut verde cel puțin în două reguli din baza de cunoştințe.

5. Introduceți schimbările corespunzătoare în programul din punctul 2., utilizând cut roşu în reguli din baza de cunoştințe. Trageți concluzii.

6. Prezentați darea de seamă.

LUCRAREA DE LABORATOR nr. 3

Structuri de date în Prolog

Scopul: Folosirea listelor, un instrument puternic al Prologului.

PARTEA 1. Noțiuni teoretice

3.1. Listele în Prolog

Lista reprezintă unul dintre tipurile cele mai utilizate de structurile de date atât în Prolog, cât şi în alte limbaje declarative. O listă este o secvență ordonată de obiecte de acelaşi tip. În Prolog, elementele unei liste se separă între ele prin virgulă şi întreaga secvență este închisă între paranteze drepte.

Exemple:
`[]` – listă vidă;
`[X, Y, Y]` – listă ale cărei elemente sunt variabilele **`X, Y şi Z`**;
`[[0, 2, 4], [1, 3]]` – listă de liste de numere întregi.

Tipurile de liste utilizate într-un program Prolog trebuie declarate în secțiunea domains sub forma:

```
tip_lista = tip*
```

unde tip este un tip standard sau definit de utilizator. O listă este compusă conceptul din două părți:

• ***cap*** (head), care desemnează primul element din listă;

• ***rest*** (tail), care desemnează lista elementelor rămase după eliminarea primului element.

Restul unei liste se mai numeşte corpul sau coada unei liste şi este întotdeauna o listă. Exemplele următoare ilustrează modul în care se structurează o listă (tabelul 3.1).

Tabelul 3.1. Exemple de structurare a listelor

Lista	Cap	Coada
['a','b','c']	'a'	['b','c']
['a']	'a'	[]
[]	nedefinit	nedefinit
[[1,2,3],[2,3,4],[	[1,2,3]	[[2,3,4],[]

Această dihotomie este utilizată în predicatele care prelucrează liste folosindu-se de avantajul regulii de unificare (identificare, substituire):

• singurul termen care se identifică cu `[]` este `[]`.

• o listă de forma `[H1|T1]` se va identifica numai cu o listă de forma `[H2|T2]` dacă `H1` se poate identifica cu `H2` şi `T1` se poate identifica cu T2.

În tabelul 3.2 sunt date câteva exemple care ilustrează această regula de unificare.

Tabelul 3.2. Exemple ce ilustrează regula de unificare

Lista1	Lista2	Legarea variabilelor
[X,Y,Z]	[Ion, Maria, Vasile]	X=Ion, Y=Maria, Z=Vasile
[7]	[X,Y]	X=7, Y=[]
[1,2,3,4]	[X,Y|Z]	X=1, Y=2, Z=[3,4]
[1,2]	[3|X]	esec

3.2. Exemple de utilizare

Majoritatea predicatele care utilizează (procesează) liste sunt recursive şi sunt definite pentru:

• cazul de bază: listă vidă `[]`

• cazul recursiv: pentru o listă de forma `[H|T]`, se efectuează anumite acţiuni asupra capului `H`, şi ulterior se apelează predicatul recursiv cu coada `T`.

Utilizând listele, putem colecta informații şi/sau date într-un singur obiect Prolog. De exemplu, fie faptele:

```
luna (1, ianuarie). luna(2, februarie). luna(3,
martie). luna(4, aprilie). luna(5, mai). luna(6,
iunie).
```

Ele pot fi redate folosind o singură listă sub forma faptului:

```
luni_prima_jumatate_an([ianuarie, februarie, martie,
aprilie, mai, iunie]).
```

Pentru o comparație mai amplă, considerăm următoarele două programe şi câteva interogări asupra lor:

```
/* program_1 */
predicates
        luna(integer,symbol)
        afis
        afis_p(integer)
        afis_n(integer)
clauses
        luna(1,ianuarie). luna(2, februarie). luna(3,
        martie).
        luna(4, aprilie). luna(5, mai). luna(6, iunie).
        afis:-luna(_,X),write(X),nl,fail.
        afis.
        afis_p(1):-luna(1,X), write(X),nl.
        afis_p(N):-N1=N1,afis_p(N1),N2=N1+1,
        luna(N2,X),write(X),nl.
        afis_n(N):-luna(N, X), write(X),nl.

Goal: afis
ianuarie februarie martie aprilie mai iunie
yes

Goal: afis_p(1)
ianuarie
yes

Goal: afis_p(3)
ianuarie februarie martie
yes

Goal: afis_n(3)
```

```
Martie
yes
```
Figura 3.1. Cod listing. Exemplu de folosire a listelor

```
/* program_2 */
Domains
 luni=symbol*

predicates
 prima_jumat_an(luni)

clauses

prima_jumat_an([ianuarie,februarie,martie,aprilie,mai,
iunie]).

Goal: prima_jumat_an(X)
X = [“ianuarie”, “februarie”, “martie”, “aprilie”,
“mai”,“iunie”]
1 Solution

Goal: prima_jumat_an([X1, X2, X3, X4, X5, X6])
X1 = ianuarie, X2 = februarie, X3 = martie, X4 =
aprilie, X5 = mai, X6 = iunie
1 Solution

Goal: prima_jumat_an([X|Y])
X = ianuarie, Y = [ “februarie”, “martie”, “aprilie”,
“mai”, “iunie”]
1 Solution
Goal: prima_jumatate_an([X|_])
X = ianuarie
1 Solution

Goal: prima_jumat_an([X,Y,Z| R])
X = ianuarie, Y = februarie, Z = martie, R =
[“aprilie”, “mai”, “iunie”]
1 Solution

Goal: prima_jumat_an([X,Y,Z |_])
 X = ianuarie, Y = februarie, Z =martie
1 Solution
```

```
Goal: prima umat_an([_,_,X|_])
X = martie
1 Solution
```

Figura 3.2. Cod listing. Exemplu de folosire a listelor eterogene

Din exemplele prezentate mai sus limbajul Prolog permite să se aleagă nu doar primul element al unei liste, ci mai multe. De asemenea, se permite lucrul cu listele în care elementele nu sunt de acelaşi tip.

În exemplul ce urmează obiectele de tip `lista` sunt liste ale căror elemente pot fi numere întregi, reale sau complexe.

```
domains
      complex=z(real,real)
      numar=r(real);i(integer);c(complex)
      lista=numar*
predicates
      p(lista)
clauses.
      p([r(2.8),i(9),r(0.89),i(77),c(z(2,6))]).

Goal: p([X|Y])
X = r(2.8), Y = [i(9),r(0.89),i(77),c(z(2,6))]
1 Solution

Goal: p([X|_]) X = r(2.8)
1 Solution

Goal: p([X, i(9) |Y])
X = r(2.8), Y = [r(0.89),i(77),c(z(2,6))]
1 Solution

Goal: p([X, r(0.89) |Y])
No Solution
```

Figura 3.3. Cod listing. Exemplu de folosire a listelor cu numere întregi, reale şi complexe

PARTEA 2. Desfăşurarea lucrării

1. Se propun spre rezolvare următoarele probleme (precizați varianta corespunzătoare la profesor):

1.1. Liniarizarea listelor. Scrie predicatul `liniar (ListaListe, Lista)`, unde `ListaListe` este o listă de elemente care pot fi la rândul lor liste, iar în `Lista` se construieşte liniarizarea listei `ListaListe`. Astfel, `liniar([1, 2, [3, 4], [5, [6, 7], [[8], 9]]], L)` va returna în `L` lista `[1, 2, 3, 4, 5, 6, 7, 8, 9]`.

1.2. Scrieți un predicat `descomp(N, Lista)` care recepționează un număr întreg N şi returnează o listă a factorilor primi ai numărului N; de exemplu: `descomp(12, [2, 2, 3])` este adevarat.

1.3. Scrieți un predicat `invers(Lista, ListaInversata)` care inversează elementele unei liste; să se scrie două variante ale predicatului de inversare a unei liste: o variantă în care lista inversată este calculata pe ramura de revenire din recursivitate şi o variantă în care lista inversată este calculată pe ramura de avans în recursivitate.

1.4. Scrieți un predicat `palindrom(Lista)` care verifică dacă o listă este palindrom (Un palindrom este o secvență care, dacă este parcursă de la stânga la dreapta sau de la dreapta la stânga, este identică, de exemplu: `[a, b, c, b, a]` sau `[a, b, c, c, b, a]`). Să se modifice predicatul anterior astfel, încât să genereze liste palindrom cu elemente 0 şi 1.

1.5. Scrieți un predicat `rotire(Lista, Directie, Nr, ListaRez)` care roteşte `Lista` cu un număr de `Nr` elemente la stânga (`dac_ Directie = stg`) sau la dreapta `(dac_ Directie = dr)`, depunând rezultatul în `ListaRez`.

1.6. Să se scrie predicatul `substitutie(X, Y, L1, L2)`, unde `L2` este rezultatul substituirii tuturor aparițiilor lui `X` din lista `L1` cu `Y`. Ex: `substitutie(a, x, [a, [b,a,] c], L2)` va produce: `L2 = [x, [b, x], c]`.

1.7. Să se scrie predicatul `imparte(L, L1, L2)` care împarte lista `L` în două sub-liste `L1` şi `L2`, care au un număr de elemente aproximativ egal, fără a calcula lungimea listei `L`. Ex: `imparte([a, b, c, d, e], L1, L2)` va produce: `L2 = [a, b, c]` şi `L3 = [d, e]`.

1.8. Să se scrie un predicat `evenmember(Elem,Lista)` care reflectă dacă `Elem` se află în `Lista` pe poziție pară. De exemplu, apelul `evenmember(X,[1,5,3,4])` va returna pe rând soluțiile `X =5; X =4`.

1.9. Să se scrie un predicat `imparte(L1,L2,L3)` care împarte lista `L1` în două liste `L2` şi `L3`, conținând elementele de pe pozițiile impare iar `L3` pe cele de pe pozițilii pare. Ex: `imparte([a,b,c,d,e],L2,L3)` va produce `L2=[a,c,e]` şi `L3=[b,d]`.

1.10. Să se scrie un program Prolog care să calculeze media numerelor unei liste.

1.11. Să se scrie un program Prolog care calculează şi afişează cel mai mare divizor comun al tuturor numerelor dintr-o listă.

1.12. Să se scrie un program Prolog care să sorteze descrescător numerele unei liste.

1.13. Scrieți un program care utilizează predicatul `listaPara`, care are conține două argumente: o listă de numere întregi, iar al doilea argument returnează o listă cu toate numerele pare din prima listă.

1.14. Să se elimine primele N elemente de la începutul unei liste. Numărul elementelor pentru eliminare se va introduce utilizând predicatul `readln` al limbajului Prolog.

1.15. Să se elimine ultimele N elemente a unei liste. Numărul elementelor pentru eliminare se va introduce utilizând predicatul `readln` al limbajului Prolog.

2. Se va prezenta darea de seamă.

CAPITOLUL II

Exemple şi probleme în limbajul Prolog

LUCRAREA DE LABORATOR nr. 4

Sisteme expert

Scopul: Studierea principiilor de proiectare şi de organizare a sistemelor expert bazate pe logică şi reguli.

PARTEA 1. Noțiuni teoretice

4.1. Scopul şi structura generală a sistemelor expert

Sistemul expert (*SE*) este un program (pachet de programe), care simulează într-o oarecare măsură activitatea unui expert uman într-un anumit domeniu. Mai mult decât atât, acest domeniu este strict limitat. Principalul scop al SE este de a consulta în domeniul pentru care acest SE este proiectat.

Un SE este format din trei componente principale (fig. 4.1):

1) *Baza de cunoştințe (BC).* BC este partea centrală a sistemului expert. Aceasta conține o colecție de fapte şi de cunoştințe (regulile) pentru extragerea altor cunoştințe. Informațiile conținute în baza de cunoştințe sunt folosite de către SE pentru determinarea răspunsului în timpul consultării. De regulă, BC sunt separate de programul principal sau stocate pe alte mijloace fixe.

2) *Mecanismul (motorul) de inferență.* MI conține descrieri ale modului de aplicare a cunoştințelor cuprinse în baza de cunoştințe. În timpul consultării, MI inițiază SE să înceapă procesările, îndeplinind regulile determină admisibilitatea soluției găsite şi transmite rezultatele la Interfața sistemului (System User Interface).

3) *Interfața sistemului utilizatorului (ISU)* este parte a SE, care interacționează cu utilizatorul. Funcțiile ISU includ: recepționarea informațiilor de la utilizator, transferul rezultatelor în forma cea mai

convenabilă utilizatorului, explicarea rezultatelor recepționate de SE (oferă informații cu privire la atingerea rezultatelor).

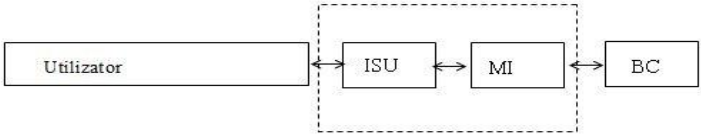


Figura 4.1. Structura generală a SE

În funcție de metoda de clasificare şi plasare a informației există mai multe tipuri de Baze de cunoştințe: *de producție, de rețea* şi *de cadru* (frame-uri) modele de reprezentare a cunoştințelor.

Modelul de rețea se bazează pe reprezentarea cunoştințelor în forma unei rețele ale cărei noduri corespund conceptelor iar arcele relațiilor dintre ele.

La baza *modelului pe cadre (frame-uri)* se află o grupare logică de atribute a obiectului, precum şi depozitarea şi prelucrarea grupurilor logice care sunt descrise în cadre.

Modelul de producție se bazează pe regulile de forma "dacă-atunci" şi permite introducerea fragmentelor de cunoştințe faptice în regulile limbajului Prolog. Anume astfel sunt construite SE bazate pe reguli.

La punerea în aplicare a SE bazat pe logică, baza de cunoştințe reprezintă un set de afirmații, fapte. Elaborarea concluziei unui expert în acest caz se bazează pe mijloacele standard de lucru cu listele.

4.2. Determinarea rezultatului (răspunsului) expert

Prin concluzia SE, se subînțelege dovada faptului că în setul de ipoteze se conține concluzia căutată. Logica de obținere a răspunsului (concluziei) sunt specificate de regulile de inferență. Concluzia (rezultatul) se obține prin căutarea şi compararea modelului.

În SE, bazat pe reguli, întrebările (scopurile) utilizatorului sunt transformate într-o formă care este comparabilă cu normele de forma BC. Motorul de inferență inițiază procesul de corelare de la regula de "top" (vârf). Recursul la reguli este numit "apelare". Apelarea regulilor relevante în procesul de corelare continuă atâta timp, cât nu există o

comparație sau nu este epuizată toată BC, iar valoarea nu este găsită. În cazul în care MI detectează că pot fi apelate mai mult decât o regulă, începe procesul de soluționare a conflictului. În soluționarea conflictului prioritate se dă regulilor care sunt mai specifice, sau regulilor ce țin de mai multe date actuale.

În SE, bazate pe logică, interogările sunt transformate în valori care sunt comparate cu listele de valori prezente în BC.

Procesul de unificare în programele Turbo Prolog şi Visual Prolog ajută la rezolvarea problemei de găsire şi comparare după exemplu şi nu necesită scrierea unor reguli suplimentare. Ca şi în sistemele bazate pe reguli, în cel bazat pe logică, utilizatorul obține răspunsuri la interogările (scopurile) sale, în conformitate cu logica stabilită în SE. Pentru punerea în aplicare a mecanismului de extragere a răspunsului expertului este suficient să se scrie specificațiile necesare.

4.3. Sistemul expert bazat pe reguli şi realizarea lui

SE pe baza regulilor permite proiectantului a construi regulile care sunt în mod natural combinate în grupuri de fragmente de cunoştințe. Independența reciprocă a regulilor de producție face ca baza de reguli să fie semantic modulară şi capabilă de dezvoltare.

Realizarea în Turbo Prolog (sau Visual Prolog) a SE bazat pe reguli conține un set de reguli care sunt invocate de către datele de intrare în momentul de corelare (comparare). Împreună cu regulile MI, SE este compus dintr-un *interpretator* care selectează şi activează diferite sisteme.

Activitatea acestui interpretator este descrisă într-o succesiune de trei etape:

1) Interpretatorul compară un exemplu de regulă cu elementele de date în baza de cunoştințe.

2) În cazul în care este posibilă apelarea mai multor reguli, pentru a selecta o regulă, se utilizează mecanismul de soluționare a conflictelor.

3) Regula selectată este folosită pentru a găsi un răspuns la această întrebare.

Acest proces în trei etape este ciclic şi se numeşte *ciclul de detectare-acțiune*. Răspunsul afirmativ al SE este rezultatul uneia din regulile de producție. Selecția regulii se efectuează, în conformitate cu datele de intrare.

Elaborarea a SE în Turbo Prolog (sau Visual Prolog), bazat pe reguli, începe cu declarația bazei de fapte. Baza de fapte stochează (păstrează) răspunsurile utilizatorului la întrebările ISU. Aceste date sunt răspunsurile pozitive sau negative. În continuare, se vor construi regulile de producție care vor descrie fragmente de cunoştințe actuale. Exemplu (SE de identificare şi selectare a rasei câinilor):

```
dog_is("buldog englez"): -
it_is("cu par scurt"),
positive("are","mai mic de 22 inch"),
positive("are", "coada atirnata"),
positive("are", "caracter bun"), !.
```

/ * În mod similar sunt descrise cunoştințele cu privire la alte rase de câini * /

```
dog_is("cocher-spaniel"): -
it_is("cu par lung"),
positive("are", " mai mic de 22 inch"),
positive("are", "coada atirnata"),
positive("are", "urechi lungi"),
positive("are", "caracter bun"), !.
```

Mai mult decât atât, în scopul de a limita spațiul de căutare care descrie fragmentele de cunoştințe de reguli, ultimele pot fi grupate în bază prin introducerea de reguli auxiliare pentru identificarea subcategoriilor. De exemplu, în SE alegerea rasei de câine va fi regula `it_is`, care identifică rasa de câine pe motive de apartenență la un grup de câini cu părul lung sau cu părul scurt:

```
it_is("cu par scurt"):-
positive("cainele are", "par scurt"), !.

it_is("cu par lung"):
positive("cainele are", "par lung"), !.
```

În exemplul considerat aici, datele pentru selectarea rasei de câini sunt cunoscute, deoarece sunt selectate rasele de câini comune. Setul de caracteristici de clasificare a speciilor este selectat în baza următoarelor criterii:

• toate atributele utilizate sunt necesare pentru a identifica rasa.

• nici unul dintre atribute nu este comun pentru toate speciile simultan.

Regulile MI compară datele utilizatorului cu datele ce se conțin în regulile de producție (regulile `pozitive` şi `negative` în acest SE), precum şi păstrarea în continuare "a traseului" răspunsurilor afirmative şi negative (regula `remember` pentru adăugarea în baza de date a răspunsurilor 1 (da) şi 2 (nu)), care sunt utilizate în comparare cu modelul:

Mecanismul de determinare a (găsirea) răspunsului

`xpositive (X, Y)` şi `xnegative (X, Y)` predicatele bazei dinamice de date păstrează respectiv, răspunsurile afirmative şi negative ale utilizatorului la întrebările care ISU le pune pe baza faptelor argumentelor predicatului `positive` în corpul regulii `dog_is`

```
positive(X, Y):
      xpositive(X,Y), !.
positive(X, Y):
      not(negative(X,Y)), !, ask(X,Y).
 negative(X, Y):
      xnegative(X,Y), !.
```

Mecanismul de consultare

Predicatul `dlg_Ask` creează o fereastră standard de dialog pentru a obține răspunsul utilizatorului la întrebarea Da/Nu.

```
ask(X, Y):
concat("Intrebare : " X, Temp),
concat(Temp, " " , Temp1),
concat(Temp1, Y, Temp2),
concat(Temp2, " "?, Quest),
Reply1=dlg_Ask("Consultare", Quest, ["Da", "Nu"]),
Reply=Reply1+1,
remember(X, Y, Reply).
```

Introducerea răspunsurilor utilizatorului în baza de date dinamică.

```
remember(X, Y, 1): !,
      assertz(xpositive(X, Y)).
remember(X, Y, 2): !, assertz(xnegative(X, Y)), fail.
```

4.4. Sistem expert bazat pe logică şi realizarea lui

În acest caz, BC constă în declaraţii sub formă de enunţuri în logica predicatelor. O parte din afirmaţii descrie obiectele, iar cealaltă parte a afirmaţiilor descrie condiţiile şi atributele, ce caracterizează diferite obiecte. Numărul de trăsături determină gradul de precizie de clasificare. Interpretatorul în cadrul sistemului îndeplineşte funcţiile sale în baza schemei următoare:

1) Sistemul conţine în baza de cunoştinţe enunţuri (fapte) care guvernează (conduc) căutarea şi compararea. Interpretatorul compară aceste enunţuri cu elementele aflate în baza de date.

2) În cazul în care este posibilă apelarea mai multor decât a unei singure reguli, pentru a rezolva conflictul sistemul utilizează mecanismul intern de unificare a Prolog-ului.

3) Sistemul recepţionează rezultatele procesului de unificare în mod automat, de aceea ele sunt trimise pe dispozitivul necesar (logic) pentru output-ul informaţiei.

La fel ca în SE, bazat pe reguli, acest proces ciclic este proces de detectare-acţiune. Principala diferenţă în structura SE bazat pe logică constă în descrierea obiectelor şi atributelor sub formă de fapte:

Condiţii-caracteristici de rase diferite.

```
cond(1, "rasa cu par lung").
cond(2, " rasa cu par lung").
cond(3, "mai mic de 22 inch").
cond(4, "mai mare de 30 inch").
cond(5, "coada atirnata").
cond(6, "urechi lungi").
cond(4, "caracter bun").
cond(8, "greutate mai mare de 100 pounds").
```

Datele despre tipurile de rase:

```
topic("cu par scurt").
topic("cu par lung").
```

Datele despre rasele concrete:

```
rule(1, "ciine", "cu par scurt", [1]).
rule(2, "ciine", "cu par lung", [2]).
```

```
rule(3, "cu par scurt", "buldog englez", [3,5,4]).
rule(4, "cu par scurt", "copoi", [3,6,4]).
rule(5, "cu par scurt", "dog danez", [5,6,4,8]).
rule(6, "cu par scurt", "foxterier american", [4,6,4]).
rule(4, "cu par lung", "cocher-spaniel", [3,5,6,4]).
rule(8, "cu par lung", "setter irlandez", [4,6]).
rule(9, "cu par lung", "colli", [4,5,4]).
rule(10, "cu par lung", "senbernar", [5,4,8]).
```

Al treilea argument al predicatului `rule` reprezintă o listă de numere întregi - condiții, din enunțurile tipului `cond`. Fiecare enunț (fapt) specifică tipul de `cond` şi stabileşte condiția de selecție a clasificării de rase de câini utilizate aici. În SE bazate pe logică, interpretatorul utilizează aceste numere de condiție pentru a efectua alegerile (selecțiile) adecvate.

MI conține regulile de tratare a listelor de atribute în descrierea obiectelor. Prin utilizarea predicatului `go` MI verifică afirmațiile BC `rule` şi `cond` pentru a clarifica cu ajutorul regulii `check` existența sau lipsa de valori adecvate de date:

Regula inițială a mecanismul de determinare a răspunsului:

```
go(_, Mygoal):
      not(rule(_, Mygoal, _, _)), !, concat("Rasa
      recomandata : ", Mygoal, Temp), concat(Temp, ".",
      Result),
      dlg_Note("Concluzia expertului : ", Result).
go(History, Mygoal):
      rule(Rule_number, Mygoal, Type_of_breed,
      Conditions), check(Rule_number, History,
      Conditions), go([Rule_number|History],
      Type_of_breed).
```

Compararea datelor de intrare a utilizatorului cu listele de atribute a raselor aparte de câini:

```
check(Rule_number, History,
[Breed_cond|Rest_breed_cond_list]):
      yes(Breed_cond), !,
      check(Rule_number, History,
Rest_breed_cond_list).
check(_, _, [Breed_cond|_]):
      no(Breed_cond), !, fail.
```

```
check(Rule_number, History,
[Breed_cond|Rest_breed_cond_list]):
      cond(Breed_cond, Text),
      ask_question(Breed_cond, Text),
      check(Rule_number, History,
Rest_breed_cond_list).
check(_, _, [ ]).
      do_answer(Cond_number, 1): !,
      assertz(yes(Cond_number)).

      do_answer(Cond_number, 2): !,
      assertz(no(Cond_number)), fail.
```

Solicitarea şi recepționarea răspunsurilor „da” şi „nu” de la utilizator:

```
ask_question(Breed_cond, Text):
      concat("Intrebare : ", Text, Temp),
      concat(Temp, " ", Temp1),
      concat(Temp1, "?", Quest),
      Response1=dlg_Ask("Consultare", Quest,
      ["Da","Nu"]), Response=Response1+1,
      do_answer(Breed_cond, Response).
```

Regula `go` încearcă să potrivească obiectele care sunt clasificate prin numărul de condiții. Dacă ”potrivirea” se produce, modulul programului ar trebui să adauge la baza de cunoştințe valorile potrivite şi să continue procesul cu noile date recepționate din partea utilizatorului. În cazul în care ”potrivirea” nu se produce, mecanismul opreşte procesul actual şi alege o altă cale. Căutarea şi compararea continuă până când toate posibilitățile sunt epuizate.

Avantajul SE bazat pe logică constă în capacitatea de a stoca faptele bazei de cunoştințe în afirmațiile bazei de date dinamice. Baza de cunoştințe poate fi plasată într-un fişier pe disc, ceea ce o face independentă de codul sursă.

PARTEA 2. Desfăşurarea lucrării

1. Pe baza materialului însuțit în cadrul acestui curs şi exemplului discutat anterior se va studia realizarea ambelor tipuri de sisteme de expert prin intermediul limbajului Prolog.

2. Se va elabora un sistem expert pentru un anumit domeniu selectat în conformitate cu numărul variantei din Tabelul 4.1., sau în orice alt domeniu corelat cu profesorul. Numărul de obiecte descrise ar trebui să fie cel puțin 12, iar descrierea atributele lor nu mai puțin de 8.

3. Se va analiza realizarea unui sistem expert bazat pe logică şi sistem expert care se bazează pe reguli, realizând mecanismul deductiv şi inductiv.

Tabelul 4.1. Domeniul pentru sistemul expert

Varianta	**Domeniul**
1,5	Microprocesoare
2,6	Dispozitive mobile
3,10	Sisteme operaționale
4,11	Limbaje de programare
7,12,19	Jocuri pe computere
8,15,20	Viruşi de computere
9,16	Rețele de calculatoare
13,17	Algoritmi de sortare
14,18	Algoritmi de căutare

Raportul trebuie să conțină:

1) formularea scopului şi obiectivelor de cercetare;
2) descrierea caracteristicilor sistemului expert elaborat;
3) diagramele datelor de intrare şi diagrama structurală a sistemului expert atât bazat pe reguli, cât şi pe logică;
4) listing-ul programului cu comentarii şi explicații, justificări;
5) descrierea mecanismelor de determinarea răspunsului în sistemul expert;
6) concluzii cu privire la experimentele efectuate.

LUCRAREA DE LABORATOR nr. 5

Prelucrarea limbajului natural

Scopul: Însuşirea principiilor fundamentale de prelucrarea limbajului natural.

PARTEA 1. Noțiuni teoretice

5. 1. Prelucrarea limbajului natural

Prelucrarea limbajului natural (PLN) reprezintă o tehnologie (un ansamblu de procese, metode operații) care creează modalități de a executa diferite sarcini referitoare la limbajul natural cum ar fi construcția unor interfețe bazate pe limbaj natural ca baze de date, traducerea automată etc. Procesarea limbajului natural reprezintă şi astăzi o problemă dificilă şi în cea mai mare parte nerezolvată. Găsirea unei tehnologii adecvate este extrem de grea datorită naturii multidisciplinare a problemei, fiind implicate următoarele ştiințe şi domenii: lingvistică, lingvistica computațională, informatică şi inteligență artificială.

Aplicațiile procesării limbajului natural se înscriu în trei mari categorii:

a. Aplicațiile bazate pe text:
    - clasificarea documentelor (găsirea documentelor legate de anumite subiecte);
    - regăsirea informației (căutarea unor cuvinte-cheie);
    - extragerea informației (legate de un anumit subiect);
    - înțelegerea textelor (ce presupune o analiză profundă a structurii acestora);
    - traducerea automată şi traducerea asistată de calculator dintr-o limbă în alta;
    - alcătuirea de sinteze;
    - achiziția de cunoştințe.

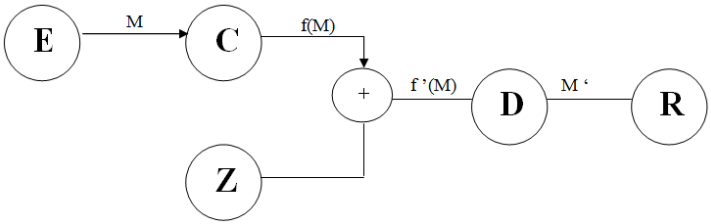
b. Aplicațiile bazate pe dialog, care implică comunicarea între om şi maşină, aplicații cum ar fi sistemele de învățare, sistemele de interogare şi răspuns la întrebări, rezolvarea problemelor etc.
c. Procesarea vorbirii.

Comunicarea este schimbul intenționat de informație generat de producerea şi perceperea semnelor dintr-un sistem partajat de semne convenționale.

Pentru comunicare trebuie folosit un limbaj ce poate fi de mai multe tipuri: verbal (auditiv), text, mimică, etc.

5.1.1. Componentele comunicării

Schema de bază a comunicării:



E – emițător
C - bloc de codificare
Z - zgomot extern
D - bloc de decodificare
R – receptor
M - mesaj
f(M) – mesajul codificat în limbaj
$f^1(M)$ – codificarea modificată de zgomot
M^1 – mesajul nou

Emițător:

- intenție - dorința emițătorului de a transmite un mesaj către un receptor;
- generare - crearea intenției de a comunica şi a conținutului ei într-un limbaj;
- sinteză - procesul efectiv de trimitere a mesajului.

Receptor:

- percepție - procesul de recepționare a mesajului;
- analiza - interpretarea mesajului din punct de vedere lexical, sintactic, semantic, pragmatic;
- dezambiguizarea – selecția dintre posibilele rezultate ale analizei;
- încorporare – introducerea conținutului mesajului.

Actul de comunicare

Acesta conține mai multe interpretări/sensuri:

- Locuția – fraza, așa cum e spusă de locutor (emițător);
- Ilocuție – înțelesul frazei ce se doreşte a fi comunicat;
- interlocuțiune – acțiunea ce rezultă din locuție.

Exemplu:

Maria i-a spus lui Ionică: "Te rog închide uşa"

locuție — ilocuție — interlocuțiune ⟶ **uşa închisă**

Categorii locuționale:

- asertive – ascultătorul este informat în legătură cu fraza;
- directive – ascultătorul efectuează o acțiune după aflarea informației;
- comisive (angajamente) – ascultătorul află de acțiuni viitoare;
- expresive – mesaje ce exprimă atitudinea emițătorului legat de un fapt.

5.1.2. Definirea unui limbaj

Un prim element într-un limbaj este *lexiconul***.** Acesta conține termenii vocabularului pentru limbajul respectiv.

Exemplu:

Lexicon:

Noun $\rightarrow$ *breeze* | *wumpus* | *ball*
Verb $\rightarrow$ *is* | *see* | *smell* | *hit*
Adjective $\rightarrow$ *right* | *left* | *smelly ...*
Adverb $\rightarrow$ *here* | *there* | *ahead ...*
Pronoun $\rightarrow$ *me* | *you* | *I* | *it*

RelPronoun → *that* | *who*
Name → *John* | *Mary*
Article → *the* | *a* | *an*
Preposition → *to* | *in* | *on*
Conjunction → *and* | *or* | *but*

Pe baza lexiconului se construiesc propoziții/fraze în limbajul respectiv. Aceste fraze se construiesc pe baza unor *gramatici*.

Aceste fraze (construcții lexicale) se pot analiza din mai multe perspective:

- *analiza lexicală* – procesul de conversie a frazei în atomi (realizat de un procesor)
- *analiza semantică* – procesul de asociere a unui sens pentru fiecare atom; sens folosit ulterior pentru a reține cunoştințe dobândite din frază
- *analiza pragmatică* - procesul de considerare al contextului în analiza unei fraze. Este necesar pentru a elimina ambiguitățile (apar ambiguități când un cuvânt poate avea mai multe sensuri, cel efectiv depinzând de frază sau de contextul dialogului).

Gramaticile definesc regulile pe care o frază trebuie să le respecte, din punct de vedere al structurii, pentru a face parte dintr-un limbaj.

Ele se pot defini în mai multe moduri, unul dintre ele folosind următoarele concepte:

a) *Neterminali* – expresii care se substituie cu alte expresii/ elemente din gramatică sau lexicon:
- sentence **S**
- noun phrase **NP**
- verb phrase **VP**
- prepositional phrase **PP**

Exemplu:

S → NP VP | S Conjunction S
NP → Pronoun | Noun | Article Noun | NP PP
VP → Verb | VP NP | VP Adjective | VP PP | VP Adverb
PP → Preposition NP

b) *Terminali* – expresii care nu se mai substituie cu alte expresii din gramatică (ei reprezintă direct categoriile din lexicon):

Noun | Verb | Adjective | Adverb | Pronoun

c) *Reguli de rescriere* sunt regulile de extindere a gramaticii.

d) *Simbol de început* este un neterminal din care poate fi derivată orice frază corectă din punct de vedere gramatical.

5.1.3. Analiza sintactică

Procesul de recunoaştere a structurii unei propoziții de către un calculator se numeşte **parsing** *(procesare)***.** Se poate realiza în mai multe moduri:

- *top-down parsing* (de sus în jos);
- *bottom-up parsing* (de jos în sus).

Top-Down parsing – se pleacă de la regulile gramaticii până când se identifică fraza.

Exemplu: "John hit the ball"

1. S
2. S → NP, VP
3. S → Noun, VP
4. S → John, Verb, NP
5. S → John, hit, NP
6. S → John, hit, Article, Noun
7. S → John, hit, the, Noun
8. S → John, hit, the, ball

Bottom – down parsing **-** se pleacă de la frază şi se încearcă obținerea simbolului de început al gramaticii.

Exemplu: "John hit the ball"

1. John, hit, the, ball
2. Noun, hit, the, ball
3. Noun, Verb, the, ball
4. Noun, Verb, Article, ball
5. Noun, Verb, Article, Noun
6. NP, Verb, Article, Noun

7. NP, Verb, NP
8. NP, VP
9. S

5.1.4. Definite Clause Grammar (DCG)

Există mai multe tipuri de reprezentare pentru gramatici pe care le expunem în continuare:

a) Gramatici în forma BNF (*Backus Normal Form* sau *Backus–Naur Form)* **–** modul formal matematic de descriere a limbajului.

Sunt gramatici dependente de context, bazate pe expresii de forma:

```
<simbol> ::= _expresie_
```

unde,

`<simbol>` - un neterminal, iar `_expresie_`– o secvență de simboluri (terminali şi neterminali).

Exemplu:

```
<syntax> : : = <rule> | <rule> <syntax>
<rule> : : = <whitespace> „<”<rule_name>„>”
<whitespace> : : = „”<whitespace >|„” etc.
```

BNF este o gramatică dependentă de context (GDC), ceea ce e dificil a fi transpus într-un program automat.

b) Gramatici cu clauze definite DCG – *Definite Clause Grammar*

În modul implicit, DCG sunt gramatici independente de context folosite pentru prelucrarea de limbaje formale sau naturale, cu proprietatea că se pot integra foarte uşor în limbaje de programare logică (cum e Prolog de exemplu). Aceasta se realizează prin faptul că DCG se bazează pe FOPL (First Order Predicate Logic) în reprezentare.

În DCG:

- fiecare regulă din gramatică poate fi văzută ca o regulă din DCG;
- fiecare categorie sintactică se reprezintă printr-un *predicat* cu un *argument şir.*

Proprietăți:

- aceeaşi gramatică se foloseşte şi pentru analiză (recunoaştere) şi pentru generare;
- procesul de analiză lexicală (parsing) se realizează exclusiv prin inferențe logice;
- ca abordare:
    a. *Bottom-up parsing* – forward chaining;
    b. *Top-down parsing* – backward chaining.

Folosirea predicatelor are loc astfel:
"NP(s) este adevărat dacă s este NP "
deci,

S → NP VP

va deveni

NP(s1) ∧ VP(s2) ⇒ S(append(s1,s2)).

În BNF:

S → NP VP

În LP / DGC:

NP(s1) ∧ VP(s2) ⇒ S(append(s1, s2)).

În BNF:

NP → Pronoun | Noun

În LP / DGC:

Pronoun(s) ∨ Noun(s) ⇒ NP(s)

În BNF:

Noun → ball | book

În LP / DGC

(s = "ball" ∨ s = "book") ⇒ Noun(s) .

Tabelul 5.1. Soluția comparativă cu limbajul Prolog.

BNF	FOPL/DCG	PROLOG

S → NP VP
 	
NP → Noun
 	
Noun → stench
Noun → wumpus
 	
VP → Verb
Verb → smells
Verb → kills
	NP(s1) ∧ VP(s2) ⇒ S(append(s1,s2))
	
	Noun(s) ⇒ NP(s)
	
	(s = "stench" ∨ s = "wumpus") ⇒ Noun(s)
	
	Verb(s) ⇒ VP(s)
	(v = "smells" ∨ v = "kills") ⇒ Verb(v)
		`sentence([S1, S2]) :- np(S1), vp(S2).`
		
		`np(S):- noun(S).`
		
		`noun(stench).`
		`noun(wumpus).`
		
		`vp(S):- verb(S).`
		`verb(smells).`
		`verb(kills).`
		
		`?- sentence([wumpus, smells]).`
		`?-sentence([S1, S2]).`

5.2. Modele de analiză sintactică a propozițiilor

5.2.1. Rețele de tranziție cu stări finite - Finite State Transition Network

MODEL: *Moldova has a rich culture.*
English is the most widespread language.
I enjoy learning new things.

Pentru a implementa o rețea de tranziție cu stări finite RTSF în limbajul Prolog trebuie să efectuăm o descriere a rețelei, precum şi a modului în care se efectuează parcurgerea ei. O descriere a RTSF constă din trei componente: numele rețelei, o mulțime de declarări şi o mulțime de descrieri ale arcelor. Exemplu de rețea este dat prin automatul de mai jos figura 5.1.

Acest automat are 8 stări, starea 1 fiind starea inițială, iar starea 8 starea finală. Cele opt stări sunt conectate prin 10 arce etichetate.

Numele rețelei nu joacă un rol anume şi nici o altă componentă nu face referire la el. El este introdus din motive de consistență şi joacă un rol important în definirea rețelelor de tranziție recursive.

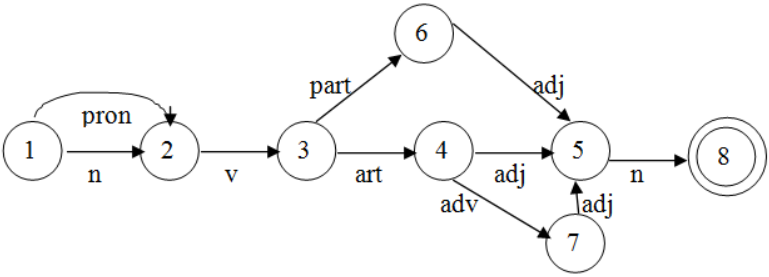


Figura 5.1. Automat

Cea de-a doua componentă constă dintr-o unică declarare obligatorie a unor stări inițiale şi finale. Cea de-a treia componentă majoră a reprezentării constă dintr-o mulțime alcătuită din una sau mai multe descrieri ale arcelor, fiecare din acestea având aceeaşi formă.

În Prolog, RTSF vor fi prezentate ca rețele în mod declarativ, ca nişte structuri de date (fapte din baza de fapte), care pot fi examinate şi manipulate. Având o astfel de descriere, se vor putea scrie programe generale de **recunoaştere** şi de **generare**.

Reprezentarea RTSF ca structuri de date va specifica: nodurile inițiale, nodurile finale, arcele unde fiecare arc este definit prin nodul de plecare, nodul de destinație şi eticheta arcului.

Aceste informații pot fi comunicate în Prolog, utilizând predicatele de forma:

```
initial(Nod).
final(Nod).
arc(Nod_Plecare,Nod_Destinație,eticheta).
```

Exemplu:

```
initial(1).
final(8).
arc(1,n,2).
arc(1,pron,2).
arc(2,v,3).
```

Desigur, descrierea rețelei trebuie completată prin specificarea semnificației abrevierii pe care o reprezintă cel de-al treilea argument al predicatului **arc.** Aceasta va fi indicată prin utilizarea predicatului:

```
word(moldova,n).
```

```
word(has,v).
word(a,art).
```

Atunci când rețeaua este utilizată pentru recunoaştere, fiecare moment al parcurgerii ei este caracterizat de două elemente:

R1. *numele unui nod (locația curentă)*

R2. *şirul de intrare rămas*

Exemplu:

```
?- recognize([i,enjoy,learning,new,things]).
```

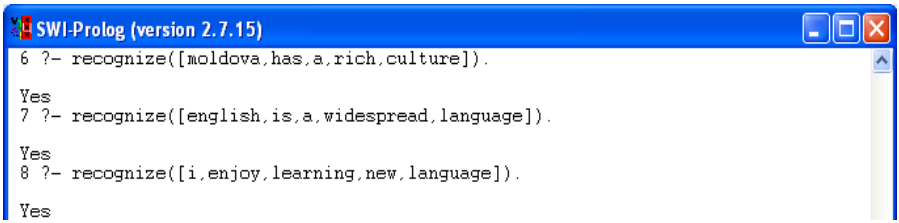


Figura 5.2. Rezultatul execuției *recognize()*.

Atunci când rețeaua este utilizată pentru **generare** aceste elemente vor fi înlocuite cu următoarele

G1. *numele unui nod (locația curentă)*

G2. *şirul de ieşire generat până la acel moment*

Exemplu:

```
?- recognize(X). /*Figura 5.3*/
```

Lexicon:

Noun (n) - Moldova | culture | language | things | english
Verb (v) - has | is | enjoy |
Participle (part) - learning
Article (art) – a | the | I
Adjective (adj) – rich | most | widespread | new

Gramatica:

S → NP VP
NP → N | Adj N | Art Adj Adj N |
VP → V | V Part

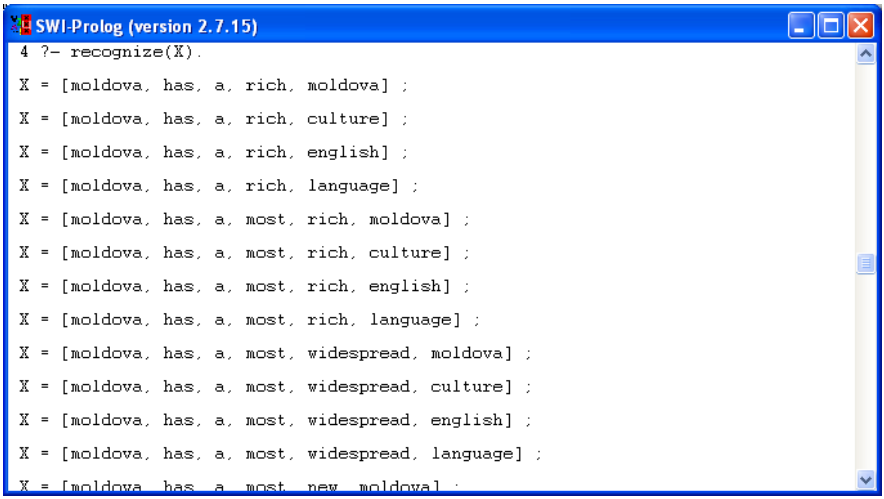


Figura 5.3. Rezultatul execuției *recognize(X).*

Codul sursă:

```
        config([],State):-final(State).
        config([Word|Rest],State):-
                word(Word,Cat),
                arc(State,Cat,State1),
                config(Rest,State1).
        recognize(String):-
                initial(State),
                config(String,State).
        initial(1).                     word(a,art).
        final(8).                       word(rich,adj).
        arc(1,n,2).                     word(culture,n).
        arc(1,pron,2).                  word(english,n).
        arc(2,v,3).                     word(is,v).
        arc(3,part,6).                  word(the,art).
        arc(3,art,4).                   word(most,adv).
    arc(4,adj,5).                       word(widespread,adj).
    arc(5,n,8).                         word(language,n).
    arc(6,adj,5).                       word(i,pron).
    arc(4,adv,7).                       word(enjoy,v).
    arc(7,adj,5).                       word(learning,part).
    word(moldova,n).                    word(new,adj).
    word(has,v).                        word(things,n).

?- recognize([i,enjoy,learning,new,things]).
Yes.
```

5.2.3. Analiza propoziției bazată pe cadre - Sentence Frame Grammars.

MODEL: *Moldova has a rich culture.*
English is the most widespread language.
I enjoy learning new things.

Codul sursă pentru `sfg_rec.pl`**:**

```
/* sentence frame grammar */
recognize(String):-
      transform(String,CatString),
      frame(CatString).
/*transformarea unui sir de cuvinte intr-un sir de
categrorii lexicale*/
transform([],[]).
transform([Word|String],[Cat|CatString]):-
      word(Word,Cat),
transform(String,CatString).
```

Codul sursă pentru `lex.pl`***:***

```
 /*baza de fapte a cuvintelor cu categoriile
lexicale*/
word(moldova,n).                word(most,adv).
word(has,v).                    word(widespread,adj).
word(a,art).                    word(language,n).
word(rich,adj).                 word(i,pron).
word(culture,n).                word(enjoy,v).
word(english,n).                word(learning,part).
word(is,v).                     word(new,adj).
word(the,art).                  word(things,n).
```

Codul sursă pentru `sf.pl`**:**

```
/*lista propoziţiilor cadre*/
frame([n,v,art,adj,n]).
frame([pron,v,part,adj,n]).
frame([n,v,art,adv,adj,n]).
```

Deschideți fişierul `sfg_rec.pl`

```
/*incarcarea fisierelor cu baza de fapte si lista cadrelor*/
      ?-consult('lex.pl').
      ?-consult('sf.pl').
```

```
?-
transform([english,is,the,most,widespread,language],Ca
tString).

CatString=[n,v,art,adj,n]
```

5.2.4. Definite Clause Grammars (DCG)

Conversia regulilor structurii propoziției (phrase structure – PS) în clauze Prolog este atât de simplă, încât se poate efectua în mod automat. Prologul include o facilitate pentru a realiza acest lucru, şi anume, o extindere a notației numită notație DCG (Definite-Clause Grammar). Aceasta reprezintă o notație specială pentru regulile unei gramatici. Exemple de clauze scrise prin **notația DCG** sunt:

```
s --> np, vp.
np --> adj, n.
n --> [elev].
```

Aceste clause vor fi automat convertite, în faza de consultare, la:

```
s(X,Z):-np(X,Y),vp(Y,Z).
np(X,Z) :-adj(X,Y), n(Y,Z).
n([elev|X],X).
```

Sistemul DCG reprezintă un compilator pentru regulile gramaticii pe care le traduce direct în clauze Prolog executabile.

Prin **gramatici DC** se înțelege acele gramatici ale căror reguli de rescriere sunt exprimate în notația DCG. O gramatică scrisă în DCG reprezintă un program de parsing pentru ea însăşi.

Un program Prolog poate conține atât reguli DCG, cât şi clauze Prolog obişnuite. Translatorul DCG afectează numai clauzele ce conțin functorul „`-->`". Toate celelalte clauze sunt presupuse a fi clauze Prolog obişnuite şi sunt lăsate neschimbate. Translatorul transformă regulile DCG în clauze Prolog prin adăugarea a două argumente suplimentare corespunzător fiecărui simbol care nu se află inclus între paranteze sau acolade. Argumentele sunt în mod automat aranjate astfel, încât să poată fi corect utilizate în procesul de analiză sintactică.

O **regulă DCG** are forma:

`Simbol neterminal --> extindere`

unde `extindere` constă în unul dintre următoarele elemente:

- un simbol neterminal (ex. `np`);

- o listă de simboluri terminale (ex. `[husband]` sau `[husband, drinks]`);
- un scop Prolog inclus între acolade cum ar fi {`write ("Gasit NP")`};
- un element vid reprezentat prin `[ ]`;
- o serie de oricare dintre aceste elemente, separate prin virgule.

În ce privește sintaxa DCG, e de notat că simbolurile neterminale nu se mai află între paranteze, în timp ce simbolurile terminale sunt incluse între paranteze drepte, ceea ce le transformă în liste Prolog. Simbolurile sunt separate prin virgule și fiecare regulă se încheie prin punct.

Fiecare regulă DCG este tradusă într-o clauză Prolog conform următoarei scheme:

- dacă regula DCG conține în membrul drept numai simboluri neterminale, este de forma:

  n->n1,n2,...,nn cu *n1,n2,...,nn* simboluri neterminale, atunci ea este tradusă în clauza Prolog

  ```
  n(X,Y):-n1(X,Y1), n2(Y1,Y2),...,
  nn(Yn-1,Y).
  ```

- dacă regula DCG conține în membrul drept atât simboluri neterminale, cât și terminale, are forma:

  ```
  n-->n1, [t2],n3, [t4].
  ```

  cu *n1,n3 – simboluri neterminale* și *t2,t4 simboluri terminale*, atunci ea este tradusă în clauza Prolog:

  ```
  n(X,Y):-n1(X, [t2|Y1]), n3(Y1, [t4|Y])
  ```

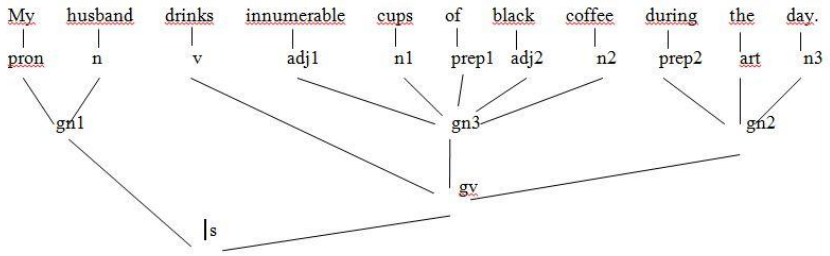


Figura 5.4. Analiza propoziției

Codul sursă:

```
s(s(GN1,GV))-->gn1(GN1),gv(GV).
gn1(gn1(PRON,N))-->pron(PRON),n(N).
gv(gv(V,GN3,GN2))-->v(V),gn3(GN3),gn2(GN2).
gn3(gn3(ADJ1,N1,PREP1,ADJ2,N2)-->
adj1(ADJ1),n1(N1),prep1(PREP1),adj2(ADJ2),n2(N2).
gn2(gn2(PREP2,ART,N3))-->
prep2(PREP2),art(ART),n3(N3).
pron(pron(my))-->[my].
n(n(husband))-->[husband].
v(v(drinks))-->[drinks].
adj1(adj1(innumerable))-->[innumerable].
n1(n1(cups))-->[cups].
prep1(prep1(of))-->[of].
adj2(adj2(black))-->[black].
n2(n2(coffee))-->[coffee].
prep2(prep2(during))-->[during].
art(art(the))-->[the].
n3(n3(day))-->[day].

?-phrase(s(X),[my, husband, drinks, innumerable, cups,
of, black, coffee, during, the, day]).

X = s(gn1(pron(my), n(husband)), gv(v(drinks),
gn3(adj1(innumerable), n1(cups), prep1(of),
adj2(black), n2(coffee)), gn2(prep2(during), art(the),
n3(day))))

Yes.
```

5.2.5. Recursive Transition Network

MODEL: Our students enjoy learning new things.

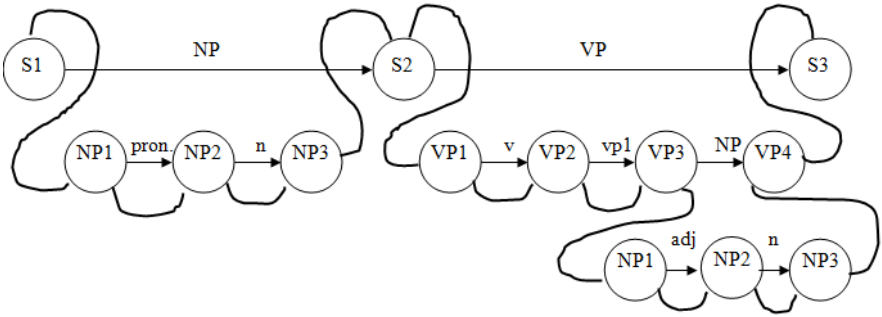


Figura 5.5. RTN

Tabelul 5.2. RTN soluție

String	State	Stack	Comment
Our students enjoy	S1	-	Push to NP-net
learning new things	NP1	S2	Recognize
Our students enjoy	NP2	S2	“pron”
learning new things	NP3	S2	Recognize
students enjoy learning	S2	-	“noun”
new things	VP1	S3	Pop to S-net
enjoy learning new	VP2	S3	Push to VP-net
things	VP3	S3	Recognize
enjoy learning new	NP1	VP4,S3	“verb”
things	NP2	VP4,S3	Recognize
enjoy learning new	NP3	VP4,S3	“verb part1”
things	VP4	S3	Push to NP-net
learning new things	S3	-	Recognize
new things			“adj”
new things			Recognize
things			“noun”
-			Pop to VP-net
-			Pop to S-net
-			SUCCESS!

Codul sursă pentru `pdr.pl`**:**

```
% Definirea predicatului ce seteaza configurarea
initiala
recognize(String):-
       initial(s,State),
       config(s:State,String,[]).

% Configurarea finala
config(s:State,[],[]):-
       final(s,State).

% If in final state for network pop back to previous
network
config(Network:State,String,[Network1:State1|Stack]):-
       final(Network,State),
       config(Network1:State1,String,Stack).

% Process next lexical item
config(Network:State,[Word|String],Stack):-
       word(Word,Cat),
       arc(Network,State,Cat,State1),
       config(Network:State1,String,Stack).

% If next arc label refers to a network push to it
config(Network:State,String,Stack):-
       arc(Network,State,Network1,State1),
       initial(Network1,State2),
       config(Network1:State2,String,[Network:State1|S
tack]).
```

Codul sursă pentru `rtn.pl`**:**

```
initial(s,1).
final(s,3).
arc(s,1,np,2).
arc(s,2,vp,3).

initial(np,1).
final(np,3).
arc(np,1,pron,2).
arc(np,2,n,3).
arc(np,1,adj,2).
```

```
initial(vp,1).
final(vp,4).
arc(vp,1,v,2).
arc(vp,2,vp1,3).
arc(vp,3,np,4).

initial(np,1).
final(np,3).
arc(np,1,adj,2).
arc(np,2,n,3).
```

Codul sursă pentru `lex.pl`**:**

```
word(our,pron).
word(students,n).
word(enjoy,v).
```

```
word(learning,vp1).
word(new,adj).
word(things,n).

?-consult('rtn.pl').
?-consult('lex.pl').
?-recognize([our,students,enjoy,learning,new,things]).

Yes.
```

PARTEA 2. Desfăşurarea lucrării

Să se elaboreze un program în limbajul Prolog, care ar analiza sintactic trei propoziții/fraze în limba română după exemplele prezentate mai sus (implementarea a cel puțin 2 metode).

Raportul trebuie să conțină:
1) formularea scopului şi obiectivelor de cercetare;
2) descrierea lexiconului şi gramaticii;
3) analiza propozițiilor prin metodele utilizate;
4) listing-ul programul cu comentarii şi explicații, justificări;
5) concluzii cu privire la experimentele efectuate.

LUCRAREA DE LABORATOR nr. 6

Algoritmi de recunoaştere

Scopul: Însuşirea şi consolidarea cunoştinţelor şi deprinderilor practice de lucru cu algoritmi simpli de recunoaştere a obiectelor după caracteristicile de calitate.

PARTEA 1. Noțiuni teoretice

În cele mai multe cazuri, imaginile sunt caracterizate cu ajutorul unor caracteristici cantitative: dimensiuni geometrice, greutate, arie, volum, şi altele.
În aceste cazuri schimbările cantitative ce caracterizează o anumită imagine, de obicei nu duc imediat la schimbarea de comportament. Atingând doar anumite limite pentru fiecare imagine, schimbările cantitative cauzează un salt cuantic pentru a trece la următoarea imagine.

Imaginile propriu-zise şi imaginile specifice pot fi caracterizate nu numai cantitativ, dar şi de caracteristicile calitative (proprietăți, semne, atribute).

Aceste semne nu pot fi descrise (sau de obicei nu au fost descrise) cantitativ cum ar fi culoarea, gustul, simțul, mirosul.

Imaginile pot avea sau nu avea unele caracteristici de calitate. Între caracteristicile calitative şi cantitative ale imaginilor există o diferență importantă, cu toate acestea, diferența în multe cazuri nu e necesar să fie evidențiată, deoarece pentru fiecare atribut calitativ există în anumite zone ale parametrilor cantitativi, limite care schimbă şi atributul de calitate.

De exemplu, o imagine cu o culoare specifică corespunde unei game de lungimi de undă specifică undelor electromagnetice, în limitele căreia se schimbă culoarea.

Există abordări diferite privind recunoaşterea caracteristicilor calitative ale imaginii. În acest laborator vom lua în considerare una dintre ele, bazându-ne pe codarea binară, prezența sau absența unei trăsături de calitate.

În cadrul acestei abordări analizăm o imagine concretă Xk.

Caracteristicile calitative pot fi reprezentate ca un vector binar:

$$X_k = (x_{k1}, x_{k2}, \ldots, x_{kn}),$$

unde: n - dimensiunea spațiului de caracteristică.

Dacă imaginea Xk conține j-m semne, atunci Xkj = 1, în caz contrar avem Xkj = 0. Aici este identificată imaginea şi vectorul binar descriptiv.

Luăm ca exemplu patru obiecte (vişină, portocală, măr, pepene galben), fiecare obiect având trei caracteristici: culoare, prezența sâmburilor sau semințelor (tabelul 6.1). În tabelul 6.2 sunt date valorile numerice ale atributelor pentru probă după codificarea lor binară.

Cea mai simplă metodă pentru rezolvarea problemelor de recunoaştere a obiectelor cu caracteristici de calitate, după codificarea binară a atributelor constă în reducerea problemei inițiale la rezolvarea problemei recunoaşterii imaginii cu caracteristicile cantitative ale spațiului vectorial n-dimensional.

Pentru aceasta e necesar ca la fiecare tip de atribut de calitate să fie introdusă o axă în spațiul vectorial n-dimensional. Dacă pentru analiza imaginii semnul există, atunci pe axă este indicată unitatea, dacă nu, atunci zero.

Tabelul 6.1. Prezența sâmburilor sau semințelor în imagini

	Vector	galben	oranj	roşu	sâmbure	semințe
Vişină	X1	nu	nu	da	da	nu
Portocală	X2	nu	da	nu	nu	da
Măr	X3	da	nu	da	nu	da
Pepene	X4	da	nu	nu	nu	da

Tabelul 6.2. Trăsăturile calitative ale imaginilor

	Vector semn	galben	oranj	roşu	sâmbure	semințe
Vişină	X1	x11 = 0	x12=0	x13= 1	x14= 1	x15=0
Portocală	X2	x21= 0	x22= 1	x23=0	x24=0	x25= 1
Măr	X3	x31= 1	x32=0	x33= 1	x34=0	x35= 1
Pepene	X4	x41= 1	x42=0	x43=0	x44 =0	x45= 1

Drept rezultat obținem un spațiu multidimensional caracteristic binar, unde poate fi calculată o gamă de distanțe, utilizată pentru recunoașterea obiectelor cu caracteristici cantitative.

În acest exemplu, ca urmare a caracteristicilor cantitative, sau mai bine spus a trăsăturilor calitative (tabelul 6.2), obținem un spațiu de cinci dimensiuni cu valori binare, unde putem utiliza distanța lui Euclid (1) şi Minkowski (2), distanțe ce folosesc suma diferențelor absolute dintre componentele respective ale vectorilor n-dimensionali (3):

$$L_1(S_i, X_j) = \sqrt{\sum_{k=1}^{n} (s_{ik} - x_{jk})^2}; \quad (1)$$

$$L_2(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} (s_{ik} - x_{jk})^\lambda}; \quad (2)$$

$$L_3(S_i, X_j) = \sum_{k=1}^{n} |s_{ik} - x_{jk}|, \quad (3)$$

unde:

$$L_p(S_i, X_j), \quad p = \overline{1,3} \quad (4)$$

corespunde distanței dintre imaginea de intrare:

$$S_i = (s_{i1}, ..., s_{in}) \quad (5)$$

cu imaginea model:

$$X_i = (x_{i1}, ..., x_{in}) \quad (6)$$

iar parametrul j reprezintă numărul modelului, iar λ – un rezultat pozitiv întreg, mai mare ca doi.

Distanțele (1) - (3) pot fi, de asemenea, utilizate cu coeficienți de greutate. Codificarea binară a caracteristicilor calitative poate fi aplicată şi la distanța Hamming, care este introdusă pentru toți vectorii binari. Distanța Hamming dintre doi vectori binari este numărul de componente distincte de vectori binari. În cazul în care vectorii au aceleaşi componente, distanța dintre ei este zero, în cazul în care vectorul nu are o componentă echivalentă, distanța este egala cu dimensiunea vectorilor.

O clasificare mai fină a obiectelor cu caracteristici calitative se obține prin introducerea compatibilității sau diferenței, pentru fiecare pereche de obiecte Xj, Xi pentru care sunt introduse caracteristici calitative binare de codare, utilizând tabelul 6.3.

Tabelul 6.3. Compatibilitatea şi diferența

Xj	Xi	
	1	0
1	a	h
0	g	b

Variabila *a* în tabelul 6.3 este proiectată pentru a contoriza numărul de caracteristici comune ale obiectelor Xj şi Xi. Ea poate fi calculată utilizând relația:

$$a = \sum_{k=1}^{n} x_{jk} x_{ik}; \qquad (7)$$

unde: xjk, xik sunt componente binare de vectori care descriu Xj obiecte, şi Xi obiecte.

Cu ajutorul variabilei *b* calculăm numărul de cazuri în care obiectele Xj, şi Xi nu au acelaşi semn:

$$b = \sum_{k=1}^{n} (1 - x_{jk}) x_{ik}. \qquad (8)$$

Variabilele *g* şi *h* sunt folosite, respectiv, pentru a calcula numărul de caracteristici care sunt atestate la obiectul Xi şi lipsesc în Xj şi sun atestate la obiectul Xj şi lipsesc din Xi:

$$g = \sum_{k=1}^{n} x_{ik} (1 - x_{jk}); \qquad (9)$$

$$h = \sum_{k=1}^{n} (1 - x_{ik}) x_{jk}. \qquad (10)$$

Din analiza variabilelor *a, b, g, h* rezultă următoarele: cu cât este mai mare asemănarea dintre obiectele Xj şi Xi, cu atât mai mare trebuie să fie variabila *a*. Măsura de apropiere a obiectelor sau a funcției de asemănare trebuie să fie o funcție crescătoare a lui *a*, funcția de asemănare trebuie să fie simetrică în raport cu variabilele *g* şi *h*.

Relativ la încheiere, variabila *b* nu poate fi utilizată, pentru că pe de o parte lipsa caracteristicilor similare în obiecte poate fi un indiciu al asemănării lor, pe de altă parte, în cazul în care obiectul nu are aceleaşi semne, înseamnă că aceste obiecte nu se pot referi la aceeaşi clasă.

De cele mai multe ori sunt utilizate în mod obişnuit funcţiile de similitudine reprezentate mai jos:

Funcţia de similitudine de Russell şi Rao:

$$S_1(X_i, X_j) = \frac{a}{a+b+g+h} = \frac{a}{n}; \quad (11)$$

Funcţia de similitudine Zhokara şi Nidmena:

$$S_2(X_i, X_j) = \frac{a}{n-b}; \quad (12)$$

Funcţia de similitudine Dice:

$$S_3(X_i, X_j) = \frac{a}{2a+g+h}; \quad (13)$$

Funcţia de similitudine de Snifa şi Sokal:

$$S_4(X_i, X_j) = \frac{a}{a+2(g+h)}; \quad (14)$$

Funcţia de similitudine de Sokal şi Michener:

$$S_5(X_i, X_j) = \frac{a+b}{n}; \quad (15)$$

Funcţia de similitudine Kulzhinskogo:

$$S_6(X_i, X_j) = \frac{a}{g+h}; \quad (16)$$

Funcţia de similitudine Yule:

$$S_7(X_i, X_j) = \frac{ab-gh}{ab+gh}; \quad (17)$$

PARTEA 2. Desfăşurarea lucrării

1. Se va proiecta un algoritm şi un program care simulează recunoaşterea diferitor obiecte cu caracteristici calitative şi a caracteristici de asemănare S1-S7.

2. Se va seta numărul de caracteristici calitative ale obiectelor n şi numărul de tone a imaginilor, având ca referinţă imaginile (gropile trebuie să fie de cel puţin 4). Se va adresa la câteva obiecte şi caracteristici de similitudine S1-S7, astfel indicând apartenenţa la un şablon anumit.

3. Se va oferi o funcţie unică de similitudini ale obiectelor cu caracteristici de calitate şi se va arăta performanţa de lucru, utilizând exemplele din punctul 2.

4. Se va compune o funcţie proprie în aşa fel, încât să fie similară cu exemplele de recunoaştere prezentate în S1-S7, totodată una din funcţii trebuie să ia o valoare minimă, iar alta - maximă.

5. Se vor sugera câteva exemple de recunoaştere folosind distanţa Hamming. Într-un exemplu distanţa Hamming ar trebui să ia o valoare egală cu numărul de pe listă din grupă, conform registrului.

6. Se va propune un exemplu de recunoaştere în cadrul căruia valoarea distanţei Hamming este egală cu una dintre funcţiile de similitudine S1-S7.

LUCRAREA DE LABORATOR nr. 7

Algoritmi de recunoaştere a imaginilor

Scopul: Obținerea cunoştințelor şi abilităților practice pentru lucrul cu cei mai simpli algoritmi de recunoaştere, care au la bază prezentarea imaginilor în formă de puncte sau de vectori *n*-dimensionali într-un spațiu vectorial

PARTEA 1. Noțiuni teoretice

Există un număr mare de forme de reprezentare ale imaginilor în dispozitivele de recunoaştere sau în aplicații. Una dintre cele mai simple şi pe înțelesul tuturor este forma, care foloseşte reprezentarea imaginilor ca fiind nişte puncte sau vectori într-un oarecare spațiu ***n***-dimensional.
Fiecare axă a acestui spațiu, în mod natural, corespunde cu una din ***n*** intrări sau cu unul din ***n*** receptori ai rețelei de recunoaştere. Fiecare receptor poate să se afle în una dintre ***m*** stări, dacă ele sunt discrete sau să posede o infinitate de stări în cazul când receptorii sunt continui.

În funcție de tipul receptorilor utilizați, se poate genera un spațiu vectorial *n*-dimensional continuu, discret sau continuu-discret.

În această lucrare de laborator vom examina un spațiu *n*-dimensional continuu.

Măsura asemănării imaginilor în spațiul vectorial *n*-dimensional se introduce ca funcție de două variabile $L(S_k, S_i)$, unde $S_k, S_i \in S$;

$S = \{S_1, S_2, ..., S_n\}$ – mulțimea finală de imagini în spațiul examinat.
În acelaşi timp, funcția $L(S_k, S_i)$ posedă următoarele caracteristici:

– este simetrică, adică $L(S_k, S_i) = L(S_i, S_k)$;
– valorile funcției sunt numere nenegative;
– măsura asemănării imaginii cu sine însuşi, ia o valoare extremă în comparație cu oricare altă imagine, adică, în funcție de modul introducerii măsurii de asemănare, se îndeplineşte una din următoarele două expresii:

$$L(S_k, S_k) = \max_i L(S_k, S_i) \qquad (1)$$

$$L(S_k, S_k) = \min_i L(S_k, S_i); \qquad (2)$$

– în cazul imaginilor compacte, funcția $L(S_k, S_i)$ este o funcție monotonă la funcția de îndepărtare a punctelor S_k şi S_i în spațiul ***n***-dimensional.

În spațiul *n*-dimensional, măsura de asemănare a imaginilor poate fi introdusă în multe moduri. Vom examina mai multe din ele. În fiecare mod se considera că imaginea etalon $X_1, X_2, .. , X_r$. **r** clase diferite de imagini în spațiul vectorial ***n***-dimensional, se introduc în formă de vectori cu proiecțiile la axele de coordonate: $X_1 = (x_{11}, x_{12}, \ldots, x_{1n}), X_2 = (x_{21}, x_{22}, \ldots, x_{2n}), \ldots, X_m = (x_{m1}, x_{m2}, \ldots, x_{mn})$. Orice imagine de intrare $S_i \in S$ la fel se reprezintă în formă de vector $S_i = (S_{i1}, S_{i2}, \ldots, S_{in})$ în această dimensiune.

7.1. Recunoaştere în funcție de unghiul dintre vectori

Măsura asemănării dintre doi vectori într-un spațiu *n*-dimensional poate fi reprezentată sub formă de unghi. Dacă este dată imaginea de intrare $S_i = (S_{i1}, S_{i2}, \ldots, S_{in})$ şi vectorii imaginilor - etalon $X_1=(x_{11}, x_{12}, \ldots, x_{1n}), X_2 = (x_{21}, x_{22}, \ldots, x_{2n}), \ldots, X_m = (x_{m1}, x_{m2}, \ldots, x_{mn})$, atunci măsura asemănării între imaginile de intrare şi cele etalon, se determină din relația

$$L(S_i, X_j) = \arccos\left(\frac{s_{i1}x_{j1} + s_{i2}x_{j2} + \ldots + s_{in}x_{jn}}{\sqrt{s_{i1}^2 + s_{i2}^2 + \ldots + s_{in}^2} \cdot \sqrt{x_{j1}^2 + x_{j2}^2 + \ldots + x_{jn}^2}}\right) = \arccos\left(\frac{\sum_{k=1}^{n} s_{ik}x_{jk}}{|S_i| \cdot |X_j|}\right), \qquad (3)$$

unde: $|S_i|$, $|X_j|$ - lungimile vectorilor S_i şi X_j.

Apartenența imaginii de intrare S_i la una dintre ***m*** imagini se determină din următoarea regulă decisivă:

$$S_i \in X_j, \quad \text{dacă} \quad L(S_i, X_j) = \min_j L(S_i, X_j), \qquad (4)$$

În acelaşi timp, în această regulă şi mai departe în text, pentru notarea imaginii **j** şi a imaginii - etalon se foloseşte aceeaşi notare $X_j (j = \overline{1,m})$.

7.2. Recunoaşterea imaginilor după produsul lor scalar

Măsura asemănării imaginilor după unghiul dintre (3) se bazează pe produsul scalar al vectorilor:

$$< S_j, X_j >= |S_i| \|X_j| \cos\alpha = \sum_{k=1}^{n} s_{ik} x_{jk} . \qquad (5)$$

Unele sisteme de recunoştere folosesc nemijlocit produsul scalar în calitate de măsură de asemănare a imaginilor într-un spaţiu *n*-dimensional vectorial:

$$L(S_i, X_j) = \sum_{k=1}^{n} s_{ik} x_{jk} . \qquad (6)$$

În acest caz apartenenţa imaginii de intrare S_i la o oarecare imagine - etalon, se determină din următoarea regulă decisivă:

$$S_i \in X_j, \quad dacă \quad L(S_i, X_j) = \max_j (L(S_i, X_j)) \qquad (7)$$

7.3. Recunoaşterea imaginilor după apartenenţa acestora la o zonă de spaţiu dată

La acest mod de recunoaştere, tot spaţiul de imagini **V** se împarte în zone care nu se intersectează $\mathbf{V_1, V_2, ..., V_m, V_{m+1}}$, unde $\mathbf{V_1, V_2, ..., V_m}$ - zone ce conţin imagini numai la o imagine virtuală corespunzătoare $\mathbf{X_1, X_2, ..., X_r}$; $\mathbf{V_{m+1}}$ - zonă care nu conţine imagini, ce se referă la imaginile virtuale date. În acest caz, apartenenţa imaginii de intrare $S_i = (S_{i1}, S_{i2}, ..., S_{in})$ la o oarecare imagine virtuală j ($j = \overline{1,m}$) se determină din regula decisivă:

$$S_i \in X_j, \quad dacă \quad (s_{i1}, s_{i2}, ..., s_{in}) \in V_j . \qquad (8)$$

Dacă zona V_j ($j = \overline{1,m}$) este reprezentată în spaţiul euclidian în formă de sfere cu centrul în punctele $(x^*_{j1}, x^*_{j2}, ..., x^*_{jn})$ şi cu razele R_j, atunci regula decisivă (8) va lua forma:

$$S_i \in X_j, \quad dacă \quad L(S_i, X_j) = \sqrt{\sum_{k=1}^{n} (x_{jk}^* - s_{ik})^2} \le R_j . \quad (9)$$

Pentru construirea zonelor în spațiul de imagini se pot folosi oricare din metodele de asemănare, ca de exemplu, dinstanțele cu coeficiente care posedă o capacitate (10)-(12), distanța după Camberr (13) ș.a.m.d.:

$$L(S_i, X_j) = \sqrt{\sum_{k=1}^{n} \eta_k (s_{ik} - x_{jk})^2}; \quad (10)$$

$$L(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} \eta_k (s_{ik} - x_{jk})^\lambda}; \quad (11)$$

$$L(S_i, X_j) = \sum_{k=1}^{n} \eta_k |s_{ik} - x_{jk}|; \quad (12)$$

$$L(S_i, X_j) = \sum_{k=1}^{n} \left| \frac{s_{ik} - x_{jk}}{s_{ik} + x_{jk}} \right|, \quad (13)$$

unde: $\eta_k (k = \overline{1,n})$ — coeficienții cu capacitate;

λ — număr întreg pozitiv, mai mare decât 2.

Regula decisivă (8) pentru distanțele (10)-(12) va lua forma:

$$S_i \in X_j, \quad dacă \quad L(S_i, X_j) = R_{ij} \le R_j , \quad (14)$$

unde: R_{ij} — distanța dată de una din relațiile (10)-(13), între imaginea prezentată S_i şi centrul sferei, care conține imaginea virtuală j;

R_i – raza sferei care conține imaginea virtuală j.

În cazul folosirii pentru recunoașterea unghiului dintre vectorii zoneleor ce nu se intersectează V_j ($j = \overline{1,m}$), se reprezintă în formă de conuri, iar regula decisivă va avea forma: $S_i \in X_j$, dacă

$$L(S_i, X_j) \Rightarrow \varphi_{ij} = \arccos\left(\frac{s_{i1}x_{j1} + s_{i2}x_{j2} + ... + s_{in}x_{jn}}{\sqrt{s_{i1}^2 + s_{i2}^2 + ... + s_{in}^2} \cdot \sqrt{x_{j1}^2 + x_{j2}^2 + ... + x_{jn}^2}}\right) \le \varphi_{j\max}, \quad (15)$$

unde: φ_{ij} - unghiul dintre imaginile prezentate *Si* şi imaginile etalon $\boldsymbol{X_j}$, $\varphi_{j\max}$ - unghiul maxim admisibil pentru imaginea virtuală ***j*** dintre etalon şi imaginile pentru recunoaştere.

PARTEA 2. Desfăşurarea lucrării

1. Se va elabora algoritmul şi programul care modelează recunoaşterea a câtorva obiecte diferite într-un spaţiu vectorial *n*-dimensional după unghiul dintre vectori şi după produsul lor scalar.

2. Se va stabili dimensiunea vectorului *n*-dimensional al spaţiului cu numărul **T** de obiecte etalon (***n*** şi ***m*** trebuie să fie nu mai mici decât 5) şi cu câteva obiecte pentru recunoaştere. Cu ajutorul unghiului dintre vectori şi produsul lor scalar se va determina apartenenţa obiectelor prezentate pentru testare la un oarecare obiect - etalon.

3. Se va elabora algoritmul şi programul care modelează recunoaşterea a câtorva obiecte diferite la apartenenţa lor la zonele sferice sau conice într-un spaţiu vectorial *n*-dimensional.

4. Se va stabili dimensiunea vectorului *n*-dimensional al spaţiului cu numărul **T** de obiecte - etalon şi cu câteva obiecte pentru recunoaştere. Prin intermediul zonelor sferice sau conice ce conţin obiectele - etalon determinaţi apartenenţa obiectelor prezentate pentru testare la un oarecare obiect - etalon.

LUCRAREA DE LABORATOR nr. 8

Rețelele neuronale Hamming

Scopul: Însuşirea şi consolidarea cunoştinţelor, precum şi formarea deprinderilor practice privind lucrul cu rețelele neuronale Hamming.

PARTEA 1. Noțiuni teoretice

Rețeaua Hamming este una dintre cele mai performante rețele neuronale pentru recunoaştere şi clasificare. În această rețea imaginile alb-negru sunt prezentate sub formă de vectori bipolari *m*-dimensionali. Denumirea rețelei provine de la distanța Hamming, care se foloseşte în rețea pentru măsurarea asemănării a **R** imagini de intrare şi a celor - etalon, păstrate cu ajutorul balanței de legături a rețelei. Măsura asemănării se determină din relația:

$$R = m\text{-}R_x, \qquad (1)$$

unde: **m** – componenta numerică a vectorilor de intrare şi etalon;

$\mathbf{R_x}$ – distanța Hamming dintre vectori.

Definiție:
Distanța Hamming dintre doi vectori binari este componenta numerică în care vectorii sunt diferiți.

Din definiție rezultă că măsura de asemănare a imaginilor (1) poate fi redată prin numărul *a*, care este componenta vectorilor binari în care ei coincid: $\mathbf{R = a}$.

Vom deduce pentru vectorii bipolari $\mathbf{S = (s_1 ,...,s_m)}$ и $\mathbf{Z = (z_1 ,...,z_m)}$ produsul lor scalar prin numărul de componente care coincid şi de componentele care sunt diferite:

$$SZ = \sum_{i=1}^{m} S_i Z_i = a - d, \qquad (2)$$

unde: ***a*** – numărul de componente identice ale vectorului;

d – numărul de componente diferite ale vectorilor **S şi Z.**

Deoarece ***m*** – dimensiunea vectorilor, atunci $m = a + d$ şi rezultă că produsul scalar (2) poate fi scris în forma:

$$SZ = 2a - m.$$

De aici nu este greu de determinat:

$$a = \frac{m}{2} + \frac{SZ}{2} = \frac{m}{2} + 1/2 \sum_{i=1}^{m} S_i Z_i . \qquad (3)$$

Partea dreaptă a expresiei (3) poate fi privită ca semnalul de intrare a neuronului, care are *m* sinapse, cu coeficientul de greutate $z_i/2$ ***(i = 1,m)*** şi cu deplasamentul ***m*/2**. Sinapsele neuronului recunosc m componente ale vectorului de intrare $S = (s_1, .., s_m)$. Această interpretare a părții drepte a expresiei (3) reprezintă arhitectura unei subrețele neuronale, care este redată de partea de jos a figurii 8.1. În unele surse rețeaua din figura 8.1. este numită *rețea Hamming*, altele numesc rețea Hamming doar partea de jos a figurii, considerând că rețeaua este formată din două subrețele - Hamming şi MaxNet. Noi vom considera că este o rețea Hamming.

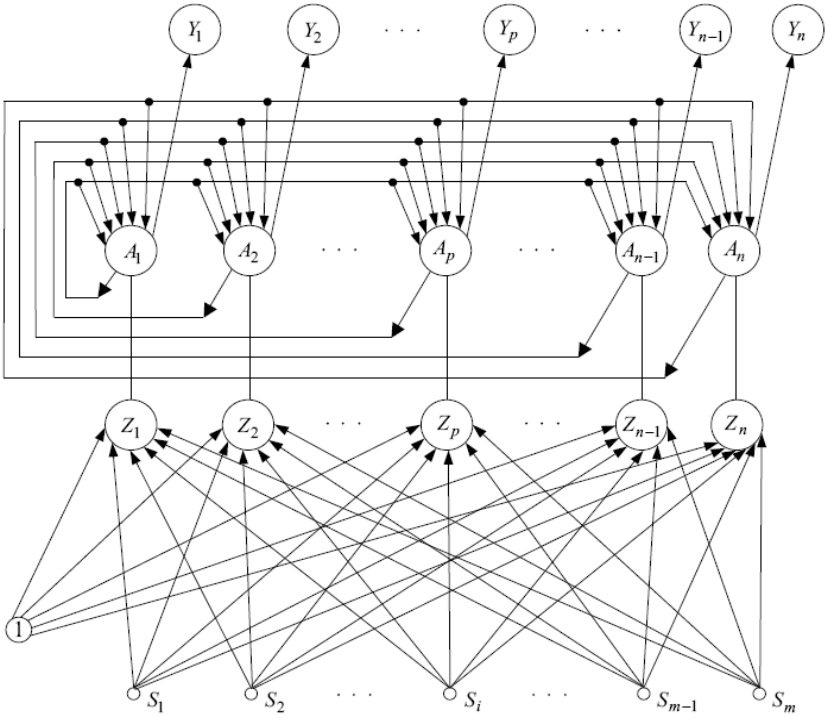


Figura 8.1. Exemplu de Rețea Hamming

Rețeaua Hamming are *m* neuroni de intrare $\mathbf{S_1}$, **…,** $\mathbf{S_m}$, care recunosc componentele bipolare $\mathbf{s_1^q}$**, …,** $\mathbf{s_m^q}$ ale imaginilor de intrare $\mathbf{S^q}$ **(q=1,L).** Semnalele de ieşire ale elementelor S se determină din relația:

$$U_{out} = S_i = \begin{cases} +1, \text{ dacă } S_i^q = 1 \\ -1, \text{ dacă } S_i^q = -1 \end{cases} \quad (4)$$

unde: $\boldsymbol{U_{out}}$ **–>** $\boldsymbol{U}$ ***ieşire;***

$\boldsymbol{U_{in}}$ **->** $\boldsymbol{U}$ ***intrare .***

Rezultă că semnalul de ieşire a elementului S repetă semnalul lui de intrare:

$$\boldsymbol{U_{out}\, S_i = U_{in}\, S_i = s_i^q}\ .$$

Fiecare neuron $\mathbf{S_j}$ **(j=1, m)** este legat cu intrarea fiecărui element $\mathbf{Z_k}$ **(k=1, n).** Capacitatea acestor legături $\mathbf{w_{1k}, \dots, w_{mk}}$ conține informația despre imaginea - etalon k:

$$Vk = (v1k, \dots, vmk): \quad \mathbf{w1k} = v1k/2, \dots, wmk = vmk/2. \tag{5}$$

Funcția de activare a Z-elemente este descrisă de relația:

$$g_z(U_{in}) = \begin{cases} 0, \text{ dacă } U_{in} \le 0 \\ k_1 U_{in}, \text{ dacă } 0 < U_{in} \le U_n \\ U_n, \text{ dacă } U_{in} > U_n \end{cases} \tag{6}$$

unde: $\boldsymbol{U_{in}}$ – semnalul de intrare a neuronului; $\boldsymbol{k_1}$ şi $\boldsymbol{U_n}$ – constante.

La descrierea imaginii $\mathbf{S^* = (s_1^*, \dots, s_m^*)}$ fiecare Z-neuron calculează semnalul lui de intrare în conformitate cu expresia (3):

$$U_{in} z_k = \frac{m}{2} + \sum_{i=1}^{m} w_{ik} s_i^* \tag{7}$$

şi cu ajutorul funcției de activare determină semnalul de ieşire $\mathbf{U_{inZk}}$. Semnalele de ieşire $\mathbf{U_{inZ1}, \dots, U_{inZn}}$ a Z-elemente sunt semnale de intrare $\mathbf{a_1, \dots, a_n}$ a subrețelei de sus, care este o rețea MaxNet. Funcția de activare a neuronilor $\mathbf{A_p}$ **(p=1,n)** şi capacitatea lor de legătură sunt date de relația:

$$g(U_{in}) = \begin{cases} U_{in}, \text{ dacă } U_{in} > 0 \\ 0, \text{ dacă } U_{in} \le 0 \end{cases}$$

$$w_{ij} = \begin{cases} 1, \text{ dacă } i = j \\ -\varepsilon, \text{ dacă } i \ne j, i, j = \overline{1,n} \end{cases}$$

unde: ε – constanta care satisface inegalitatea $\mathbf{0 < \varepsilon \le 1/n}$.

Rețeaua funcționează ciclic, iar dinamica neuronilor este redată de relația iterativă:

$$U_i(+1) = g\left(U_i(t) - \varepsilon \sum_{j=1,\, j\neq i}^{n} U_j(t)\right), i = \overline{1,n} \qquad (8)$$

cu condițiile inițiale:

$$U_i(0)=a_i=U_{inZi},\ i=1,n.$$

Dacă printre semnalele de intrare $\mathbf{a_1}$ **,...,** $\mathbf{a_n}$ alr neuronilor $\mathbf{A_1}$**, ...,** $\mathbf{A_n}$ este un semnal cel mai înalt $\mathbf{a_p}$ **(p** $\in$**{1, 2, ..., n})**, atunci în urma procesului iterativ, în subrețeaua MaxNet va figura un singur neuron $\mathbf{A_p}$, care va rămâne ca semnal de ieşire, mai mare ca **0**, adică va deveni *"învingător"*, deşi semnalele de ieşire $\mathbf{U_1}$**, ...,** $\mathbf{U_p}$**, ...,** $\mathbf{U_n}$ ale **A**-elemente vin la intrările a **Y**-neuroni care au funcția de activare de forma:

$$g_Y(U_{in}) = \begin{cases} 1, \textit{ dacă } U_{in} > 0 \\ 0, \textit{ dacă } U_{in} \le 0 \end{cases} \qquad (9)$$

Atunci la ieşirea rețelei Hamming doar un singur neuron $\mathbf{Y_p}$ va avea un singur semnal de ieşire. Ieşirea unică a acestui neuron şi ieşirile nule ale celorlalți neuroni vor indica că imaginea redată $\mathbf{S^*=(s_1^*, \ldots, s_m^*)}$ este cea mai aproape, în sensul măsurii de asemănare date (1) de imaginea- etalon $\mathbf{V^p=(v_1^p, \ldots, v_m^p)}$**.**

Un avantaj esențial al rețelei Hamming constă în faptul că nu necesită proceduri complicate de calcul pentru învățarea sa. Un neajuns esențial al rețelei constă în faptul că aceasta nu produce două sau mai multe imagini de referință, având acelaşi nivel de asemănare cu imaginile impuse.

Exemple:

Să se elaboreze o rețea Hamming, având în calitate de etalon 5 imagini alb-negru $\mathbf{V^1}$**, ...,** $\mathbf{V^5}$ arătate în figura 8.2. Să se determine influența imaginilor prezentate în figura 8.3.

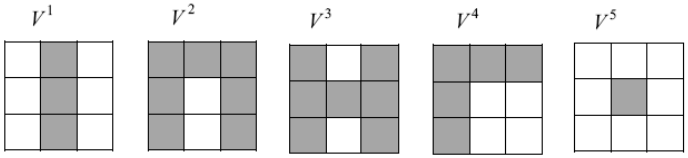


Figura 8.2. Imagini-etalon

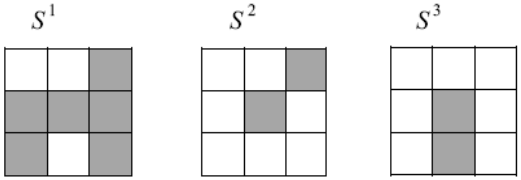


Figura 8.3. Imagini pentru testare

1	2	3
4	5	6
7	8	9

Figura 8.4. Numerotarea elementelor-imaginii

Întrucît avem doar 5 imagini-etalon, rețeaua va trebui să aibă cîte 5 **Z-, A-, Y**-neuroni. Prezența a 9 elemente alb-negru în imaginile din figura 8.2 şi figura 8.3 determină 9 **S**-neuroni, care recunosc elementele imaginilor de intrare.

Vom numerota elementele imaginilor din Figura 8.2. şi Figura 8.3. în conformitate cu Figura 8.4. şi vom prezenta imaginile $\mathbf{V^p}$ **(p = 1,s)** în formă vectorială, folosind reprezentarea bipolară a vectorilor:

$V^1 = (-1,1, -1,-1, 1,-1, -1,1,-1);$

$V^2=(1, 1,1, 1,-1, 1,1,-1,1);$

$V^3=(1,-1,1, 1,1, 1,1,-1,1);$

$V^4=(1, 1,1, 1,-1,-1, 1,-1,-1);$

$V^5=(-1, -1,-1, -1,1,-1,-1,-1,-1).$

Cunoscând vectorii imaginilor de intrare şi numărul lor în raport cu (5), determinăm matricea $|\mathbf{W_{ik}}|$ **(i = 1,9, k = 1,5)** de capacităţi a legăturilor părţii de jos a subreţelei reţelei Hamming:

$$|W_{ik}| = \begin{array}{c|ccccc} & V^1(Z_1) & V^2(Z_2) & V^3(Z_3) & V^4(Z_4) & V^5(Z_5) \\ S_1 & -0.5 & 0.5 & 0.5 & 0.5 & -0.5 \\ S_2 & 0.5 & 0.5 & -0.5 & 0.5 & -0.5 \\ S_3 & -0.5 & 0.5 & 0.5 & 0.5 & -0.5 \\ S_4 & -0.5 & 0.5 & 0.5 & 0.5 & -0.5 \\ S_5 & 0.5 & -0.5 & 0.5 & -0.5 & 0.5 \\ S_6 & -0.5 & 0.5 & 0.5 & -0.5 & -0.5 \\ S_7 & -0.5 & 0.5 & 0.5 & 0.5 & -0.5 \\ S_8 & 0.5 & -0.5 & -0.5 & -0.5 & -0.5 \\ S_9 & -0.5 & 0.5 & 0.5 & -0.5 & -0.5 \end{array} \qquad (10)$$

unde pentru claritate rândurile şi coloanele matricei sunt numerotate corespunzător cu ajutorul elementelor **S** şi a imaginilor-etalon $\mathbf{V^P}$ sau neuronilor $\mathbf{Z_p}$, luaţi în paranteze.

Deplasările $\mathbf{b_1,..., b_5}$ neuronilor **Z** se calculează cu ajutorul expresiei (11):

$$b_1 = b_2 = \ldots = b_5 = m/2 = 9/2 = 4{,}5. \qquad (11)$$

Funcţia de activare a neuronilor **Z** este redată de expresia (6) cu $\mathbf{k_1 = 0{,}1}$ şi $\mathbf{U_п = 1/k_1 = 1/0.1 = 10.}$ Funcţiile de activare a neuronilor **Y** le determinăm ca funcţii (9). Constanta ε determină capacitatea inversă a legăturilor în subreţeaua MaxNet. Vom determina egalităţile $\boldsymbol{\varepsilon = 1/n,}$ unde $\mathbf{п = 5}$ şi $\boldsymbol{\varepsilon = 0{,}2.}$

Cunoscând toţi parametrii reţelei Hamming, vom urmări funcţionarea ei la prezentarea imaginii $\mathbf{S^1 = (-1, -1, 1, 1, 1, 1, 1,-1, 1)}$ (figura 8.3).

După prezentarea imaginii $\mathbf{S^1}$ la ieşirea neuronilor **S**, cu toate semnalele lor de ieşire ce le repetă pe cele de intrare [relația (4)], vom obține vectorul de semnale $\mathbf{S_{out\ s} = S^1}$. Folosind semnalele de ieşire a elementelor semnalelor **S**, fiecare neuron **S** calculează semnalul propriu de intrare conform relației (7), matricea capacităților (10) şi deplasamentul $\mathbf{b_k, k=1,5}$ (11):

$$U_{in.Z1} = \frac{9}{2} + \sum_{i=1}^{9} w_{i1} S_i^1 = 4{,}5 + (-0{,}5)\cdot(-1) + 0{,}5\cdot(-1) + (-0{,}5)\cdot 1 + (-0{,}5)\cdot 1 + $$
$$+ 0{,}5\cdot 1 + (-0{,}5\cdot 1) + (-0{,}5)\cdot 1 + (+0{,}5)\cdot(-1) + (-0{,}5)\cdot 1 = 2;$$

$$U_{in.Z2} = \frac{9}{2} + \sum_{i=1}^{9} w_{i2} S_i^1 = 4{,}5 + 0{,}5\cdot(-1) + 0{,}5\cdot(-1) + 0{,}5\cdot 1 + 0{,}5\cdot 1 + $$
$$+ (-0{,}5)\cdot 1 + 0{,}5\cdot 1 + 0{,}5\cdot 1 + (-0{,}5)\cdot(-1) + 0{,}5\cdot 1 = 6;$$

$$U_{in.Z3} = \frac{9}{2} + \sum_{i=1}^{9} w_{i3} S_i^1 = 4{,}5 + 0{,}5\cdot(-1) + (-0{,}5)\cdot(-1) + 0{,}5\cdot 1 + 0{,}5\cdot 1 + $$
$$+ 0{,}5\cdot 1 + 0{,}5\cdot 1 + (-0{,}5)\cdot(-1) + 0{,}5\cdot 1 = 8;$$

$$U_{in.Z4} = \frac{9}{2} + \sum_{i=1}^{9} w_{i4} S_i^1 = 4{,}5 + 0{,}5\cdot(-1) + 0{,}5\cdot(-1) + 0{,}5\cdot 1 + 0{,}5\cdot 1 + $$
$$+ (-0{,}5\cdot 1) + (-0{,}5)\cdot 1 + 0{,}5\cdot 1 + (-0{,}5)\cdot(-1) + (-0{,}5)\cdot 1 = 4;$$

$$U_{in.Z5} = \frac{9}{2} + \sum_{i=1}^{9} w_{i5} S_i^1 = 4{,}5 + (-0{,}5)\cdot(-1) + (-0{,}5)\cdot(-1) + (-0{,}5)\cdot 1 + $$
$$+ (-0{,}5)\cdot 1 + 0{,}5\cdot 1 + (-0{,}5)\cdot 1 + (-0{,}5)\cdot 1 + (-0{,}5)\cdot(-1) + (-0{,}5)\cdot 1 = 4.$$

După semnalul de intrare $\mathbf{U_{in\ Zk}}$, folosind funcția proprie de activare (6) cu $\mathbf{k_1 = 0{,}1}$ şi $\mathbf{U_n = 10}$, fiecare neuron Z determină semnalul său de ieşire:

$$U_{out.Z1} = k_1 U_{in.Z1} = 0{,}1\cdot 2 = 0{,}2;$$
$$U_{out.Z2} = k_1 U_{in.Z2} = 0{,}1\cdot 6 = 0{,}6;$$
$$U_{out.Z3} = k_1 U_{in.Z3} = 0{,}1\cdot 8 = 0{,}8;$$
$$U_{out.Z4} = k_1 U_{in.Z4} = 0{,}1\cdot 4 = 0{,}4;$$
$$U_{out.Z5} = k_1 U_{in.Z5} = 0{,}1\cdot 4 = 0{,}4.$$

Vectorul: $U_{out\,Z} = (0{,}2;\ 0{,}6;\ 0{,}8;\ 0{,}4;\ 0{,}4)$

reprezintă vectorul de intrarea a subrețelei MaxNet şi începe procesul iterativ de alocare a semnalului maxim de ieşire cu condițiile inițiale (12). Pentru **t =1** vom avea:

$$U_{out.A1}(1) = g(U_{outA1}(0) - \varepsilon\sum_{k=2}^{5} U_{outAk}(0)) = g(0{,}2 - 0{,}2(0{,}6 + 0{,}8 + 0{,}4 + 0{,}4)) = g(-0{,}24) = 0;$$

$$U_{out.A2}(1) = g(U_{outA2}(0) - \varepsilon\sum_{k=1,\,k\neq 2}^{5} U_{outAk}(0)) = g(0{,}6 - 0{,}2(0{,}2 + 0{,}8 + 0{,}4 + 0{,}4)) = g(0{,}24) = 0{,}24;$$

$$U_{out.A3}(1) = g(U_{outA3}(0) - \varepsilon\sum_{k=1,\,k\neq 3}^{5} U_{outAk}(0)) = g(0{,}8 - 0{,}2(0{,}2 + 0{,}6 + 0{,}4 + 0{,}4)) = g(0{,}48) = 0{,}48;$$

$$U_{out.A4}(1) = g(U_{outA4}(0) - \varepsilon\sum_{k=1,\,k\neq 4}^{5} U_{outAk}(0)) = g(0{,}4 - 0{,}2(0{,}2 + 0{,}6 + 0{,}8 + 0{,}4)) = g(0) = 0;$$

$$U_{out.A5}(1) = g(U_{outA5}(0) - \varepsilon\sum_{k=1}^{4} U_{outAk}(0)) = g(0{,}4 - 0{,}2(0{,}2 + 0{,}6 + 0{,}8 + 0{,}4)) = g(0) = 0;$$

Folosind vectorul ($\mathbf{U_{out\ A1}(1)}$**, ...,** $\mathbf{U_{out\ A5}}$ **(1))** al semnalelor de ieşire a elementelor A cu **t = 1,** analogic se determină semnalele de ieşire a neuronilor A cu **t = 2, 3 şi 4**. Rezultatele calculelor sunt redate în tabelul 8.1.

Procesul iterativ în subrețeaua MaxNet se termină când **t =5.** În cadrul acestui pas funcționalitatea subrețelei nu modifică nici un semnal de ieşire a elementelor **A**. Vectorul semnalelor de ieşire a elementelor **A**, înscris în ultimul rând din tabelul 8.1. vine la intrarea elementelor **Y**. Neuronii **Y,** având funcția de activare (9), la ieşirea unui singur element $\mathbf{Y_3}$ va condiționa apariția semnalului unitate.

Apariția acestui semnal ne vorbeşte despre faptul că, imaginea redată $\mathbf{S^1}$ cel mai mult se aseamănă cu imaginea-etalon $\mathbf{V^3}$. Comparația vizuală a figurii 8.2 şi figurii 8.3 confirmă funcționarea corectă a rețelei.

Tabelul 8.1. Rezultatele calculelor procesului iterativ în subrețeaua MaxNet

Timpul	**Dimensiunea semnalelor de ieşire a neuronilor A_k (k=1,5)**				
	$U_{out\ A1}$	$U_{out\ A2}$	$U_{out\ A3}$	$U_{out\ A4}$	$U_{out\ A5}$
0	0,200	0,600	0,800	0,400	0,400
1	0,000	0,240	0,480	0,000	0,000
2	0,000	0,144	0,432	0,000	0,000
3	0,000	0,058	0,403	0,000	0,000
4	0,000	0,000	0,402	0,000	0,000

Acum vom determina funcționarea rețelei la redarea imaginii $\mathbf{S^2}$ **= (-1, -1, 1,-1, 1,-1,-1,-1, -1).** Calculele sunt identice, de aceea vom prezenta doar cele mai importante rezultate intermediare :

$$U_{in.Z1}(S^2) = 6,\ U_{in.Z2}(S^2) = 2, U_{in.Z3}(S^2) = 4,$$

$$U_{in.Z4}(S^2) = 4, U_{in.Z5}(S^2) = 8;$$

$$U_{out.Z1}(S^2) = a_1 = 0{,}6, U_{out.Z2}(S^2) = a_2 = 0{,}2,$$

$$U_{out.Z3}(S^2) = U_{out.Z4}(S^2) = a_3 = a_4 = 0{,}4,$$

$$U_{out.Z5}(S^2) = a_5 = 0{,}8.$$

Deoarece vectorul de intrare *(0,6; 0,2; 0,4; 0,4; 0,8)* a subrețelei MaxNet conține doar un singur element maxim $\mathbf{a_5 = 0{,}8}$, atunci în urma procesului iterativ, numai la ieşirea elementului $\mathbf{A_5}$ vom avea un semnal pozitiv, care va provoca un semnal unitate la ieşirea neuronului $\mathbf{Y_5}$. Rezultă că imaginea redată se aseamănă foarte mult cu imaginea-etalon $\mathbf{V^5}$, ceea ce confirmă comparația vizuală a figurii 8.2şi figurii 8.3.

Vom determina reacția rețelei Hamming la imaginea de intrare $\mathbf{S^3}$**= (-1,-1,-1,-1,1,-1,-1,1,-1)** figura 8.3. La prezentarea imaginii $\mathbf{S^3}$ vom obține:

$$U_{in.Z1}(S^3) = 8,\ U_{in.Z2}(S^3) = 0, U_{in.Z3}(S^3) = 1,$$
$$U_{in.Z4}(S^3) = 2, U_{in.Z5}(S^3) = 8.$$

Deoarece semnalele $\mathbf{U_{in\ Z1}(S^3) = U_{in\ Z5}(S^3)}$ sunt maximal identice cu semnalele de intrare, atunci vor fi identice şi semnalele maxime de la ieşirea elementelor **Z** $\mathbf{(U_{out\ Z1}(S^3) = U_{out\ Z5}(S^3) = 0,8)}$ şi la intrarea neuronilor **A** $\mathbf{(a_1\ (S^3) = a_5(S^3) = 0,8).}$ În consecință, subrețeaua MaxNet nu va putea crea un semnal maximal unic şi în urma funcționării la ieşirile neuronilor **A, Y** vom avea semnale nule.

Rezultă că rețeaua Hamming nu va putea determina pentru fiecare imagine-etalon imaginea redată de vectorul S^3.

PARTEA 2. Desfăşurarea lucrării

1. Elaborați o rețea neuronală Hamming, care va putea identifica nu mai puțin de 6 caractere diferite ale numelui şi prenumelui d-voastră, în același timp motivați alegerea:
   - numărului de receptori ai neuronilor;
   - numărului de neuroni ai stratului de intrare;
   - dimensiunea parametrului ε în subrețeaua MaxNet;
   - tipul funcțiilor de activare a neuronilor fiecărui strat;
   - dimensiunea capacităților (ponderilor) de legătură şi de deplasare în subrețeaua Hamming.

2. Studiați rețeaua neuronală a imaginilor-etalon a caracterelor. Examinați posibilitățile rețelei pentru determinarea imaginilor distorsionate (eronate).

3. Alegeți imaginea de intrare egal îndepărtată pe distanța Hamming de la două imagini-etalon. Observați efectul analizei acestei imagini. Propuneți o metodă de comportare mai informativă a rețelei la prezentarea unor astfel de imagini.

LUCRAREA DE LABORATOR nr. 9

Rețelele neuronale Hebb

Scopul: Însuşirea şi consolidarea cunoştinţelor, precum şi formarea deprinderilor practice privind lucrul cu rețelele neuronale Hebb.

PARTEA 1. Noțiuni teoretice

9.1. Neuroni formali ai rețelelor neuronale artificiale

În timpul modelării rețelelor neuronale, în calitate de neuroni artificiali se foloseşte un simplu element procesor, ilustrat în figura 9.1.

La intrările lui vine vectorul $\mathbf{X=(x_1, ..., x_n)}$ de semnale de intrare, care la rândul lor sunt semnale de ieşire al altor neuroni. Aceasta mai conține şi un semnal unic de deplasare.

Toate semnalele de intrare, inclusiv ale semnalul de deplasare, se înmulțesc la coeficienții ponderilor de legătură şi se sumează:

$$S = \sum_{i=1}^{n} x_i w_i + w_0, \qquad (1)$$

unde:**S –** suma semnalului de intrare**;** $\mathbf{w_i\ (i = 1..n)}$ – coeficienții ponderilor legăturilor semnalelor de intrare $\mathbf{x_1, ..., x_n}$; $\mathbf{w_0}$ – coeficientul ponderii de legătură al semnalului de deplasare.

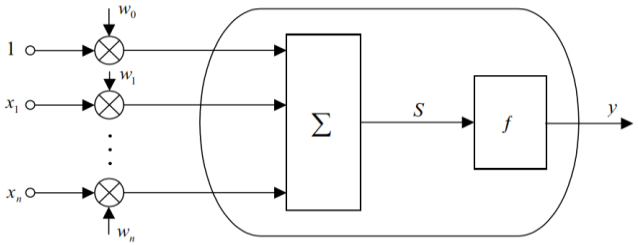


Figura 9.1. Element procesor folosit în rețelele neuronale simple

Semnalul recepționat **S** vine la intrarea blocului care realizează funcția ***f*** de activare a neuronului. Funcțiile tipice de activare sunt binare:

$$y = \begin{cases} 1, dacă\ S > 0 \\ 0, dacă\ S \le 0 \end{cases}. \quad (2)$$

sau bipolare:

$$y = \begin{cases} 1, dacă\ S > 0 \\ -1, dacă\ S \le 0 \end{cases}. \quad (3)$$

Mulți autori, la descrierea modelului neuronului, nu folosesc nu semnalul de deplasare, dar pragul θ al neuronului, ceea ce conduce la modelul elementului echivalent. În acest caz, expresiile (2) şi (3) vor avea forma:

$$y = \begin{cases} 1, dacă\ S > \theta \\ 0, dacă\ S \le \theta \end{cases}; \quad (4)$$

$$y = \begin{cases} 1, dacă\ S > \theta \\ -1, dacă\ S \le \theta \end{cases}, \quad (5)$$

unde:

$$S = \sum^{n} w_i x_i . \quad (6)$$

Imaginea grafică a funcției de activare binare şi bipolare pentru acest caz este arătată în figura 9.2 *a* şi *b*.

Din relațiile (1)-(3) şi (4)-(6) rezultă că pentru fiecare valoare de prag **θ** a neuronului poate fi prezentat în conformitate cu capacitatea coeficientului w_0 de legătură a semnalului de deplasare, şi invers.

Însă mai rar se utilizează funcții liniare binare şi liniare bipolare de activare (figura 9.2 *c* şi *d*):

$$y = \begin{cases} -a\ \ pentru S < \theta_1, \\ kS + a_0\ \ pentru \theta_1 \le S \le \theta_2 \\ 1\, pentru S > \theta_2 \end{cases} \quad (7)$$

unde ***a*** este egal cu 0 pentru semnalele de intrare binare ale neuronilor iar ***a*** este egal cu -1 pentru semnalele bipolare; ***k, a_0*** – coeficienți permanenți.

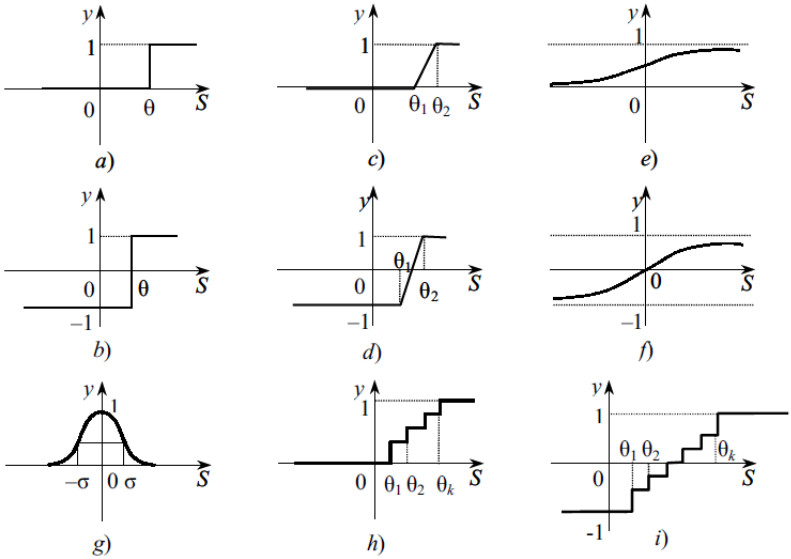


Figura 9.2. Funcția de activare a neuronilor

În afară de cele expuse mai sus, în teoria rețelelor neuronale se folosesc şi următoarele funcții neliniare de activare:
Sigmoida binară sau sigmoida logică (figura 9.2, *e*):

$$y = \frac{1}{1+e^{-\tau S}}, \tag{8}$$

unde: **τ** – coeficient permanent.

Sigmoida bipolară (figura 9.2 *f*):

$$y = \frac{2}{1+e^{-\tau S}} - 1. \tag{9}$$

Radial – simetrică (figura 9.2 *g*):

$$y = e^{-\frac{S^2}{\tau^2}}. \qquad (10)$$

Binară de rangul k (figura 9.2 *h*):

$$y = \begin{cases} 0 \ \ pentru S < \theta_{1,} \\ 1/(K-1) \ pentru \theta_1 \le S \le \theta_2, \\ 2/(K-1) \ pentru \theta_2 \le S < \theta_3, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \dfrac{K-2}{K-1} \ pentru \theta_{k-1} \le S < \theta_k, \\ 1 \ \ pentru S \ge \theta_k. \end{cases} \qquad (11)$$

*Bipolară de rangul k (*figura 9.2 *i):*

$$y = \begin{cases} -1 \ \ pentru S < \theta_{1,} \\ -1 + 2/(K-1) \ pentru \theta_1 \le S \le \theta_2, \\ -1 + 4/(K-1) \ pentru \theta_2 \le S < \theta_3, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ -1 + \dfrac{2(K-2)}{K-1} \ pentru \theta_{k-1} \le S < \theta_k, \\ 1 \ \ pentru S \ge \theta_k. \end{cases} \qquad (12)$$

Modelele neuronilor artificiali prezentate anterior ignorează multe proprietăți cunoscute ale prototipurilor biologice. De exemplu, nu iau în considerație reținerile în timp ale neuronilor, efectele modulării în frecvență, excitării locale și consecințele legate de ele a sumării spațiale și temporale de prag, atunci când celula nu se excită imediat după venirea impulsurilor, dar cu secvențe de semnale de excitare care vin în intervale scurte de timp. De asemenea nu se iau în considerație perioadele absolute de refractare în timpul cărora celulele nervoase nu pot fi excitate, adică ca și cum ar avea o perioadă de excitare infinită, care apoi timp de câteva milisecunde după trecerea semnalului se micșorează la nivelul normal. Această listă de deosebiri, pe care mulți biologi le socot decisive, poate fi

uşor continuată, numai că reţelele neuronale artificiale totuşi depistează un şir de proprietăţi interesante, caracteristice pentru prototipurile biologice.

9.2. Rezolvarea problemelor de recunoaştere în baza neuronilor independenţi. Regula lui Hebb

Reţelele neuronale artificiale destinate pentru soluţionarea diferitor probleme, pot conţine de la câţiva neuroni până la mii şi milioane de elemente. Însă un neuron aparte (figura 9.1) cu funcţia de activare bipolară sau binară, poate fi folosit pentru rezolvarea problemelor simple de recunoaştere şi clasificare a imaginilor. Alegerea reprezentării bipolare (1, -1) sau binare (1, 0) a semnalelor în reţele neuronale se face reieşind din problema ce se rezolvă şi în multe cazuri este echivalentă.Totodată avem un spectru de probleme în care codificarea binară a semnalelor este mai comodă, însă este mai util de ales o prezentare bipolară a informaţiei.

Întrucât semnalul de ieşire a neuronului binar (figura 9.1) ia numai două valori, atunci acest neuron poate fi folosit pentru clasificarea imaginilor prezentate în două clase.

Fie că avem o mulţime M de imagini pentru care sunt cunoscute clasificarea corectă în două clase $X^1 = \{X^{11}, X^{12}, \ldots, X^{1q}\}$, $X^2 = \{X^{21}, X^{22}, \ldots, X^{2p}\}$, $X^1 \cup X^2 = M$, $X^1 \cap X^2 = \varnothing$, şi fie că primei clase X^1 îi corespunde semnalul de ieşire $y=1$, iar clasei X^2 semnalul $y = -1$. Dacă, de exemplu este prezentată o imagine oarecare $X^\alpha = (x_1^\alpha, \ldots, x_n^\alpha)$, $X^\alpha \in M$ şi suma capacităţilor semnalelor de intrare depăşeşte valoare zero:

$$S = \sum_{i=1}^{n} x_i^\alpha w_i + w_0 > 0,$$

atunci semnalul de ieşire $y=1$ şi, respectiv, imaginea de intrare X^α corespunde clasei X^1. Dacă $S \le 0$, atunci $y = -1$ şi imaginea prezentată aparţine clasei a doua.

Este posibilă folosirea unui neuron aparte şi pentru rezervarea din mulţimea de clase $M = \{ X^1 = \{X^{11}, \ldots, X^{1k}\}, \ldots, X^i = \{X^{i1}, \ldots, X^{iq}\}, \ldots, X^p = \{X^{p1}, \ldots, X^{pm}\}\}$ a imaginilor din clasa unică X^i. În acest caz se consideră că unul din două semnale posibile de ieşire ale neuronului (de exemplu, 1) corespunde clasei X^i**,** al doilea – tuturor celorlalte clase. De aceea, dacă imaginea de intrare X^α duce la apariţia semnalului $y = 1$, atunci $X^\alpha \in X^i$, dacă $y = -1$ (sau $y = 0$**,** dacă se foloseşte codificarea binară), atunci aceasta ne arată că imaginea prezentată nu corespunde clasei atribuite.

Sistemul de recunoaştere în baza neuronului unic împarte toată suprafaţa de soluţii posibile în două suprafeţe cu ajutorul hiperplanului:

$$x_1 w_1 + x_2 w_2 + ... + x_n w_n + w_0 = 0.$$

Pentru vectorii de intrare bidimensionali, pragul dintre două clase de imagini este o linie dreaptă: vectorii de intrare, care se află mai sus de această dreaptă, aparţin unei singure clase, iar cei care se alfă mai jos – altei clase.

Pentru adaptarea, setarea sau învăţarea ponderilor (capacităţilor) legăturilor neuronilor pot fi folosite mai multe metode. Vom descrie una dintre ele care se numeşte *regula lui Hebb*. Hebb, studiind mecanismele de funcţionare a sistemului nervos central (SNC), a presupus că învăţarea se produce pe calea augmetării ponderilor legăturilor dintre neuroni, activitatea cărora coincide în timp. Dar în sistemele biologice această presupunere nu se îndeplineşte întotdeauna şi nu epuizează toate tipurile de învăţare, deşi la învăţarea reţelelor neuronale pe un strat cu semnale bipolare este foarte efectivă.

În conformitate cu regula lui Hebb, dacă imaginii bipolare prezentate $X=(x_1, ..., x_n)$ îi corespunde un semnal incorect de ieşire y, atunci ponderile $w_i (i = \overline{1,n})$ ale legăturilor neuronului se adaptează după formula:

$$w_i(t+1) = w_i(t) + x_i y, i = \overline{0,n}, \qquad (13)$$

unde: $w_i\ (t)$, $w_i\ (t+1)$ respectiv ponderea legăturii i a neuronului până şi după adaptare; $x_i (i = \overline{1,n})$ – componentele imaginii de intrare; $x_0 \equiv 1$ – semnalul de deplasare; y – semnalul de ieşire a neuronului.

Într-o formă mai completă, algoritmul de setare a ponderilor legăturilor neuronului cu folosirea regulii lui Hebb arată în felul următor:

Pasul 1. Se dă mulţimea $M = \{ (X^1, t^1), ..., (X^m, t^m)\}$ care constă din perechi (imaginea de intrare $X^k = (X1^k, ..., Xn^k)$, este necesar semnalul de ieşire a neuronului t^k), $k = \overline{1,m}$. Se iniţiază ponderile legăturilor neuronilor:

$$w_i = 0, i = \overline{0,n}.$$

Pasul 2. Pentru fiecare pereche (X^k, t^k), $k = \overline{1,m}$ până ce nu se îndeplinesc condiţiile de stopare, se îndeplinesc paşii 3-5.

Pasul 3. Se inițiază mulțimea de intrări a neuronului:

$$x_0 = 1,\ x_i = x_i^k,\ i = \overline{1,n}.$$

Pasul 4. Se inițiază semnalul de ieşire a neuronului: $y = t^k$.

Pasul 5. Se corectează ponderile legăturilor neuronului după regula:

$$w_i(new) = w_i(old) + x_i y,\ i = \overline{0,n}.$$

Pasul 6. Controlul condițiilor de stopare.

Pentru fiecare imagine de intrare X^k se determină semnalul corespunzător de ieşire y^k:

$$y^k = \begin{cases} 1, \textit{dacă}\ S^k > 0 \\ -1, \textit{dacă}\ S^k \le 0 \end{cases} \qquad k = \overline{1,m},$$

unde:

$$S^k = \sum_{i=1}^{n} x_i^k w_i + w_0.$$

Dacă vectorul $(y^1, \ldots, y^m)$ semnalelor de ieşire este identic cu vectorului $(t^1, \ldots, t^m)$ semnalelor inițiale ale neuronului, adică fiecărei imagini de intrare îi corespunde un semnal de ieşire dat anterior, atunci calculele se sfârşesc (trecem la pasul 7), dar dacă $(y^1, \ldots, y^m) \neq (t^1, \ldots, t^m)$, atunci trecem la pasul 2 al algoritmului.

Pasul 7. Stop.

Exemplu:

Fie că trebuie să învățăm un neuron bipolar să recunoască imaginiler X^1 şi X^2**,** ilustrate în figura 9.3.

Figura 9.3. Imaginile de intrare

În acelaşi timp vom cere ca imaginii X^1 să-i corespundă semnalul de ieşire **+1** al neuronului, iar imaginii X^2 – semnalul **-1**.

Aplicarea algoritmului Hebb ne oferă următoarele rezultate:

Pasul 1. Se dau mulțimile:

$$M = \{(X^1 = (-1,1,1,1,-1,-1,1,1);$$
$$(X^2 = (1,1,1,1,-1,1,1,-1,1),-1)\};$$

se inițiază ponderile legăturilor neuronului: $w_i = 0$**,** $i = \overline{0,9}$**.**

Pasul 2. Pentru fiecare din perechile $(X^1,1)$, $(X^2,-1)$ se îndeplinesc paşii 3-5.

Pasul 3. Se inițiază mulțimea intrărilor neuronului pentru imaginea primei perechi: $x_0 = 1$, $x_i = x_i^1$, $i = \overline{0,9}$.

Pasul 4. Se inițiază semnalul de ieşire al neuronului pentru imaginea primei perechi: $y = t^1 = 1$**.**

Pasul 5. Se corectează ponderile legăturilor neuronului după regula lui Hebb $w_i = w_i + x_i^1 y$ $(i = \overline{0,n})$:

$$w_0 = w_0 + x_0 y = 0 + 1 \cdot 1 = 1;$$
$$w_1 = w_1 + x_1^1 y = 0 + 1 \cdot 1 = 1;$$
$$w_1 = w_3 = w_4 = w_5 = w_6 = w_9 = 1;$$
$$w_2 = w_2 + x_2^1 y = 0 + (-1) \cdot 1 = -1;$$
$$w_2 = w_7 = w_8 = -1.$$

Pasul 3. Se inițiază mulțimea de intrări ale neuronului pentru imaginea X^2 a perechii a doua: $x_0 = 1,\ x_i = x_i^2,\ i = \overline{0,9}$.

Pasul 4. Se inițiază semnalul de ieşire a neuronului pentru imaginea din a doua pereche (X^2, t^2):

$$y = t^2 = -1.$$

Pasul 5. Se corectează ponderile legăturilor neuronului:

$$w_0 = w_0 + x_0 y = 1 + 1 \cdot (-1) = 0;$$

$$w_1 = w_1 + x_1^2 y = 1 + 1 \cdot (-1) = 0;$$

$$w_1 = w_3 = w_4 = w_6 = w_9 = 0;$$

$$w_2 = w_2 + x_2^2 y = -1 + 1 \cdot (-1) = -2;$$

$$w_2 = w_7 = -2;$$

$$w_5 = w_5 + x_5^2 y = 1 + (-1) \cdot (-1) = 2;$$

$$w_8 = w_8 + x_8^2 y = -1 + (-1) \cdot (-1) = 0;$$

Pasul 6. Se verifică condițiile de stopare.

Se calculează semnalele de intrare şi semnalele de ieşire ale neuronului la prezentarea imaginii X^1:

$$S^1 = \sum_{i=1}^{9} x_i^1 w_i + w_0 = 1 \cdot 0 + (-1) \cdot (-2) + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 2 + 1 \cdot 0 +$$

$$+ (-1) \cdot (-2) + (-1) \cdot 0 + 1 \cdot 0 + 0 = 6;$$

$$y^1 = 1, \text{ deoarece } S^1 > 0.$$

Se calculează semnalele de ieşire şi de intrare ale neuronului la prezentarea imaginii X^2:

$$S^2 = \sum_{i=1}^{9} x_i^2 w_i + w_0 = 1 \cdot 0 + 1 \cdot (-2) + 1 \cdot 0 + 1 \cdot 0 + (-1) \cdot 2 + 1 \cdot 0 +$$

$$+ 1 \cdot (-2) + (-1) \cdot 0 + 1 \cdot 0 + 0 = -6;$$

$$y^2 = -1, \text{ deoarece } S^2 < 0.$$

Întrucât vectorul $(y^1, y^2) = (1, -1)$ este egal cu vectorul (t^1, t^2), atunci calculele se stopează, deoarece a fost atins scopul – neuronul corect recunoaşte imaginile prezentate.

Ideea esențială a regulii (13) de a mări legăturile care leagă neuronii cu aceeaşi activitate în timp şi de a micşora legăturile, care leagă elementele cu activitate diferită, poate fi folosită la setarea rețelelor neuronale cu elemente binare. Regula lui Hebb (13) pentru rețelele binare cu un singur strat poate fi scrisă în forma:

$$w_i(t+1) = w_i(t) + \Delta w_i, \tag{14}$$

unde:

$$\Delta w_i = \begin{cases} 1, \ dacă\ x_i y = 1, \\ 0, \ dacă\ x_i = 0, \\ -1, \ dacă\ x_i \neq 0\ şi\ \ y = 0. \end{cases} \tag{15}$$

Exemplu:

Fie că trebuie să învățăm un neuron binar pentru recunoaşterea imaginilor X^1 şi X^2 ale exemplului precedent. Imaginii $\mathbf{X^1}$ fie că-i va corespunde semnalul de ieşire +1 al neuronului, iar imaginii X^2 – 0**.**

Aplicarea regulii lui Hebb în acest caz ne va da următoarele rezultate:

Pasul 1. Se dă mulțimea:

$M = \{(X^1 = (1, 0, 1, 1, 1, 1, 0, 0, 1), 1), (X^2 = (1, 1, 1, 1, 0, 1, 1, 0, 1), 0)\}$ **,**

şi se inițiază ponderile legăturilor neuronului $w_i = 0$**,** $i = \overline{0,9}$**.**

Pasul 2. Pentru perechile $(X^1, 1)$, $(X^2, 0)$ se îndeplinesc paşii 3-5.

Pasul 3. Se inițiază mulțimea intrărilor neuronului cu elementele imaginii X^1:

$$x_0 = 1,\ x_i = x_i^1,\ i = \overline{0,9}.$$

Pasul 4. Se inițiază semnalul de ieşire a neuronului pentru imaginea X^1:

$$y = t^1 = 1.$$

Pasul 5. Se corectează ponderile legăturilor neuronului cu ajutorul relațiilor (14, 15):

$$w_0 = w_0 + \Delta w_0 = 0 + 1 = 1;$$

$$w_1 = w_1 + \Delta w_1 = 0 + 1 = 1;$$

$$w_1 = w_3 = w_4 = w_5 = w_6 = w_9 = 1;$$

$$w_2 = w_2 + \Delta w_2 = 0 + 0 = 0;$$

$$w_2 = w_7 = w_8 = 0.$$

Pasul 3. Se inițiază mulțimea intrărilor neuronului cu elementele imaginii X^2:

$$x_0 = 1,\ x_i = x_i^2,\ i = \overline{0,9}.$$

Pasul 4. Se inițiază semnalul de ieşire al neuronului pentru imaginea X^2:

$$y = t^2 = 0.$$

Pasul 5. Se corectează ponderile legăturilor neuronului cu ajutorul relațiilor (14,15):

$$w_0 = w_0 + \Delta w_0 = 1 + (-1) = 0;$$

$$w_1 = w_1 + \Delta w_1 = 1 + (-1) = 0;$$

$$w_1 = w_3 = w_4 = w_6 = w_9 = 0;$$

$$w_2 = w_2 + \Delta w_2 = 0 + (-1) = -1;$$

$$w_5 = w_5 + \Delta w_5 = 1 + 0 = 1;$$

$$w_7 = w_2 = -1;$$

$$w_8 = w_8 + \Delta w_8 = 0 + 0 = 0.$$

Pasul 6. Verificarea condițiilor de stopare.

Se determină semnalele de intrare şi de ieşire ale neuronului la prezentarea imaginilor X^1 şi X^2 :

$$S^1 = \sum_{i=1}^{9} x_i^1 w_i + w_0 = 1\cdot 0 + 0\cdot(-1) + 1\cdot 0 + 1\cdot 0 + 1\cdot 1 + 1\cdot 0 +$$

$$+ 0\cdot(-1) + 0\cdot 0 + 1\cdot 0 + 0 = 1;$$

$$y^1 = 1, \text{ deoarece } S^1 > 0.$$

$$S^2 = \sum_{i=1}^{9} x_i^2 w_i + w_0 = 1\cdot 0 + 1\cdot(-1) + 1\cdot 0 + 1\cdot 0 + 0\cdot 1 + 1\cdot 0 +$$

$$+ 1\cdot(-1) + 0\cdot 0 + 1\cdot 0 = -2;$$

$$y^2 = -1, \text{ deoarece } S^2 < 0.$$

Întrucât vectorul $(y^1, y^2) = (1, 0)$ este egal cu vectorul (t^1, t^2), atunci calculele se stopează, deoarece a fost atins scopul.

9.3. Rețeaua neuronală Hebb

Utilizarea grupului din *m* neuroni bipolari sau binari $A_1, \ldots A_m$ (Figura 9.4 permite de a mări considerabil posibilitățile rețelei neuronale şi a

recunoaşte până la 2^m imagini diferite. Însă aplicarea acestei rețele pentru recunoaşterea a 2^m (sau aproape de 2^m) de imagini diferite poate duce la probleme nerezolvabile privind adaptarea ponderilor legăturilor rețelei neuronale. De aceea, deseori se recomandă folosirea acestei arhitecturi doar pentru recunoaşterea numai a *m* imagini diferite, fiecărei atribuindu-i o ieşire unică numai la ieşirea unui element A (ieşirile celorlalți în acelaşi timp trebuie să ia valoarea **-1** pentru neuroni bipolari şi valoarea **0** – pentru cei binari).

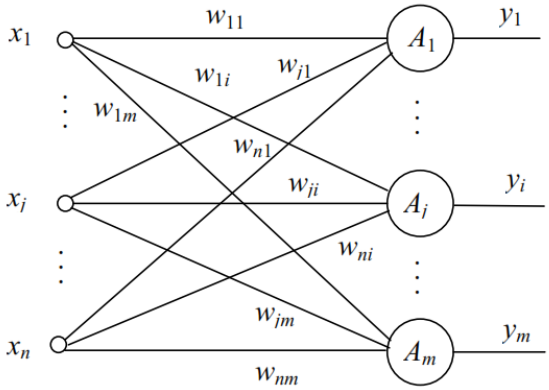


Figura 9.4. Rețea neuronală din *m* elemente

Rețeaua neuronală pe un singur strat cu neuroni binari, ilustrată în Figura 9.4, poate fi învățată cu ajutorul unui algoritm bazat pe regula Hebb. În acest caz se numeşte rețea Hebb. Folosirea în denumirea rețelei a algoritmului de învățare este caracteristic pentru teoria rețelelor neuronale.

Pentru prezentarea bipolară a semnalelor este posibilă învățarea rețelei neuronale cu ajutorul următorului algoritm:

Pasul 1. Se dă mulțimea $M = \{(X^1, t^1), \ldots, (X^m, t^m)\}$, care constă din perechi (imaginea de intrare $X^k=(X1^k, Xn^k)$, semnalul necesar de ieşire al neuronului t^k, $k = \overline{1,m}$). Se inițiază ponderile legăturilor neuronului:

$$w_{ji} = 0,\ j = \overline{1,n},\ i = \overline{1,m}.$$

Pasul 2. Fiecare pereche (X^k, t^k) se verifică la corectitudinea reacției rețelei neuronale la imaginea de intrare. Dacă vectorul de ieşire recepționat al rețelei $(y1^k, \ldots, ym^k)$, se deosebeşte de cel inițial $t^1 = (t1^k, tm^k)$, atunci îndeplinim paşii 3-5.

Pasul 3. Se inițiază mulțimea de intrari ale neuronilor:

$$x_0 = 1,\ x_j = x_j^k,\ j = \overline{1,n}.$$

Pasul 4. Se inițiază semnalele de ieşire ale neuronilor: $y_i = t_i^k$, $i = \overline{0,m}$.

Pasul 5. Se corectează ponderile legăturilor neuronilor după regula:

$$w_{ji}(new) = w_{ji}(old) + x_j y_i,\ j = \overline{0,n},\ i = \overline{0,m}.$$

Pasul 6. Se verifică condiția de stopare, adică corectitudinea funcționării rețelelor la prezentarea fiecărei imagini de intrare. Dacă condițiile nu se îndeplinesc, atunci trecem la *pasul 2* al algoritmului, altfel se sfârşesc calculele (trecem la *pasul 7*).

Pasul 7. Stop.

PARTEA 2. Desfăşurarea lucrării

1. Elaborați structura rețelei Hebb care va fi capabilă să recunoască 4 litere diferite ale numelui şi prenumelui dvs. şi motivați alegerea:
   - numărului de receptori ai neuronilor(numărul n de elemente ale rețelei trebuie să fie în limitele $12 \leq n \leq 30$);
   - numărului de neuroni de ieşire;
   - alegerea vectorului de semnale de ieşire.

2. Elaborați algoritmul şi programul de modelare al rețelei Hebb. În algoritm numaidecât luați în considerare posibilitatea apariției situației cu probleme nerezolvabile la adaptarea ponderilor legăturilor în rețeaua neuronală.

3. Învățați rețeaua neuronală Hebb a recunoaşterii a 4 caractere date.

4. Prezentați un set de simboluri de intrare şi semnale de ieşire dorite, când în rețea apare situația cu proleme nerezolvabile de adaptare a ponderilor legăturilor.

5. Se va prezenta darea de seamă.

BIBLIOGRAFIE

Cărți:

- A. Florea, A. Boangiu. Elemente de inteligența artificială.
- A. Florea. Bazele logice ale inteligenței artificiale. Cap. 4, 6.
- A. Florea, B. Dorohonceanu, C. Francu. Programare în Prolog.

Articole on-line:

- Turing, A.M. Computing Machinery and Intelligence, Mind, 59, 1950, 433-460.
- AI's Greatest Trends and Controversies, IEEE Intelligent Systems January/February 2000.
- Marvin Minsky. A Framework for Representing Knowledge, MIT-AI Laboratory Memo 306, June, 1974.
- J. Austin - How to do things with words, 1962, J. Searle - Speech acts, 1969

ANEXE

Anexa 1

MEDIUL GNU Prolog

Introducere

GNU Prolog este un compilator Prolog gratuit, dezvoltat de Daniel Diaz. GNU Prolog reprezintă un compilator bazat pe maşina abstractă Warren (WAM). GNU Prolog, mai întâi, compilează programul într-un fişier WAM, care apoi este translat în limbajul *mini-assembly*, elaborat special pentru GNU Prolog. Fişierul obținut este la fel translat, însă deja în limbajul assembler specific stației pe care rulează. Acest fapt permite GNU Prolog să producă un executabil nativ, independent dintr-un fişier sursă Prolog (.pl, .pro). Avantajul principal al acestei scheme de compilare constă în producerea codului nativ şi viteza de compilare. O altă caracteristică este că executabilele sunt de dimensiune mică.

Caracteristicile GNU Prolog:

- corespunde standardului ISO pentru Prolog;
- posedă extensii ca: variabile globale, sockets API, OS API, suport pentru gramatica DC etc;
- peste 300 de predicate Prolog predefinite;
- Prolog debugger şi WAM debugger de nivel jos;
- facilitatea de editare a liniilor de cod prin intermediul interpretatorului interactiv;
- interfață bidirecțională între Prolog şi C;
- generarea directă de cod assembler de 15 ori mai rapidă decât în *wamcc + gcc.*

Interpretatorul GNU Prolog

GNU Prolog oferă două posibilități de executare a programelor:
- interpretarea lor, utilizând interpretatorul interactiv GNU Prolog;
- compilarea lor, utilizând compilatorul GNU Prolog.

Rularea programului în cadrul interpretatorului permite utilizatorului vizualizarea şi depanarea codului. Compilarea programelor în cod nativ permite producerea executabilelor de dimensiune mică şi viteză de execuție optimizată. Rularea unui program compilat e de 3-5 ori mai

rapidă decât rularea unui program interpretat. Însă nu e posibil a fi depanat programul precum permite aceasta interpretatorul şi nici a fi vizualizat codul sursă. De regulă se recomandă rularea programului, utilizând interpretatorul pe durata dezvoltării lui, permițând astfel depanarea uşoară. Apoi, produsul final urmează a fi compilat în cod nativ, pentru a produce un executabil autonom.

Lansarea/închiderea interpretatorului

GNU Prolog oferă un interpretator interactiv Prolog clasic numit *top-level*. Acesta permite utilizatorului să execute interogări, să consulte un program Prolog, să-l vizualizeze şi să-l depaneze. *Top-level* poate fi invocat utilizând următoarea comandă din terminal:

```
% gprolog [OPTION]…
```

Opțiuni:

-init-goal *GOAL* execută scopul înainte să intre în *top-level;*
-consult-file *FILE* consultă fişierul în cadrul *top-level;*
-entry-goal *GOAL* execută scopul în cadrul *top-level;*
-query-goal *GOAL* execută scopul drept interogare pentru *top-level;*
-help afişează comenzile disponibile;
-version afişează numărul versiunii curente;
- evită parsarea liniei curente.

Rolul principal al comenzii gprolog este de a executa însăşi interpretatorul, cu alte cuvinte, a executa predicatul predefinit top_level/0 care va produce următorul lucru:

```
GNU Prolog 1.4.0
By Daniel Diaz
Copyright (C) 1999-2012 Daniel Diaz
| ?-
```

Interpretatorul este pregătit să execute interogările prestate de utilizator. Pentru a părăsi interpretatorul trebuie tastată secvența de taste pentru *end-of-file* (Ctrl-D) sau end_of_file. La fel, e posibil a fi folosit predicatul predefinit halt/0 (cifra de după „/” indică numărul de argumente ale

predicatului).

Ciclul interactiv de citire-execuție-afişare al interpretatorului

GNU Prolog *top-level* se bazează pe un ciclu interactiv de citire-execuție-afişare care permite reexecutări după cum urmează:

- afişarea promptă'| ?-';
- citirea unei interogări;
- executarea interogării;
- în caz de succes se afişează valorile variabilelor din interogare;
- dacă mai rămân alternative (cu alte cuvinte interogarea nu este deterministă), se afişează *?* unde utilizatorul poate interveni cu una din comenzile disponibile: RETURN pentru a stopa execuția, *;* pentru a estima următoarea soluție sau a pentru estimarea tuturor soluțiilor rămase.

Mai jos este prezentat un exemplu de execuție a unei interogări (găseşte listele *X* şi *Y* astfel încât concatenarea lor să fie *[a,b,c]*):

```
| ?- append(X,Y,[a,b,c]).
```

X = []
`Y = [a,b,c] ? ;` (aici utilizatorul tastează ';' ca să estimeze o soluție)

X = [a]
`Y = [b,c] ?` a (aici utilizatorul tastează 'a' ca să estimeze toate soluțiile rămase)

`X = [a,b]`
`Y = [c]` (aici utilizatorul nu mai este interogat şi soluția următoare este estimată automat)

`X = [a,b,c]`
`Y = []` (aici utilizatorul nu mai este interogat şi soluția următoare este estimată automat)
no (nu mai sunt soluții)

Încărcarea unui program Prolog

Interpretatorul permite utilizatorului să încarce fişiere sursă Prolog. Predicatele încărcate pot fi vizualizate, executate şi depanate, în timp ce cele compilate – nu. Pentru a încărca în memorie un program Prolog, utilizăm predicatul predefinit consult/1 argumentul căruia este calea relativă sau absolută spre fişierul ce conține programul Prolog. Aceasta permite utilizatorului să introducă predicatele direct în terminal. O scurtătură pentru apelul consult(*FILE*) este [*FILE*], de exemplu următoarele sunt echivalente:

```
| ?- consult('C:\example.pro').
| ?- ['C:\example.pro'].
```

Când predicatul consult/1 este invocat asupra unui fişier sursă, GNU Prolog întâi compilează acest fişier pentru a genera un fişier WAM temporar. Dacă compilarea nu eşuează, programul se încarcă în memoria calculatorului utilizând predicatul load/1.

Predicate de depanare a programelor

Predicatul `trace/0` activează regimul de depanare a interpretatorului. Orice următoare invocare de predicate va fi depanată.

Predicatul `debug/0` activează regimul de depanare a interpretatorului. Doar următoarea invocare a unui oarecare predicat va fi depanată.

Predicatul `debugging/0` afişează în terminal informații de depanare referitoare la starea curentă.

Predicatele notrace/0 şi nodebug/0 dezactivează regimul de depanare.

Predicatul wam_debug/0 invocă regimul de depanare specific structurilor de date WAM. Acest regim la fel mai poate fi invocat, utilizând comanda W.

Predicate predefinite în Prolog

Operatorii de unificare:

`(=)/2` – operatorul de unificare în Prolog
`Term1 = Term2`

`(\=)/2` – operatorul care verifică dacă operatorii nu pot fi unificați
`Term1 \= Term2`

Operatorii de comparare generală a termenilor în GNU Prolog:

`(==)/2` – operatorul de egalitate
`(\==)/2` – operatorul diferit de…
`(@<)/2` – operatorul mai mic ca…
`(@=<)/2` – operatorul mai mic sau egal cu…
`(@>)/2` – operatorul mai mare ca…
`(@>=)/2` – operatorul mai mare sau egal cu…

Operatorii aritmetici în GNU Prolog:

`(=:=)/2` – operatorul de egalitate
`(=\=)/2` – operatorul diferit de…
`(<)/2` – operatorul mai mic ca…

`(=<)/2` – operatorul mai mic sau egal cu…
`(>)/2` – operatorul mai mare ca…
`(>=)/2` – operatorul mai mare sau egal cu…

`(is)/2` – evaluarea unei expresii aritmetice, de exemplu:
Result is Expression1 + Expression2

Predicatele de prelucrare a listelor:

`append/3`
`append(List1, List2, List12)` – concatenează lista `List1` şi lista `List2` în `List12`.

`member/2`
`member(Element, List)` – verifică dacă elementul Element aparţine listei `List`.

`reverse/2`
`reverse(List1, List2)` – inversează lista `List1` în `List2`.

`delete/3`
`delete(List1, Element, List2)` – elimină toate apariţiile Element în lista `List1` în cadrul listei `List2`.

`prefix/2`
`prefix(Prefix, List)` – verifică dacă lista `Prefix` este prefixul listei `List`.

`suffix/2`
`suffix(Suffix, List)` – verifică dacă lista `Suffix` este sufixul listei `List`.

`sublist/2`
`sublist(List1, List2)` – verifică dacă în lista `List1` toate

elementele apar în aceeaşi ordine ca şi în `List2`.

`last/2`
`last(List, Element)` – verifică dacă elementul `Element` este ultimul în cadrul listei `List`.

`length/2`
`length(List, Length)` – estimează lungimea listei `List` în `Length`.

`sort/2`
`sort(List1, List2)` – sortează lista `List1` în cadrul listei `List2` (elementele duplicate sunt eliminate).

`msort/2`
`msort(List1, List2)` – sortează lista `List1` în cadrul listei `List2` (elementele duplicate nu sunt eliminate).

`nth/3`
`nth(N, List, Element)` – returnează al N-lea element din lista `List` în `Element`.

`min_list/2`
`min_list(List, Min)` – returnează elementul minimal din lista `List` în `Min`.

`max_list/2`
`max_list(List, Max)` – returnează elementul maximal din lista `List` în `Max`.

`sum_list/2`
`sum_list(List, Sum)` – returnează suma elementelor listei `List` în `Sum`.

Anexa 2

Codul sursă al programului care realizează o rețea neuronală Hamming în limbajul de programare C++

```
#include <iostream.h>
#include <string.h>
#define true 1
#define false 0
using namespace std;
class Hemming
{
private:
	float* Uin;
	float* Uout;
	float** W;
	int N;
	int M;
	int* s;
	float e;
	float b;
	float k;

	void InitUin();
	void InitUout();
	float g(float x);
	bool Compare(float* v1, float* v2);
public:
	int GetVectorLength(); //returns Image's length
	int GetEtalonsCount();// returns Etalon count
	void SetImage(int* image);// Set recognizing Image's
vector
	void SetEtalons(int** images);// set etalons

	void Hemming::Final(int* rezult, int &counter);//
recognize Image
	Hemming(int n,int m);
	Hemming(Hemming& hemming);
	~Hemming();
};

	void Hemming::InitUin()
	{	int i,j;
		for (i = 0; i < N; i++)
		{	float temp = 0;
			for ( j = 0; j < M; j++)
				temp += s[j] * W[j][i];
				Uin[i] = temp;
```

```
            }
        }
        void Hemming::InitUout()
        {       int i;
                for (i = 0; i < N; i++)
                Uout[i] = Uin[i] * k;
        }
        float Hemming::g(float x)
        {       return x < 0 ? 0 : x;
        }
        bool Hemming::Compare(float* v1, float* v2)
        {       int i;
                for ( i = 0; i < N; i++)
                        if (v1[i] != v2[i]) return false;
                return true;
        }
        int Hemming::GetVectorLength(){ return N;}
        int Hemming::GetEtalonsCount(){ return M;}
        void Hemming::SetImage(int* image)
        {       int i;
                for( i=0; i<M; i++)
                        s[i]= image[i];
        }
        void Hemming::SetEtalons(int** images)
        {       int i,j;
                for ( i = 0; i < M; i++)
                        for ( j = 0; j <N; j++)
                                W[i][j] = (float)images[j][i] / 2;
        }
        void Hemming::Final(int* rezult, int &counter)
        {       InitUin();
                InitUout();
                int i,j;
                while (!Compare(Uin, Uout))
                {       for(i=0; i<N; i++)
                                Uin[i]=Uout[i];
                        for (i = 0; i < N; i++)
                        {       //Uout[i] = g(Uin[i] - e *
(Uin.Where((r, index) => index != i).Sum()));
                                float temp=0;
                                for(j=0; j<N; j++)
                                if(i!=j) temp+=Uin[j];
                                Uout[i] = g(Uin[i] - e*temp);
                        }
                }
                int max = 0;
                for ( i = 1; i < N; i++)
```

```
                if (Uout[max] < Uin[i]) max = i;
                counter = 0;
                for( i=max; i<N; i++)
                        if(Uout[max]==Uout[i])
                        {       rezult[counter]=i;
                                counter++;
                        }
}
Hemming::Hemming(int n,int m)
{       N=n; M=m;
        b = M / 2;
        e = 1 / N;
        k = 0.1;
        s= new int[M];
        Uin = new float[N];
        Uout = new float[N];
        W= new float*[M];
        for(int i=0; i<M; i++)
                W[i] = new float[N];
}
Hemming::Hemming(Hemming& hemming)
{       N=hemming.N;
        M=hemming.M;
        b = M / 2;
        e = 1 / N;
        k = 0.1;
        int i;
        s= new int[N];
        Uin = new float[N];
        Uout = new float[N];
        W= new float*[M];
        for( i=0; i<M; i++)
                W[i] = new float[N];
                for( i=0; i<N; i++)
                {       s[i] = hemming.s[i];
                        Uin[i]=hemming.Uin[i];
                        Uout[i]=hemming.Uout[i];
                }
        int j;
        for( i=0; i<M; i++)
                for( j=0; j<N; j++)
                        W[i][j] = hemming.W[i][j];
}
Hemming::~Hemming()
{       if(s!=NULL) delete s;
        if(Uin!=NULL) delete Uin;
        if(Uout!=NULL) delete Uout;
```

```
                if(W!=NULL)
                {       for (int i=0; i<M; i++)
                                if (W[i]!=NULL) delete W[i];
                        delete W;
                }
        }
int _tmain(int argc, _TCHAR* argv[])
{       int rezult[5];
        int n,m,i,j;
        n=5; m=9;
        int etalons[5][9] =
        {       { -1, 1, -1, -1, 1, -1, -1, 1, -1 },
                { 1, 1, 1, 1, -1, 1, 1, -1, 1 },
                { 1, -1, 1, 1, 1, 1, 1, -1, 1 },
                { 1, 1, 1, 1, -1, -1, 1, -1, -1 },
                { -1, -1, -1, -1, 1, -1, -1, -1, -1 }
        };
        int** etalonPt= new int* [5];
        for( i =0; i<5; i++)
                etalonPt[i]=new int[9];
        for( i=0; i<5; i++)
                for( j=0; j<9; j++)
                        etalonPt[i][j] = etalons[i][j];
        int s[9]= {-1,-1,1,1,1,1,1,-1,1};
        //s3={-1,-1,-1,1,-1,-1,-1,1,-1};
        Hemming h(n,m);
        h.SetEtalons(etalonPt);
        h.SetImage(s);
        int counter;
        h.Final(rezult,counter);
        cout<<"Mached image indexes: "<<endl;
        for(i=0; i<counter; i++)
                cout<<rezult[i]<<",";
                cout<<endl;
        char c; cin>>c;
        for( i =0; i<5; i++)
                delete etalonPt[i];
        delete etalonPt;
        return 0;
}
```

Anexa 3

Codul sursă al programului care realizează rețeaua neuronală Hamming, în limbajul de programare C#

Clasa neuronului:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Plia_Interface
{ public class Newrons
 { protected int _m = 9;
  protected int _n;
  public Newrons(int n, int m)
  { _m = m; _n = n;
  b = m / 2;
  e = 1 / _n;
  k = 0.1;
  }
  private double b;
  private double e;
  private sbyte[][] etalons;
  public sbyte[][] Etalons
  { get
  { return etalons;
  }
  set
  { if (value.GetLength(0) != _n
    || value.Any(v => v.Length != _m))
    throw new ArgumentException("Wrong size of vector");
   etalons = value;
   InitW();
  }
  }
  private double[,] W;
  private double[] Uin;
  private double[] Uout;
  private sbyte[] _s;
  public sbyte[] S
  { get
  { return _s;
  }
  set
  { if (value.Length != _m)
    throw new ArgumentException("Wrong vector size");
   _s = value;
```

```
  }
  }
  private double k;
  protected void InitW()
  { W = new double[_m, _n];
  for (int i = 0; i < W.GetLength(0); i++)
   for (int j = 0; j < W.GetLength(1); j++)
    W[i, j] = (double)etalons[j][i] / 2;
  }
  protected void InitUin()
  { Uin = new double[_n];
                    for (int i = 0; i < _n; i++)
  { double temp = 0;
   for (int j = 0; j < _m; j++)
    temp += _s[j] * W[j, i];
   Uin[i] = temp;
  }
  }
  protected void InitUout()
  { Uout = new double[_n];
  for (int i = 0; i < _n; i++)
   Uout[i] = Uin[i] * k;
  }
  protected double g(double x)
  { return x < 0 ? 0 : x;
  }
  private bool Compare(double[] v1, double[] v2)
  { if (v1.Length != v2.Length) return false;
  for (int i = 0; i < v1.Length; i++)
   if (v1[i] != v2[i]) return false;
  return true;
  }
  public List<int> Final()
  { InitUin();
  InitUout();
  while (!Compare(Uin, Uout))
  { Array.Copy(Uout, Uin, Uout.Length);
   for (int i = 0; i < Uout.Length; i++)
    Uout[i] = g(Uin[i] - e * (Uin.Where((r, index) => index
!= i).Sum()));
  }
  int max = 0;
  for (int i = 1; i < Uout.Length; i++)
   if (Uout[max] < Uin[i]) max = i;
  return Uout.Select((u, index) => new { U = u, Index =
index }).Where(u => u.U == Uout[max]).Select(u =>
u.Index).ToList();
```

```
  }
 }
}
```

Clasa colecției de şabloane:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
namespace Plia_Interface
{ [Serializable]
 public class Symbol
 { public string Name { get; set; }
  public sbyte[] Vector { get; set; }
 }
 [Serializable]
 public class SymbolCollection
 { public SymbolCollection()
  { _symbols = new List<Symbol>();
  _n=0; _m=0;
  }
  private List<Symbol> _symbols;
  public List<Symbol> Symbols
  { get
  { return _symbols;
  }
  set
  { _symbols = value;
  }
  }
  private int _n, _m;
  public int N
  { get
  { return _n;
  }
  set
  { _n = value;
   _symbols.Clear();
  }
  }
  public int M
  { get
  { return _m;
  }
  set
  { _m = value;
```

```
    _symbols.Clear();
   }
   }
   public sbyte[][] GetVectors()
   { return _symbols.Select(s => s.Vector).ToArray();
   }
   public void AddSymbol(Symbol s)
   { if (s.Vector.Length != _n * _m)
    throw new ArgumentException();
   }
   private static string path = "data.dat";
   public void LazyInit()
   { _n = 3;
   _m = 3;
   var etalons = new sbyte[5][];
   etalons[0] = new sbyte[9] { -1, 1, -1, -1, 1, -1, -1, 1, -
1 };
   etalons[1] = new sbyte[9] { 1, 1, 1, 1, -1, 1, 1, -1, 1 };
   etalons[2] = new sbyte[9] { 1, -1, 1, 1, 1, 1, 1, -1, 1 };
   etalons[3] = new sbyte[9] { 1, 1, 1, 1, -1, -1, 1, -1, -1
};
   etalons[4] = new sbyte[9] { -1, -1, -1, -1, 1, -1, -1, -1,
-1 };
   for (int i = 0; i < etalons.Length; i++)
    _symbols.Add(
     new Symbol()
     { Name = i.ToString(),
     Vector = etalons[i]
     });
   }
   public static SymbolCollection Load()
   { BinaryFormatter serializer = new BinaryFormatter();
   var file = File.Open(path, FileMode.Open);
   var result =
(SymbolCollection)serializer.Deserialize(file);
   return result;
   }
   public static void Save(SymbolCollection collection)
   { BinaryFormatter serializer = new BinaryFormatter();
   var file = File.Create(path);
   serializer.Serialize(file, collection);
   }
  }
}
```

Clasa de redactare a elementelor:

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows;
using System.Windows.Shapes;
using System.Windows.Media;
namespace Plia_Interface
{ public class SymbolEditor
 { private int _n, _m;
  private Grid _grid;
  private SolidColorBrush selected;
  private SolidColorBrush deselected;
  private LinearGradientBrush hylighted;
  private Rectangle mask;
  public SymbolEditor(int n, int m, Grid grid)
  { _n = n; _m = m; _grid = grid;
  #region Fiture Initialization
  _grid.ShowGridLines = true;
  selected = new SolidColorBrush(Colors.DarkBlue);
  deselected = new SolidColorBrush(Colors.AntiqueWhite);
  hylighted = new
   LinearGradientBrush(Colors.Azure, Colors.Crimson, 0.5);
  mask = new Rectangle();
  mask.Fill = hylighted;
  mask.RadiusX = 5;
  mask.RadiusY = 5;
  mask.Opacity = 0.35;
  mask.Visibility = Visibility.Visible;
  #endregion
  InitGrid();
  }
  protected void InitGrid()
  { _grid.Children.Clear();
  _grid.RowDefinitions.Clear();
  _grid.ColumnDefinitions.Clear();
  for (int i = 0; i < _n; i++)
  { var row = new RowDefinition();
   _grid.RowDefinitions.Add(row);
  }
  for (int j = 0; j < _m; j++)
  { var column = new ColumnDefinition();
   //column.Width = new GridLength(0, GridUnitType.Star);
   _grid.ColumnDefinitions.Add(column);
  }
  for (int i = 0; i < _n; i++)
  { for (int j = 0; j < _m; j++)
```

```
    { var panel = new StackPanel();
     panel.Background = deselected;
     panel.SetValue(Grid.RowProperty, i);
     panel.SetValue(Grid.ColumnProperty, j);
     panel.MouseEnter += new
MouseEventHandler(Canvas_MouseEnter);
     panel.MouseLeave += new
MouseEventHandler(Canvas_MouseLeave);
     _grid.Children.Add(panel);
    }
   }
   _grid.MouseDown += new
MouseButtonEventHandler(MyGrid_MouseDown);
  }
  public sbyte[] GetVector()
  { var panels = _grid.Children.OfType<StackPanel>()
    .Select(p =>
     p.Background == selected ? (sbyte)1 : (sbyte)-1
    ).ToArray();
  if (panels.Length < _n * _m)
   throw new Exception("Something wrong here");
  return panels;
  }
  public void ApplyVector(sbyte[] v)
  { if (v.Length != _n * _m)
   throw new ArgumentException("argument size is not much to
the grid");
  var panels = _grid.Children.OfType<StackPanel>();
  for (int i = 0; i < v.Length; i++)
  { var panel = panels.Single(p =>
    ((int)p.GetValue(Grid.RowProperty) * _m
    + (int)p.GetValue(Grid.ColumnProperty)) == i);
   if (v[i] == 1)
    panel.Background = selected;
   else panel.Background = deselected;
  }
  }
  #region Gui Event Hendlers
  private void switchColor(StackPanel canv)
  { if (canv.Background.Equals(selected))
   canv.Background = deselected;
  else
   canv.Background = selected;
  }
  private void Canvas_MouseEnter(object sender,
MouseEventArgs e)
  { if (!(e.OriginalSource is StackPanel)) return;
```

```
  var canv = (StackPanel)e.OriginalSource;
  mask.Width = canv.ActualWidth;
  mask.Height = canv.ActualHeight;
  if (e.LeftButton == MouseButtonState.Pressed)
   switchColor(canv);
  canv.Children.Add(mask);
  }
  private void Canvas_MouseLeave(object sender,
MouseEventArgs e)
  { if (!(e.OriginalSource is StackPanel)) return;
  var canv = (StackPanel)e.OriginalSource;
  canv.Children.Remove(mask);
  }
  private void MyGrid_MouseDown(object sender,
MouseButtonEventArgs e)
  { if (e.OriginalSource != mask) return;
  var canv = mask.Parent as StackPanel;
  if (canv == null) return;
  switchColor(canv);
  }
  #endregion
 } }
```

PROGRAMAREA LOGICĂ ŞI INTELIGENȚA ARTIFICIALĂ

Îndrumar de laborator