Ministerul Educaţiei al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică şi Microelectronică
Catedra Automatică şi Tehnologii Informaţionale

RAPORT

Lucrare de laborator nr 4

Disciplina: Programarea Funcțională și Inteligența Artificială

Tema: Sistem expert

A efectuat:

Vovc Artemie st. TI-133

A verificat:

Lazu Victoria lect. super.

Chișinău 2016

Cuprins

1 Scopul lucrării

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

2 Noțiuni teoretice

2.1. 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. 2.1):

1) *Baza de cunoştinţe (BC).* BC este partea centrală a sistemului expert. Aceasta conţine o colecţie de fapte şi de cunoştinţe (regulile) pentru extragerea altor cunoştinţe. Informaţiile conţinute în baza de cunoştinţe 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 inferenţă.* MI conţine descrieri ale modului de aplicare a cunoştinţelor cuprinse în baza de cunoştinţe. În timpul consultării, MI iniţiază SE să înceapă procesările, îndeplinind regulile determină admisibilitatea soluţiei găsite şi transmite rezultatele la Interfaţa sistemului (System User Interface).

3) *Interfaţa sistemului utilizatorului (ISU)* este parte a SE, care interacţionează cu utilizatorul. Funcţiile ISU includ: recepţionarea informaţiilor de la utilizator, transferul rezultatelor în forma cea mai convenabilă utilizatorului, explicarea rezultatelor recepţionate de SE (oferă informaţii cu privire la atingerea rezultatelor).

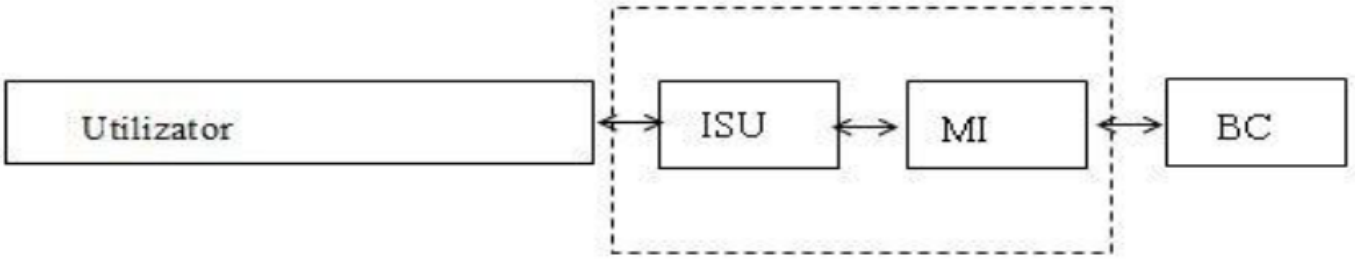

Figura 2.1. Structura generală a SE

În funcţie de metoda de clasificare şi plasare a informaţiei, există mai multe tipuri de baze de cunoştinţe: *de producţie, de reţea* şi *de cadru* (frame-uri) modele de reprezentare a cunoştinţelor. *Modelul de reţea* se bazează pe reprezentarea cunoştinţelor în forma unei reţele ale cărei noduri corespund conceptelor, iar arcele relaţiilor 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 producţie* se bazează pe regulile de forma "dacăatunci" şi permite introducerea fragmentelor de cunoştinţe faptice în regulile

limbajului Prolog. Anume astfel sunt construite SE bazate pe reguli. La punerea în aplicare a SE bazat pe logică, baza de cunoştinţe reprezintă un set de afirmaţii, fapte. Elaborarea concluziei unui expert în acest caz se bazează pe mijloacele standard de lucru cu listele.

2.2. Determinarea rezultatului (răspunsului) expert

Prin concluzia SE, se subînţelege dovada faptului că în setul de ipoteze se conţine concluzia căutată. Logica de obţinere a răspunsului (concluziei) sunt specificate de regulile de inferenţă. Concluzia (rezultatul) se obţine 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 inferenţă iniţiază 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 comparaţie 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 soluţionare a conflictului. În soluţionarea conflictului prioritate se dă regulilor care sunt mai specifice, sau regulilor ce ţin 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 obţine 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 specificaţiile necesare.

2.3. 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 cunoştinţe. Independenţa reciprocă a regulilor de producţie 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 conţine 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 cunoştinţe.

2) În cazul în care este posibilă apelarea mai multor reguli, pentru a selecta o regulă, se utilizează mecanismul de soluţionare 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-acţiune*. Răspunsul afirmativ al SE este rezultatul uneia din regulile de producţie. Selecţia regulii se efectuează, în conformitate cu datele de intrare.

Elaborarea a SE în Turbo Prolog (sau Visual Prolog), bazat pe reguli, începe cu declaraţia 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 producţie care vor descrie fragmente de cunoştinţe 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 cunoştinţele 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 spaţiul de căutare care descrie fragmentele de cunoştinţe 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 apartenenţă 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 conţin în regulile de producţie (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 obţine 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.
```

2.4. 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]).
36
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 - condiţii, din enunţurile tipului cond. Fiecare enunţ (fapt) specifică tipul de cond şi stabileşte condiţia de selecţie a clasificării de rase de câini utilizate aici. În SE bazate pe logică, interpretatorul utilizează aceste numere de condiţie pentru a efectua alegerile (selecţiile) adecvate. MI conţine regulile de tratare a listelor de atribute în descrierea obiectelor. Prin utilizarea predicatului go MI verifică afirmaţiile BC rule şi cond pentru a clarifica cu ajutorul regulii check existenţa sau lipsa de valori adecvate de date: Regula iniţială 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 attribute 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.
37
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 recepţionarea 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 condiţii. Dacă ”potrivirea” se produce, modulul programului ar trebui să adauge la baza de cunoştinţe valorile potrivite şi să continue procesul cu noile date recepţionate 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 posibilităţile sunt epuizate. Avantajul SE bazat pe logică constă în capacitatea de a stoca faptele bazei de cunoştinţe în afirmaţiile bazei de date dinamice. Baza de cunoştinţe poate fi plasată într-un fişier pe disc, ceea ce o face independentă de codul sursă.

3 Realizarea

Funcțiile yes, no, retractall, read sunt funcții standarde din swi prolog. Yes setează memorie partajată ca și no. Retractall eliberează memoria. Read este o funcție ce așteaptă introducere de la utilizator. Pozitiv și negativ setează baza de date cu 1 sau 0 dacă răspunsul utilizatorului este afirmativ sau nu.

```
%seteaza baza de date
setez(X, 1) :- assertz(yes(X)), !.
setez(X, _) :- assertz(no(X)).
%sterge baza de date
sterge_db :- retractall(yes(_)), retractall(no(_)).
%verifica raspunsul utilizatorului
pozitiv(X) :- yes(X), ! ;no(X),  !, fail ;setezator(X), pozitiv(X).
negativ(X) :- no(X),  ! ;yes(X), !, fail ;setezator(X), negativ(X).
%seteaza baza de date cu raspunsul utilizatorului
setezator(X) :- intrebare(X, RET), setez(X, RET).
%formeaza intrebari spre utilizator si salveaza raspunsul
intrebare(X, RET) :- format("~nAlgoritmul de sortare are: ~w ?~n",[X]),introdu(RET).
introdu(RET) :- read(T), !, raspunsul(T, RET).
introdu(0) .
%modifica raspunsul utilizatorului formand din y -> 1
% si din alt caracter -> 0
raspunsul(y, 1) :- !.
raspunsul(_, 0) .
%punctul de start a sistemului expert
start :- sterge_db, algoritm_de_sortare(X),
       format("~nAlgoritmul potrivit este: ~w .",[X]).
```

Rezultatul:

```
24 ?- start.

Algoritmul de sortare are: performanta de Q(nlogn) ?
|: y.

Algoritmul de sortare are: performanta de Q(n^2) ?
|: n.

Algoritmul de sortare are: memorie adaugatoare n ?
|: y.

Algoritmul de sortare are: memorie adaugatoare 1 ?
|: n.

Algoritmul de sortare are: este stabil ?
|: n.

Algoritmul de sortare are: nu este stabil ?
|: y.

Algoritmul potrivit este: QuickSort .
true.
```

Figura 3.1 – Rezultatul interogării

Concluzia

Lucrarea dată a avut ca scop să ne facă cunoscuți cu programarea logică la general și programarea în prolog esențial. În lucrarea dată am implementat un sistem expert ce recomandă un algoritm de sortare după specificațiile clientului. În baza de cunoștințe sunt doar șapte algoritmi de sortare cu caracteristicele sale.

Bibliografia

1 **Prolog la general :** http://biblioteca.regielive.ro/referate/limbaje-de-programare/prolog-limbaj-de-programare-logica-114586.html

2 **Studierea SWI-Prolog :** http://www.swi-prolog.org/pldoc/index.html

Anexe A
Listingul

```
%seteaza baza de date
setez(X, 1) :- assertz(yes(X)), !.
setez(X, _) :- assertz(no(X)).
%sterge baza de date
sterge_db :- retractall(yes(_)), retractall(no(_)).
%verifica raspunsul utilizatorului
pozitiv(X) :- yes(X), ! ;no(X),  !, fail ;setezator(X), pozitiv(X).
negativ(X) :- no(X),  ! ;yes(X), !, fail ;setezator(X), negativ(X).
%seteaza baza de date cu raspunsul utilizatorului
setezator(X) :- intrebare(X, RET), setez(X, RET).
%formeaza intrebari spre utilizator si salveaza raspunsul
intrebare(X, RET) :- format("~nAlgoritmul de sortare are: ~w ?~n",[X]),introdu(RET).
introdu(RET) :- read(T), !, raspunsul(T, RET).
introdu(0) .
%modifica raspunsul utilizatorului formand din y -> 1
% si din alt caracter -> 0
raspunsul(y, 1) :- !.
raspunsul(_, 0) .
%punctul de start a sistemului expert
start :- sterge_db, algoritm_de_sortare(X),
       format("~nAlgoritmul potrivit este: ~w .",[X]).
%baza de cunostinte
algoritm_de_sortare('QuickSort') :-
       pozitiv("performanta de Q(nlogn)"),
       negativ("performanta de Q(n^2)"),
       pozitiv("memorie adaugatoare n"),
       negativ("memorie adaugatoare 1"),
       negativ("este stabil"),
       pozitiv("nu este stabil")
       ,!.
algoritm_de_sortare('MergeSort') :-
       pozitiv("performanta de Q(nlogn)"),
       negativ("performanta de Q(n^2)"),
       pozitiv("memorie adaugatoare n"),
       negativ("memorie adaugatoare 1"),
       pozitiv("este stabil"),
       negativ("nu este stabil")
       ,!.
algoritm_de_sortare('HeapSort') :-
       pozitiv("performanta de Q(nlogn)"),
       negativ("performanta de Q(n^2)"),
       negativ("memorie adaugatoare n"),
       pozitiv("memorie adaugatoare 1"),
       negativ("este stabil"),
       pozitiv("nu este stabil")
       ,!.
algoritm_de_sortare('BubbleSort') :-
       negativ("performanta de Q(nlogn)"),
       pozitiv("performanta de Q(n^2)"),
       negativ("memorie adaugatoare n"),
       pozitiv("memorie adaugatoare 1"),
       pozitiv("este stabil"),
       negativ("nu este stabil")
       ,!.
algoritm_de_sortare('BinaryTreeSort') :-
       pozitiv("performanta de Q(nlogn)"),
       negativ("performanta de Q(n^2)"),
```

```
	pozitiv("memorie adaugatoare n"),
	negativ("memorie adaugatoare 1"),
	pozitiv("este stabil"),
	negativ("nu este stabil")
	,!.
algoritm_de_sortare('CocktailSort') :-
	negativ("performanta de Q(nlogn)"),
	pozitiv("performanta de Q(n^2)"),
	negativ("memorie adaugatoare n"),
	pozitiv("memorie adaugatoare 1"),
	pozitiv("este stabil"),
	negativ("nu este stabil")
	,!.
algoritm_de_sortare('GnomeSort') :-
	negativ("performanta de Q(nlogn)"),
	pozitiv("performanta de Q(n^2)"),
	negativ("memorie adaugatoare n"),
	pozitiv("memorie adaugatoare 1"),
	pozitiv("este stabil"),
	negativ("nu este stabil")
	,!.
algoritm_de_sortare('Nu am gasit nici un algoritm potrivit pentru cazul tau. ').
```