
1. Sisteme expert

1.1. Elementele componente ale unui SE

Elementul central al prelucrării inteligente îl constituie *raționamentul artificial*, ca imitare a celui natural, efectuat de creierul uman. În orice domeniu de activitate există probleme cu un grad ridicat de dificultate, care pot fi rezolvate numai de către experți umani, formați ca specialiști în urma unei vaste experiențe în domeniul respectiv. Se pot defini sistemele expert ca reprezentând sisteme de programare bazate pe tehnici de inteligență artificială (IA), care înmagazinează cunoștințele experților umani dintr-un domeniu bine definit și apoi le folosesc pentru rezolvarea problemelor din acest domeniu.

Conceptele de bază ale unui sistem expert sunt reprezentate în figura 1-1:

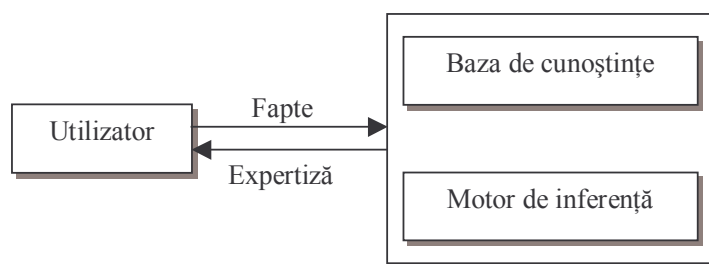


figura 1-1. Concepte de bază ale SE

Elementele principale ale unui SE sunt prezentate în figura 1-2 sau detaliat în figura 1-3, unde:

- *baza de cunoștințe* – conține ansamblul cunoștințelor specializate introduse de către expertul uman (cunoașterea din domeniul obiect). Este formată din baza de fapte și baza de reguli;
 - *baza de fapte* – conține faptele inițiale ce descriu enunțul problemei de rezolvat și rezultatele intermediare produse în cursul procedurii de deducție;
 - *baza de reguli* – reproduce raționamentul expertului;
- *motorul (mecanismul) de inferențe* – este componenta de bază a sistemului expert. Motorul de inferențe preia cunoștințele din baza de cunoștințe, realizează raționamente, indicând care reguli sunt satisfăcute de fapte, care este prioritatea acestor reguli și determină execuția regulii cu prioritatea cea mai mare;
- *modulul explicativ* – explică utilizatorului motivele raționamentului SE permițând trasarea drumului urmat în raționare de SE și generarea justificărilor pentru soluțiile obținute;

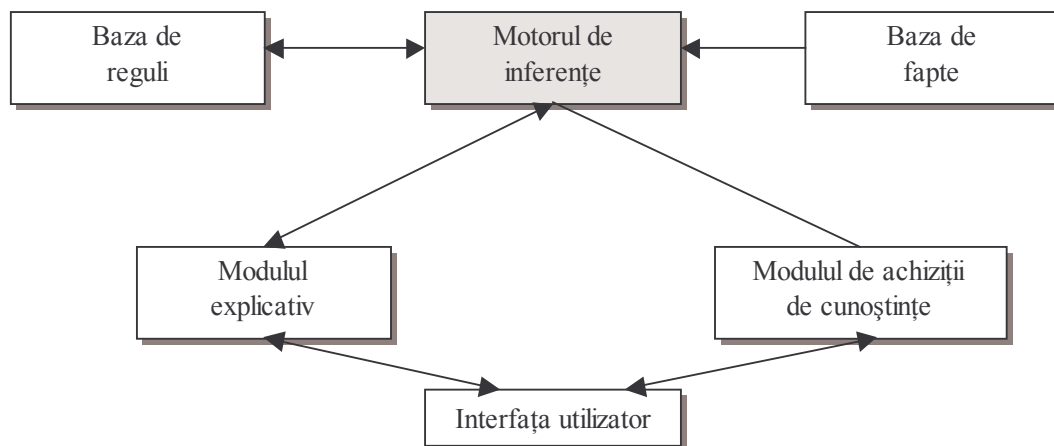


figura 1-2. Elementele principale ale unui sistem expert

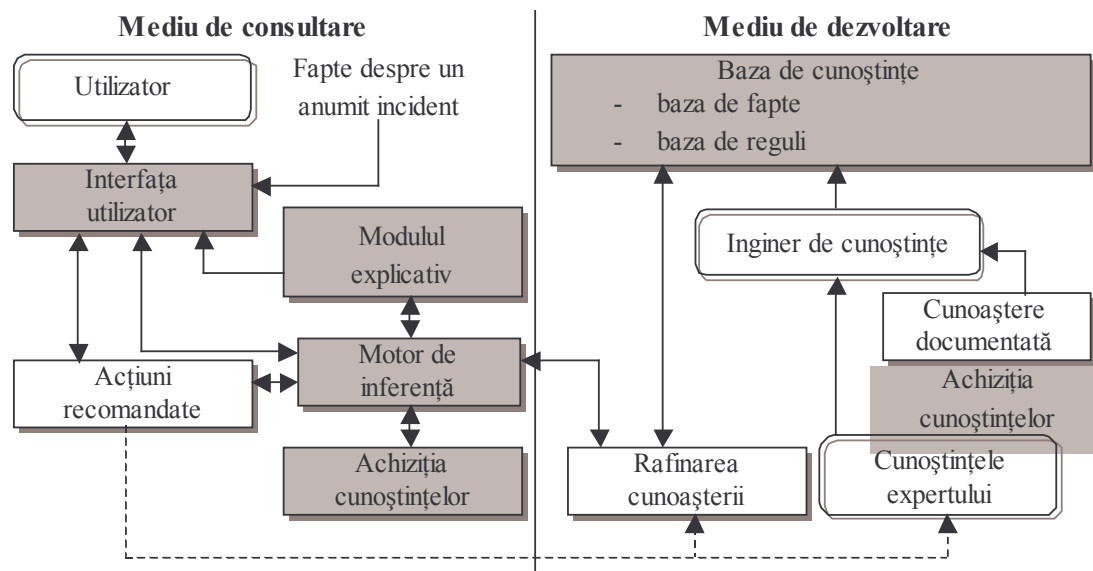


figura 1-3. Structura detaliată a unui sistem expert

- *modulul de achiziție a cunoștințelor* – preia cunoștințele specializate furnizate de expertul uman într-o formă ce nu este specifică reprezentării interne, verifică validitatea acestora, generând în final o bază de cunoștințe coerentă;
- *interfața utilizator* – permite comunicarea între SE și utilizator.

În continuare va fi descrisă fiecare componentă a unui SE:

Baza de cunoștințe

- conține ansamblul de cunoștințe specializate într-un anumit domeniu, preluate de inginerul de cunoștințe (specialistul care va face legătura dintre expertul uman și SE) de la expertul uman.
- cunoștințele reflectă "obiectele" lumii reale și relațiile dintre ele.

Baza de fapte

- conține:
 - o datele unei probleme concrete care urmează să fie rezolvată (formularea problemei),
 - o faptele rezultate în urma raționamentelor efectuate de motorul de inferențe asupra bazei de cunoștințe.

In unele publicații, baza de fapte este inclusă în baza de cunoștințe.
- una dintre cerințele esențiale la care trebuie să răspundă baza de fapte este de a reflecta cât mai fidel realitatea și de a înregistra, cu promptitudine, modificările intervenite în aceasta.
- În funcție de domeniul concret în care se utilizează și de condițiile de exploatare, faptele pot fi introduse într-un SE prin una din următoarele căi:
 - prin tastare la terminal, înaintea declanșării procesului de raționament;
 - în cursul procesului de raționament, prin chestionarea utilizatorului, caz în care, întrebările puse de SE urmează fidel traseul din procesul inferențial;
 - prin consultarea unei baze de date proprii sau aparținând altor aplicații informatice;
 - prin preluare directă de la diverși senzori.
- Baza de fapte mai este denumită și *memorie de lucru* a sistemului.

Baza de reguli

- regulile exprimă cunoștințe generale despre domeniul problemei de rezolvat. Ordinea regulilor din baza de reguli este nesemnificativă.

Motorul de inferențe

- este elementul efectiv de prelucrare în SE, care, pornind de la fapte (datele de intrare ale problemei), activează cunoștințele corespunzătoare din baza de cunoștințe, construind astfel raționamente care conduc la noi fapte (de ieșire).
- construiește un plan de rezolvare în funcție de specificul problemei, utilizând cunoștințele din domeniul respectiv.
- În urma acțiunii motorului de inferențe într-un anumit context, baza de fapte se îmbogățește fie prin adăugarea unor elemente noi, fie prin modificarea celor existente.
- În ultimă instanță, motorul de inferențe este un program care implementează algoritmi de raționament deductiv, inductiv și mixt, dar care este independent de baza de cunoștințe. Astfel, algoritmi de raționament implementați în motorul de inferențe sunt în general aceiași, indiferent de SE pe care dorim să-l dezvoltăm, dar cu toate acestea, nu se poate construi un mecanism universal de inferențe pentru toate domeniile de expertiză.

La realizarea motorului de inferențe dificultatea constă în găsirea celui mai eficient mod de declanșare a regulilor. Pentru aceasta ciclul de bază al motorului de inferență parcurge următoarele etape mari:

- *selecția* – în această etapă are loc selecția unui subansamblu al bazei de fapte și reguli de interes pentru raționament, în scopul economisirii timpului la fazele următoare;
- *filtrarea* – în cadrul acestei etape, numită filtrare (pattern matching), motorul de inferențe caută acele fapte care sunt declanșabile (sau cele care verifică un filtru particular definit în sistem) comparând premisele fiecărei reguli cu cele din baza de fapte. Regulile vor fi reținute în timpul acestei etape într-o listă numită “ansamblul

de conflicte” și care va conține acele reguli ce vor putea fi declanșate și cărora li se va aplica un criteriu de arbitraj pentru a se stabili care sunt regulile ce vor fi aplicate efectiv.

- *rezolvarea conflictelor* – în această fază, motorul alege din ansamblul de conflicte regulile ce vor fi efectiv declanșate. În numeroase sisteme expert această fază se reduce la declanșarea tuturor regulilor din ansamblul de conflicte, în ordinea în care ele se prezintă. De fapt, atunci când sunt prea multe reguli declanșate sunt deduse prea multe fapte ceea ce va conduce la declanșarea a tot mai multe reguli și astfel performanțele sistemului se degradează rapid. Pentru a se evita acest lucru se poate alege o strategie de control. Există mai multe astfel de strategii însă cele mai des utilizate se aleg:
 - în funcție de prioritate (este aleasă regula cea mai prioritară);
 - în funcție de coeficientul de încredere (varianta cea mai fiabilă).

Fiecare sistem are propria sa tehnică de rezolvare a conflictelor aleasă în funcție de domeniul pe care îl abordează sistemul expert. Această alegere poate fi făcută în mod “inteligent” prin utilizarea metaregulilor, cu alte cuvinte reguli ce permit sistemului selecționarea dinamică a regulilor de declanșat în funcție de cunoștințele ce le deține despre problema tratată;

- *execuția* – în această fază, motorul de inferențe acționează asupra părții acțiune a regulilor reținute în faza de rezolvare a conflictelor. În general, această fază constă în adăugarea de noi fapte în baza de fapte, dar, pentru unele sisteme, este posibilă extragerea faptelor din baza de fapte sau declanșarea procedurilor legate de o regulă.

Acest ciclu <filtrare - rezolvarea conflictelor - execuție> este repetat de motorul de inferențe cât timp există reguli declanșabile sau condiția de oprire nu este verificată. Când nu mai există reguli declanșabile, deci sistemul a dedus tot ce era posibil de dedus, se spune că baza de cunoștințe este saturată și ciclul se încheie.

Modulul explicativ

- are rolul de a prezenta într-o formă larg accesibilă (limbaj natural) justificarea raționamentelor efectuate de motorul de inferențe și totodată întrebările la care trebuie să răspundă utilizatorul.
- este util și expertului uman pentru verificarea coerenței bazei de cunoștințe.
- a fost dezvoltat ca o consecință a observării tendinței utilizatorilor de a folosi sistemul pe care l-au înțeles cel mai bine.

Modulul de achiziție a cunoștințelor

- are rolul de a transforma cunoștințele din forma în care le exprimă cognitivianul (inginerul de cunoștințe) în forma internă de memorare pe suport.
- asigură și interfața de comunicare cu baza de date sau alte sisteme.

Interfața cu utilizatorul

- realizează dialogul utilizatorului cu SE, în sensul specificării datelor de intrare și al furnizării rezultatelor problemei de rezolvat.
- se recomandă să se țină cont de modul de procesare al inteligenței umane, care trebuie descompusă în raționamente cât mai simple și, în același timp, simplu de modelat matematic.

Se poate observa că modelul ia în considerare și aspectul cantitativ al volumului de cunoștințe vehiculate de om în timp.

Interfața între utilizator și sistemul expert trebuie să îndeplinească următoarele funcții:

- să ofere opțiuni prin meniuri;
- să utilizeze un limbaj adecvat utilizatorului prin întrebări, comenzi și comentarii;
- să poată afișa rapid grafice, animație etc.;
- să permită intervenția rapidă și facilă a utilizatorului în orice etapă;
- limbajul de comunicare să fie cât mai apropiat de cel uman;
- să țină cont de erorile umane, să prevină și să afișeze cât mai multe mesaje de eroare;
- formatarea ecranului să fie cât mai convenabilă utilizatorului.

1.2. Reprezentarea cunoașterii

Pentru un SE, reprezentarea cunoștințelor experților în domeniul de expertiză considerat, are o importanță deosebită, deoarece un SE va fi conceput pentru un anumit tip de reprezentare a cunoștințelor și modul în care SE reprezintă cunoștințele afectează dezvoltarea, eficiența, viteza de lucru și mentenanța respectivului sistem.

Cunoștințele pot fi clasificate în:

- cunoștințe procedurale;
- cunoștințe declarative;
- cunoștințe subconștiente - care nu pot fi exprimate prin limbaj (de exemplu, cum știm să mișcăm mâna).

O altă clasificare împarte cunoștințele în:

- cunoștințe afirmative, care sunt datele primare aflate în baza de fapte;
- cunoștințe operatorii - reprezintă cunoștințele conținute în reguli și ele indică modul în care sunt folosite faptele;
- strategii (cunoștințe) de control, care sunt constituite din cunoștințele ce se referă la modul de rezolvare a problemei și care indică ordinea în care se aplică regulile.

Prelucrarea cunoștințelor impune definirea de structuri pentru memorare și prelucrare care să facă posibilă efectuarea de raționamente. În consecință, pentru stocarea și utilizarea cunoștințelor se folosesc *structuri de cunoștințe*, tot așa cum pentru stocarea și prelucrarea datelor se utilizează structuri de date.

Integrarea cunoștințelor într-un formalism de reprezentare ridică o serie de probleme specifice. Există, în primul rând, mari diferențe în privința statutului informațiilor prelucrate (ele pot fi intangibile sau modificabile, certe sau incerte, valide sau perimate, empirice sau teoretice), diferențiere la care formalismul de reprezentare trebuie să răspundă cât mai bine.

Cunoștințele

- sunt, în marea majoritate a cazurilor, *incomplete*, fie deoarece, fiind implicite pentru expertul uman, sunt omise în reprezentarea pe calculator, fie datorită faptului că sunt dificil de transmis sau de formulat.
- se schimbă o dată cu modificările survenite în domeniul în care sunt utilizate.
- exprimă, în general, informații specifice unor clase sau grupuri de "obiecte" materiale sau conceptuale.

1.2.1. Metode de reprezentare a cunoașterii

1.2.1.1. Introducere

Pentru a fi utilizată de un program, cunoașterea este memorată sub forma unor *piese de cunoaștere* care descriu obiectele, faptele, fenomenele, procesele și evenimentele din domeniul de expertiză.

O arhitectură de reprezentare a cunoștințelor poate fi considerată ca fiind structurată pe trei nivele:

- *nivelul intern*, care este constituit din schema internă ce descrie structura de stocare fizică a cunoștințelor în baza de cunoștințe. Astfel, sunt descrise detaliile complete ale stocării, precum și modul de acces la cunoștințe;
- *nivelul conceptual* - descrie complet structura întregii baze de cunoștințe, ascunzând detaliile legate de stocarea fizică. La acest nivel, descrierea bazei de cunoștințe se concentrează asupra descrierii entităților, tipurilor de date, relațiilor dintre ele, precum și a restricțiilor asociate;
- *nivelul extern sau nivelul utilizator* - include o colecție de scheme externe ce descriu baza de cunoștințe prin prisma diferiților utilizatori, fiecare descriind baza de cunoștințe prin prisma propriilor interese.

1.2.1.2. Reprezentarea cunoștințelor în limbajul calculului cu predicate de ordin I

Utilizarea logicii simbolice ca model de reprezentare a cunoștințelor:

- este importantă deoarece oferă o abordare formală a raționamentului, cu fundamente teoretice riguroase.
- formalismul logic permite derivarea unor cunoștințe noi, plecând de la cele existente, pe baza deducției și a demonstrării teoremelor, facilitând automatizarea proceselor de raționament și execuția inferențelor corecte și logic valide.
- logica simbolică este suficient de expresivă și flexibilă pentru a permite reprezentarea cu acuratețe a cunoștințelor problemei de rezolvat.
- această metodă de reprezentare este indicată în cazul în care piesele de cunoaștere și componentele acestora sunt implicate în probleme cu numeroase inferențe.

Logica predicatelor aprofundează studiul propozițiilor, prin descompunerea propozițiilor elementare în părțile lor constitutive, logica propozițională constituind un caz particular al logicii cu predicate de ordinul I.

În calculul propozițional, propozițiile sunt presupuse a fi bine alcătuite după reguli de formare care nu aparțin limbajului calculului propozițional, ci limbajului natural. Pentru rafinarea cunoașterii, astfel încât să poată fi surprinsă semnificația componentelor fiecărei propoziții, se poate considera următoarea fază, și anume reprezentarea predicativă. Pentru această etapă trebuie făcute următoarele *observații*:

- orice propoziție are structura gramaticală de forma subiect - predicat, sau, considerând termeni mai apropiați de obiectivul urmărit în procesul de reprezentare, de forma grup nominativ- grup predicativ;
- se poate considera grupul predicativ ca având semnificația abstractizată printr-un simbol de predicat, în ale cărui locuri sunt plasate obiectele din grupul nominal;
- unele componente ale propoziției au rolul de a modifica valoarea de adevăr în funcție de acoperirea domeniului în care variabilele iau valori, dând astfel o semnificație cantitativă care este surprinsă prin cuantificarea universală sau existențială a variabilelor.

De exemplu, dacă în propoziția "Traductorul este defect" se elimină subiectul, se ajunge la o formulare de forma: "... este defect". Expresiile de acest tip, cu un singur element liber sunt denumite *predicate* sau proprietăți. Expresiile cu două sau mai multe elemente libere sunt denumite *relații*. Relațiile pot fi concepute și ca predicate. Dacă în poziția elementului liber se introduce o variabilă, atunci în funcție de valorile care i se atribuie acesteia, se pot obține propoziții care să fie întotdeauna adevărate, uneori adevărate și alteori false sau întotdeauna false. Prin urmare, tratarea unor asemenea expresii impune ca, alături de modalitățile de legare a propozițiilor (*și* \wedge , *sau* \vee , *negație* \neg , *implicație* \Rightarrow , *echivalență* \Leftrightarrow) să se ia în considerare și *cuantificatorul universal* (\forall - pentru orice valori) și *cuantificatorul existențial* (\exists - există cel puțin o valoare). Utilizarea cuantificatorilor este legată de transcrierea în calculul cu predicate de ordinul întâi a unor propoziții în care acești cuantificatori acționează ca operatori unari asupra variabilelor pe care le prefixează. Folosirea predicatelor pentru a rezolva problema de apartenență la o mulțime pentru entități, care sunt, la rândul lor, termeni ai altor predicate, are dezavantajul de a conduce, în caz de substituție, la predicate de ordin superior.

În acest context, reprezentarea cunoștințelor se poate face în două moduri:

- pe baza logicii predicatelor de ordinul unu, în care utilizarea variabilelor în formularea faptelor și a regulilor este permisă numai pentru subiecte;
- pe baza logicii predicatelor de ordinul doi, în care utilizarea variabilelor este permisă atât pentru subiecte, cât și pentru predicate.

Se poate spune că în acest mod de reprezentare piesele de cunoaștere sunt descrise prin expresii ale căror componente sunt formule ale limbajului. O piesă de cunoaștere exprimată în limbaj natural este descompusă în propoziții elementare adecvate numite *asertiuni* ce specifică fapte (proprietăți, relații legate de piesa de cunoaștere). Ținând cont de definițiile anterioare se poate defini o propoziție elementară ca fiind generată de un predicat cu un număr finit de locuri în care se specifică variabile formale sau obiecte în mulțimea suport.

Acest mod de reprezentare este constituit din două etape:

- reprezentarea propozițională, în cadrul căreia piesele de cunoaștere sunt descompuse în asertiuni legate prin conectivele logice specifice calculului propozițional;
- reprezentarea predicativă, în care fiecare asertiune este descompusă în componentele sale, predicate sau obiecte.

Calculul cu predicate reprezintă un model al raționării cu două valori, model ce implică analiza conexiunilor între clauze. Modul de raționare este asemănător raționării umane.

Setul obiectelor ce reprezintă variabile în propoziție poartă denumirea de *universul discursului*.

Cea mai răspândită modalitate de reprezentare a regulilor de inferență folosește *forma clauzală* a expresiilor în limbajele de ordinul întâi.

Se numește *clauză*, o expresie de forma:

$$A_1, A_2, \dots, A_m \mid - B_1, \dots, B_n$$

în care A_1, A_2, \dots, A_m reprezintă *antecedentul*, iar B_1, \dots, B_n reprezintă *succedentul* clauzei respective.

Clauza este o expresie echivalentă cu formula:

$$(A_1 \wedge A_2 \wedge \dots \wedge A_m) \rightarrow (B_1 \vee \dots \vee B_n)$$

din calculul propozițional și din calculul cu predicate de ordinul întâi, în care A_i și B_j (cu $1 \leq i \leq m$, $1 \leq j \leq n$) sunt formule bine formate în aceste limbaje.

Cea mai răspândită formă de reprezentare clauzală este aceea care conține o singură formulă în succedent, de tipul:

$$A_1, A_2, \dots, A_m \vdash B,$$

cunoscută sub denumirea de *clauză Horn*.

Forma generală a regulilor de inferență este:

$$\frac{A_1, \dots, A_m}{B_1, \dots, B_n} \quad (m \geq 1, n \geq 1)$$

în care A_1, A_2, \dots, A_m sunt clauzele care alcătuiesc *premise* regulii, iar B_1, \dots, B_n sunt clauzele care alcătuiesc *concluzia*.

Principalul *avantaj* al metodei constă în faptul că piesele de cunoaștere pot fi introduse direct în sistemul rezolutiv pentru a fi folosite la efectuarea de inferențe, dacă și acest sistem este conceput pe baza unor mecanisme inferențiale definite în calculul cu predicate de ordinul întâi (cum este cazul, de exemplu, cu diferitele metode de rezolvare bazate pe demonstrarea automată a teoremelor).

Unul din *dezavantajele* metodei este reprezentat de implementarea dificilă în practică a acestei metode, datorită lucrului cu elemente de logică. De asemenea, un alt dezavantaj îl constituie captarea trăsăturilor relevante ale pieselor de cunoaștere în predicate.

1.2.1.3. Reprezentarea cunoștințelor prin rețele de producție

Modelul regulilor de producție a fost utilizat pentru prima dată în cadrul sistemelor expert în sistemele DENDRAL și MYCIN.

Această metodă este indicată a se folosi în situațiile în care cunoștințele pot fi modularizate în segmente mici, relativ independente și în care conținutul cunoștințelor este procedural (cunoștințe de tip *cum*). De asemenea, se recomandă această reprezentare în cazul în care se constată existența condiționărilor sau a restricțiilor în utilizarea pieselor de cunoaștere, exprimate în vederea restrângerii spațiului problemei la acele elemente care reprezintă candidații valabili la participarea în procesele inferențiale. În momentul de față acest mod de reprezentare a cunoștințelor este unul dintre cele mai utilizate în cadrul sistemelor expert.

Cunoașterea în rețelele (sistemele) de producție este de natură *procedurală* și se pot defini următoarele componente:

- *cunoaștere declarativă* sau *factuală*, ce reprezintă piese de cunoaștere stocate sub forma unor structuri de date într-o colecție numită *context*;
- *cunoaștere procedurală*, care este reprezentată sub forma regulilor de producție, de tip *condiție - acțiune*, reguli ce formează *baza de reguli*;
- *cunoașterea strategică* sau *de control*, formată din reguli ce privesc secvențele de acțiuni în procesul de rezolvare.

Reprezentarea regulilor de producție se bazează pe logica propozițiilor, astfel încât atât faptele cât și regulile pot conține numai entități invariabile (constante). Datorită limitărilor inerente unei asemenea soluții, s-a trecut la o altă modalitate de reprezentare, bazată pe logica predicatelor, în care faptele și regulile pot include entități generice, conferindu-le astfel un grad mai ridicat de generalitate. Deoarece entitățile generice sunt specificate prin intermediul variabilelor, această metodă de reprezentare este denumită, prin extensie de limbaj, reprezentare prin reguli de producție cu variabile.

Se consideră sistemul de producții ca fiind compus dintr-o bază de date și un set de reguli. Ordinea în care regulile sunt introduse și stocate în sistem este nesemnificativă. Condițiile unei reguli pot fi considerate ca o bază de date, ce returnează un indicator de succes sau eroare. Concluzia unei reguli este o acțiune ce manipulează date din baza de date, și în plus, o strategie de control a sistemului determină secvența regulilor utilizate.

Baza de date este constituită dintr-un set de termeni, iar o regulă de producție este formată din două componente: partea stângă a regulii (numită și antecedent, premisă, condiție sau situație) – notată LHS și partea dreaptă a regulii (numită și consecință, concluzie, acțiune sau răspuns) – notată RHS. Legătura logică între partea stângă și partea dreaptă a regulii este implicația, în sensul că adevărul părții stângi determină adevărul părții drepte a regulii, deci $LHS \rightarrow RHS$. O regulă de producție are forma generală *IF c THEN t*, (tradus în limba română, *DACĂ c ATUNCI t*), în care condiția *c* este constituită din termeni, paranteze, conective, \vee , \wedge , \neg iar concluzia *t* este formată dintr-un singur termen. În general, nu se permite o conjuncție de termeni într-o concluzie.

Exemple:

- R1: **daca** Coco zboara
si Coco are pene
atunci Coco este pasare.
- R2: **daca** pacientul are temperatura mare
si tipul organismului este gram-pozitiv
si pacientul are gitul uscat
atunci organismul este streptococ
- R3: **daca** masina nu porneste
si farurile nu se aprind
atunci bateria este consumata
sau bornele bateriei nu fac contact
- R4: **daca** temperatura > 95° C
atunci deschide valva de protectie

Formularea regulilor și a faptelor din exemplele date s-a făcut în limbaj natural. Cele mai multe limbaje bazate pe reguli de producție au o sintaxa fixă care, deși apropiată de limbajul natural, definește un limbaj independent de context. Utilizând o astfel de sintaxă, exemplele de reguli R2÷R4 vor fi de fapt exprimate într-un posibil limbaj de reprezentare astfel:

- R2': **daca** Temperatura-Pacient = mare
si Tip-Organism = gram-pozitiv
si GitUscat-Pacient
atunci Identitate-Organism = streptococ

R3': **daca** NuPorneste-Masina
si NuAprinde-Masina = far
atunci Stare-Baterie = consumata
sau Borne-Baterie = fara-contact

R4': **daca** Temperatura > 95
atunci DeschideValva

Probleme ale reprezentării cunoștințelor prin reguli

Operatorul SAU în ipoteză

Se recomandă transformarea regulilor ce conțin operatorul *SAU* în ipoteză prin mai multe reguli ce sunt în forma conjunctivă. Acest lucru este totdeauna posibil, deoarece în logica propozițională formulele:

- (F1) $(P_1 \text{ sau } P_2 \text{ sau } \dots \text{ sau } P_n) \rightarrow C$
(F2) $(P_1 \rightarrow C) \text{ sau } (P_2 \rightarrow C) \text{ sau } \dots \text{ sau } (P_n \rightarrow C)$

sunt echivalente. În general, o regulă de forma:

- (R) **dacă** f_{11} **și** f_{12} ... **și** f_{1n} **sau**
 f_{21} **și** f_{22} ... **și** f_{2n2} **sau** ...
 f_{m1} **și** ... **și** $f_{m,nm}$
atunci C

se expandează în mai multe reguli de forma:

- (R₁) **dacă** f_{11} **și** f_{12} **și** ... **și** $f_{1,n1}$ **atunci** C
.
.
.
(R_{m1}) **dacă** f_{m1} **și** f_{m2} **și** ... **și** $f_{m,nm}$ **atunci** C

Operatorul SAU în concluzie

Se consideră regula:

- (Rx) **dacă** *lipsă_intrare* **atunci** *Scos_din_funcție* **sau** *Avarie*

Se ridică în mod evident problema: cărui fapt din concluzie să-i fie atribuită valoarea adevărat? Și cum să se rezolve această situație din punct de vedere informatic? Există soluțiile:

- se dublează memoria de lucru, și într-o copie a acesteia se consideră un fapt, iar în cealaltă, cel de al doilea fapt. Se efectuează procesul inferențial separat pentru fiecare din aceste copii, utilizatorul trebuind să decidă în final asupra uneia din variantele de rezultat propuse de sistem;
- se consideră mai întâi faptul *Scos_din_funcție* ca fiind adevărat, se continuă procesul inferențial, apoi se revine și se adaugă *Avarie* = adevărat.

Ambele variante sunt complicate. Din acest motiv cele mai multe SE comerciale nu acceptă operatorul **sau** în concluzie. Rezolvarea o poate da inginerul de cunoștințe cu ajutorul unui fapt suplimentar, în cazul considerat, de exemplu *Nefuncțional*:

dacă *Nefuncțional* **și not** *Scos_din_funcție* **atunci** *Avarie*

dacă *Nefuncțional* **și not** *Avarie* **atunci** *Scos_din_funcție*

dacă *Scos_din_funcție* **atunci** *Nefuncțional*

dacă *Avarie* **atunci** *Nefuncțional*

dacă **not** *Scos_din_funcție* **și not** *Avarie* **atunci not** *Nefuncțional*

Primele două reguli sunt cele mai necesare. În general, această problemă trebuie evitată prin introducerea unor concepte de o semnificație cunoscută.

Utilizarea regulilor de producție prezintă următoarele *avantaje* din punct de vedere al modelării cunoștințelor în sistem:

- separarea cunoștințelor generale despre problemă, de datele specifice unei instanțe a problemei de rezolvat;
- partiționarea cunoștințelor în unități de cunoștințe independente, facilitând, astfel, dezvoltarea incrementală a bazei de cunoștințe;
- posibilitatea menținerii a două forme de expresie a regulilor: o formă internă sistemului, adecvată procesului de rezolvare și o formă externă, apropiată limbajului natural, pentru interfața utilizatorului cu sistemul.

1.2.1.3.1. Mecanismul interpretativ al regulilor de producție

Dacă într-o regulă condiția este satisfăcută, se spune că regula este selectată pentru declanșare sau regula este *aplicabilă*. Regulile aplicabile sunt introduse într-o mulțime a regulilor aplicabile, mulțime din care este selectată regula cu cea mai mare prioritate.

Mecanismul interpretativ al regulilor de producție, prezentat în figura 1-4, conține următoarele etape:

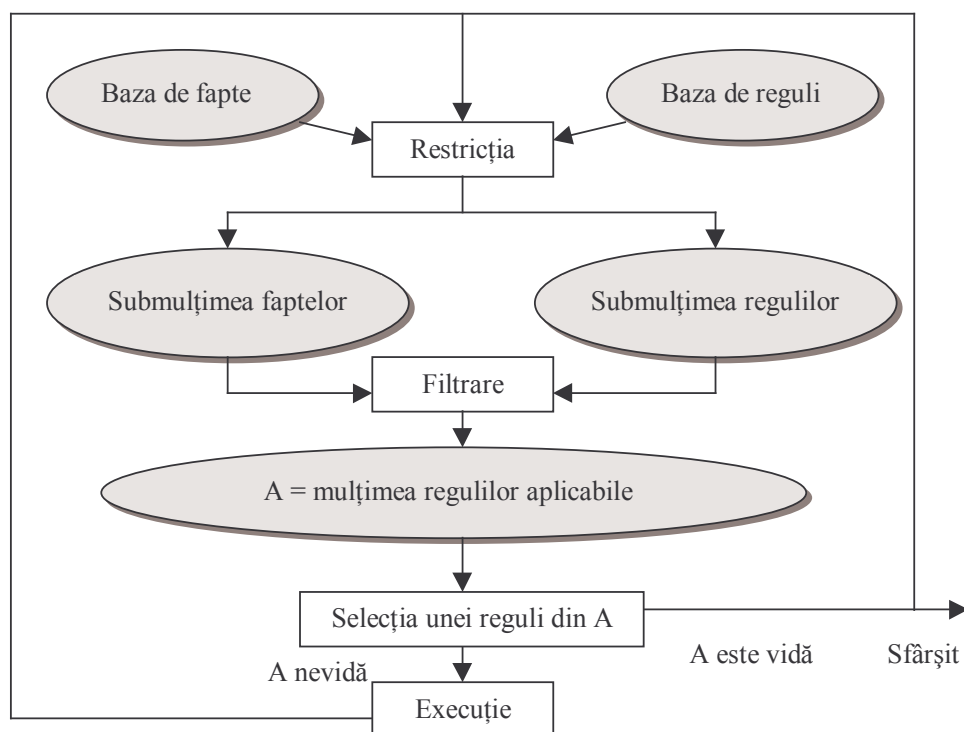


figura 1-4. Mecanismul interpretativ al regulilor de producție

- faza de *restricție* - reprezintă selecția unui subansamblu al bazei de fapte și reguli de interes. Selecția are ca efect reducerea considerabilă a timpului pentru fazele următoare;
- faza de *filtrare* (pattern matching), ce are ca efect comparația între partea de premisă a regulii considerate și faptele bazei de fapte pentru determinarea regulilor aplicabile. În urma acestei etape pot rezulta una, mai multe sau nici o regulă declanșabilă. Dacă nu se obține nici o regulă declanșabilă, atunci rezultă o situație de eșec, care trebuie explicată, sau în care utilizatorul trebuie să răspundă la o serie de întrebări puse de către SE, în scopul completării formulării problemei. De asemenea, se poate face observația conform căreia aplicarea după faza de selecție a fazei de filtrare ajută la reducerea substanțială a ansamblului regulilor ce sunt filtrate cu elemente din baza de fapte;
- faza de *rezolvare a conflictelor* are ca obiectiv alegerea acelor reguli ce sunt aplicate efectiv. Problema este rezolvată printr-o strategie ce poate fi foarte simplă în raport cu contextul, sau mai complexă ținând cont de context în sensul aplicării celei mai promițătoare reguli. Printre principalele criterii de alegere care pot fi utilizate în această etapă, se pot aminti: prima regulă din listă, cea mai complexă regulă (cu cel mai mare număr de fapte în premisă), cea mai utilizată regulă sau cea mai prioritară (regula selectată conform criteriului priorității), regula cea mai fiabilă (regulă selectată conform coeficientului de încredere);
- faza de *execuție* constă în aplicarea regulilor ce au fost selectate în faza anterioară, acțiunea constând, în general, din adăugarea de noi fapte în baza de fapte. Este posibil ca aplicarea regulii să facă apel la proceduri externe având ca

- efect modificări ale bazei de fapte sau/și formularea de întrebări către utilizator sau execuția acțiunii indicate de partea dreaptă a regulii;
- pornind de la contextul modificat în urma aplicării regulilor anterioare, se reia ciclul începând cu faza de restricție, atât timp cât în cadrul ciclului are loc modificarea contextului.

Oprirea mecanismului interpretativ poate avea loc în cazul în care acțiunea unei producții specifică concret oprirea, sau dacă se selectează o producție vidă.

Etapa de selecție este cea care stabilește de fapt forma de cautare utilizată de sistemul bazat pe reguli, deci strategia de control. *Strategia de control* este un element important al ciclului de inferență al unui sistem bazat pe reguli. Într-un sistem bazat pe reguli strategia de control are două componente: *stabilirea criteriului de selecție a regulilor* din mulțimea de conflicte și *direcția de aplicare a regulilor*: înlăturarea înainte sau înlăturarea înapoi a regulilor.

Rezolvarea unei probleme de către un sistem bazat pe reguli este de fapt un proces de cautare a soluției în care operatorii sunt reprezentați de regulile sistemului. În consecință, toate tehnicile de cautare prezentate în cursul de IA pot fi aplicate în cazul sistemelor bazate pe reguli. Din punct de vedere al direcției de aplicare a regulilor, înlăturarea înainte a regulilor corespunde unei reprezentări a soluției problemei prin spațiul stărilor, iar înlăturarea înapoi a regulilor corespunde unei reprezentări prin grafuri SI/SAU a soluției problemei. Rezolvarea conflictelor se referă la ordinea de selecție și preferarea unui operator față de alți operatori aplicabili într-un context dat.

La fel ca în orice problemă de cautare, strategia unui sistem bazat pe reguli poate fi irevocabilă, i.e. fără posibilitatea revenirii în stări anterioare, sau tentativă, i.e. se aplică regula dar se menține informația necesară unei posibile revenirii în punctul anterior aplicării regulii. De asemenea, strategia sistemului poate avea un grad de informare mai mare sau mai mic, dacă există cunoștințe suficiente pentru a alege regulile potrivite, sau poate fi complet neinformată, caz în care selecția se face conform unei ordini stabilite a priori.

Costul computațional al rezolvării unei probleme utilizând reguli de producție implică, pe lângă costul controlului și cel al aplicării regulilor, și costul procesului de identificare. În urma studiilor efectuate, s-a observat că identificarea este etapa cea mai consumatoare de timp din ciclul de inferență al unui sistem bazat pe reguli.

Criterii de selecție a regulilor

Rezultatul etapei de identificare este mulțimea de conflicte, deci mulțimea tuturor regulilor care au identificat cu descrierea stării curente a rezolvării problemei, descriere conținută în memoria de lucru. Rezolvarea conflictelor din etapa de selecție are rolul alegerii uneia sau mai multor reguli care vor fi aplicate. Există diverse criterii de selecție, de exemplu:

(a) *Selecția primei reguli aplicabile*

Considerând ordinea fizică a regulilor din baza de cunoștințe, se alege prima regulă aplicabilă, deci prima care a identificat, și se aplică acea regulă. În acest caz, nu se creează de fapt o mulțime de conflicte, regimul de control fiind un control numit *focalizarea atenției*. Această strategie este aplicată de exemplu, de sistemul Prolog și corespunde unei strategii de tip "backtracking".

(c) *Aplicarea tuturor regulilor din mulțimea de conflicte*

O astfel de strategie aplică toate regulile din mulțimea de conflicte și produce mai multe stări care vor fi memorate și prelucrate independent. Ea se numește strategie de tipul "*încearcă toate regulile*" și poate fi aplicată în cazul în care se poate evita un cost ridicat al exploziei

combinationale. Criteriul "incearca toate regulile" se aplica, in general, in cazul sistemelor bazate pe reguli cu rationament incert. In acest tip de sisteme, toate datele (faptele) au asociat un coeficient de certitudine care indica increderea sistemului in acele valori, iar sistemul calculeaza noi coeficienti de certitudine pentru datele nou inferate. Executia unor secvente de reguli diferite, pornind de la aceeasi stare, poate duce la deductia unor date diferite, fiecare avind insa asociat un alt coeficient de certitudine. Un astfel de sistem bazat pe reguli este sistemul MYCIN.

Directia de aplicare a regulilor

A doua componenta a strategiei de control a unui sistem bazat pe reguli este directia de aplicare a regulilor. Regulile pot fi aplicate utilizind *inlantuirea inainte* prin identificarea partii stingi a regulii cu memoria de lucru, sau utilizind *inlantuirea inapoi* prin identificarea partii drepte a regulii cu memoria de lucru si incercarea satisfacerii partii stingi a regulii. Cele doua abordari sint prezentate sintetic in Figura 4.2.

INLANTUIRE INAINTE INLANTUIRE INAPOI

daca A atunci B	determina C
daca B atunci C	daca B atunci C
<u>A (data)</u>	<u>daca A atunci B</u>
C (concluzie)	(daca A atunci C , implicit)
	Este A adevarata? (data)

Figura 4.2 Inlantuirea executiei regulilor de productie

Exemplu. Fie urmatorul set de reguli de productie:

- R1: **daca** A
si B
atunci C
- R2: **daca** C
atunci D
- R3: **daca** E
atunci B
- R4: **daca** A
si E
atunci C

Memoria de lucru contine initial faptele A si E care reprezinta datele de caz ale instantei problemei de rezolvat, si starea scop D, D reprezentind solutia cautata.

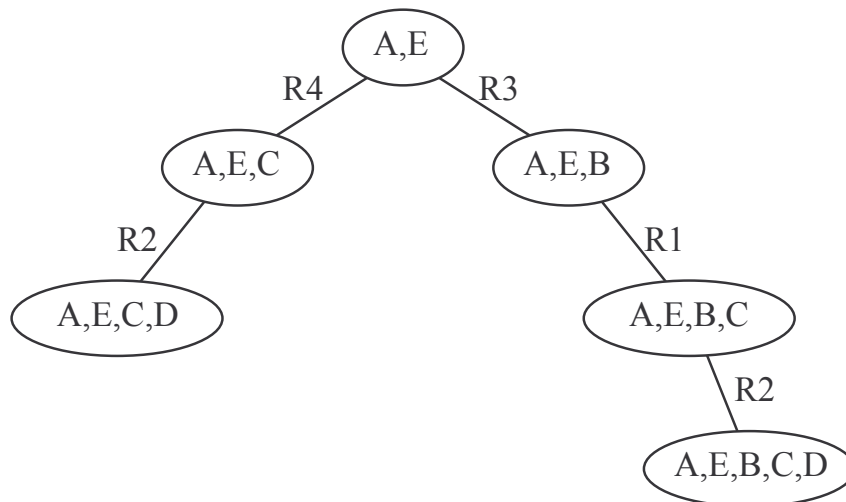
Aplicind inlantuirea inainte a regulilor se poate gasi secventa de reguli R3, R1, R2, prezentata in Figura 4.3 (a), care produce in memoria de lucru stare scop D cautata. Considerind toate regulile aplicabile la un moment dat, exemplul genereaza arborele de cautare in spatiul starilor din Figura 4.3 (b). Se observa ca starea initiala, definita prin continutul A, E al memoriei de lucru, creeaza multimea de conflicte {R4, R3}. Fiecare aplicare de regula, pe o cale de cautare, va adauga noi fapte la memoria de lucru astfel incit, in final, aceasta contine fie secventa A, E, C, D fie A, E, B, C, D. Din spatiul de cautare al solutiei s-au eliminat aplicari de reguli care nu modifica continutul memoriei de lucru, de exemplu nu s-a figurat regula R4 care s-ar fi putut aplica secventei A, E, C dar care nu ar fi produs nici o schimbare.

Continut initial al memoriei de lucru: A,E

Stare scop: D

$$A,E \xrightarrow{R3} A,E,B \xrightarrow{R1} A,E,B,C \xrightarrow{R2} A,E,B,C,D$$

(a)



(b)

Figura 4.3 Inlantuirea inainte a regulilor de productie

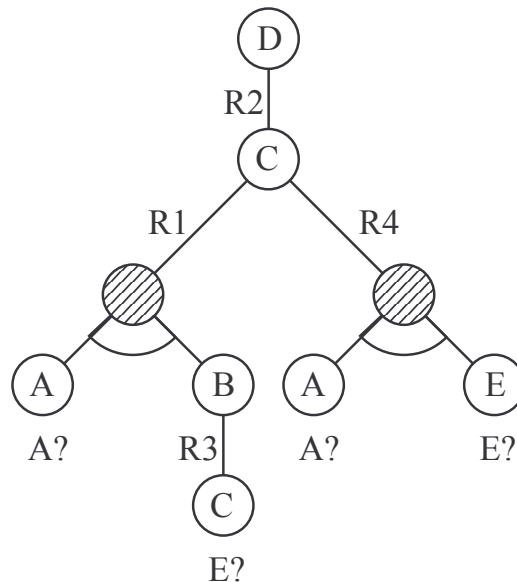
Aplicind inlantuirea inapoi a regulilor de productie, se incearca satisfacerea scopului D prin aplicarea unei reguli care mentioneaza in concluzie D. O astfel de regula este R2. Pentru ca R2 sa poata fi executata trebuie ca memoria de lucru sa contina faptele care satisfac premisa regulii. Pentru aceasta trebuie indeplinite scopurile din premisa, deci cautate regulile care refera aceste scopuri in concluzie. In cazul in care nu exista astfel de reguli, fie faptele sint deja in memoria de lucru, si regula poate fi direct aplicata, fie, in caz contrar, se solicita utilizatorului informatii despre adevarul acestor fapte. O posibila secventa de satisfacere a scopului D prin aplicarea inlantuirii inapoi a regulilor este prezentata in Figura 4.4 (a). Daca se considera toate posibilitatile de satisfacere a scopului D se obtine arborele SI/SAU din Figura 4.4 (b).

Continut initial al memoriei de lucru: A,E

Stare scop: D

$D? \xrightarrow{R2} C? \xrightarrow{R1} A?, B? \xrightarrow{R3} A?, E?$

(a)



(b)

Figura 4.4 Inlantuirea inapoi a regulilor de productie

Sisteme cu inlantuire inainte a regulilor

Sisteme cu inlantuire inapoi a regulilor

Intr-un sistem cu inlantuire inapoi a regulilor de productie functionarea se porneste de la scopul de demonstrat sau de aflat. De obicei acest scop este aflarea valorii unui obiect sau a unui atribut al unui obiect.

Se considera un sistem bazat pe reguli care functioneaza cu inlantuire inapoi a regulilor. Baza de cunostinte contine urmatoarele reguli:

- R1: **daca** X are par
atunci X este mamifer
- R2: **daca** X hraneste puii cu lapte
atunci X este mamifer
- R3: **daca** X este mamifer
si X are dinti ascutiti
si X are falci
atunci X este carnivor
- R4: **daca** X este carnivor

si X este maroniu
si X are pete
atunci X este leopard

R5: **daca** X este carnivora
si X este maroniu
si X are dungii
atunci X este tigru

Scopul de demonstrat este "Ce este X?". Pentru acesta se caută regulile care referă în concluzie tipul lui X; acestea sînt R5 și R4. Se încearcă aplicarea regulii R5, pentru care se încearcă aplicarea regulii R3, pentru care se încearcă aplicarea regulilor R1 și R2. Deoarece nu mai există reguli care să refere atributele premiselor regulilor R1 și R2, se întreabă dacă X are par și la răspunsul afirmativ al utilizatorului, se îndeplinește regula R1 și se revine la validarea ipotezelor regulii R3. Presupunind că utilizatorul răspunde în consecință, se încearcă satisfacerea restului condițiilor regulii R4. Dacă utilizatorul răspunde că X este maroniu și are dungii, regula R4 nu este satisfăcută, dar se îndeplinește regula R5, deci tipul lui X este tigru. Se observă că în cazul în care nu există nici o regulă care să refere în concluzie un atribut din ipoteza regulii curente de verificat, se întreabă utilizatorul. Sistemul ar fi putut funcționa și cu introducerea a o parte (sau toate) din datele inițiale ale problemei în memoria de lucru; în acest caz utilizatorul ar fi fost întrebat numai despre valorile de atribut care nu sînt prezente în memoria de lucru.

În continuare se prezintă un exemplu de program bazat pe reguli cu înlănțuire înapoi care folosește strategia de tip "încearcă toate regulile". Un sistem care folosește o astfel de strategie se mai numește și *sistem cu acumulare de probe*. Se pune problema unei decizii gastronomice în care se cere alegerea vinului potrivit pentru un meniu cu componente specificate de utilizator. Această problemă prezintă o caracteristică întâlnită și în alte probleme de decizie sau diagnosticare, cum ar fi diagnosticul medical, și anume existența unei incertitudini asupra valorilor corecte de atribute. De exemplu, dacă meniul conține sos alb, se poate combina atît un vin sec cit și un vin demisec. Pentru a modela această incertitudine, se asociază valorilor din sistem o măsura a încrederii, numită *factor de certitudine* sau *coeficient de certitudine*, notat cu CF.

Cunostintele sînt reprezentate sub forma de obiect-atribut, și valorile asociate atributelor, și sub forma de reguli care referă obiectele și atributele din baza de cunostinte. Această înseamnă că, pe lîngă reguli, baza de cunostinte conține fapte reprezentate sub forma de triplete atribut-obiect-valoare. Atributele obiectelor pot fi de două tipuri: monovaloare și multivaloare. Un atribut monovaloare este un atribut care, în final, va avea o singură valoare asociată, de exemplu culoarea vinului. Este evident că un vin nu poate avea mai multe culori în același timp. Atributele multivaloare sînt atributele care în final pot avea mai multe valori, toate admisibile. De exemplu, atributul vin poate avea mai multe valori (chardonnay, riesling), interpretarea acestor valori fiind aceea că sistemul a indicat mai multe posibilități de vinuri care se potrivesc la meniul indicat.

Exemplul prezentat este un model tipic de raționament incert sau statistic. Raționamentul incert implementat folosește coeficienții de certitudine asociați faptelor în două moduri:

- Valoarea fiecărui atribut este memorată împreună cu coeficientul de certitudine asociat, coeficientul de certitudine indicînd încrederea sistemului în acea valoare. Coeficienții de certitudine sînt valori pozitive în intervalul $[0,1]$. De exemplu, (vin chardonnay 0.8 riesling 0.6) indică faptul că vinul potrivit este Chardonnay cu încrederea 0.8 și Riesling cu încrederea 0.6.

- O regula poate avea asociat un coeficient de certitudine care indica increderea sistemului in concluzia regulii, in cazul in care premisa este adevarata. De exemplu, regula:

daca sos-meniu = sos-alb
atunci culoare-vin = alba 0.6

indica increderea de 0.6 in faptul ca un vin alb se potriveste unui meniu cu sos alb. Daca regulile nu au un coeficient de certitudine asociat, acesta se considera implicit 1, indicind incredere totala in concluzie.

Continutul bazei de cunostinte a problemei deciziei gastronomice este prezentat in continuare, utilizind o sintaxa asemanatoare celei din sistemului bazat pe reguli M1.

R11: **daca** componenta-meniu = curcan
atunci culoare-vin = rosie 0.7
si culoare-vin = alba 0.2

R12: **daca** componenta-meniu = peste
atunci culoare-vin = alba

R13: **daca** sos-meniu = sos-alb
atunci culoare-vin = alba 0.6

R14: **daca** componenta-meniu = porc
atunci culoare-vin = rosie

R21: **daca** sos-meniu = sos-alb
atunci tip-vin = sec 0.8
si tip-vin = demisec 0.6

R22: **daca** sos-meniu = sos-tomat
atunci tip-vin = dulce 0.8
si tip-vin = demisec 0.5

R23: **daca** sos-meniu = necunoscut
atunci tip-vin = demisec

R24: **daca** componenta-meniu = curcan
atunci tip-vin = dulce 0.6
si tip-vin = demisec 0.4

R31: **daca** culoare-vin = rosie
si tip-vin = dulce
atunci vin = gamay

R32: **daca** culoare-vin = rosie
si tip-vin = sec
atunci vin = cabernet-sauvignon

R33: **daca** culoare-vin = rosie
si tip-vin = demisec

atunci vin = pinot-noir

R34: **daca** culoare-vin = alba
si tip-vin = dulce
atunci vin = chenin-blanc

R35: **daca** culoare-vin = alba
si tip-vin = sec
atunci vin = chardonnay

R36: **daca** culoare-vin = alba
si tip-vin = demisec
atunci vin = riesling

scop(vin) /* se indica atributul a carei valoare trebuie aflata */
 monovaloare(componenta-meniu)
 monovaloare(culoare-vin)
 monovaloare(sos-meniu)
 multivaloare(tip-vin)
 multivaloare(vin)
 valori-legale(componenta-meniu) = [curcan, peste, porc] /* domeniul de valori */
 valori-legale(sos-meniu) = [sos-alb, sos-tomat]
 valori-legale(tip-vin) = [sec, demisec, dulce]
 valori-legale(vin) = [gamay, cabernet-sauvignon, pinot-noir,
 chenin-blanc, chardonnay, riesling]
 valori-legale(culoare-vin) = [rosie, alba]

Se considera existenta unui interpretor de reguli pentru aceasta baza de cunostinte, interpretor care lucreaza cu inlantuirea inapoi a regulilor si acumulare de probe. Pe parcursul rezolvarii problemei un atribut monovaloare poate avea mai multe valori competitive, cu diversi coeficienti de certitudine asociati, dar in final se va selecta o singura valoare pentru acesta. Daca se deduce o valoare cu coeficientul 1 (sigura) pentru un atribut monovaloare se abandoneaza acumularea de valori pentru acel atribut si nu se mai incearca aplicarea altor reguli alternative care il refera.

Atributele multivaloare pot avea mai multe valori posibile asociate, atat pe parcursul rezolvarii, cit si in final deci in solutia problemei. Se presupune ca interpretorul de reguli intreaba utilizatorul ori de cite ori intilneste un atribut care nu este referit de nici o regula. Deci orice atribut care nu apare in concluzia unei reguli este considerat data primara.

Pe linga acestea, motorul de inferenta realizeaza o forma de rationament incert pe baza coeficientilor de certitudine asociati valorilor de atribute din memoria de lucru si regulilor din sistem. Regulile de inferenta incerta sint urmatoarele:

- (1) Coeficientul de certitudine asociat valorii de atribut dedusa de o regula R este egal cu produsul dintre coeficientul de certitudine al premisei regulii ($CF_{\text{premise R}}$) si coeficientul de certitudine asociat regulii (CF_R):

$$CF_{\text{atr dedus R}} = CF_{\text{premise R}} \cdot CF_R$$

- (2) Coeficientul de certitudine al premisei unei reguli R este valoarea minima a coeficientilor de certitudine asociati fiecarei ipoteze I_j din premisa acelei reguli. Coeficientul de certitudine al unei ipoteze este stabilit de procesul de

identificare a regulii cu memoria de lucru si este egal cu coeficientul valorii de atribut care a satisfacut ipoteza. Factorul de corectitudine al intregii premise se calculeaza astfel:

$$CF_{\text{premise } R} = \min_j \{CF_{\text{ipoteza}_i R}\}$$

- (3)Daca un atribut A are deja o valoare V cu un coeficient de certitudine CF_1 in memoria de lucru, si pe o ramura de deductie alternativa, se obtine aceeasi valoare V pentru atributul A cu un coeficient de certitudine CF_2 , memoria de lucru este actualizata, valoarea V avind un coeficient de certitudine asociat CF pe baza formulei:

$$CF = CF_1 + CF_2 \cdot (1 - CF_1)$$

In continuare se urmareste functionarea exemplului utilizind interpretorul de reguli descris, pentru cazul in care utilizatorul doreste aflarea vinului recomandat unui meniu care contine peste si sos-alb. Scopul de satisfacut este vin. Regulile care refera vin in concluzie sint R31÷R36. Pentru a satisface aceste reguli trebuie satisfacute subscopurile culoare-vin si tip-vin. Regulile care refera culoare-vin in concluzie sint R11÷R14. Pentru a satisface R11 sistemul intreaba utilizatorul valoarea atributului componenta-meniu, intrebare la care utilizatorul raspunde peste. In acest moment R11 nu este satisfacuta, dar R12 reuseste si se adauga in memoria de lucru faptul

(culoare-vin alb 1)

Deoarece culoare-vin este un atribut monovaloare pentru care s-a dedus o valoare cu factor de certitudine 1, se opreste acumularea de valori pentru acest atribut. Se continua cu satisfacerea scopului tip-vin care apare in concluziile regulilor R21÷R24. Pentru a satisface regula R21 se intreaba utilizatorul valoarea atributului sos-meniu si se primeste raspunsul sos-alb. Regula R21 reuseste si adauga la memoria de lucru faptul

(tip-vin sec 0.8 demisec 0.6)

Se observa aplicarea regulii (1) de inferenta incerta care asociaza coeficientul de certitudine valorilor de atribut deduse. Se incearca aplicarea celorlalte reguli care refera tip-vin in concluzie, respectiv R22, R23 si R24, dar nici una nu reuseste. Se revine apoi la satisfacerea scopului vin, deci a regulilor R31÷R36. Prima regula dintre acestea care reuseste este R35, regula care ar trebui sa adauge in memorie faptul (vin chardonnay). Tinind cont de regulile de inferenta (1)÷(2) faptul adaugat este

(vin chardonnay 0.8)

Deoarece vin este un atribut monovaloare, se continua acumularea de valori si se incearca si ultima regula care refera vin in concluzie, R36. Pe baza acestei reguli, care reuseste, si utilizind regulile de inferenta (1)÷(2), se actualizeaza memoria de lucru cu faptul

(vin chardonnay 0.8 riesling 0.6)

Deoarece nu mai exista reguli aplicabile, sistemul se opreste si raspunsul este

vin = chardonnay 0.8
riesling 0.6

1.2.2. Reprezentarea cunoștințelor și limbajele de programare

Limbajul utilizat pentru a implementa un SE nu depinde de forma de reprezentare a cunoștințelor adoptată, ci este permisă utilizarea oricărui limbaj existent. Astfel, un SE se poate elabora în *BASIC*, *C*, *PASCAL*, *LISP*, *PROLOG*, *SMALLTALK*, *LOOPS*, *KOOL* etc.

Tipic pentru reprezentarea procedurală a cunoașterii este limbajul *PLANNER*, sau subsetul acestuia cunoscut sub denumirea de *MICROPLANNER*. Ca o extensie a facilităților acestui limbaj a fost dezvoltat limbajul *CONNIVER*, care oferă unele posibilități suplimentare de reprezentare și o mai mare flexibilitate în utilizare.

Limbajul *PROLOG* este cel care a fost ales de japonezi ca bază de dezvoltare pentru programul de cercetare *calculatoare din generația a cincea*. De aici a rezultat limbajul *ESP* care s-a inspirat foarte mult din *PROLOG*. Limbajul *PROLOG* este bazat pe logica de ordinul întâi.

Limbajul *LISP* este considerat limbajul inteligenței artificiale și al sistemelor expert. Astfel, majoritatea generatoarelor de SE funcționează cu interpretoare sau compilatoare *LISP*.

Combinăția dintre abordarea frame și limbajele orientate pe obiect a dat naștere unor limbaje destinate dezvoltării sistemelor expert, și anume: *LOOPS* (Bobrow și Stfrik, 1983), *MERING* (Ferber, 1983), *LRO* (Roche, 1984). Unele medii de dezvoltare de sisteme expert sunt construite în jurul acestor limbaje orientate pe obiect.

- Limbajele orientate pe obiect: *Smalltalk*, *C++*.

1.3. Exemple de sisteme expert

Există numeroase domenii în care SE au fost utilizate cu succes. Dintre acestea pot fi menționate:

- conducerea proceselor de producție care utilizează celule flexibile de fabricație, roboți, piloți automați, controlul și conducerea tratamentelor termice;
- SE în traduceri dintr-o limbă în alta;
- SE de diagnostic și diagnostic în medicină;
- SE utilizate în contabilitatea financiară, datorită creșterii exponențiale a cunoștințelor și specialităților pe care trebuie să le stăpânească specialistul financiar-contabil sau expertul contabil;
- SE folosite în proiectarea sistemelor informatice financiar-contabile, stabilind modalitățile și restricțiile de agregare a informației;
- SE pentru asistarea operatorului uman în procesele de teleghidare spațială realizat de NASA.

Din multitudinea de sisteme expert construite pot fi menționate:

- ETS - este un SE care servește la realizarea unui interviu automatizat. A fost inventat de Boose în 1984 și îmbunătățit în 1986. Sistemul permite acumularea rapidă a vocabularului de bază și a relațiilor analitice asupra acestuia. Sistemul a fost folosit pentru acumularea de informații în diagnoze psihopatologice, pentru alegerea unei calități de vin etc.;

- SIS - Este un SE realizat de Kawagushi, pentru a ajuta un ziarist să construiască un interviu chiar în timpul derulării acestuia. SE recepționează mesajele auditive, le traduce într-un limbaj simplificat, ziaristul validează informația, repetând astfel informația percepută, iar SE avertizează asupra descrierilor incomplete, contradicții, ambiguități, determină incertitudinile, verifică restricțiile enunțate etc.;
- TAXADVISOR - SE pentru rezolvarea problemelor relative la impozitarea persoanelor fizice;
- TAXMAN - destinat analizei consecințelor fiscale, ca urmare a reorganizării întreprinderilor de grup;
- INVESTOR - SE destinat selectării investițiilor neimpozabile;
- AUDITOR - este unul dintre primele SE destinate controlului financiar. Accentul este pus pe analiza creanțelor provenite de la debitori greu solvabili;
- PRICE ANALYSIS - este destinat examinării cheltuielilor și analizei prețurilor;
- PAYPER - este destinat verificării cheltuielilor cu salariile.

1.4. Generatoare de SE

Un *generator de sisteme expert* (GSE) reprezintă un mediu de dezvoltare a SE. Astfel, un GSE este, de fapt, un program informatic complex, care integrează ansamblul cunoștințelor necesare elaborării și consultării sistemelor expert, fiind privit ca un instrument de elaborare de tip cadru (shell). În literatura de specialitate se mai utilizează și denumirea de *sisteme esențiale*, tocmai pentru că un astfel de sistem conține toate componentele unui sistem expert, mai puțin baza de cunoștințe. Dar, chiar dacă un GSE nu are o bază de cunoștințe, el este capabil să o construiască și să o exploateze. În plus, un GSE trebuie să conțină motorul inferențial, precum și utilitare de elaborare și exploatare indispensabile. De fapt, în cadrul GSE sunt utilizate motoare de inferențe standardizate, care devin, practic, autonome și care pot fi aplicate pentru prelucrarea cunoștințelor din diverse domenii. Acest lucru este posibil datorită distincției clare între cunoștințe și mecanismele de prelucrare a acestora din sistemele expert.

Prima realizare a constituit-o în 1979 EMYCIN (Essential MYCIN), care se baza pe motorul de inferențe MYCIN, la care s-au adăugat utilitarele pentru constituirea bazei de cunoștințe. EMYCIN putea fi utilizat pentru construirea unor sisteme expert și din alte domenii decât domeniul medical în care se aplica MYCIN.

Un generator de sisteme expert trebuie să îndeplinească trei mari cerințe:

- achiziționarea cunoștințelor experților, care nu este posibilă decât utilizând reguli simple în raport cu formele cunoașterii folosite de experți;
- exploatarea cunoștințelor prin motorul de inferențe care permite: evaluări, pronosticuri, decizii, precum și explicarea raționamentelor efectuate;
- menținerea și actualizarea cunoștințelor, prin posibilități de adăugare, suprimare și modificare a regulilor.

În decursul timpului, pentru dezvoltarea GSE s-au utilizat limbajele de programare specifice domeniului inteligenței artificiale, LISP, PROLOG, SMALLTALK, precum și extensiile acestora (INTERLISP, COMMON LISP, FRANZLISP, MACLISP, ZETALISP etc.), dar și C, FORTRAN, respectiv limbaje orientate pe obiect, cum ar fi C++.

În continuare este prezentată, pe scurt, arhitectura generală a unui GSE, care cuprinde (v. figura 1-5):

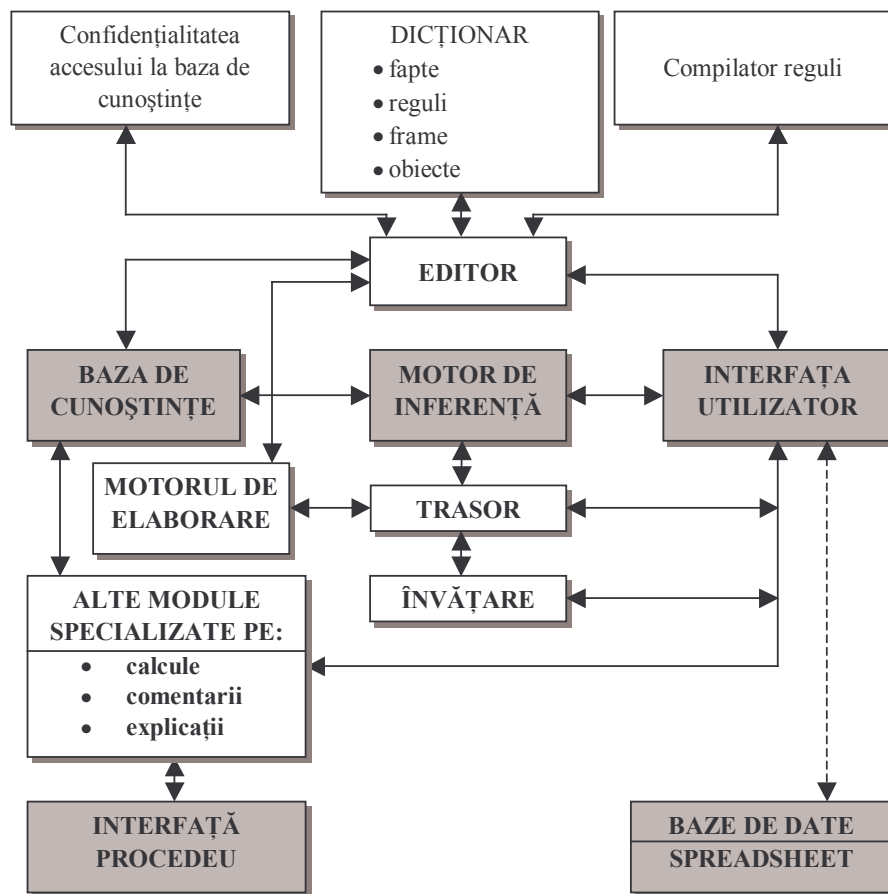


figura 1-5. Arhitectura specifică unui GSE

- *motorul de inferențe* – este conectat la baza de cunoștințe din care își preia cunoștințele pe care le prelucerează și le rememorează tot la nivelul bazei de cunoștințe;
- *baza de cunoștințe* – conține regulile și faptele necesare dezvoltării unui SE;
- *editorul* – asigură schimbul de cunoștințe între utilizator și SE prin intermediul unei interfețe și a unui modul de dialog, într-o manieră și structură apropiată limbajului natural. Practic, cu ajutorul lui se încarcă baza de cunoștințe și el este cel care asigură corespondența cunoștințelor introduse de operator cu structurarea predefinită a cunoștințelor. În plus, editorul realizează o compilare a regulilor, permițând detectarea eventualelor erori de sintaxă în cadrul regulilor, și astfel, implicit, accelerarea activității motorului inferențial;
- *trasorul* - urmărește șirul raționamentelor care se efectuează în motorul de inferențe și, de asemenea, actualizează baza de fapte, pe măsură ce faptele sunt dovedite;
- *motorul de elaborare* - are rolul de a coordona acțiunea trasorului și a editorului, permițând facilitarea lucrului pentru programator. Astfel, motorul de elaborare oferă toate facilitățile pentru efectuarea corecțiilor și realizarea verificării sintaxei, asigurând calitatea bazei de reguli (din punct de vedere al completitudinii, coerenței și neredundanței);

- *modulul de învățare*, care presupune achiziția de noi reguli și, pe de altă parte, reperarea euristicilor performante (urmărindu-se optimizarea sau simplificarea numărului de reguli);
- *interfețe utilizator* – care permit comunicarea și dialogul eficient cu utilizatorul. Trebuie să se precizeze faptul că pot exista mai multe module de interfață utilizator;
- *dicționarul*, componentă auxiliară care conține toate informațiile particulare și specifice faptelor, regulilor, obiectelor etc.
- *opțional*, module specializate în explicații, comentarii și calcule.

Toate aceste componente sunt organizate și exploatabile într-o manieră unitară în accepțiunea unui mediu de elaborare al unui SE.

Un sistem expert este consultat, în general, de două categorii de utilizatori, și anume, utilizatorii obișnuiți (care caută doar un răspuns la o anumită problemă) și utilizatorii experți (care, ca și utilizatorii obișnuiți pot căuta răspunsuri, dar, în plus, pot realiza și perfecționarea cunoștințelor). Generatoarele de sisteme expert oferă instrumente de dialog pentru ambele categorii de utilizatori.

Dintre GSE existente se pot aminti: Guru, Argument-Decidex, First-Class, Intelligence service, M1, Nexpert-Object, Crystal, Personal Consultant, VP EXPERT, H-EXPERT, XI PLUS, G2, CLIPS, JESS, CORVID (EXSYS) , K-Vision, Expert System Builder, ESIEWin, e2gLite etc.

Concluzie

Extinderea considerabilă a utilizării sistemelor expert se datorează și dezvoltării generatoarelor de sisteme expert, care reprezintă sistemele expert vide, aplicabile în diferite domenii de activitate. Utilizarea generatoarelor de sisteme expert conduce la mutații esențiale în domeniu: sfera utilizatorilor de sisteme expert se extinde, accesul la construirea bazei de cunoștințe nu mai este rezervat exclusiv specialiștilor.