Capitolul 5

Rationament nemonoton. Logici nemonotone

O mare parte a logicii se bazeaza pe rationamentul deductiv, adica pe utilizarea inferentelor valide care mentin adevarul asertiunilor. De multe ori, logica simbolica este identificata cu logica deductiva. Cu toate acestea, exista mai multe forme de rationament nedeductiv care au fost studiate în logica si aplicate în inteligenta artificiala. Tehnicile de rationament pur deductiv, prezentate pîna acum, au fost recunoscute de mult ca fiind insuficiente pentru reprezentarea tuturor formelor de comportament inteligent.

Exista diverse forme de rationament nedeductiv, numit si rationament nevalid. Rationamentul inductiv, bazat pe forma de inferenta inductie, este o posibila extensie a rationamentului deductiv, iar introducerea lui în sistemele de învatare automata s-a dovedit a fi utila în rezolvarea problemelor. Acest tip de rationament si aplicatiile lui în sistemele de inteligenta artificiala sînt prezentate în Capitolul 7. Alte posibilitati de extindere a rationamentului deductiv sînt *rationamentul nemonoton*, formalizat în logicile nemonotone, si *rationamentul statistic*.

În rationamentul nemonoton axiomele si regulile de inferenta sînt extinse astfel încît sa permita efectuarea rationamentului pe baza cunostintelor incomplete. În logica simbolica deductiva, informatiile incomplete sînt de obicei reprezentate prin disjunctii de termeni în care un termen individual nu poate fi nici demonstrat, nici contrazis pe baza axiomelor sistemului. Adaugarea de noi informatii în sistem poate conduce la contradictii, contradictii care necesita reorganizarea si revizuirea cunostintelor existente. Logica monotona nu ofera instrumente capabile sa rezolve inconsistentele cauzate de adaugarea de noi cunostinte. Logicile nemonotone încearca sa ofere solutii acestei probleme.

În rationamentul statistic, reprezentarea cunostintelor este extinsa cu o masura numerica a încrederii sau a incertitudinii asociata fiecarei asertiuni. Prin definirea unor reguli de combinare a acestor masuri ale incertitudinii, rationamentul statistic permite rezolvarea problemelor care implica cunostinte nesigure, incerte. Acest tip de rationament include modelul probabilistic, modelul factorilor de certitudine, modelul Dempster-Shafer si logicile vagi [Kruse,s.a.,1991].

Unii cercetatori considera cunostintele incerte tot ca o forma de cunostinte incomplete si vad rationamentul nemonoton si cel statistic ca solutii alternative în rezolvarea problemelor bazate pe acest tip de cunostinte. John McCarthy [1993], în articolul sau "History of circumscription", sustine ideea conform careia modelele de rationament nemonoton si cele de rationament statistic

sînt mai degraba formalisme complementare decît rivale. Metodele nemonotone sînt necesare pentru a reprezenta o lume în schimbare la care ulterior, se pot adauga probabilitati sau masuri ale incertitudinii, conform modelelor statistice, pentru a modela neîncrederea în aceasta lume. Discutia care urmeaza se orienteaza spre prezentarea unor formalisme logice nemonotone si a diverselor modele de implementare a rationamentului nemonoton în sistemele de inteligenta artificiala.

5.1 Introducere în rationamentul nemonoton

Una dintre problemele dificile ale inteligentei artificiale este reprezentarea si simularea rationamentului de bun simt. Încercarea de a solutiona aceasta problema a dus atît la formalizari ale rationamentului de bun simt în cadrul logicilor nemonotone, prin extinderea logicii clasice, cît si la dezvoltarea unor programe de rezolvare a problemelor care implica un astfel de rationament. Rationamentul de bun simt difera de logica clasica prin aceea ca necesita obtinerea de noi fapte pe baza informatiilor partiale, incomplete. În matematica sau în logica clasica, o concluzie nu este acceptata decît daca aceasta poate fi demonstrata pe baza legilor de inferenta si a axiomelor initial acceptate. În viata reala, oamenii sînt pusi în fata unor situatii în care trebuie sa actioneze pe baza unor cunostinte limitate despre contextul actiunilor lor. Se accepta concluzii pentru care nu exista demonstratii, dar care par a fi plauzibile. Un sistem care poate executa un astfel de tip de rationament se numeste *sistem nemonoton* [Minsky,1975;Winograd,1980].

Rationamentul nemonoton este un tip de rationament care implica considerarea unor presupuneri care ar putea fi abandonate în urma obtinerii de noi informatii. În rationamentul deductiv, monoton, toate faptele noi, inferate pe parcursul rezolvarii problemei, se adauga în baza de cunostinte, baza de cunostinte crescînd continuu. În rationamentul nemonoton, anumite concluzii inferate anterior pot fi invalidate ulterior, trebuind deci sa fie eliminate din baza de cunostinte. Notînd cu BC baza de cunostinte a unui sistem bazat pe cunostinte, si cu Th(BC) toate cunostintele care pot fi deduse din BC prin aplicarea regulilor de inferenta, rationamentul monoton si respectiv cel nemonoton, se specifica, informal, astfel:

- daca $F = \mathrm{Th}(BC)$, noua baza de cunostinte obtinuta prin aplicarea regulilor de inferenta monotona este $BC1 = BC \cup F$
- daca $F = \mathrm{Th}(BC)$, noua baza de cunostinte obtinuta prin aplicarea regulilor de inferenta nemonotona este $BC1 = (BC - \{A\}) \cup F$, unde $A \subset BC$ este multimea de fapte, posibil vida, invalidate de procesul inferential.

Definitia formala a rationamentului monoton si nemonoton, si a operatorului Th( ) se va face în Sectiunea 5.2.1.

5.1.1 De ce este nevoie de rationament nemonoton?

Se considera în continuare urmatorul exemplu de rationament, numit *crima ABC*, exemplu asemanator celui prezentat de Quine si Ullian [1978] si de Rich si Knight [1991]. Fie Alecu, Barbu si Cezar trei suspecti într-o crima, toti trei beneficiind de pe urma crimei. Alecu are un alibi, fiind înscris în registrul unui hotel respectabil din Arad. Barbu are de asemenea un alibi, deoarece cumnatul lui depune marturie ca Barbu îi facea o vizita acasa la Buzau, în momentul crimei. Cezar pretinde ca are si el un alibi, deoarece urmarea un concurs de schi la Cioplea, dar nu are nici o dovada în acest sens. Din cele relatate se poate infera ca:

(1) Alecu nu a comis crima.

(2) Barbu nu a comis crima.

(3) Fie Alecu, fie Barbu, fie Cezar a comis crima.

Între timp, Cezar aduce o dovada în sprijinul celor pretinse de el. El a avut norocul sa fie înregistrat pe o videocaseta a televiziunii în timp ce urmarea concursul de schi de la Cioplea. Un nou fapt se impune:

(4) Cezar nu a comis crima.

Faptele (1)÷(4), numite în rationamentul nemonoton *convingeri*, sînt inconsistente, deci una dintre ele trebuie selectata si eliminata. Daca se elimina convingerea (3), atunci trebuie sa se presupuna ca mai exista si alti beneficiari ai crimei. Daca se elimina convingerea (1), trebuie în acelasi timp sa se adauge o noua convingere, de exemplu ca registrul hotelului a fost falsificat. Daca se elimina convingerea (2), trebuie sa se elimine si convingerea (implicita) conform careia spusele cumnatului lui Barbu sînt adevarate.

Acest proces de validare si, respectiv, invalidare a convingerilor, si de adaugare a unor noi convingeri pe masura obtinerii de noi informatii ar putea continua mult timp. Multimea de convingeri (1)÷(4) s-ar putea baza pe o multime de convingeri cu mult mai mare, convingeri care ar trebui revizuite individual la detectarea unei contradictii. Problema care apare este: unde trebuie sa se opreasca demonstrarea probelor? Teoretic, acest lucru ar trebui sa nu se întîmple niciodata; în practica însa, atunci cînd s-a ajuns la un set de convingeri necontradictorii, demonstrarea probelor înceteaza. Exemplul considerat ilustreaza problemele care apar în cazul formalizarii cunostintelor incomplete, incerte, sau dinamice. Asa cum s-a mentionat anterior, cele doua tehnici importante de rezolvare a acestui gen de probleme sînt rationamentul nemonoton si rationamentul statistic. Rationamentul nemonoton poate rezolva urmatoarele probleme:

(1) Cum poate fi extinsa baza de cunostinte astfel încît sa permita realizarea inferentelor atît pe baza prezentei anumitor fapte, cît si pe baza absentei lor din baza de cunostinte.

De exemplu, în cazul crimei ABC, reprezentarea cunostintelor în baza de cunostinte trebuie sa permita reprezentarea unor convingeri cum ar fi: "Daca nu exista nici un motiv datorita caruia o persoana ar fi putut comite crima, se presupune ca nu a comis-o" sau "Daca nu exista nici un motiv pe baza caruia sa se creada ca cineva nu se întelege bine cu rudele sale atunci se presupune ca rudele persoanei vor încerca sa o ajute". În particular, trebuie facuta distinctia între:

- se stie ca ~P
- nu se stie daca P.

Logica cu predicate de ordinul I permite rationamente bazate pe "se stie ca ~P". Este nevoie, deci, de un cadru logic si de un sistem care sa permita si rationamente bazate pe "nu se stie daca P". Un astfel de sistem este un sistem cu rationament nemonoton. Inferentele care depind de lipsa unor anumite informatii se numesc *inferente nemonotone*. Tocmai de la acest tip de inferente decurge caracterul nemonoton al rationamentului. Adaugarea de noi fapte poate invalida concluzii anterioare, concluzii la care s-a ajuns tocmai pe baza absentei acestor fapte din baza de cunostinte.

(2) Cum se poate actualiza eficient baza de cunostinte la adaugarea sau la eliminarea unui fapt?

În sistemele nemonotone invalidarea unei convingeri poate duce la invalidarea tuturor concluziilor inferate pe baza acesteia, deci la necesitatea invalidarii unei întregi multimi de convingeri. Solutia uzuala a acestei probleme este aceea de a memora deductiile facute sub forma de *justificari* asociate faptelor din baza de cunostinte, deci de a înregistra dependentele între cunostinte. Acest lucru face posibila gasirea tuturor justificarilor care depind de absenta noii convingeri adaugate si marcarea acestora ca invalide. Un mecanism de înregistrare a dependentelor de acest fel poate fi util si în cazul implementarii rationamentului monoton daca, ocazional, o parte din axiomele folosite trebuie eliminate pentru a reflecta o lume în schimbare. De exemplu, în cazul crimei ABC, Alecu este în oras saptamîna aceasta si poate depune marturie dar, daca se asteapta pîna saptamîna viitoare, el va pleca din oras. În consecinta, cînd se discuta problema mentinerii justificarilor valide într-un sistem, se considera atît rationamentul nemonoton, cît si rationamentul monoton într-o lume în schimbare.

(3) Cum pot fi utilizate cunostintele existente pentru rezolvarea conflictelor, în cazul în care se pot executa mai multe inferente nemonotone cu rezultate contradictorii?

Deoarece inferentele nemonotone se bazeaza pe absenta unor anumite informatii, într-un sistem nemonoton pot apare contradictii mult mai des decît într-un sistem logic clasic. Cum poate decide sistemul care convingeri trebuie invalidate în cazul aparitiei unei contradictii? Într-un sistem nemonoton, exista uneori portiuni ale bazei de cunostinte care sînt (local) consistente dar care împreuna (global) sînt inconsistente. Asa cum se va vedea în continuare, multe sisteme formale de rationament nemonoton pot defini alternativele plauzibile, dar foarte putine sisteme

ofera modalitati de a alege între alternative mutual exclusive. De cele mai multe ori este necesar sa se recurga la informatii specifice problemei pentru a alege între o alternativa sau alta.

În legatura cu aceasta problema de alegere, McDermott si Doyle [1980] afirma ca scopul regulilor de inferenta nemonotone nu este acela de a adauga cunostinte sigure acolo unde ele nu exista, ci de a ghida selectia convingerilor considerate la un moment dat. Convingerile obtinute, desi s-ar putea sa se dovedeasca ulterior incorecte, pot fi utilizate pentru avansul spre solutia problemei.

De exemplu, în cazul crimei ABC, se poate elimina conflictul generat de faptele (1)÷(4), alegînd ca autor al crimei persoana cu alibiul cel mai putin credibil, Barbu, plecînd de la presupunerea ca spusele cumnatului lui Barbu sînt neadevarate. Acest lucru implica de fapt încrederea în alte convingeri, de exemplu, aceea ca unul dintre suspecti a mintit. Chiar daca acest lucru va fi invalidat de obtinerea ulterioara a unor cunostinte noi, convingerea adaugata permite eliminarea conflictelor.

Observatie. Desi rationamentul nemonoton lucreaza tot cu doua valori de adevar, adevarat si fals, se utilizeaza termenii de convingere valida sau nevalida, justificare valida sau nevalida, pentru a pune în evidenta faptul ca statutul de adevarat sau fals al acestor asertiuni se poate schimba în timp. Aceasta abordare este diferita fata de cea din logica cu predicate de ordinul I, în care o asertiune este fie adevarata, fie falsa pe tot parcursul procesului de rezolvare a problemei.

5.1.2 Forme de rationament nemonoton

Abordarile rationamentului nemonoton în inteligenta artificiala pot fi în general considerate ca facînd parte din doua clase. Prima abordare, reprezentata mai ales de rezultatele lui McCarthy, Reiter si McDermott si Doyle, extinde logica clasica în diverse moduri, construind sisteme logice formale de rationament nemonoton. A doua abordare considera sistemul logic ca un obiect al universului discursului si extinde sistemul de rationament cu o multime de meta-instrumente, reprezentînd o abordare pragmatica a rationamentului nemonoton.

Extinderi ale logicii

În abordarea lui McDermott si Doyle [1980] logica este extinsa prin introducerea simbolului modal **M** care semnifica consistenta unei asertiuni. Aceasta perspectiva urmareste traditia logicilor modale si are corespondent în operatorii de timp utilizati în axiomatizarea faptelor care implica secvente temporale.

Reiter [1980] pastreaza notatia standard a formulelor din calculul cu predicate de ordinul I, dar extinde logica prin adaugarea unei reguli de inferenta implicita la multimea uzuala de axiome. Solutiile lui McDermott si Doyle, si, respectiv, Reiter formeaza ceea ce se numeste *rationament implicit*. Acest tip de rationament nemonoton este prezentat în Sectiunea 5.2.

McCarthy [1980;1986] dezvolta teoria circumscrierii pornind tot de la logica cu predicate de ordinul I si tratînd situatiile ca obiecte despre care se poate rationa. În modelul lui, McCarthy considera extensia partiala curenta a unui predicat drept o extensie totala, care poate fi ulterior invalidata prin obtinerea unor noi exemple. În acest fel, el dezvolta o forma de rationament numit *rationament circumscris* sau *rationament minimal*, care va fi prezentat în Sectiunea 5.3.

Din punct de vedere al construirii unor sisteme de rationament bazate pe aceste formalisme exista o problema semnificativa comuna celor trei modele de rationament nemonoton propuse. Fiecare formalism ofera o forma de rationament echivalenta unei inferente nemonotone de tipul: "Daca P este consistent cu ceea ce se cunoaste atunci infera Q". În logica cu predicate de ordinul I, problema stabilirii faptului ca o formula bine formata este sau nu teorema este nedecidabila, dar se poate decide daca o secventa de formule este sau nu demonstrarea unei teoreme. Daca se adauga unui sistem logic o regula de inferenta de tipul celei specificate mai sus, atunci problema stabilirii daca o secventa de formule este demonstratia unei teoreme nu mai este decidabila, deoarece un pas din demonstratie implica stabilirea realizabilitatii unei formule arbitrare. Intuitiv, se încearca executia unui pas de rationament de forma: "Daca nu se cunoaste X atunci Y", unde cunoasterea lui X este definita în termenii posibilitatii de demonstrare a lui X. Acest rezultat cam trist genereaza concluzia conform careia orice abordare computationala a rationamentului nemonoton necesita o componenta euristica care, ocazional, va duce la convingeri false.

Meta-abordari

A doua clasa de abordari ale rationamentului nemonoton se desprinde de cadrul sistemului logic, considerînd obiectele acestuia ca fiind elementele unui metasistem. O prima categorie de solutii implementeaza un tip de rationament nemonoton care depinde de cantitatea de cunostinte existente sau de epuizarea unui efort predefinit de calcul pentru efectuarea deductiei. Întrebarea "Este P consistent?" este convertita în încercarea de a deduce P sau ~P pe baza cunostintelor existente, în limita unui anumit efort de calcul. O a doua categorie de solutii construiesc *sisteme de mentinere a consistentei datelor* [Doyle,1979;McDermott,1983;deKleer,1986a]. Un sistem de mentinere a consistentei este un sistem formal de restrictii între obiecte care reprezinta elemente ale unei teorii logice de ordinul I. Regulile de inferenta ale teoriei logice pot fi utilizate pentru a genera noi restrictii, care au în general forma "Daca se crede A atunci trebuie sa se creada B". În orice moment, un sistem de mentinere a consistentei are o multime finita de teoreme si de restrictii. În acest caz, el poate implementa efectiv inferente nemonotone de tipul "Daca P este consistent atunci Q", deoarece consistenta este considerata din punctul de vedere al multimii curente de teoreme si nu din acela a tot ceea ce se poate demonstra pornind de la aceste teoreme.

În timp ce cele trei extinderi ale logicii clasice au scopuri similare, si anume construirea unor instrumente de rationament nemonoton, cele doua categorii de meta-abordari ale rationamentului nemonoton se concentraeza pe aspecte diferite ale rezolvarii problemei. Prima categorie de solutii se ocupa de efectuarea rationamentului în termenii unor resurse limitate, pe cînd cea de a doua trateaza problema revizuirii convingerilor si mentinerii restrictiilor de

consistenta. Sistemele de mentinere a consistentei datelor, cît si abordarea înrudita utilizînd multimi de lumi posibile, sînt prezentate în Sectiunile 5.4, 5.5 si, respectiv, 5.6.

5.2 Rationament implicit

O mare parte din ceea ce se cunoaste despre lume este "aproape întotdeauna" adevarat, cu cîteva exceptii. Astfel de cunostinte au de cele mai multe ori forma "Cei mai multi P sînt Q" sau "Cei mai multi P au proprietatea Q". De exemplu, se stie ca cele mai multe pasari zboara, cu exceptia pinguinilor, strutilor si a soimilor maltezi. Avînd de a face cu o anumita pasare, se poate concluziona ca ea zboara în cazul în care nu apartine acestor exceptii. Într-o reprezentare folosind logica cu predicate de ordinul I, aceste cunostinte ar trebui exprimate prin enumerarea tuturor exceptiilor astfel

$$(\forall x)\ (\text{Pasãre}(x) \wedge \sim \text{Pinguin}(x) \wedge \sim \text{Struþ}(x) \wedge ... \rightarrow \text{Zboarã}(x)).$$

Dar într-o astfel de reprezentare nu se poate concluziona ca pasarile zboara, în general. Aceasta deoarece, daca se doreste sa se demonstreze Zboarã(Coco) si tot ceea ce se stie despre Coco este faptul ca este pasare, demonstratia nu se poate face. Exista deci necesitatea reprezentarii (implicite) a faptului ca, în general, toate pasarile zboara. Altfel spus, exista necesitatea de a formula rationamente de tipul "Daca x este pasare atunci, în absenta unei informatii contrare, se poate infera ca x zboara". Rationamentul implicit încearca sa rezolve aceste probleme.

În cadrul rationamentului implicit se considera ca adevarate anumite fapte atît timp cît nu s-a demonstrat contrariul lor. În acest mod se obtin concluzii bazate pe ceea ce este cel mai plauzibil a fi adevarat.

În continuare se discuta doua modalitati de formalizare a rationamentului implicit:

- logica nemonotona a lui McDermott si Doyle, si
- logica implicita a lui Reiter.

Se descriu, de asemenea, doua forme de inferenta nemonotone care pot fi definite în aceste formalisme logice: abductia si mostenirea proprietatilor.

Observatie. Nu trebuie facuta confuzia între rationamentul nemonoton, care reprezinta o forma fundamentala de rationament, rationamentul implicit, care este un tip de rationament nemonoton si logica nemonotona sau logica implicita, care reprezinta posibile formalizari logice ale rationamentului implicit.

5.2.1 Logica nemonotona a lui McDermott si Doyle

În logica nemonotona a lui McDermott si Doyle [McDermott,Doyle,1980;Hanks,McDermott, 1987] se pleaca de la limbajul calculului cu predicate de ordinul I si se extinde acest limbaj cu operatorul modal **M**, care are semnificatia "este consistent" sau "nu poate fi infirmat". Valorile implicite sînt reprezentate cu ajutorul acestui operator modal. De exemplu formula

$$(\forall x)(\forall y)\ (\text{Rude}(x,y) \wedge \mathbf{M}\ \text{SeÎnþeleg}(x,y) \rightarrow \text{VaApãra}(x,y))$$

semnifica: "Pentru orice x si y, daca x si y sînt rude si daca faptul ca x si y se înteleg este consistent (nu poate fi infirmat), atunci concluzioneaza ca x va apara pe y". Cel mai simplu exemplu de rationament implicit în acest formalism este

$$\mathbf{M}\ \text{Predicat} \rightarrow \text{Predicat}$$

ceea ce înseamna ca, daca un predicat nu poate fi infirmat, acest predicat este adevarat.

Din punct de vedere formal, logica nemonotona a lui McDermott si Doyle se poate defini pornind de la axiomele logicii cu predicate de ordinul I si de la regula de inferenta Modus Ponens, asa cum se prezinta în continuare.

Definitie. Fie $L =< A, F, A, \Re >$ un sistem (formal) logic si Γ o multime de formule, $\Gamma \subset F$. Operatorul $\text{Th}(\Gamma)$ calculeaza *multimea formulelor* P *deductibile din* Γ în sistemul logic L, $\text{Th}(\Gamma) = \{P \mid P \in F, \Gamma \xrightarrow[L]{} P\}$.

Operatorul Th are proprietatea de *monotonicitate*, definita de urmatoarele doua relatii

(1) $\Gamma \subseteq \text{Th}(\Gamma)$

(2) daca $\Gamma \subseteq \Gamma_1$ atunci $\text{Th}(\Gamma) \subseteq \text{Th}(\Gamma_1)$

si proprietatea de *idempotenta*, definita de relatia

(3) $\text{Th}(\text{Th}(L)) = \text{Th}(L)$

Observatie. Sfatuim cititorul mai putin constiincios sa revada notiunile prezentate în Capitolul 2, daca notatiile si semnificatiile notiunilor utilizate îi sînt neclare.

Orice sistem de inferenta clasica (monotona) satisface aceste proprietati. Proprietatea de idempotenta poate fi vazuta si ca o ecuatie de punct fix, care spune ca multimea de teoreme derivabile monoton dintr-un sistem logic este un punct fix al operatorului Th. Acest operator calculeaza închiderea unei multimi de formule dupa regulile de inferenta monotone.

Pentru a trata cazul logicii nemonotone, este necesara însa o noua regula de inferenta, de exemplu o regula de derivare nemonotona. Aceasta ar putea fi

daca ~P în L **atunci** $\xrightarrow[L]{}$ **M** P

ceea ce înseamna ca daca negatia unei formule nu este derivabila în sistem, atunci formula este consistenta. Aceasta regula nu poate fi însa utilizata deoarece este circulara, definind derivabilitatea unei formule în termenii derivabilitatii altei formule. Din acest motiv se retine $\xrightarrow[L]{}$ pentru indicarea derivabilitatii monotone a unei formule si se defineste operatorul NM dupa cum urmeaza.

Definitie. Fie un sistem formal (logic) L =< A, F, A, $\Re$ > si o multime de formule Γ, $\Gamma \subset F$. Fie $A_\Gamma(\Gamma)$ multimea presupunerilor din Γ, definita de relatia

$$A_\Gamma(\Gamma) = \{\mathbf{M}\ \mathrm{P} \mid \mathrm{P} \in F, \sim \mathrm{P} \notin \Gamma\} - \mathrm{Th}(A)$$

Se defineste operatorul NM prin relatia

$$\mathrm{NM}(\Gamma) = \mathrm{Th}(\mathrm{L} \cup A_\Gamma(\Gamma))$$

cu urmatoarea semnificatie: NM porneste de la multimea de formule Γ si produce o noua multime de formule, care include Th(Γ) dar contine si toate formulele deductibile din setul original de axiome reunit cu toate presupunerile care nu sînt infirmate de Γ.

Se observa ca teoremele din L de forma **M** P nu sînt niciodata considerate presupuneri. Pornind de la definitia de mai sus, Th(A) poate fi vazut ca

Th(A) = "cel mai mic punct fix al lui NM"

Aceasta definitie încearca sa surprinda ideea adaugarii regulii de inferenta nemonotona prezentata la începutul acestei sectiuni si justificata ca fiind inadecvata într-un sistem logic de ordinul I.

În consecinta, formulele în care apare operatorul modal **M** pot fi tratate ca orice alte formule din logica predicatelor de ordinul I. De exemplu, se poate aplica rezolutia asupra formulelor care contin operatorul **M** astfel

$$\frac{\mathrm{A} \wedge \mathbf{M}\,\mathrm{B} \rightarrow \mathrm{B} \qquad \sim \mathrm{A} \wedge \mathbf{M}\,\mathrm{B} \rightarrow \mathrm{B}}{\mathbf{M}\,\mathrm{B} \rightarrow \mathrm{B}}$$

Formalismul prezentat pune însa urmatoarele doua probleme. În primul rînd definirea notiunii de "este consistent" se face prin echivalenta cu "nu poate fi infirmat". Dar logica cu predicate de ordinul I nu este decidabila, deci, daca se doreste ca acest formalism sa fie macar semidecidabil, trebuie definita consistenta în termenii unor criterii euristice, de exemplu "nereusita de a demonstra inconsistenta în limitele unui efort predefinit". În al doilea rînd, mai multe inferente nemonotone, luate separat, indica moduri de extindere a bazei de cunostinte care,

daca ar fi luate împreuna, ar fi inconsistente. Pentru a exemplifica aceasta problema se considera urmatoarea baza de cunostinte:

$$(\forall x)\ (\text{Democrat}(x) \wedge \mathbf{M}\ \text{Pacifist}(x) \rightarrow \text{Pacifist}(x))$$

$$(\forall x)\ (\text{Anarhist}(x) \wedge \mathbf{M}\ \sim \text{Pacifist}(x) \rightarrow \sim \text{Pacifist}(x))$$

$$\text{Democrat}(\text{Vlad})$$

$$\text{Anarhist}(\text{Vlad})$$

Definitia data logicii nemonotone a lui McDermott si Doyle ofera doua posibilitati distincte de extindere a bazei de cunostinte. Prima posibilitate este aplicarea primei asertiuni, din care se deduce faptul $\text{Pacifist}(\text{Vlad})$, care se adauga la baza de cunostinte. În aceste conditii, cea de a doua implicatie bine formata nu se mai aplica. A doua posibilitate de extindere a bazei de cunostinte este aplicarea celei de-a doua asertiuni, din care se deduce $\sim \text{Pacifist}(\text{Vlad})$, ceea ce împiedica apoi aplicarea primei reguli. Deci baza de cunostinte BC se poate extinde fie la baza de cunostinte $BC_1 = BC \cup \{\text{Pacifist}(\text{Vlad})\}$, fie la $BC_2 = BC \cup \{\sim \text{Pacifist}(\text{Vlad})\}$.

Care este extensia corecta a bazei de cunostinte, i.e. care concluzie este implicata de fapt de multimea de axiome din baza de cunostinte? Conform definitiei formale, logica nemonotona defineste multimea teoremelor care pot fi derivate dintr-o multime de axiome A ca fiind intersectia multimilor de teoreme ce rezulta din diversele moduri posibile de combinare a axiomelor din A. Pentru exemplul anterior, rezulta ca baza de cunostinte extinsa BC' este identica cu cea originala, $BC' = BC$, nici una din extensiile BC_1 sau BC_2 nefiind cea corecta. În consecinta, nu se poate spune nimic despre pacifismul lui Vlad. Un sistem logic mai recent, numit logica autoepistemica [Moore,1985;Moore,1993], dezvoltat pornind de la formalismul lui McDermott si Doyle, solutioneaza o parte dintre aceste probleme.

5.2.2 Logica implicita a lui Reiter

Reiter [1980] introduce în formalismul logicii implicite o noua clasa de reguli de inferenta pentru a reprezenta rationamentul implicit. Aceste reguli au forma

$$\frac{P : Q}{R} \quad \text{sau} \quad P : Q / R$$

si au urmatoarea semnificatie: "Daca P este adevarat si daca este consistent sa se presupuna Q, atunci concluzioneaza R", unde P, Q si R sînt formule bine formate în logica cu predicate de ordinul I. Cel mai simplu rationament implicit în acest caz este P / P.

Observatie. Notatia originala a lui Reiter pentru regulile de inferenta nemonotona este **:M** în loc de **:**, dar ulterior **M** a fost abandonat deoarece nu aducea nici o informatie suplimentara.

În definitia logicii sale, Reiter pleaca, de asemenea, de la cadrul logicii cu predicate de ordinul I si introduce regula de inferenta nemonotona descrisa informal mai sus, pe care o numeste regula implicita. Cadrul formal al logicii implicite a lui Reiter este prezentat în continuare. În definitiile urmatoare, prin teorie se întelege multimea de axiome a unui sistem formal $L =< A, F, A, \Re >$.

Definitie. Se numeste *regula implicita* orice expresie de forma

$$\frac{\alpha(x) : \beta_1(x),...,\beta_m(x)}{w(x)}$$

unde $\alpha(x),\beta_1(x),...,\beta_m(x),w(x)$ sînt formule bine formate în sistemul logic al calculului cu predicate de ordinul I. $\alpha(x)$ se numeste *preconditia* regulii implicite, iar $w(x)$ se numeste *consecinta* regulii.

Definitie. O teorie implicita este o pereche (D,W) în care D este o multime de reguli implicite care se adauga la multimea de reguli de inferenta $\Re$ a sistemului logic si W este o multime de formule bine formate, $W \subset F$.

Definitie. Fie $\Delta = (D,W)$ o teorie implicita astfel încît fiecare regula implicita din D are forma $(\alpha : \beta_1,...,\beta_m / w)$, unde $\alpha,\beta_1,...,\beta_m,w$ sînt formule bine formate apartinînd limbajului F al sistemului logic L. Pentru orice multime de formule $\Gamma \subseteq F$, fie $S(\Gamma)$ cea mai mica multime care satisface urmatoarele proprietati:

(1) $W \subseteq S(\Gamma)$

(2) $Th(S(\Gamma)) = S(\Gamma)$, unde $Th(\Gamma) = \{P \mid P \in F, \Gamma \xrightarrow[L]{} P\}$

(3) Daca $(\alpha : \beta_1,...,\beta_m / w) \in D$ si $\alpha \in S(\Gamma)$, si $\sim\beta_1,...,\sim\beta_m \notin \Gamma$ atunci $w \in S(\Gamma)$.

O multime de formule bine formate $E \subseteq F$ este o *extensie* a lui Δ daca si numai daca $S(E) = E$, i.e. daca si numai daca E este un punct fix al operatorului S.

Definitiile de mai sus formalizeaza urmatoarea idee: orice extensie a unei teorii implicite este interpretata ca o multime acceptabila de convingeri pe care cineva le are despre o lume L incomplet specificata. Extensia unei teorii implicite nu este unica. În sprijinul acestei idei, Reiter discuta urmatorul exemplu. Fie urmatoarele doua reguli implicite, exprimate atît în limbaj natural cît si în logica implicita.

(1) "Orasul în care locuieste o persoana este orasul în care locuieste sotul, respectiv sotia persoanei respective."

$$(\forall x)(\forall y)(\forall z)\ (Soþ(x,y) \wedge Ora^o(y,z) : Ora^o(x,z) / Ora^o(x,z))$$

(2) "Orasul în care locuieste o persoana este orasul în care se afla serviciul acestei persoane."

$$(\forall x)(\forall y)(\forall z)\ (\text{Serviciu}(x,y) \wedge \text{Localizat}(y,z) : \text{Ora}^{o}(x,z) / \text{Ora}^{o}(x,z))$$

Se presupune acum ca sotul Cleopatrei locuieste la Cairo si serviciul Cleopatrei este localizat la Bucuresti. În urma primei reguli implicite, Cleopatra locuieste la Cairo, iar în urma celei de-a doua reguli, ea locuieste la Bucuresti, aceste doua convingeri reprezentînd cele doua extensii posibile ale teoriei implicite. Evident, este inconsistent sa se creada ambele extensii în simultan. Din punct de vedere intuitiv, acest exemplu este corect, deoarece în logica implicita, se lucreaza cu convingeri. Cineva poate sa creada ca locuinta Cleopatrei este la Cairo si altcineva poate sa creada ca ea locuieste la Bucuresti. Deci logica implicita a lui Reiter nu poate spune nimic despre modul în care trebuie aleasa extensia teoriei.

O solutie posibila a acestei probleme o reprezinta *modelul lumilor posibile*. În acest model valoarea unei asertiuni A este necunoscuta în lumea curenta, devine adevarata daca se presupune P si falsa daca se presupune Q. În modelul lumilor posibile, din lumea curenta L se creeaza doua lumi noi: L_1 care este identica cu L dar în care P este adevarat, si L_2 care este la fel cu L dar în plus Q este adevarat. Atunci A este adevarata în lumea L_1 si falsa în lumea L_2. O discutie detaliata a acestui model si solutiile de implementare asociate sînt prezentate în Sectiunea 5.6.

Dupa cum se observa, formalismul logicii implicite a lui Reiter seamana cu formalismul logicii nemonotone a lui McDermott si Doyle, dar exista si diferente semnificative. Prima diferenta este aceea ca exprimarea rationamentului nemonoton se face cu ajutorul unei reguli de inferenta în formalismul logicii implicite a lui Reiter, în timp ce în formalismul logicii nemonotone a lui McDermott si Doyle se face cu ajutorul unui operator al limbajului. De exemplu, din $A : B / B$ si $\sim A : B / B$ nu se poate deduce B deoarece nu se poate aplica nici o regula de inferenta, cum ar fi rezolutia. A doua diferenta consta în faptul ca, desi în formalismul logicii implicite ca si în cel al logicii nemonotone se pot construi mai multe extensii diferite ale bazei de cunostinte, formalismul logicii implicite nu spune nimic despre cum trebuie aleasa extensia corecta a bazei de cunostinte.

5.2.3 Inferente nevalide utilizate în rationamentul implicit

Doua forme de inferente nevalide sînt utilizate preponderent în implementarea sistemelor care folosesc rationament implicit: abductia si mostenirea proprietatilor.

Abductia

În logica cu predicate de ordinul I, inferenta valida (deductiva) standard Modus Ponens este definita, asa cum s-a mai spus, în modul urmator

$$\frac{P(a) \quad (\forall x)\ (P(x) \rightarrow Q(x))}{Q(a)}$$

Abductia sau inferenta abductiva se defineste ca implicatia inversa

$$\frac{Q(a) \quad (\forall x)\ (P(x) \rightarrow Q(x))}{P(a)}$$

O astfel de concluzie poate fi eronata, dar este consistenta atît timp cît nu exista fapte care sa o infirme. Abductia este o forma de inferenta nevalida care poate fi utilizata pentru formalizarea rationamentului nemonoton, cum ar fi rationamentul de bun simt. De exemplu, se considera urmatoarea baza de cunostinte:

(1) $(\forall x)\ (\text{Pasãre}(x) \rightarrow \text{Zboarã}(x))$

(2) Zboarã(AirBus)

(3) Zboarã(Coco)

Utilizînd abductia se poate infera din (1) si (2) faptul Pasãre(AirBus), si din (1) si (3) faptul Pasãre(Coco). Este consistent sa se concluzioneze Pasãre(Coco) dar si Pasãre(AirBus), atît timp cît nu se poate demonstra ca AirBus nu este pasare.

Observatii:

- Abductia este o forma de inferenta ce poate fi modelata atît în formalismul logicii nemonotone cît si în formalismul logicii implicite.
- Abductia poate fi combinata cu rationamentul statistic. De exemplu, se poate adauga faptului inferat prin abductie, P(a), o probabilitate sau un coeficient de certitudine pentru a reprezenta încrederea sistemului în adevarul lui P(a).

Valori implicite si mostenirea proprietatilor

Una dintre modalitatile de reprezentare a cunostintelor în inteligenta artificiala este reprezentarea structurata. Doua modele importante ale acestui tip de reprezentare sînt retelele semantice [Quillian,1969;Winston,1984] si limbajele bazate pe cadre (sau unitati) [Minsky,1975;Bobrow, Winograd,1977;Fikes,Kehler,1985]. Ambele modele realizeaza, desi utilizînd forme sintactice diferite, inferente specifice care modeleaza o forma de rationament implicit.

Într-o reprezentare structurata a cunostintelor, entitatile descrise sînt obiecte particulare (instante) si obiecte generice (clase de obiecte), care descriu toti membrii unei multimi de obiecte particulare. Legaturile între aceste entitati se stabilesc pe baza unor relatii (binare) între obiecte si

pe baza valorilor proprietatilor (atributelor) obiectelor respective. Proprietatile obiectelor pot fi privite din mai multe puncte de vedere, diversele valori asociate acestor perspective diferite numindu-se fatete. Exemple de fatete des întîlnite sînt: fateta valoare, fateta valoare implicita, fateta procedura efectiva. Exista doua relatii importante, prezente în orice model structurat, relatii care descriu taxonomia obiectelor reprezentate. Aceste doua relatii sînt: relatia de apartenenta a unui obiect la o multime, care defineste o relatie individual-generic, sau instanta-clasa, notata de obicei cu ISA (IS A), si relatia de incluziune a unei multimi de obiecte într-o alta multime de obiecte, care defineste o relatie generic-generic, notata de obicei cu AKO (A Kind Of). Pentru o prezentare mai detaliata a acestui model cititorul poate consulta Fikes si Kehler [1985] si Florea [1993].

Figura 5.1 prezinta o retea semantica care contine clase de obiecte, cum ar fi Persoana, Barbat, Jucator–de–baschet, si instante ale acestor clase, cum este obiectul particular j1. Reteaua semantica din figura contine relatii generic-generic, sau subclasa-clasa, între clasele Jucator–de–baschet si Barbat, respectiv Barbat si Persoana, si relatia individual-generic între jucatorul j1 si clasa Jucator–de–baschet. Se observa existanta proprietatilor Înaltime, Echipa si Medie–aruncari care caracterizeaza clasa Jucator–de–baschet. Acestea reprezinta relatii specifice ale retelei semantice. Proprietatea Echipa are valoarea cunoscuta MîiniLungi si toate instantele clasei Jucator–de–baschet vor avea aceeasi valoare a proprietatii Echipa. Nodurile retelei semantice catre care puncteaza proprietatile obiectelor contin valori ale acestor atribute, în cazul în care aceste valori se cunosc. În Figura 5.2 este prezentata aceeasi retea semantica, îmbogatita cu fatetele valoare implicita, pe lînga fatetele valoare. Astfel, se considera ca valoarea implicita a înaltimii unui barbat este 1.80 m, si cea a unui jucator de baschet 2.00 m. În plus, se presupune ca media aruncarilor unui jucator de baschet are valoarea implicita 5.

În logica cu predicate de ordinul I, relatiile subclasa-clasa definite în reteaua semantica din Figura 5.1 pot fi reprezentate astfel:

$$(\forall x)\ (\text{Jucător} - \text{de} - \text{baschet}(x) \rightarrow \text{Bărbat}(x))$$

$$(\forall x)\ (\text{Bărbat}(x) \rightarrow \text{Persoană}(x))$$

iar relatia instanta-clasa între un jucator j si clasa careia îi apartine se reprezinta prin

$$(\exists j)\ \text{Jucător} - \text{de} - \text{baschet}(j)$$

Considerînd clasa Jucator–de–baschet si proprietatile ei asociate, instanta j1 a acestei clase se poate reprezenta în logica cu predicate de ordinul I astfel:

$$\text{Jucător} - \text{de} - \text{baschet}(j1) \wedge \text{Nume}(j1, \text{Stancu}) \wedge \text{Înălțime}(j1, 2.10) \wedge \text{Echipa}(j1, \text{MîiniLungi})$$

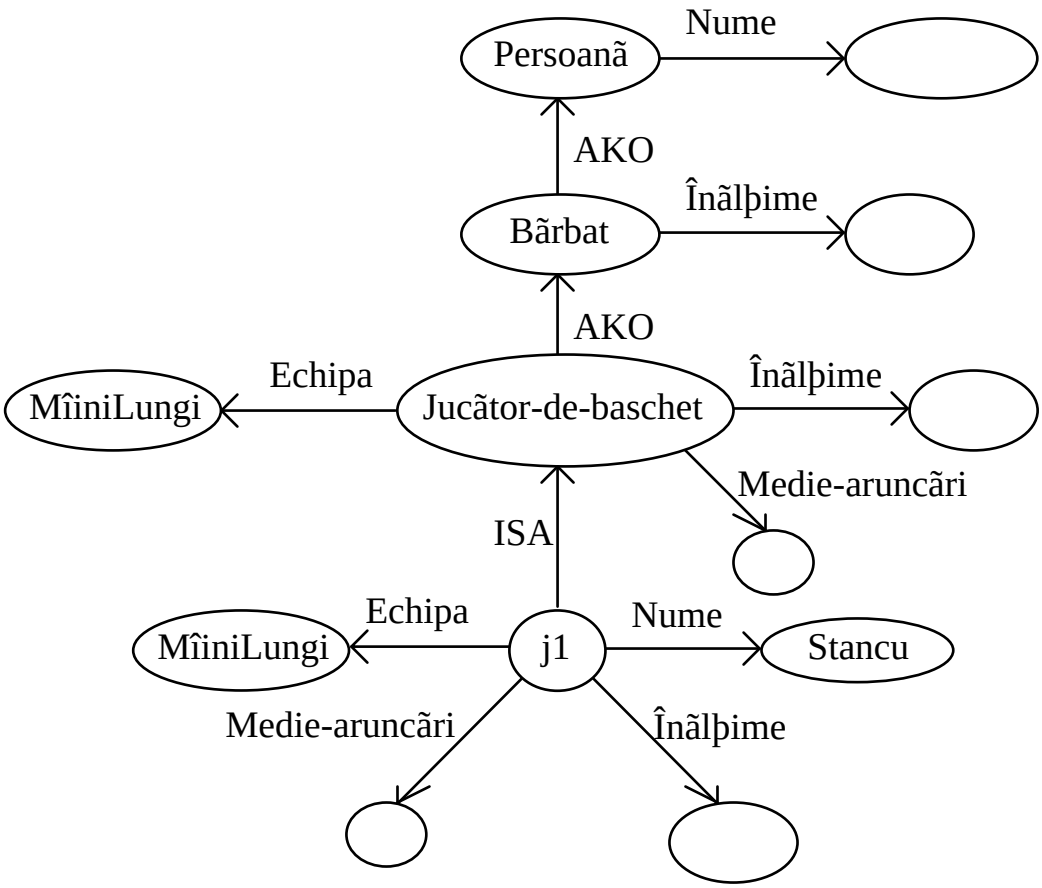


Figura 5.1 Retea semantica cu ierarhie de clase

Metoda de inferenta de baza a unor astfel de reprezentari este *mostenirea proprietatilor* de-a lungul relatiilor ISA si AKO. Aceasta metoda de inferenta specifica reprezentarii structurate a cunostintelor, suporta atît mostenirea monotona, realizata foarte eficient pentru aceste modele, cît si mostenirea nemonotona a valorilor implicite. Într-o reprezentare structurata, instantele mostenesc atributele claselor carora apartin, iar clasele mostenesc atributele superclaselor în care sînt incluse. O fateta valoare a atributului unei clase se va mosteni ca valoare a atributului unei instante a acelei clase numai în cazul în care nu se cunoaste valoarea atributului pentru acea instanta. Aceasta mostenire reprezinta o forma de inferenta nemonotona.

Regula generala de mostenire este: un obiect mosteneste valori de atribute de la toate clasele a caror membru este, cu conditia ca valorile obtinute sa nu conduca la o contradictie. În cazul în care o astfel de contradictie apare, o valoare apartinînd unei clase mai restrînse are prioritate fata de o valoare care apartine unei clase mai largi de obiecte. În cazul în care se considera mai multe fatete asociate unui atribut, este rolul strategiei de control sa specifice ordinea de utilizare a acestor valori.

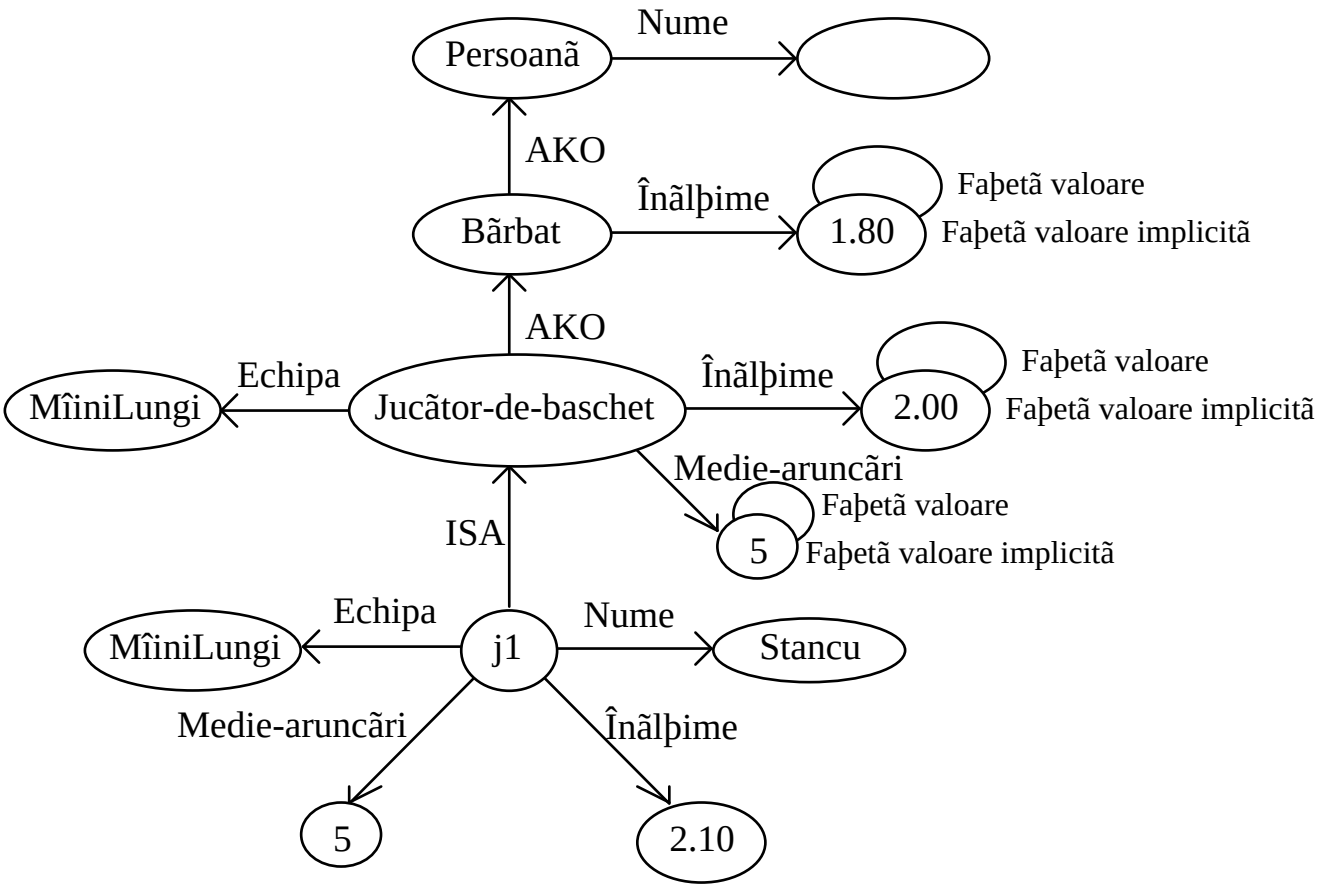


Figura 5.2 Retea semantica cu fatete valoare si valoare implicita

În exemplul din Figura 5.2, instanta j1 mosteneste proprietatile Echipa, Înaltime si Medie–aruncari de la clasa Jucator–de–baschet si atributul Nume de la clasa Persoana. Se observa ca atributul Înaltime este la rîndul lui mostenit de clasa Jucator–de–baschet de la superclasa Barbat. Pentru instanta j1 se cunoaste valoarea atributelor Nume si Înaltime, ca fiind Stancu si, respectiv, 2.10 m (se presupune ca au fost specificate explicit) si se mosteneste monoton valoarea atributului Echipa. Prin mostenire nemonotona se poate infera faptul ca media aruncarilor jucatorului j1 este 5. Daca nu s-ar fi indicat explicit valoarea 2.10 m ca înaltime a jucatorului j1, atunci j1 ar fi mostenit nemonoton valoarea atributului Înaltime de 2.00 m de la clasa Jucator–de–baschet.

Cum s-ar putea reprezenta mostenirea valorilor pentru acest exemplu în formalismul logicii implicite a lui Reiter? Se poate scrie o regula de inferenta implicita pentru a modela mostenirea valorii implicite a slotului Înaltime, de forma

$$(\forall x)\ (\text{Jucător} - \text{de} - \text{baschet}(x) : \text{Înălțime}(x, 2.00) / \text{Înălțime}(x, 2.00))$$

si se adauga asertiunea $\text{Jucător} - \text{de} - \text{baschet}(\text{Stancu})$ în baza de cunostinte. În aceste conditii, se poate infera faptul $\text{Înălțime}(\text{Stancu}, 2.00)$. Daca se adauga în baza de cunostinte o alta asertiune, $\text{Înălțime}(\text{Stancu}, 2.10)$, prin care se specifica explicit înaltimea lui Stancu, asa cum s-a

considerat în exemplu, atunci trebuie sa se adauge o regula prin care sa se precizeze ca înaltimea unui individ este unica. Aceasta regula poate avea forma:

$$(\forall x)(\forall y)(\forall z)\ (\text{Înãlþime}(x,y) \wedge \text{Înãlþime}(x,z) \rightarrow \text{Egal}(y,z))$$

Conform acestei reguli, odata ce Stancu are o înaltime de 2.10 m, nu mai este consistent ca el sa aiba si o înaltime de 2.00 m. Aceasta regula va împiedica aplicarea regulii implicite. Astfel, o valoare a înaltimii specificata explicit va bloca procesul de mostenire a valorilor implicite, obtinînd astfel efectul dorit. Evident, obtinerea înaltimii de 2.10 sau de 2.00 depinde de ordinea de aplicare a regulilor implicite.

Lucrurile se complica daca se considera valorile implicite specificate de toate superclasele. În exemplul prezentat, relatia între clasele Jucator–de–baschet si Barbat cît si valorile implicite ale atributului Înaltime din aceste doua clase pot fi reprezentate în logica implicita a lui Reiter astfel:

$$(\forall x)\ (\text{Jucãtor} - \text{de} - \text{baschet}(x) : \text{Înãlþime}(x,2.00) / \text{Înãlþime}(x,2.00))$$

$$(\forall x)\ (\text{Bãrbat}(x) : \text{Înãlþime}(x,1.80) / \text{Înãlþime}(x,1.80))$$

$$(\forall x)\ (\text{Jucãtor} - \text{de} - \text{baschet}(x) \rightarrow \text{Bãrbat}(x))$$

Presupunînd, din nou, ca nu exista nici o valoare explicit specificata a înaltimii lui Stancu, atunci exista doua extensii posibile ale bazei de cunostinte:

- prima extensie adauga la baza de cunostinte faptul $\text{Înãlþime}(\text{Stancu}, 2.10)$
- a doua extensie adauga la baza de cunostinte faptul $\text{Înãlþime}(\text{Stancu}, 1.80)$

Nici una din aceste extensii nu va fi preferata. Pentru a indica faptul ca se prefera o anumita extensie, adica ca se prefera categoria cu specificitate mai mare, Jucator–de–baschet, se poate rescrie regula implicita care defineste înaltimea barbatilor astfel:

$$(\forall x)\ (\text{Bãrbat}(x) : \sim \text{Jucãtor} - \text{de} - \text{baschet}(x) \wedge \text{Înãlþime}(x,1.80) / \text{Înãlþime}(x,1.80))$$

În acest fel, se blocheaza efectiv a doua extensie a bazei de cunostinte.

Aceasta abordare devine impracticabila daca multimea de exceptii de la regula generala creste. Daca se doreste extinderea valabilitatii regulii implicite de definire a înaltimii barbatilor, aceasta poate fi reformulata, de exemplu, astfel:

$$(\forall x)\ (\text{Bãrbat}(x) : \sim \text{Jucãtor} - \text{de} - \text{baschet}(x) \wedge \sim \text{Chinez}(x) \wedge \sim \text{Jocheu}\ (x) \wedge$$

$$\text{Înãlþime}(x,1.80) / \text{Înãlþime}(x,1.80))$$

Noua forma a regulii încearca de fapt sa surprinda cunostintele generale despre clasa barbatilor si sa enumere toate exceptiile acestei clase. O formulare mai clara poate fi obtinuta exprimînd cunostintele de mai sus astfel: "Barbatii au, de obicei, înaltimea de 1.80 m, daca nu sînt diferiti într-un anumit sens". În acest fel, se pot introduce clase distincte, informatia specifica exceptiilor se poate asocia acestor clase si reprezentarea se simplifica prin utilizarea unei singure reguli implicite, asa cum se arata în continuare.

$$(\forall x)\ (\text{Bãrbat}(x) \wedge \sim \text{Diferit}(x, \text{aspect1}) \rightarrow \text{Înãlþime}(x, 1.80))$$

$$(\forall x)\ (\text{Jucãtor} - \text{de} - \text{baschet}(x) \rightarrow \text{Diferit}(x, \text{aspect1}))$$

$$(\forall x)\ (\text{Chinez}(x) \rightarrow \text{Diferit}(x, \text{aspect1}))$$

$$(\forall x)\ (\text{Jocheu}(x) \rightarrow \text{Diferit}(x, \text{aspect1}))$$

$$(\forall x)(\forall y)\ (\sim \text{Diferit}(x, y) / \sim \text{Diferit}(x, y))$$

Regulile scrise mai sus surprind cazul implicit si exceptiile sale, si au o claritate si generalitate crescuta.

Observatie vesela. Ambiguitatile generate de diversele formalisme logice nemonotone sînt rezolvate în sistemele cu reprezentare structurata a cunostintelor la nivelul strategiei de aplicare a inferentelor specifice. Odata stabilita strategia de mostenire a valorilor, sistemul poate lucra consistent cu valori implicite. Solutiile de implementare pentru rezolvarea acestei probleme sînt prezentate în Sectiunea 5.4.

5.3 Rationament minimal

O alta forma de rationament nemonoton este rationamentul bazat pe ideea *modelului minimal*. Prin definitie, un *model* este o interpretare a unei formule bine formate pentru care aceasta este adevarata. Toate interpretarile unei formule bine formate pentru care aceasta are valoarea adevarat formeaza multimea de modele posibile ale formulei. Un model al unei multimi de formule este considerat minimal daca el contine numarul minim de fapte adevarate necesare pentru a rezolva o problema. Diversele formalizari ale rationamentului implicit, prezentate în sectiunea anterioara, au indicat metode pentru a descrie fapte care sînt "în general" adevarate. Abordarile rationamentului minimal specifica metode de descriere a unei clase restrînse de lucruri care sînt adevarate.

Utilizînd rationamentul minimal, se pot rezolva diverse probleme dificile ale inteligentei artificiale, generate de reprezentarea cunostintelor de bun simt. Rationamentul despre actiunile cotidiene genereaza trei probleme considerate clasice în cadrul comunitatii de inteligenta artificiala: problema cadrului, problema calificarii si problema ramificarii. *Problema cadrului*

[McCarthy,Hayes,1969] consta în determinarea tuturor faptelor care nu se modifica în urma producerii unui eveniment, a unei actiuni sau a unei serii de evenimente si actiuni, odata cu trecerea timpului. *Problema calificarii* [McCarthy,1977] consta în determinarea preconditiilor necesare executiei unei actiuni. De cele mai multe ori este imposibil sa se specifice explicit toate preconditiile care trebuie sa fie adevarate pentru executia unei actiuni sau producerea unui eveniment. *Problema ramificarii* [Finger,1987] consta în specificarea tuturor efectelor executiei unei actiuni. Aceste probleme pot fi (partial) rezolvate de diversele modele de rationament minimal propuse.

Utilizarea modelului minimal ca suport al rationamentului nemonoton se bazeaza pe urmatoarea idee: "Exista mai putine asertiuni adevarate decît false. Daca un fapt este adevarat si relevant pentru problema, este consistent sa se presupuna ca acel fapt a fost enuntat, deci adaugat la baza de cunostinte". Se pleaca deci de la presupunerea ca se cunosc toate faptele care sînt adevarate. În continuare se vor prezenta doua abordari ale rationamentului minimal:

- presupunerea lumii închise
- rationamentul circumscris

5.3.1 Presupunerea lumii închise

O modalitate simpla de realizare a rationamentului minimal este *presupunerea lumii închise* [Reiter,1978]. În acest model, se presupune ca numai obiectele care satisfac un anumit predicat P sînt obiectele pentru care P este adevarat.

Un exemplu tipic al presupunerii lumii închise este reprezentat de cunostintele codificate într-o baza de date, asa cum arata Reiter [1978]. Fie o baza de date care contine informatii despre cursele unei companii aeriene. Daca se pune întrebarea "Exista o cursa între Bucuresti si Paris?", un sistem deductiv de întrebare-raspuns va trata baza de date ca o multime de fapte pe baza carora va încerca sa demonstreze $\text{Conectat}(\text{Bucure}^{o}\text{ti}, \text{Paris})$. Daca demonstratia reuseste sistemul va raspunde "da", iar daca nu reuseste sistemul raspunde tipic "nu", i.e. concluzioneaza $\sim \text{Conectat}(\text{Bucure}^{o}\text{ti}, \text{Paris})$. Utilizarea modelului presupunerii lumii închise simplifica considerabil reprezentarea problemei. Numai informatiile pozitive despre problema trebuie reprezentate explicit în baza de date. Informatiile negative nu sînt reprezentate explicit (în exemplu nu se indica toate orasele între care nu exista curse aeriene) si sînt inferate pe baza lipsei informatiilor pozitive.

Un alt exemplu de model al presupunerii lumii închise este întîlnit în limbajul Prolog. În acest limbaj nu se poate reprezenta *negatia logica pura.* De exemplu, nu se poate codifica direct negatia logica pura $(\forall x)\,(\text{Cãþel}(x) \rightarrow \sim \text{Pisicã}(x))$. În locul acesteia, negatia este reprezentata implicit prin absenta faptelor; predicatul Prolog predefinit not(P) este adevarat numai daca nu exista fapte sau reguli care satisfac predicatul P. Acest lucru duce la o strategie de rezolvare a problemelor numita *negarea ca insucces*. Raspunsul sistemului Prolog la întrebarile

?– pisica(grivei) si ?– pisica(pusi) va fi negativ atît timp cît nu exista fapte explicite în baza de date Prolog care sa indice ca Pusi (cel mai probabil) si Grivei sînt pisici. Deci conceptul de negare ca insucces implica modelul lumii închise, i.e. se presupune ca toate faptele relevante pentru problema exista în baza de date Prolog. Orice asertiune care nu este prezenta poate fi considerata ca fiind falsa.

Acest model functioneaza în Prolog deoarece limbajul modeleaza clauze Horn. Pentru clauze generale, modelul lumii închise poate genera probleme de inconsistenta, asa cum se va prezenta mai departe.

Mostenirea valorilor într-un limbaj de reprezentare structurata a cunostintelor poate fi considerata, de asemenea, pe lînga un exemplu de rationament implicit, si un exemplu de presupunere a lumii închise. Considerînd ierarhia de obiecte prezentata în Sectiunea 5.2.3, daca clasa Jucator–de–baschet are ca generalizare clasa Barbat, procedura de mostenire a proprietatilor va cauta valorile atributelor numai în superclasa Barbat, fara sa tina cont de faptul ca Jucator–de–baschet ar putea avea ca generalizare si clasa Femeie si ca aceasta legatura ar putea fi adaugata ulterior.

Modelul presupunerii lumii închise poate fi util în rezolvarea problemei cadrului. Fie urmatoarea problema, numita *problema misionarilor si canibalilor*. Trei misionari si trei canibali ajung la un rîu. Exista o barca de doua locuri cu care se poate traversa rîul. Daca numarul canibalilor este mai mare decît numarul misionarilor pe unul din malurile rîului, misionarii vor fi mîncati de canibali. Cum pot trece toti misionarii si canibalii rîul fara ca misionarii sa fie mîncati?

În aceasta problema, definitia unei actiuni de traversare a rîului specifica modificarea pozitiei barcii si a numarului de canibali si misionari de pe fiecare mal. Nicaieri în problema nu se specifica explicit fapte ca: la traversarea rîului capacitatea barcii nu se modifica, acelasi numar de persoane care pleaca de pe un mal ajung pe celalalt (nici canibalii nici misionarii nu sar în apa între timp), distanta dintre maluri nu se modifica ca urmare a traversarii rîului, nedevenind atît de mare încît sa fie nevoie de un vapor pentru traversare si nu de o barca.

Aceste cunostinte, considerate implicite în descrierea problemei, exemplifica dificultatile care pot apare în momentul în care se doreste formalizarea executiei unei actiuni din viata de zi cu zi. O solutie practica a acestei probleme a fost data de sistemele de planificare automata STRIPS [Fikes,Nilsson,1971], sistem prezentat în Capitolul 6, PLANNER [Hewitt,1971] si MICRO-PLANNER [Sussman,s.a.,1970]. Aceste sisteme, dedicate sintetizarii unui plan de actiune pentru realizarea unui scop dat, abordeaza problema cadrului asociind cu fiecare actiune liste de fapte care au fost adaugate si fapte care au fost sterse prin executia actiunii. Sistemele STRIPS, PLANNER si MICRO-PLANNER reprezinta un alt exemplu de model al presupunerii lumii închise. Se presupune ca singurele fapte care se schimba sînt acelea explicit specificate în descrierea actiunii. De exemplu, mutarea unui cub dintr-o camera în alta va schimba pozitia cubului, si acest lucru este explicit specificat în descrierea actiunii, dar nu va modifica nici

culoarea, nici greutatea cubului, culoarea si greutatea nefacînd parte dintre faptele adaugate sau sterse. Problema acestei abordari este aceea ca, pentru ca actiunile care au schimbat starea sistemului sa functioneze corect, trebuie sa se modifice si toate faptele care au fost deduse pe baza faptelor sterse de o actiune. De exemplu, daca într-o anumita stare s-a dedus (prin mai multi pasi inferentiali) ca un cub este sub o lustra, într-o noua stare obtinuta prin mutarea cubului în alta camera, deductia trebuie eliminata. O posibila solutie eficienta a acestei probleme este înregistrarea dependentelor, asa cum se va discuta în Sectiunea 5.5.

Desi modelul lumii închise este deosebit de simplu si de puternic, el poate esua în reprezentarea unui model adecvat, din doua motive. Întîi, anumite parti ale lumii reale nu pot fi într-adevar închise. De exemplu, în problema crimei ABC, prezentata la începutul acestui capitol, existau fapte relevante, dar care nu fusesera înca descoperite pîna la un moment dat. În al doilea rînd, aplicarea modelului pe clauze generale, si nu numai pe clauze Horn, cum este cazul limbajului Prolog, poate genera inconsistente.

Fie baza de cunostinte formata dintr-o singura asertiune $\text{Bun}(\text{Vlad}) \vee \text{Înalt}(\text{Vlad})$. Modelul lumii închise permite deducerea atît a faptului $\sim \text{Bun}(\text{Vlad})$, deoarece $\text{Bun}(\text{Vlad})$ nu poate fi dedus din baza de cunostinte, cît si a faptului $\sim \text{Înalt}(\text{Vlad})$, din acelasi motiv. Din nefericire, baza de cunostinte care rezulta

$\text{Bun}(\text{Vlad}) \vee \text{Înalt}(\text{Vlad})$

$\sim \text{Bun}(\text{Vlad})$

$\sim \text{Înalt}(\text{Vlad})$

este inconsistenta.

Aceasta problema deriva din faptul ca, în presupunerea lumii închise, s-a asociat un statut special instantelor pozitive ale predicatelor, spre deosebire de instantele negative. De fapt, presupunerea lumii închise forteaza (implicit) completarea bazei de cunostinte prin adaugarea asertiunii negate ~P, ori de cîte ori este consistent sa se procedeze astfel. În multe cazuri, o astfel de completare poate fi arbitrara. De exemplu, daca se defineste predicatul $\text{Burlac}(x)$ prin faptele:

$\text{Burlac}(\text{Nostradamus})$

$\text{Burlac}(\text{Miticã})$

si se încearca stabilirea adevarului faptului $\text{Burlac}(\text{Napoleon})$, în modelul lumii închise raspunsul este $\sim \text{Burlac}(\text{Napoleon})$. Dar daca, în locul predicatului $\text{Burlac}(x)$, se alege predicatul $\text{Cãsãtorit}(x)$ si se exprima aceleasi cunostinte sub forma

$\sim \text{Cãsãtorit}(\text{Nostradamus})$

$\sim \text{Cãsãtorit}(\text{Miticã})$

raspunsul dedus va fi ~Cãsãtorit(Napoleon). În consecinta, aceleasi cunostinte pot genera concluzii contradictorii, în functie de semnificatia predicatelor alese. Aceste inconsistente apar în modelul lumii închise datorita acceptarii clauzelor generale, deci si a clauzelor care nu sînt clauze Horn distincte.

5.3.2 Rationament circumscris

Pentru a formaliza rationamentul minimal si a elimina limitarile aduse de presupunerea lumii închise, McCarthy [1980] propune *teoria circumscrierii*. Circumscrierea este o conjectura care poate fi utilizata de o persoana sau de un program pentru a obtine concluzii imediate. Circumscrierea are la baza urmatoarea idee: obiectele pentru care se poate arata ca satisfac o proprietate (predicat) P, plecînd de la anumite asertiuni (fapte) A, reprezinta toate obiectele ce satisfac proprietatea P. Mai general, circumscrierea poate fi utilizata pentru a conjectura ca tuplele <x,y,...,z>, pentru care se poate arata ca satisfac o relatie P(x,y,...,z), sînt toate tuplele care pot satisface aceasta relatie. Astfel, se circumscriu multimi de tuple relevante.

Teoria circumscrierii elimina caracterul global al modelului lumii închise, iar rationamentul minimal din presupunerea lumii închise se aplica numai la o portiune a bazei de cunostinte, deci numai unui anumit predicat sau unor anumite predicate, prin procesul de circumscriere.

În plus, circumscrierea rezolva *problema calificarii*. Pentru a reprezenta complet toate conditiile necesare executiei unei actiuni, este nevoie de un numar urias de conditii, practic imposibil de reprezentat în întregime. Utilizînd circumscrierea, se poate elimina necesitatea specificarii explicite a tuturor acestor conditii.

Se considera din nou problema misionarilor si a canibalilor, prezentata în sectiunea anterioara, si se presupune ca se cere unei persoane sa o rezolve. Persoana, dupa ce se gîndeste putin, poate raspunde: "Pot trece de pe un mal pe altul daca merg în josul rîului si trec peste pod". Se poate obiecta "Dar nu exista nici un pod!", iar persoana raspunde: "Nu s-a spus explicit în problema. Iar daca nu am pod, pot sa iau un elicopter, iar daca nu am elicopter pot sa leg canibalii astfel încît sa nu manînce misionarii". Pe de alta parte, persoana poate obiecta "Cum sa rezolv problema daca barca este sparta si se poate scufunda". Desi pozitia ei enerveaza, persoana are dreptate, deoarece nu s-a spus nicaieri în problema ca barca nu este sparta, sau ca nu exista pod, elicopter, sau ca nu se pot lega canibalii pentru a-i face inofensivi. Evident, este de dorit evitarea includerii în problema a tuturor acestor preconditii. Circumscrierea poate fi utilizata pentru a realiza acest lucru. Ea permite conjectura prin care nu exista alte obiecte explicite decît cele descrise în problema si, eventual, cele care se refera la cunostinte de bun simt.

Daca se circumscrie reprezentarea problemei în logica cu predicate de ordinul I se poate deduce ca nu exista nici pod, nici elicopter. Daca se adauga si cunostinte de bun simt, cum ar fi "o barca poate fi folosita la traversarea rîului în cazul în care barca nu este defecta sau nu exista ceva

care sa-i împiedice utilizarea", si nu exista fapte care sa indice una din aceste doua situatii, dificultatile specificate mai sus sînt eliminate.

Daca se postuleaza ca o barca poate fi utilizata la traversarea unui rîu daca nu exista ceva care sa împiedice aceasta utilizare a barcii, circumscrierea se poate utiliza pentru a conjectura ca singurele lucruri care pot împiedica folosirea barcii sînt cele care se cunosc. Corectitudinea acestei concluzii depinde de considerarea sau ignorarea faptelor relevante în momentul aplicarii circumscrierii.

Circumscrierea este o regula de conjectura formalizata care poate fi utilizata împreuna cu alte reguli de inferenta ale logicii cu predicate de ordinul I. *Circumscrierea predicatelor* presupune ca anumite entitati satisfac un predicat dat numai daca trebuie sa-l satisfaca pe baza unei multimi de fapte. *Circumscrierea domeniului* conjectureaza faptul ca entitatile cunoscute reprezinta toate entitatile care exista. McCarthy arata ca circumscrierea domeniului, cunoscuta si sub numele de inferenta minimala si echivalenta cu presupunerea lumii închise, este subsumata de circumscrierea predicatelor.

Rezultatul aplicarii circumscrierii unei multimi de fapte A este o expresie care indica faptul ca singurele tuple care satisfac un predicat $P(x,y,...,z)$ sînt acelea care satisfac predicatul pe baza propozitiilor din A. Deoarece adaugarea unei propozitii suplimentare multimii A ar putea face ca P sa fie adevarat pentru mai multe tuple, circumscrierea nu este o regula de inferenta monotona. Concluziile derivate prin circumscriere sînt conjecturi conform carora A include toate faptele relevante si obiectele a caror existenta decurge din A sînt toate obiectele relevante existente. Din punct de vedere formal McCarthy defineste circumscrierea astfel:

Definitie. Fie A(P) o formula bine formata în logica cu predicate de ordinul I continînd un predicat $P(x_1,...,x_n)$, notat cu $P(\bar{x})$. Fie ϕ o expresie predicativa, care contine de obicei toate instantele cunoscute ale predicatului P. Se noteaza cu $A(\phi)$ rezultatul înlocuirii tuturor aparitiilor predicatului P în formula A prin expresia predicativa ϕ. *Circumscrierea* predicatului P în A(P) este formula:

$$A(\phi) \wedge (\forall \bar{x})\ (\phi\ (\bar{x}) \rightarrow P(\bar{x})) \rightarrow (\forall \bar{x})\ (P\ (\bar{x}) \rightarrow \phi(\bar{x}))$$

Interpretarea formulei este urmatoarea: singurele tuple $(\bar{x})$ care satisfac predicatul P sînt acele tuple care trebuie sa satisfaca predicatul P, presupunînd asertiunea A adevarata. Formula contine un parametru predicat ϕ care poate fi substituit cu orice predicat Q; daca s-ar folosi o logica de ordinul II, s-ar putea indica acest lucru prin $(\forall\phi)$. Deoarece circumscrierea lui P în A(P) este o implicatie, pentru ca circumscrierea sa fie adevarata se poate presupune ca, daca ambele conjunctii din partea stînga sînt adevarate, atunci partea dreapta a formulei este adevarata. Fiecare componenta a formulei din definitie poate fi interpretata astfel:

- $A(\phi)$ exprima presupunerea ca expresia predicativa ϕ satisface conditiile satisfacute de P, deci ca $\phi(\bar{x})$ este adevarata ori de cîte ori $P(\bar{x})$ este adevarat.

- $(\forall \bar{x})\ (\phi(\bar{x}) \rightarrow P(\bar{x}))$ exprima presupunerea ca faptele care satisfac expresia predicativa ϕ sînt o submultime a faptelor care satisfac predicatul P.
- $(\forall \bar{x})\ (P(\bar{x}) \rightarrow \phi(\bar{x}))$ reprezinta asertiunea inversa, deci toate faptele pentru care $P(\bar{x})$ este adevarat sînt fapte pentru care $\phi(\bar{x})$ este adevarata. În acest caz, se poate deduce ca ϕ coincide cu P.

În continuare sînt prezentate cîteva exemple de realizare a circumscrierii.

Se presupune ca o cunoastem pe Irina, care are parul rosu. Daca vedem o persoana cu parul rosu se poate presupune, prin circumscriere, ca acea persoana este Irina. Inferenta este nemonotona deoarece, daca între timp mai aflam ca exista o alta femeie cu parul rosu, nu se mai poate infera ca persoana vazuta este Irina. Exprimat formal, acest exemplu devine:

$$A = Păr - ro^{o}u(Irina)$$

Circumscrierea predicatului Par–rosu în A este:

$$\phi(Irina) \wedge (\forall x)\ (\phi(x) \rightarrow Păr - ro^{o}u(x)) \rightarrow (\forall x)\ (Păr - ro^{o}u(x) \rightarrow \phi(x))$$

Singura instanta cunoscuta a predicatului $Păr - ro^{o}u(x)$ este Irina, deci $\phi(x) \equiv (x = Irina)$. În formula de mai sus, se înlocuieste predicatul ϕ cu exprimarea lui echivalenta $(x = Irina)$ si se obtine

$$(Irina = Irina) \wedge (\forall x)\ (x = Irina \rightarrow Păr - ro^{o}u(x)) \rightarrow (\forall x)\ (Păr - ro^{o}u(x) \rightarrow x = Irina)$$

Tinînd cont de A, primele doua parti ale acestei formule sînt adevarate; se obtine astfel presupunerea (implicita) $(\forall x)\ (Păr - ro^{o}u(x) \rightarrow x = Irina)$. Inferenta este nemonotona deoarece, daca se adauga $Păr - ro^{o}u(Maria)$ la A, nu se mai poate infera aceeasi concluzie. În schimb, se poate infera o alta aplicare a circumscrierii, si anume

$$(\forall x)\ (Păr - ro^{o}u(x) \rightarrow x = Irina \vee x = Maria)$$

În lumea blocurilor se considera formulele care specifica ca B, C si D sînt blocuri:

$$A = (Bloc(B) \wedge Bloc(C) \wedge Bloc(D)) \qquad (i)$$

Se circumscrie predicatul Bloc în A si se obtine

$$\phi(B) \wedge \phi(C) \wedge \phi(D) \wedge (\forall x)\ (\phi(x) \rightarrow Bloc(x)) \rightarrow (\forall x)\ (Bloc(x) \rightarrow \phi(x)) \qquad (ii)$$

Daca se considera ca B, C si D sînt singurele blocuri cunoscute, atunci

$$\phi(x) \equiv (x = B \vee x = C \vee x = D) \qquad (iii)$$

si, prin substitutia lui $\phi(x)$ cu (iii) în (ii), rezulta

$$(B = B \vee B = C \vee B = D) \wedge (C = B \vee C = C \vee C = D) \wedge (D = B \vee D = C \vee D = D) \wedge$$

$$(\forall x)\ (x = B \vee x = C \vee x = D \rightarrow Bloc(x)) \rightarrow (\forall x)\ (Bloc(x) \rightarrow (x = B \vee x = C \vee x = D))\quad (\text{iv})$$

Deoarece partea stînga a implicatiei (iv) este adevarata, se obtine

$$(\forall x)\ (Bloc(x) \rightarrow (x = B \vee x = C \vee x = D)) \qquad (\text{v})$$

Semnificatia formulei (v) este aceea ca singurele blocuri din asertiunea A sînt B, C si D, adica chiar acele obiecte pe care (i) le considera blocuri. Acest exemplu este aproape trivial deoarece (i) nu ofera posibilitatea generarii de noi blocuri pe baza celor existente. Cu toate acestea, exemplul arata ca inferentele facute prin circumscriere sînt nemonotone deoarece, daca se adauga Bloc(E) la (i), atunci nu se mai poate infera (v).

Tot în lumea blocurilor fie

$$A = (Bloc(B) \vee Bloc(C)) \qquad (\text{i})$$

Se circumscrie predicatul Bloc în A si se obtine

$$(\phi(B) \vee \phi(C)) \wedge (\forall x)\ (\phi(x) \rightarrow Bloc(x)) \rightarrow (\forall x)\ (Bloc(x) \rightarrow \phi(x)) \qquad (\text{ii})$$

Apoi se substituie succesiv $\phi(x) \equiv (x = B)$ si $\phi(x) \equiv (x = C)$ în (ii), ceea ce conduce la

$$(B = B \vee B = C) \wedge (\forall x)\ (x = B \rightarrow Bloc(x)) \rightarrow (\forall x)\ (Bloc(x) \rightarrow x = B) \qquad (\text{iii})$$

$$(C = B \vee C = C) \wedge (\forall x)\ (x = C \rightarrow Bloc(x)) \rightarrow (\forall x)\ (Bloc(x) \rightarrow x = C) \qquad (\text{iv})$$

Prin simplificare se obtine:

$$Bloc(B) \rightarrow (\forall x)\ (Bloc(x) \rightarrow x = B) \qquad (\text{iii}')$$

$$Bloc(C) \rightarrow (\forall x)\ (Bloc(x) \rightarrow x = C) \qquad (\text{iv}')$$

Din (i), (iii') si (iv') rezulta

$$(\forall x)\ (Bloc(x) \rightarrow x = B) \vee (\forall x)\ (Bloc(x) \rightarrow x = C)$$

ceea ce indica faptul ca fie B este singurul bloc, fie C este singurul bloc din asertiunea A.

Circumscrierea nu este o logica nemonotona, ci o forma de rationament nemonoton care îmbogateste calculul cu predicate de ordinul I. În plus, McCarthy afirma ca rationamentul implicit oferit de anumite sisteme logice este mai putin general decît circumscrierea. În sprijinul acestei idei el da urmatorul exemplu. În lumea blocurilor, un bloc x este considerat a fi peste un bloc y numai daca acest lucru este indicat explicit; situatia implicita este aceea în care blocul x nu este peste blocul y. Atunci, pentru fiecare bloc x, se poate concluziona ca x nu este peste un alt bloc

A, dar nu se poate concluziona, cum este cazul circumscrierii, ca nu exista nici un bloc peste blocul A. În logicile implicite, pentru a ajunge la aceeasi concluzie, ar trebui adaugat un fapt implicit separat, care sa spuna ca un bloc nu are nici un alt bloc deasupra lui, cu exceptia cazului în care s-a afirmat explicit contrariul.

5.4 Rezolvarea problemelor utilizînd rationamentul nemonoton

Programele de inteligenta artificiala care implementeaza rationamentul nemonoton pornesc de la unul din formalismele teoretice prezentate si trebuie sa rezolve, la nivel pragmatic, limitarile existente în aceste formalisme, urmarind în acelasi timp eficienta computationala. Unul dintre aspectele esentiale de care trebuie tinut cont este acela al caracterului semidecidabil al rationamentului nemonoton. Aceste meta-abordari trebuie sa poata rezolva problemele (1)÷(3) prezentate în Sectiunea 5.1 si, în plus, trebuie sa urmareasca eficienta procesului de calcul.

Solutiile prezentate în continuare se bazeaza pe o aceeasi idee comuna, si anume separarea procesului de rezolvare a problemei în doua parti:

- Realizarea inferentelor nemonotone pe baza extinderii reprezentarii si a mecanismelor de functionare a motorului de inferenta.
- Mentinerea consistentei bazei de cunostinte si cresterea eficientei procesului de rezolvare a problemelor.

În aceasta sectiune se prezinta diverse modalitati de îmbogatire a reprezentarii cunostintelor si modificarile implicate de acestea în functionarea motorului de inferenta. În Sectiunile 5.5 si 5.6 se prezinta solutii pentru mentinerea consistentei bazei de cunostinte. Capitolul 6 prezinta alte exemple de programe care implementeaza o forma de rationament nemonoton: sistemele de planificare automata.

5.4.1 Mostenirea proprietatilor în reprezentarile structurate

În Sectiunea 5.2.3 s-a discutat extinderea reprezentarii structurate a cunostintelor prin introducerea fatetei valoare implicita a unui atribut. Mostenirea proprietatilor, cu forma de inferenta nemonotona a mostenirii valorilor implicite, a fost prezentata tot în sectiunea amintita.

Inconsistentele generate de reprezentarea si utilizarea inferentelor nemonotone în formalismele logice pot fi eliminate în sistemele bazate pe reprezentari structurate prin modul de implementare a strategiei de control. În continuare se va discuta o solutie de implementare care poate conduce la rezolvarea problemelor de inconsistenta sau ambiguitate aparute în cazul extensiilor multiple ale bazei de cunostinte într-un formalism nemonoton.

Procesul de rezolvare a problemelor într-o astfel de reprezentare consta în obtinerea de instante, pe baza ierarhiei si a relatiilor definite în baza de cunostinte si pe baza unor instante partial sau total specificate care reprezinta datele initiale ale problemei. Instantele astfel obtinute reprezinta solutia problemei. Pentru a obtine instante, sistemul trebuie sa aleaga o valoare pentru fiecare proprietate a obiectului particular cautat. Aceasta valoare poate fi furnizata explicit de utilizator sau poate fi obtinuta pe baza fatetelor atributului din clasele sau superclasele legate de instanta în ierarhie, conform mecanismului de mostenire descris. Ordinea de inspectare a fatetelor atributelor si de parcurgere a ierarhiei de clase pentru a obtine o valoare este data de strategia de control a sistemului. Odata stabilita aceasta strategie ambiguitatile rationamentului implicit dispar.

În continuare, atît relatiile generale de tip ISA si AKO, cît si relatiile specifice, deci atributele reprezentarii structurate, se vor numi sloturi, conform unei denumiri consacrate în reprezentarea bazata pe cadre. Reprezentarea utilizata ca exemplu este cea a retelelor semantice, dar algoritmii prezentati pot fi aplicati, cu mici modificari, si în cazul reprezentarilor bazate pe unitati (cadre). Se considera urmatoarele trei fatete posibile ale unei proprietati (slot): fateta valoare, fateta valoare implicita si fateta procedura necesara. Fateta procedura necesara este o functie sau procedura atasata unui slot, care este invocata atunci cînd se doreste calculul dinamic al valorii unui slot. În consecinta, toate cele trei fatete pot produce o valoare pentru o anumita proprietate, strategia de control fiind cea care alege între o valoare sau alta.

Se considera exemplul retelei semantice descrise în Sectiunea 5.2.3 (Figura 5.2). Se adauga clasei Jucator–de–baschet o noua proprietate, Performante, care are asociata o fateta procedura necesara, calcul. Daca se doreste aflarea valorii atributului Performante al unei instante j1 a clasei Jucator–de–baschet, se apeleaza functia de calcul care întoarce ca valoare performantele, pe baza valorilor atributelor Înaltime si Medie–aruncari ale instantei j1.

În continuare se prezinta doi algoritmi de mostenire a proprietatilor într-o reprezentare structurata a cunostintelor. Cei doi algoritmi implementeaza doua strategii de control, strategia N si strategia Z, care difera prin ordinea de inspectare a fatetelor valoare, valoare implicita si procedura necesara, în determinarea valorii unui slot.

Strategia Z inspecteaza fatetele valoare, procedura necesara si valoare implicita ale slotului pentru care se doreste gasirea unei valori, pe fiecare nivel al ierarhiei de clase, pornind de la instanta si considerînd relatiile ISA, apoi AKO. În algoritm, ierarhia de clase este construita sub forma unei liste de clase în subprogramul Ierarhie. Prima clasa din lista este chiar clasa de la care se porneste inspectarea ierarhiei de clase.

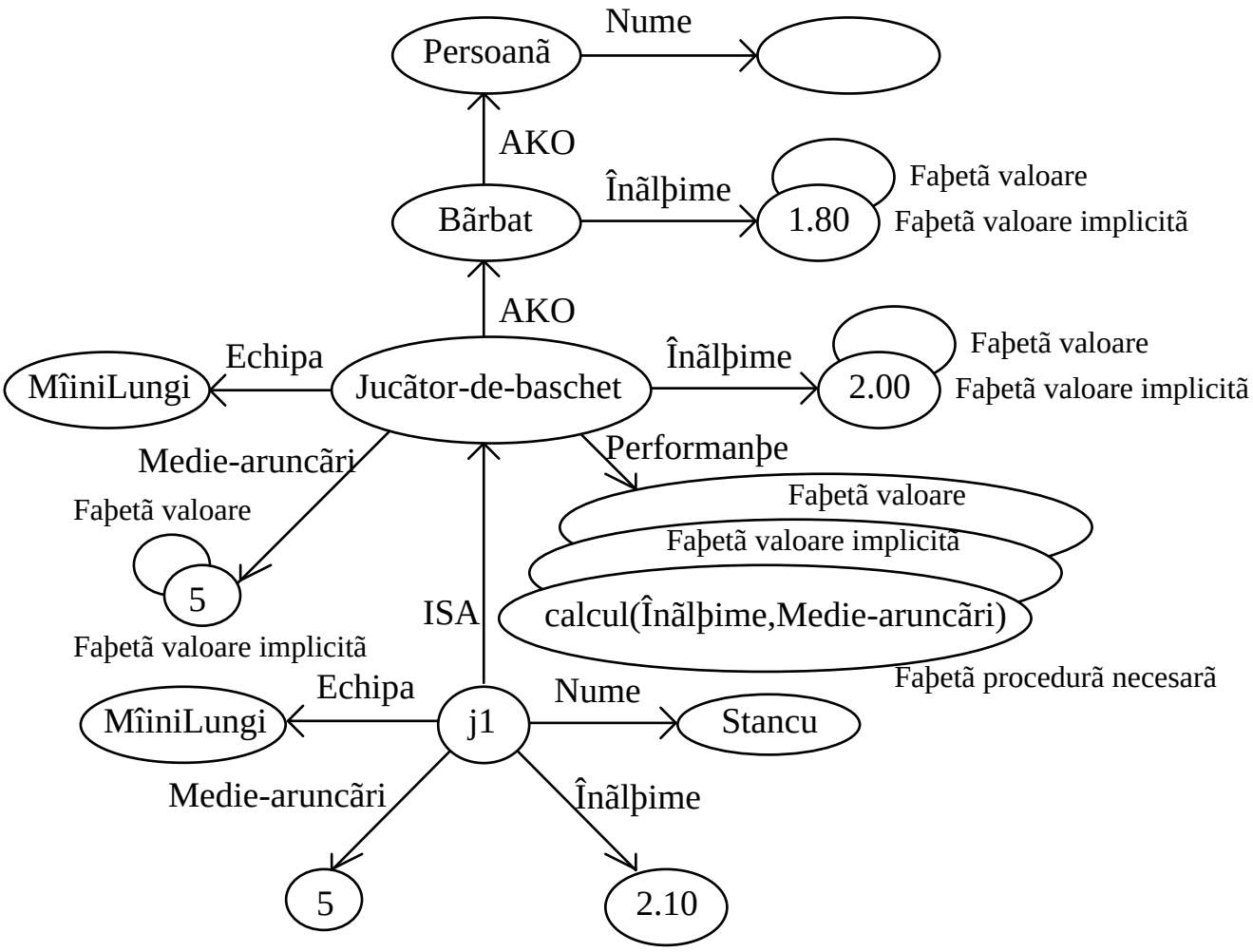


Figura 5.3 Retea semantica cu diverse fatete

Construirea ierarhiei de clase în primul pas al subprogramului ObtineValoare a fost preferata unei alte abordari posibile, de exemplu parcurgerea ierarhiei concomitent cu inspectarea fatetelor, deoarece în cazul unei ierarhii care contine clase ce mostenesc de la mai multe clase, parcurgerea ierarhiei concomitent cu inspectarea ei ar fi mai dificila, necesitînd reveniri la clasele anterioare.

Algoritm: Strategia Z

ObtineValoare(Clasa,Slot)

1. I ← Ierarhie(Clasa)
2. **cît timp** I ≠ { } **executa**
   - 2.1. Clasa ← primul element din I
   - 2.2. **daca** exista Clasa **.** Slot **.** VALOARE
     atunci întoarce Clasa **.** Slot **.** VALOARE
   - 2.3. **daca** exista Clasa **.** Slot **.** PROCEDURÃ_ NECESARÃ
     atunci întoarce Clasa **.** Slot **.** PROCEDURÃ_ NECESARÃ
   - 2.4. **daca** exista Clasa **.** Slot **.** VALOARE_ IMPLICITÃ

atunci întoarce Clasa **.** Slot **.** VALOARE_IMPLICITÃ

2.5. I ← I - {Clasa}

3. **întoarce** INSUCCES

sfîrsit.

Ierarhie(Clasa)

1. Q ← {Clasa}
2. R ← { }
3. **cît timp** Q ≠ { } **executa**
   - 3.1. Clasa ← primul element din Q
   - 3.2. **daca** Clasa ∉ R
     atunci
     - 3.2.1. R ← R + {Clasa}
     - 3.2.2. Q ← Q + {Clasa **.** AKO **.** VALOARE}
   - 3.3. Q ← Q - {Clasa}
4. **întoarce** R

sfîrsit.

În strategia N de mostenire a valorilor se executa mai întîi o inspectare a ierarhiei de clase folosind o singura fateta, respectiv fateta valoare. Daca nu se gaseste o valoare pentru slotul primit ca parametru, se executa o noua inspectare a tuturor nivelelor ierarhiei, folosind de aceasta data informatia continuta în fateta procedura necesara a sloturilor. În cazul în care nici aceasta inspectare nu stabileste o valoare pentru slotul dat, se executa o noua inspectare a ierarhiei, folosindu-se informatia continuta în fateta valoare implicita a sloturilor.

Algoritm: Strategia N

ObtineValoare(Clasa,Slot)

1. I ← Ierarhie(Clasa)
2. I_1 ← I
3. **cît timp** I_1 ≠ { } **executa**
   - 3.1. Clasa ← primul element din I_1
   - 3.2. **daca** exista Clasa **.** Slot **.** VALOARE
     atunci întoarce Clasa **.** Slot **.** VALOARE
   - 3.3. I_1 ← I_1 - {Clasa}
4. I_1 ← I
5. **cît timp** I_1 ≠ { } **executa**

5.1. Clasa ← primul element din I_1

5.2. **daca** exista Clasa **.** Slot **.** PROCEDURÃ_ NECESARÃ
atunci întoarce Clasa **.** Slot **.** PROCEDURÃ_ NECESARÃ

5.3. $I_1 \leftarrow I_1$ - {Clasa}

6. $I_1 \leftarrow I$

7. **cît timp** $I_1 \neq \{\}$ **executa**

7.1. Clasa ← primul element din I_1

7.2. **daca** exista Clasa **.** Slot **.** VALOARE_ IMPLICITÃ
atunci întoarce Clasa **.** Slot **.** VALOARE_ IMPLICITÃ

7.3. $I_1 \leftarrow I_1$ - {Clasa}

8. **întoarce** INSUCCES

sfîrsit.

Observatii:

- Algoritmul strategiei N utilizeaza acelasi subprogram Ierarhie(Clasa) ca si algoritmul strategiei Z. Acest subprogram nu a mai fost inclus în algoritmul strategiei N.

- Anumite sisteme implementeaza strategii dinamice în care, pe baza unor meta-cunostinte, sistemul poate comuta de la caz la caz, între o ordine de inspectare si alta.

- Strategiile N si Z nu sînt singurele strategii de control utilizate în modelul cunostintelor structurate. În functie de reprezentarea particulara aleasa, controlul va fi extins adecvat cu componente specifice.

- Algoritmii prezentati permit legarea unei instante de mai multe clase si, respectiv, a unei clase de mai multe subclase, deci existenta unor relatii multiple ISA si AKO. În aceste conditii, ierarhia de clase are forma unui graf si nu acea a unui arbore. Acest lucru reprezinta o generalizare importanta, dar care poate conduce la inconsistente. În cazul în care ierarhia este astfel definita încît sa permita obtinerea mai multor valori de acelasi tip (valoare, valoare implicita sau procedura necesara) pe baza reletiilor multiple, si atributele accepta numai valori unice, strategiile prezentate nu reusesc sa aleaga între aceste valori competitive. O posibila solutie a acestei probleme este modificarea algoritmului pe baza distantei inferentiale între instante si clase. Detalii suplimentare pot fi gasite în Rich si Knight [1991] si Florea [1993].

Utilizînd sistemul orientat pe obiecte CLOS (Common Lisp Object System) din Common Lisp, exemplul prezentat în Figura 5.3 poate fi descris în urmatorul mod.

(defclass Persoana ()

```
  ((Nume :initarg :Nume :reader Nume)))

(defclass Barbat (Persoana)
  ((Inaltime :initarg :Inaltime :initform 1.80 :reader Inaltime)))

(defclass Jucator-de-baschet (Barbat)
  ((Inaltime :initarg :Inaltime :initform 2.00 :reader Inaltime )
   (Medie-aruncari :initarg :Medie-aruncari :initform 5 :accessor Medie-aruncari)
   (Echipa :initarg :Echipa :initform MiiniLungi)
   (Performante :initform 0 :accessor Performante)))

(defmethod performante1 ((jucator Jucator-de-baschet))
  (setf (Performante jucator)
        (calcul (Inaltime jucator) (Medie-aruncari jucator))))

;;  creeaza jucatorul j1 cu numele Stancu, inaltimea 2.10, media aruncarilor 5 (implicita)
    si echipa MiiniLungi
(setf j1 (make-instance 'Jucator-de-baschet
                        :Nume "Stancu"
                        :Inaltime 2.10))

;; calculeaza performantele jucatorului j1
(performante j1)
```

În continuare este prezentata o solutie de implementare în limbajul Lisp a strategiilor descrise anterior. Structura unei unitati (clase) este cea prezentata mai jos. Aceasta structura este memorata ca valoare a proprietatii UNITATE a atomului <nume unitate>.

```
(<nume unitate> (<slot 1> (<fateta 1> <valoare 1> ... <valoare n>) ...
                          (<fateta m> <valoare 1> ... )) ...
                (<slot p> (<fateta 1> <valoare 1> ... ) ... ) ... )
```

Toate celelalte elemente ale structurii sînt reprezentate prin atomi Lisp. Functia *uassoc* cauta o cheie într-o lista de asociatii (cu ajutorul functiei Lisp predefinite assoc) si daca nu o gaseste o adauga "chirurgical" la sfîrsitul listei de asociatii. Functia *obtine-unitate* întoarce unitatea asociata simbolului primit ca parametru, daca aceasta unitate exista, altfel asociaza o unitate vida aceluiasi simbol. Functia *u-adauga* adauga valori într-o unitate, *u-obtine* regaseste informatia stocata într-o unitate pe baza unei cai unitate-slot-fateta, iar *u-sterge* sterge o valoare dintr-o unitate pe baza unei cai similare. Functia *u-obtine-vmi* încerca obtinerea unei valori pentru un slot dintr-o unitate, folosind informatia aflata în fatetele *VALOARE*, *PROCEDURA-NECESARA* si *VALOARE-IMPLICITA*, în aceasta ordine. În aceasta implementare se presupune

ca procedurile necesare nu au argumente, ca intorc o lista de valori posibil vida si ca pot utiliza variabilele speciale **UNITATE** si **SLOT** pentru a calcula un rezultat.

Functiile *obtine-valoare-Z*, *obtine-valoare-N* si *ierarhie* reprezinta implementarea efectiva a strategiilor prezentate anterior.

```
(defvar *UNITATE*)
(defvar *SLOT*)

(defun uassoc (cheie lista-asociatii)
  (cond ((assoc cheie (rest lista-asociatii)))
        (t (cadr (rplacd (last lista-asociatii) (list (list cheie)))))))

(defun obtine-unitate (unitate)
  (when unitate
    (cond ((get unitate 'UNITATE))
          (t (setf (get unitate 'UNITATE) (list unitate))))))

(defun u-adauga (unitate slot fateta valoare)
  (cond ((member valoare (u-obtine unitate slot fateta)) nil)
        (t (push valoare (rest (uassoc fateta (uassoc slot (obtine-unitate unitate)))))
           valoare)))

(defun u-obtine (unitate slot fateta)
  (rest (assoc fateta (rest (assoc slot (rest (get unitate 'UNITATE)))))))

(defun u-sterge (unitate slot fateta valoare)
  (let* ((sloturi (obtine-unitate unitate))
         (fatete (assoc slot (rest sloturi)))
         (valori (assoc fateta (rest fatete))))
    (when (member valoare (rest valori))
      (setf (rest valori) (delete valoare (rest valori)))
      (unless (rest valori)
        (delete valori fatete))
      (unless (rest fatete)
        (delete fatete sloturi))
      t)))

(defun u-obtine-vmi (unitate slot)
  (let ((*UNITATE* unitate)
        (*SLOT* slot))
```

```
   (cond ((u-obtine unitate slot 'VALOARE))
         ((delete nil (mapcar #'(lambda (val-activa) (funcall val-activa))
                              (u-obtine unitate slot 'PROCEDURA-NECESARA))))
         (t (u-obtine unitate slot 'VALOARE-IMPLICITA)))))

(defun ierarhie (unitate)
  (do ((coada (list unitate) (rest coada))
       stramos unitati)
      ((null coada) (reverse unitati))
    (unless (member (first coada) unitati)
      (push (first coada) unitati))
    (setf stramos (u-obtine (first coada) 'AKO 'VALOARE))
    (setf coada (append coada stramos))))

(defun  obtine-valoare-Z (unitate slot)
  (do* ((unitati (ierarhie unitate) (rest unitati))
        (unitate1 (first unitati) (first unitati))
        rezultat)
       ((cond ((null unitati))
              ((setf rezultat (u-obtine-vmi unitate1 slot))))
        rezultat)))

(defun obtine-valoare-N (unitate slot)
  (let ((unitati (ierarhie unitate) (rest unitati))
        rezultat)
    (do ((unitati1 unitati (rest unitati1)))
        ((cond ((null unitati1))
               ((setf rezultat (u-obtine (first unitati1) slot 'VALOARE))))))
    (cond (rezultat rezultat)
          ((do* ((unitati1 unitati (rest unitati1))
                 (*UNITATE* (first unitati1) (first unitati1))
                 (*SLOT* slot))
                ((cond ((null unitati1))
                       ((setf rezultat
                              (delete nil
                                      (mapcar #'(lambda (val-activa) (funcall val-activa))
                                              (u-obtine *UNITATE* slot
                                                        'PROCEDURA-NECESARA))))))
                 t))
           (cond (rezultat rezultat)
```

```
((do ((unitati1 unitati (rest unitati1)))
     ((cond ((null unitati1))
            ((setf rezultat
                   (u-obtine (first unitati1) slot 'VALOARE-IMPLICITA))))
      t))
 rezultat))))))
```

5.4.2 Extinderea modelului regulilor de productie

Reprezentarea cunostintelor utilizînd regulile de productie este un model frecvent întîlnit în programele de inteligenta artificiala [Buchanan,Shortliffe,1984;Hayes-Roth,s.a.,1983;Hayes-Roth,1985]. Regulile de productie reprezinta o varianta simplificata a regulilor de inferenta deductiva din logica cu predicate de ordinul I. Modelul acestor reguli poate fi extins pentru a surprinde si inferentele nemonotone. Pe lînga operatorii standard dintr-un limbaj bazat pe reguli: **daca**, **si**, **atunci**, se poate adauga operatorul nemonoton **dacanu**. Acesta are o semnificatie apropiata operatorului **M** din logica nemonotona a lui McDermott si Doyle. Extinderea modelului regulilor de productie prin introducerea operatorului **dacanu** a fost facuta de Sandewall [1972], într-un sistem deductiv nemonoton bazat pe reguli.

Considerînd problema crimei ABC prezentata la începutul acestui capitol, asertiunile problemei pot fi reprezentate într-un model de reguli nemonotone prin regulile si faptele din Figura 5.4.

Sistemele de rationament nemonoton bazate pe reguli permit introducerea conditiilor de tip **dacanu**, ca baza a rationamentului implicit. Regulile pot fi executate atît utilizînd înlantuirea înainte cît si înlantuirea înapoi. Controlul, inclusiv decizia asupra carei interpretari implicite se va opri sistemul, este tratat conform strategiei generale de control în astfel de sisteme, de exemplu ordonare de reguli sau utilizarea meta-regulilor pentru selectia regulii de aplicat. O discutie detaliata despre strategia de control a sistemelor bazate pe reguli, pentru cazul rationamentului monoton, poate fi gasita în Buchanan si Shortliffe [1984] si în Cooper si Wogrin [1988].

Reguli

R1: **daca** x este beneficiar
si dacanu x are alibi
atunci x este suspect

R2: **daca** x este înregistrat la hotel la y
si y este departe
si dacanu registru falsificat la y
atunci x are alibi

R3: **daca** x este aparat de y
si dacanu y minte
atunci x are alibi

R4: **daca** poza lui x la y
si y este departe
atunci x are alibi

R5: **dacanu** exista un suspect x
atunci contradictie

Fapte

Alecu este beneficiar
Barbu este beneficiar
Cezar este beneficiar

Figura 5.4 Reguli de productie nemonotone.

suspect(X) ← beneficiar(X), **dacãnu** alibi(X).
alibi(X) ← înregistrat_la_hotel(X,Y), departe(Y), **dacãnu** registru_falsificat(Y).
alibi(X) ← aparã(X,Y), **dacãnu** minte(Y).
alibi(X) ← poza(X,Y), departe(Y).
contradicþie← **dacãnu** suspect(X).

beneficiar(alecu).
beneficiar(barbu).
beneficiar(cezar).

Figura 5.5 Reguli nemonotone într-un sistem de tip Prolog

Sistemele de rationament nemonoton cu înlantuire înapoi a regulilor pot adopta doua solutii:

- introducerea conditiilor **dacanu** în reguli si rezolvarea conflictelor generate cu aceeasi strategie de rezolvare a conflictelor utilizata în general de sistem
- implementarea unui mecanism de alegere între diverse valori posibile pentru o asertiune prin construirea unor justificari pentru fiecare dintre aceste valori.

Modelul cel mai simplu de sistem cu înlantuire înapoi a regulilor este sistemul Prolog. În acest caz nu se mai construieste multimea de conflicte a regulilor aplicabile, ci se aplica întotdeauna prima regula aplicabila întîlnita. Într-un sistem gen Prolog, regulile crimei ABC ar putea fi exprimate ca în Figura 5.5. În Prolog pur, operatorul **dacanu** poate fi înlocuit cu metapredicatul standard **not**.

Pentru un astfel de sistem tip Prolog, daca se încearca demonstrarea scopului suspect(X) programul va produce X = alecu. Daca ulterior se adauga faptele

înregistrat_la_hotel(alecu,arad)

departe(arad)

programul va desemna ca suspect pe X = barbu. Un mecanism simplist, de tipul metapredicatului **not** din Prolog, nu este întotdeauna adecvat sau eficient. La adaugarea de noi fapte, care infirma faptele anterioare, sistemul nu poate determina faptele care nu mai sînt valide. Aceasta este o deficienta generala a sistemelor care lucreaza cu înlantuire înapoi a regulilor, deoarece au un control condus de scopuri, spre deosebire de cele care lucreaza cu înlantuire înainte si au un control condus de date.

Tot pentru cazul sistemelor cu înlantuire înapoi a regulilor, se poate considera o a doua paradigma de strategie de control: se încearca gasirea tuturor valorilor care îndeplinesc un scop, apoi se încearca alegerea unei valori dintre aceste variante posibile. Alegerea se poate face pe baza unei alte reguli (metaregula); pentru exemplul considerat, se poate adauga o regula care indica ca un alibi furnizat de o ruda a suspectului este mai putin credibil decît un alibi furnizat de înregistrarea persoanei în registrul unui hotel serios. Un exemplu de implementare în limbajul Lisp a modelului nemonoton al regulilor de productie va fi prezentat în Sectiunea 5.6.2.

Sistemele de rationament nemonoton cu înlantuire înapoi a regulilor pot adopta doua solutii:

- introducerea conditiilor **dacanu** în reguli si rezolvarea conflictelor generate cu aceeasi strategie de rezolvare a conflictelor utilizata în general de sistem
- implementarea unui mecanism de alegere între diverse valori posibile pentru o asertiune prin construirea unor justificari pentru fiecare dintre aceste valori.

Modelul cel mai simplu de sistem cu înlantuire înapoi a regulilor este sistemul Prolog. În acest caz nu se mai construieste multimea de conflicte a regulilor aplicabile, ci se aplica întotdeauna prima regula aplicabila întîlnita. Într-un sistem gen Prolog, regulile crimei ABC ar putea fi exprimate ca în Figura 5.5. În Prolog pur, operatorul **dacanu** poate fi înlocuit cu metapredicatul standard **not**.

Pentru un astfel de sistem tip Prolog, daca se încearca demonstrarea scopului suspect(X) programul va produce X = alecu. Daca ulterior se adauga faptele

înregistrat_la_hotel(alecu,arad)

departe(arad)

programul va desemna ca suspect pe X = barbu. Un mecanism simplist, de tipul metapredicatului **not** din Prolog, nu este întotdeauna adecvat sau eficient. La adaugarea de noi fapte, care infirma faptele anterioare, sistemul nu poate determina faptele care nu mai sînt valide. Aceasta este o deficienta generala a sistemelor care lucreaza cu înlantuire înapoi a regulilor, deoarece au un control condus de scopuri, spre deosebire de cele care lucreaza cu înlantuire înainte si au un control condus de date.

Tot pentru cazul sistemelor cu înlantuire înapoi a regulilor, se poate considera o a doua paradigma de strategie de control: se încearca gasirea tuturor valorilor care îndeplinesc un scop, apoi se încearca alegerea unei valori dintre aceste variante posibile. Alegerea se poate face pe baza unei alte reguli (metaregula); pentru exemplul considerat, se poate adauga o regula care indica ca un alibi furnizat de o ruda a suspectului este mai putin credibil decît un alibi furnizat de înregistrarea persoanei în registrul unui hotel serios. Un exemplu de implementare în limbajul Lisp a modelului nemonoton al regulilor de productie va fi prezentat în Sectiunea 5.6.2.

5.5 Sisteme de mentinere a consistentei datelor

Rezolvarea problemelor care implica rationament nemonoton necesita un proces de cautare a solutiei. Procesul de cautare este complicat suplimentar de existenta presupunerilor, adica a faptelor nemonotone, care pot fi considerate initial ca adevarate, pentru ca ulterior sa fie infirmate sau invers. În acest caz este deosebit de importanta gasirea unor solutii de cautare eficienta. Una dintre aceste solutii este reprezentata de sistemele de mentinere a consistentei datelor.

Ideea sistemelor de mentinere a consistentei datelor consta în înregistrarea dependentelor existente între concluziile sau faptele inferate de sistem si faptele initiale pe baza carora s-a ajuns la aceste concluzii. Unul dintre primele programe care a asociat dependente deductiilor facute a fost sistemul de analiza a circuitelor electrice EL, proiectat de Stallman si Sussman [1977]. Scopul înregistrarii dependentelor în acest sistem era gasirea presupunerilor care au generat

contradictii. Pe baza mecanismului de înregistrare a dependentelor, sistemul EL a introdus strategia de control *backtracking condus de dependente*, prezentata în sectiunea urmatoare.

Doyle [1979] si, independent, London au fost primii care au descoperit ca mecanismul înregistrarii dependentelor, strategia de backtracking condus de dependente si reprezentarea încrederii curente (nemonotone) în anumite presupuneri pot fi implementate în orice program bazat pe cautare, independent de forma de rationament utilizata. Doyle a numit acest mecanism *sistem de mentinere a consistentei datelor*, cu prescurtarea din limba engleza TMS. Ulterior s-a încercat utilizarea unui nume mai sugestiv pentru aceste sisteme, si anume sisteme de mentinere a rationamentului, dar prima denumire era deja impusa în limbaj, asa ca s-a renuntat la cea de a doua. În continuare, aceste sisteme sînt desemnate prin termenul de sisteme de mentinere a consistentei datelor.

Sistemele de mentinere a consistentei datelor au fost un cîmp fertil al cercetarilor de inteligenta artificiala, propunîndu-se noi solutii cum ar fi sistemele de mentinere a consistentei datelor bazate pe logica, introduse de McAllester în 1980, si sistemele propuse de deKleer. deKleer [1986a,b,c] introduce o varianta îmbunatatita a sistemelor de tip TMS, si anume *sistemele de mentinere a consistentei datelor bazate pe presupuneri*, cu prescurtarea din limba engleza ATMS. Spre deosebire de sistemele de mentinere a consistentei, care lucreaza cu justificari, un sistem bazat pe presupuneri lucreaza cu multimi de presupuneri si justificarile asociate acestora. În acest fel, sistemele de mentinere a consistentei bazate pe presupuneri permit utilizarea efectiva si eficienta a informatiilor inconsistente, asa cum se va vedea în Sectiunea 5.5.4.

5.5.1 Principiile sistemelor de mentinere a consistentei datelor

Sistemele de mentinere a consistentei datelor permit efectuarea rationamentului nemonoton si cresc eficienta procesului de cautare. Rolul lor nu este acela de a rezolva problema, deci de a face inferente, ci de a înregistra faptele (convingerile) deduse de sistem si modul în care acestea au fost deduse. Datorita acestei proprietati, un sistem de mentinere a consistentei poate fi utilizat la:

(1) validarea si invalidarea presupunerilor,

(2) refacerea concluziilor abandonate si regasite,

(3) controlul actiunilor programului,

(4) explicarea rationamentului efectuat,

(5) învatare, deoarece un program de învatare trebuie sa poata analiza cauzele succesului si insuccesului unei anumite actiuni.

În continuare se vor utiliza urmatorii termeni:

- *convingere* - expresie care poate fi adevarata sau falsa;
- *justificare* - pas de inferenta care a condus la obtinerea unei convingeri;
- *premisa* - fapt considerat întotdeauna adevarat, i.e. care nu necesita justificare;
- *presupunere* - fapt justificat ca adevarat (are o justificare valida), dar care poate fi ulterior invalidat datorita invalidarii altor fapte de care depinde.

Într-o abordare bazata pe mentinerea consistentei, procesul de rezolvare a problemelor este împartit în doua: motorul de inferenta, care realizeaza inferente, încercînd rezolvarea problemei, si sistemul de mentinere a consistentei datelor care înregistreaza starea curenta a procesului de cautare sub forma unei retele de dependente în care se memoreaza asertiunile si modul în care acestea au fost obtinute (Figura 5.6). Pe parcursul rezolvarii problemei, cele doua componente, motorul de inferenta si sistemul de mentinere a consistentei datelor, schimba informatii între ele. Astfel, motorul de inferenta trimite sistemului de mentinere a consistentei datelor concluziile la care a ajuns si justificarile asociate, iar sistemul de mentinere a consistentei informeaza motorul de inferenta despre asertiunile considerate adevarate. De exemplu, daca baza de cunostinte contine numai asertiunile P si $P \rightarrow Q$ si regula de inferenta Modus Ponens, motorul de inferenta poate concluziona Q, deci Q este adaugat la baza de cunostinte . Ulterior, daca se descopera de fapt ca ~P este convingerea curenta, ~P trebuie adaugat la baza de cunostinte. Pentru a nu exista inconsistente, P trebuie eliminat. În aceste conditii si Q trebuie eliminat, deoarece Q nu mai este adevarat, daca P nu mai este adevarat. Acest mecanism de revizuire a convingerilor este executat de sistemul de mentinere a consistentei datelor.

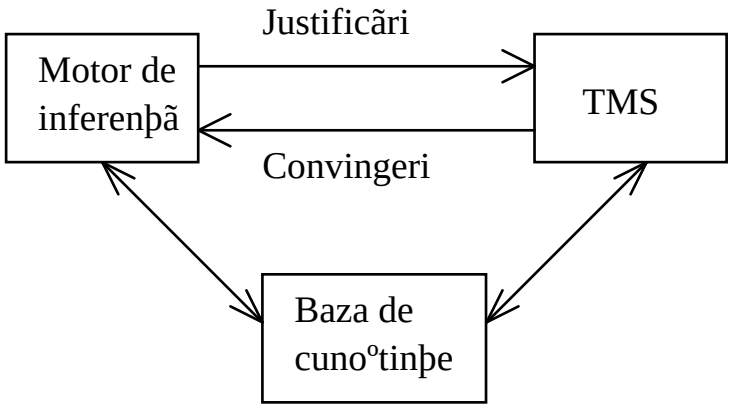


Figura 5.6 Rolul sistemului de mentinere a consistentei în rezolvarea problemelor

De fapt, sistemul de mentinere a consistentei datelor nu elimina asertiunea Q si nici asertiunea P din baza de cunostinte, deoarece, ulterior, P poate fi din nou considerat adevarat, deci si Q redevine adevarat si tot ce s-a inferat pe baza lui Q. În locul eliminarii asertiunii Q din baza de cunostinte, sistemul de mentinere a consistentei datelor considera Q drept o convingere care nu mai este curent activa.

Unul din principalele scopuri ale sistemelor de mentinere a consistentei este implementarea eficienta a startegiei de control *backtraking condus de dependente*. Strategia de backtracking condus de dependente este o combinatie a mecanismelor de backtracking cu salt si backtracking cu marcare, descrise în Capitolul 4. Pentru a explica ideea acestei strategii se considera urmatoarele doua exemple.

Fie urmatoarea problema, cunoscuta sub numele de *problema maimutei si a bananei*. O maimuta intra într-o camera unde exista mai multe cuburi si mai multe banane atîrnate de tavan. Pentru a ajunge la o banana, maimuta trebuie sa ajunga lînga un cub, sa împinga cubul pîna sub o banana, sa se urce pe cub si sa apuce banana.

Daca banana pe care o apuca maimuta este verde, deci nu poate fi mîncata, maimuta condusa de un algoritm de backtracking naiv ar încerca fiecare din cuburile din camera înainte de a ajunge la concluzia ca banana verde este motivul pentru care ea nu o poate mînca, deci motivul care creeaza contradictia. O maimuta condusa de o strategie de backtracking condus de dependente identifica sursa contradictiei si face revenire pîna în punctul în care îsi alege o alta banana la care sa ajunga. Pîna în acest moment, comportamentul este similar cu acela al strategiei de backtracking cu salt. Dar daca maimuta ar implementa numai aceasta strategie, efortul ei de a ajunge lînga cub, fiind ulterior alegerii unei banane, ar fi pierdut deoarece, prin backtracking, cubul si maimuta ar reveni în pozitia initiala. Dupa alegerea unei noi banane, maimuta trebuie sa refaca efortul facut anterior pentru a ajunge la un cub. O maimuta condusa de o strategie de backtracking condus de dependente nu va mai face acest lucru, ci va memora efectul actiunilor anterioare.

Un alt exemplu care arata importanta utilizarii unei strategii de backtracking condus de dependente este cel prezentat în continuare. Desi exemplul este usor patologic, el da o buna indicatie asupra costului computational pe care anumite decizii (alternative de cautare) îl implica în sistemele bazate pe cunostinte.

Fie urmatoarea problema de satisfacere a restrictiilor [deKleer,1986a]:

(1) $x \in \{0,1\}$ (2) $a = e_1(x)$

(3) $y \in \{0,1\}$ (4) $b = e_2(y)$

(5) $z \in \{0,1\}$ (6) $c = e_3(z)$

(7) $b \neq c$ (8) $a \neq b$

Functiile $e_i(x)$, $i = 1,2,3$ sînt de forma $e_i(x) = (x + 100000)!$, deci necesita o cantitate însemnata de calcul. Se cere sa se gaseasca valorile lui a, b si c care satisfac conditiile (1)÷(8).

Daca se începe atribuirea valorilor variabilelor în ordinea în care variabilele apar în domeniile de definitie, combinatiile $x = 0, y = 0, z = 0$ si $x = 0, y = 0, z = 1$ sînt rejectate deoarece nu satisfac restrictiile (7) si (8). Sursa contradictiei este valoarea lui y, deoarece o

solutie este gasita pentru combinatia $x = 0$, $y = 1$, $z = 0$. Dar în acest moment, efortul de calcul substantial facut pentru a calcula $c = e_3(0)$ si $c = e_3(1)$ este pierdut si trebuie refacut pentru noua valoare atribuita lui y.

Strategia de backtracking condus de dependente trebuie sa descopere decizia care a generat contradictia si sa se întoarca cu cautarea în acel punct, si, în acelasi timp, trebuie sa memoreze actiunile efectuate, respectiv calculul lui $e_3(0)$ si $e_3(1)$, pentru a nu le mai recalcula de fiecare data.

Observatie. În cazul problemei satisfacerii restrictiilor, prezentata în Capitolul 4, indicatorul principal al eficientei rezolvarii problemei era numarul de teste de consistenta efectuate pentru a gasi solutia. În cazul unor categorii de probleme de tipul celor discutate aici, se observa ca pasul de atribuire de valori variabilelor poate fi deosebit de costisitor din punct de vedere computational, deci poate avea o pondere importanta în timpul de rezolvare.

Implementarea algoritmului de backtracking condus de dependente necesita urmatoarele elemente:

- Fiecare actiune obtinuta în procesul de cautare trebuie sa aiba asociata justificarea corespunzatoare inferentelor care au generat-o.
- Existenta unui mecanism care, odata obtinuta o contradictie si justificarea ei, calculeaza *multimea minima de presupuneri* ce au generat contradictia. Aceasta multime are urmatoarea proprietate: daca se elimina o presupunere din multime, justificarea contradictiei nu mai este valida si contradictia este eliminata.
- Existenta unui mecanism care sa propage atît efectele adaugarii unei justificari, cît si cele ale eliminarii unei presupuneri, pastrînd astfel consistenta asertiunilor obtinute.
- Existenta unui mecanism de alegere a unei presupuneri din multimea minima de presupuneri care au generat contradictia, presupunere ce va fi infirmata.

Sistemele de mentinere a consistentei datelor rezolva primele trei probleme din lista de mai sus. De obicei, strategia de rezolvare a problemei trebuie sa decida cum se alege presupunerea de infirmat pentru a elimina contradictia.

5.5.2 Functionarea sistemelor de mentinere a consistentei datelor

Sistemul de mentinere a consistentei reprezinta reteaua de dependente care descrie faptele deduse si modul de deducere a acestora cu ajutorul a doua tipuri de entitati: *noduri* si *justificari*. Nodurile reprezinta fapte, reguli de inferenta, proceduri, iar justificarile reprezinta pasi de inferenta de la combinatii de noduri la alte noduri.

Se considera din nou exemplul crimei ABC prezentat la începutul capitolului. Multimea initiala de convingeri este:

Beneficiar(Alecu)

Beneficiar(Barbu)

Beneficiar(Cezar)

si reprezinta starea initiala în care nu s-au descoperit alibiuri pentru cei trei beneficiari ai crimei. Regulile de inferenta care descriu rezolvarea problemei sînt cele prezentate în Sectiunea 5.4.2. Initial se presupune ca Alecu este singurul suspect, deoarece este beneficiarul crimei si nu are alibi. Acest lucru se infera pe baza regulii:

R1: **daca** x este beneficiar
si dacanu x are alibi
atunci x este suspect.

Observatie. Reprezentata în logica implicita a lui Reiter, aceasta regula avea forma:

Beneficiar(x) **:** ~ Alibi(x) / Suspect(x)

Regula de mai sus este valabila pentru prima instanta a lui x, Alecu, atît timp cît Alecu nu are alibi. Faptul ca Alecu este suspect este o convingere în acest moment, deci ea trebuie adaugata la multimea curenta de convingeri. Daca ulterior se descopera ca Alecu are un alibi, convingerea Suspect(Alecu) trebuie eliminata si adaugata eventual o noua convingere. S-ar putea scrie o regula care sa trateze explicit acest caz, dar atunci ar trebui si o regula care sa trateze cazul în care Alecu redevine suspect daca alibiul lui nu mai este crezut (daca, de exemplu, s-a descoperit ca registrul în care era înregistrat la momentul crimei a fost falsificat). În acest fel dezvoltarea unei baze de cunostinte consistente ar deveni deosebit de dificila.

Utilizînd un sistem de mentinere a consistentei datelor, aceste probleme sînt eliminate. Cele trei asertiuni Beneficiar(Alecu), ~ Alibi(Alecu) si Suspect(Alecu) se reprezinta sub forma a trei noduri în reteaua de dependente a sistemului. Fiecare nod poate avea doua stari: starea IN, corespunzatoare încrederii în adevarul acelui nod, sau starea OUT, corespunzatoare contrariului. Multimea asertiunilor valide la un anumit moment dat este multimea nodurilor aflate în starea IN. Nodurile corespunzatoare asertiunilor inferate au asociate justificari.

O justificare este formata dintr-o multime de noduri care reprezinta ipotezele pe baza carora a fost inferata asertiunea justificata (nodul justificat). Aceasta multime este reprezentata sub forma a doua liste:

- Lista IN, care reprezinta lista nodurilor care trebuie sa fie curent valide, i.e. noduri cu starea IN, pentru ca justificarea sa fie valida;

- Lista OUT, care reprezinta lista nodurilor care trebuie sa fie curent invalide, i.e. noduri cu starea OUT, pentru ca justificarea sa fie valida.

Un nod care are o justificare valida are starea IN. Deci un nod justificat are starea IN daca toate nodurile lui din lista IN sînt noduri cu starea IN si toate nodurile din lista OUT sînt noduri cu starea OUT. Asertiunile care sînt întotdeauna valide se numesc *premise* si sînt reprezentate prin noduri care au o justificare pentru care atît lista IN cît si lista OUT sînt liste vide. Asertiunile pentru care nu exista justificari si nici nu sînt permise corespund unor noduri care au starea OUT. Un nod poate avea mai multe justificari, corespunzatoare mai multor inferente posibile pe baza carora a fost dedus.

Cele trei asertiuni Beneficiar(Alecu), ~ Alibi(Alecu) si Suspect(Alecu) pot fi reprezentate ca în Figura 5.7 sau alternativ, utilizînd o notatie de lista

(Suspect Alecu ((Beneficiar Alecu) (Alibi Alecu)))

sau

(N_1 ((N_2) (N_3)))

cu N_1 = Suspect Alecu, N_2 = Beneficiar Alecu si N_3 = Alibi Alecu. Forma generala a reprezentarii justificarii unui nod utilizînd notatia de lista este (Nod (ListaIN ListaOut)).

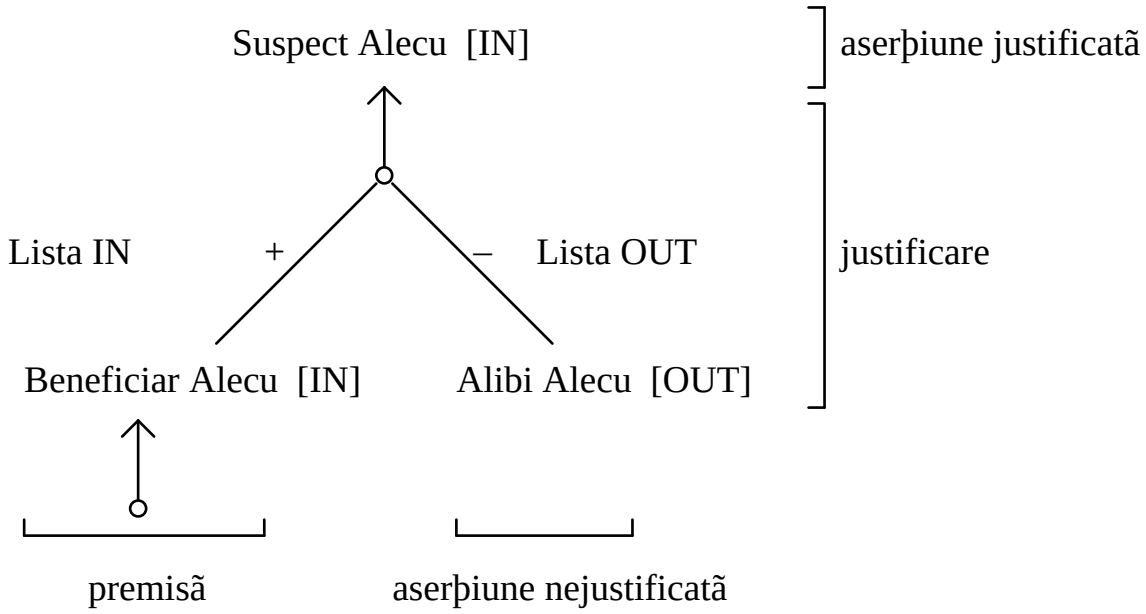


Figura 5.7 Justificare într-o retea de dependente

O premisa va avea urmatoarea reprezentare (nod (( ) ( ))). Pentru premisa din exemplu reprezentarea va fi (Beneficiar Alecu (( ) ( ))). Se observa ca un nod poate avea starea OUT datorita unei justificari invalide, cum este cazul nodului Suspect Alecu din Figura 5.8, sau poate avea starea OUT datorita faptului ca nu exista nici o informatie referitoare la acea asertiune, cum este cazul nodurilor Alibi Alecu din Figura 5.7 sau Falsificat registru din Figura 5.8.

O justificare este *nemonotona* daca lista OUT a acestei justificari este lista vida sau, recursiv, daca exista un nod în lista IN a justificarii, care are o justificare nemonotona. În caz contrar justificarea este *monotona*. O asertiune care are o justificare nemonotona se numeste *presupunere*. Nodurile sustinute de o justificare monotona pot sa-si schimbe starea din IN în OUT numai prin eliminarea explicita a unei premise. Deoarece se presupune ca premisele sînt asertiuni cu valabilitate generala în problema, acesta este un caz de exceptie, care de obicei nu se considera. Nodurile sustinute de o justificare nemonotona pot sa-si schimbe starea din IN în OUT prin adaugarea unei noi justificari în reteaua de dependente.

Sa presupunem ca, dupa reprezentarea celor trei asertiuni, se descopera ca Alecu are un alibi, fiind înregistrat în registrul unui hotel din Arad. Aplicînd regula

R2: **daca** x este înregistrat la hotel la y
si y este departe
si dacanu registru falsificat la y
atunci x are alibi

si regula R1, se poate deduce asertiunea Alibi Alecu, cu conditia ca registrul sa nu fie falsificat. Aceasta inferenta se adauga sub forma unei justificari la reteaua de dependente, iar sistemul de mentinere a consistentei datelor trebuie sa actualizeze starea nodurilor în consecinta, asa cum se prezinta în Figura 5.8. Lista IN a nodului Alibi Alecu este formata din nodurile Înregistrat la hotel Alecu si Departe Arad, iar lista OUT contine nodul Falsificat registru.

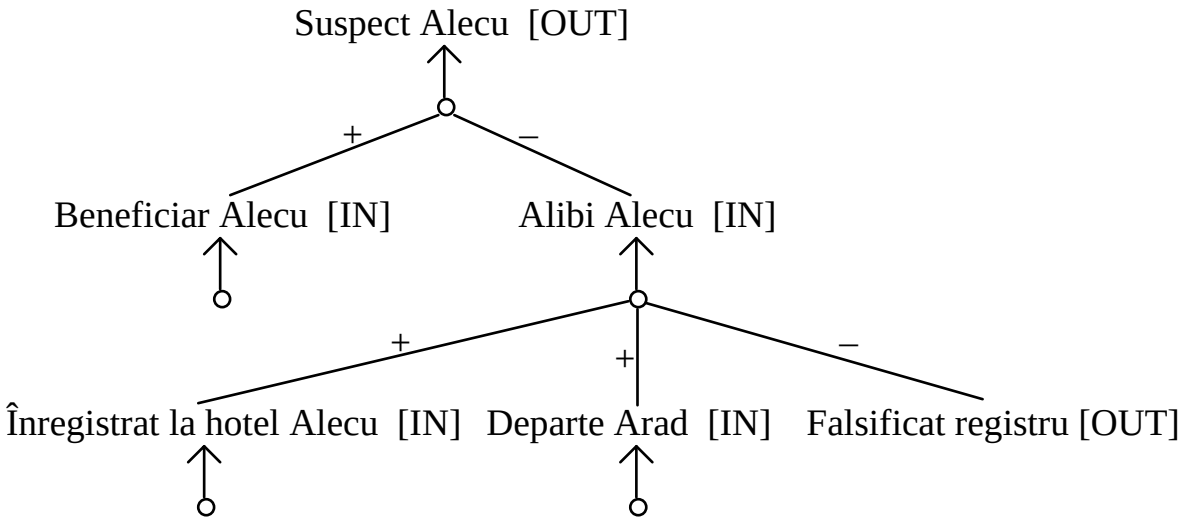


Figura 5.8 Actualizarea starii nodurilor la aparitia unei noi justificari

Similar, pe baza regulii

R3: **daca** x este aparat de y
si dacanu y minte
atunci x are alibi

si a regulii R1, se poate obtine pentru Barbu reteaua de dependente prezentata în Figura 5.9.

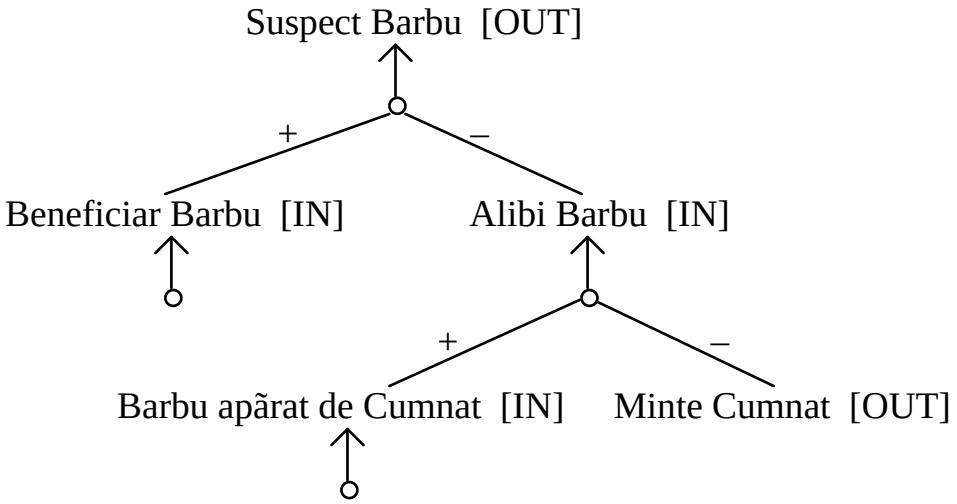


Figura 5.9 Justificarea asertiunii Suspect Barbu

Justificarea ipotezei de suspiciune a lui Cezar trebuie facuta pe baza alibiului lui. Dar alibiul lui Cezar nu are o justificare valida, deoarece nu exista nici o proba în acest sens, în afara propriei lui declaratii. În consecinta, reteaua de dependente are forma prezentata în Figura 5.10. Se observa aparitia unor justificari circulare, caz în care nodurile justificate au starea OUT.

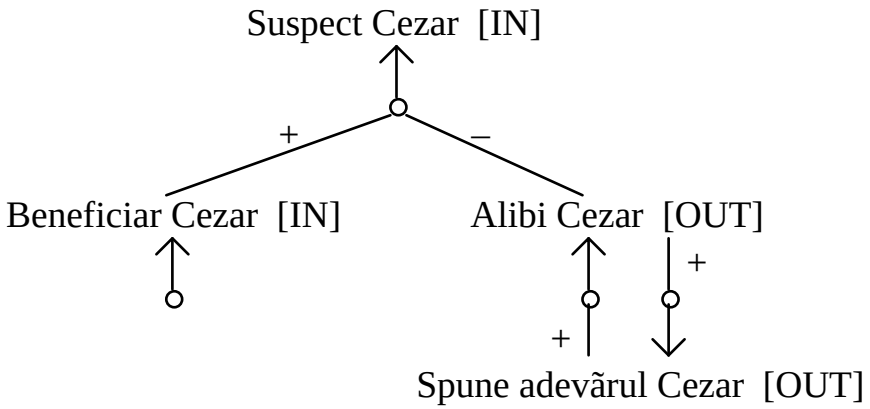


Figura 5.10 Justificarea asertiunii Suspect Cezar

Dupa cum se poate vedea din acest exemplu, primul rol al sistemului de mentinere a consistentei datelor este mentinerea consistentei faptelor inferate. Procedeul de mentinere a consistentei datelor este o problema de satisfacere a restrictiilor care calculeaza o stare valida pentru fiecare nod din reteaua de dependente. Daca contexul curent, determinat de procedura de mentinere a consistentei este inconsistent, i.e. contine contradictii, atunci asertiunile care au generat contradictiile trebuie eliminate. Al doilea rol important al sistemului de mentinere a consistentei datelor este acela de a elimina contradictiile.

În exemplul crimei ABC exista o contradictie daca nu exista nici un suspect. Acest lucru se poate reprezenta printr-un nod Contradictie cu justificarea din Figura 5.11.

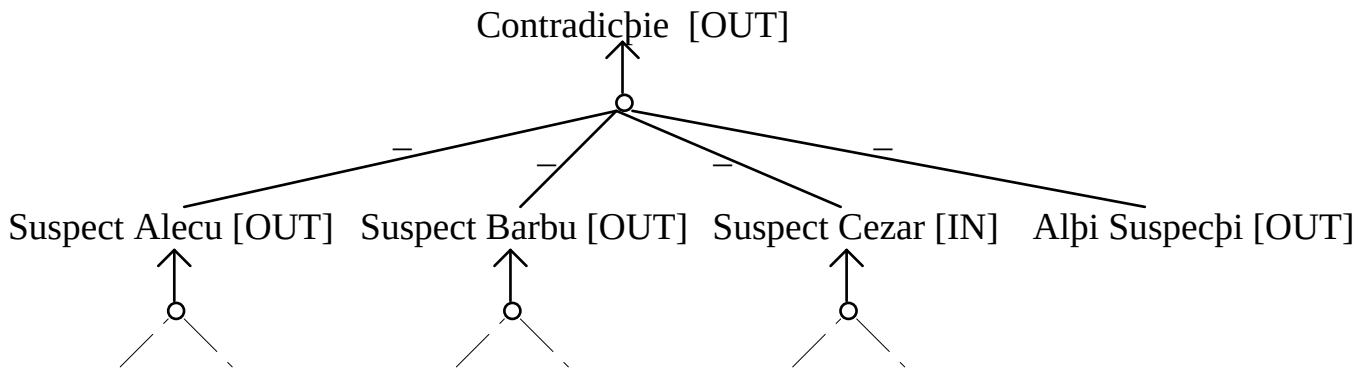


Figura 5.11 Nod contradictie în reteaua de dependente

Pîna în acest moment, Alecu si Barbu nu sînt suspecti, nu se cunosc alti suspecti, dar Cezar este considerat suspect. Nodul Contradictie are starea OUT datorita nodului Suspect Cezar cu starea IN, deci nu exista nici o contradictie. Daca se descopera probe pentru alibiul lui Cezar, se adauga o noua justificare si reteaua de dependente asociata asertiunii Suspect Cezar se modifica ca în Figura 5.12. În acest caz, nodul Suspect Cezar devine OUT, iar nodul Contradictie îsi schimba starea în IN, deoarece toate nodurile din lista OUT a nodului Contradictie au starea OUT. În acest caz, în sistem apare o contradictie, asa cum se vede în Figura 5.13.

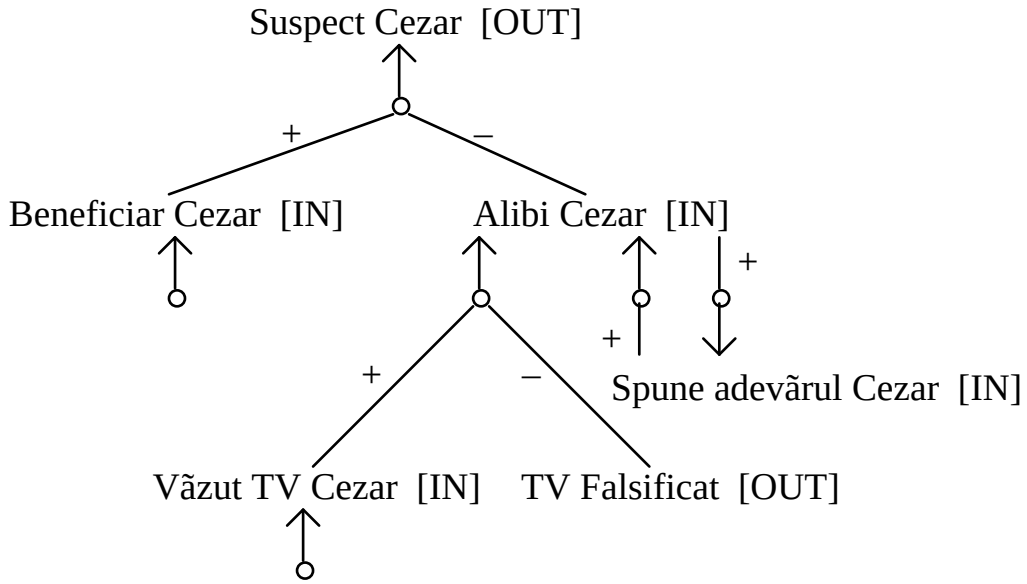


Figura 5.12 Noua justificare a asertiunii Suspect Cezar

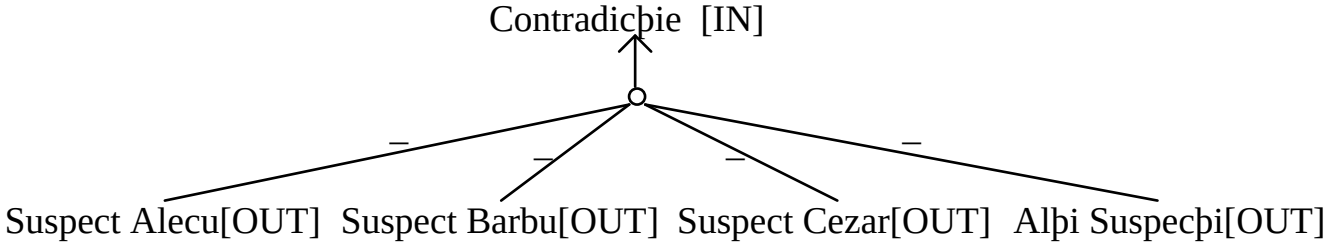


Figura 5.13 Contradictie valida în sistem

Asa cum s-a mai spus, algoritmul de backtraking condus de dependente este procedeul prin care sistemul de mentinere a consistentei datelor elimina contradictiile, identificînd multimea minima de presupuneri care au generat o contradictie. Invalidarea unei presupuneri din aceasta multime va duce la eliminarea contradictiei. Contradictia este eliminata prin alegerea unei presupuneri din aceasta multime si adaugarea unei justificari valide unui nod din lista OUT a acestei presupuneri. Recursiv, presupunerea poate fi eliminata invalidînd un nod N' din lista IN prin adaugarea de justificari unui nod N'' din lista OUT a nodului N'. Aceste justificari sînt formate din conjunctia dintre nodurile din lista IN a presupunerii si restul de presupuneri din multimea minimala de presupuneri detectata de algoritmul de backtracking condus de dependente. Adaugarea unei justificari apeleaza din nou procedura de mentinere a consistentei datelor deoarece pot apare alte contradictii (eventual alte noduri contradictie pot ajunge în starea IN).

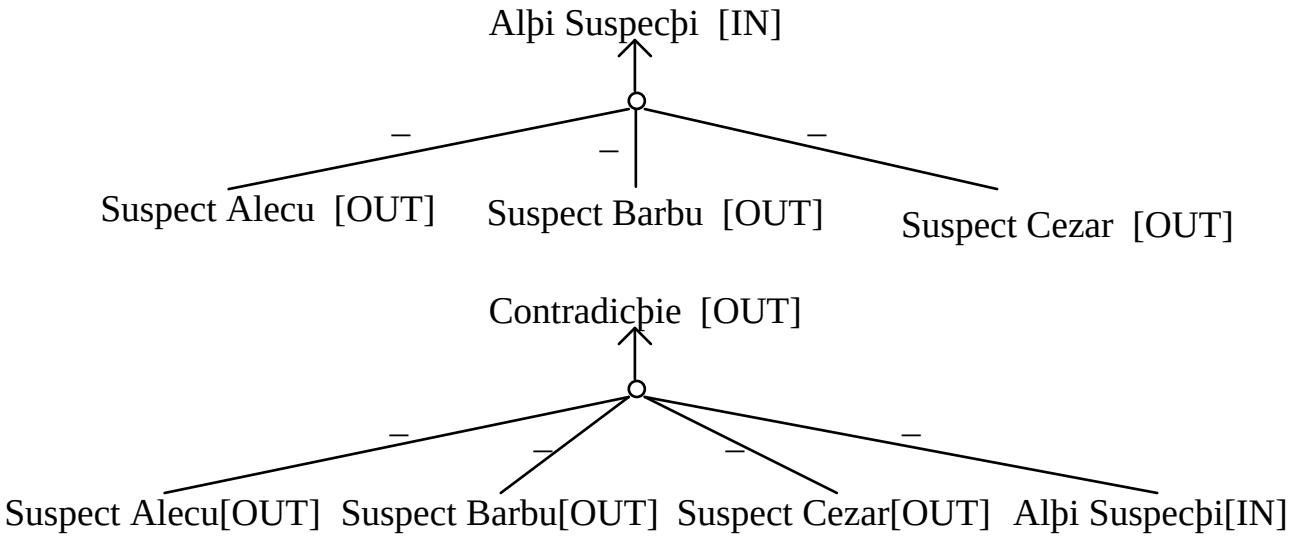


Figura 5.14 Justificare pentru a elimina contradictia

Pentru a exemplifica procesul de eliminare a contradictiei, se considera, întîi, ca multimea de presupuneri care au generat contradictia din exemplu este {Suspect Alecu, Suspect Barbu, Suspect Cezar, Alti Suspecti}, toate aceste noduri apartinînd listei OUT a nodului Contradictie. Se alege nodul Alti Suspecti, care nu are nici o justificare, si i se modifica starea din OUT în IN prin adaugarea unei justificari formata din restul nodurilor din multimea care a generat contradictia (lista IN a nodului Alti Suspecti este vida). Justificarea adaugata indica faptul ca sistemul crede ca exista alti suspecti numai pentru ca nu a reusit sa arate ca fie Alecu, fie Barbu, fie Cezar sînt suspecti. În acest caz starea nodului Contradictie se modifica din IN în OUT, deci contradictia este eliminata. Reprezentarea justificarii adaugate si efectul acestei justificari asupra contradictiei sînt prezentate în Figura 5.14.

O alta posibilitate de eliminare a contradictiei este, asa cum s-a spus, invalidarea unui nod din lista IN a nodului Contradictie. Pentru a exemplifica aceasta situatie, se considera o redefinire a dependentelor ce justifica nodul Contradictie, asa cum se arata în Figura 5.15(a). Multimea de

presupuneri care au generat contradictia se considera a fi {Nu Suspect Alecu, Nu Suspect Barbu, Nu Suspect Cezar, Nu Alti Suspecti}

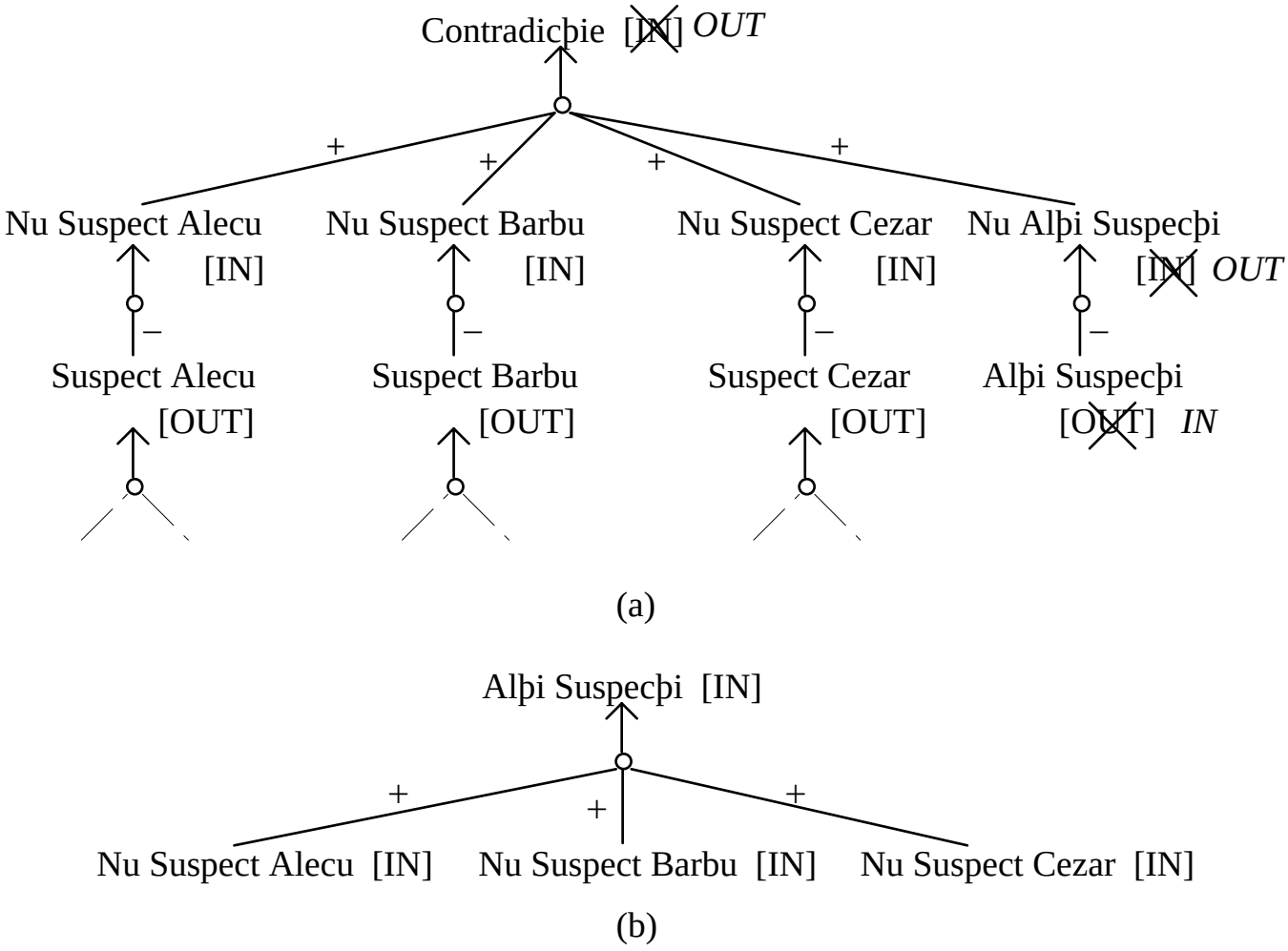


Figura 5.15 O noua formulare a contradictiei si eliminarea ei

Toate cele patru noduri din lista IN a contradictiei au justificari nemonotone, deci starea lor poate fi schimbata. Pentru a elimina contradictia se alege presupunerea Nu Alti Suspecti care se invalideaza. În acest scop trebuie adaugata o justificare unui nod din lista OUT a acestei presupuneri. Aceasta lista are, în acest caz, un singur nod, Alti Suspecti. La fel ca si în situatia anterioara, justificarea valida a nodului Alti Suspecti se face pe baza celorlalte noduri din multimea contradictorie de presupuneri, asa cum se vede în Figura 5.15(b).

Revenind la justificarea initiala a nodului Contradictie (Figura 5.11), sa presupunem ca se doreste eliminarea contradictiei prin validarea presupunerii Suspect Barbu. Validarea acestei premise nemonotone se poate face prin invalidarea presupunerii Alibi Barbu. Aceasta se poate face prin alegerea unui nod din lista OUT a nodului Alibi Barbu si transformarea acestui nod în nod cu starea IN. Lista OUT a nodului Alibi Barbu contine un singur nod, respectiv Minte Cumnat, care este o presupunere nejustificata. Pentru eliminarea contradictiei, se poate adauga acestui nod justificarea din Figura 5.16.

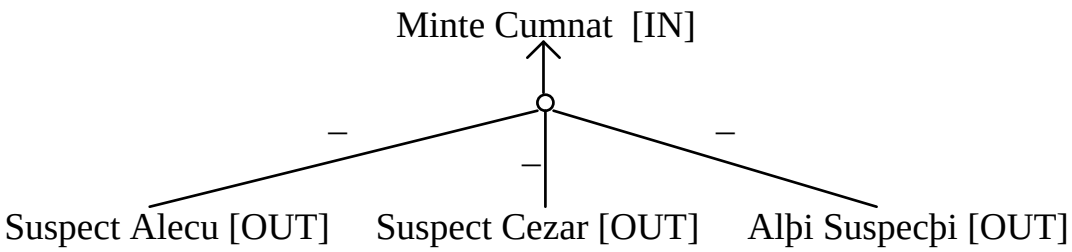


Figura 5.16 Alta justificare care elimina contradictia

Daca ulterior contradictia poate fi eliminata datorita altor presupuneri care au aparut, presupuneri care fac ca unul dintre nodurile Suspect Alecu, Suspect Cezar sau Alti Suspecti sa fie IN, presupunerea Minte Cumnat trebuie sa devina OUT. Sistemul a validat aceasta presupunere numai pentru a elimina contradictia, în lipsa altor informatii.

În cazul exemplului discutat, multimea minimala de presupuneri care au generat contradictia si care este determinata de algoritmul de backtracking condus de dependente este {Registru Falsificat, Minte Cumnat, TV Falsificat, Alti Suspecti}. Validarea oricarei dintre aceste presupuneri duce la invalidarea unei presupuneri ce a generat contradictia, deci contradictia este eliminata. Justificarea uneia dintre aceste presupuneri se face pe baza celorlalte presupuneri din multime, asa cum s-a aratat, si a nodurilor din lista IN (daca exista) a presupunerii.

Pentru cazul general, strategia de backtracking condus de dependente va genera un arbore ^{a}I/SAU de presupuneri care determina contradictia. Ramîne deschisa problema alegerii unei presupuneri de invalidat. Cum se poate face acest lucru? Sistemul de mentinere a consistentei datelor nu poate raspunde la aceasta întrebare. Primele variante de sisteme de mentinere a consistentei alegeau un nod la întîmplare. În variantele recente ale acestor sisteme, motorul de inferenta indica, în functie de startegia de control si de cunostintele specifice problemei, presupunerea care trebuie eliminata. În acest caz, sistemul de mentinere a consistentei datelor trebuie sa prezinte motorului de inferenta presupunerile nejustificate ce trebuie validate pentru a elimina contradictia.

În concluzie, operatiile de baza executate de sistemul de mentinere a consistentei datelor sînt:

- mentinerea consistentei datelor în baza de cunostinte si
- rezolvarea contradictiilor.

Sistemul de mentinere a consistentei datelor este numai o componenta ajutatoare a motorului de inferenta în rezolvarea problemei. Motorul de inferenta este cel care, în continuare, realizeaza prelucrari importante, cum ar fi:

- executarea inferentelor pentru a deduce concluzii;

- crearea de justificari ca urmare a aplicarii acestor inferente (s-a vazut ca si sistemul de mentinere a consistentei datelor poate crea justificari ca urmare a eliminarii unei contradictii);

- alegerea între diverse modalitati de rezolvare a contradictiilor.

5.5.3 Consideratii de implementare

Pentru implementarea unui sistem de mentinere a consistentei datelor se propune urmatoarea solutie. *Structurile de date ale retelei de dependente* sînt formate din doua entitati principale: nod si justificare. Acestea au urmatoarea forma:

(1) **Nod**. O structura de tip nod contine urmatoarele informatii (sloturi):

- *Valoare* este reprezentarea faptului asociat. Valoarea unui nod este unica si coincide cu reprezentarea interna din baza de cunostinte.

- *Eticheta* descrie starea nodului, i.e IN sau OUT.

- *Justificari* reprezinta lista justificarilor (referinte la obiecte de tip justificare) care explica acel nod. Se reaminteste ca un nod poate avea mai multe justificari.

- *Consecinte* reprezinta lista justificarilor în care nodul este antecedent. Aceasta structura este formata din doua liste:

  ConsecIN - lista justificarilor în care nodul apare ca antecedent în lista IN

  ConsecOUT - lista justificarilor în care nodul apare ca antecedent în lista OUT.

- *Contradictie* indica daca nodul este sau nu contradictie.

(2) **Justificare**. O structura de tip justificare contine urmatoarele informatii:

- *Tip* reprezinta descrierea inferentei, descriere data de motorul de inferenta. Tipul unei justificari poate fi premisa, Modus Ponens, mostenire, etc.

- *Consecinta* indica nodul justificat (o referinta la acel nod).

- *Antecedente* reprezinta lista nodurilor de care depinde inferenta. Aceasta structura este formata din doua liste de noduri: lista IN si lista OUT.

În plus trebuie sa existe fie o lista a tuturor nodurilor, fie un mecanism de indexare, pentru a putea cauta nodurile în reteaua sistemului de mentinere a consistentei datelor.

Operatiile de baza executate de sistemul de mentinere a consistentei datelor, pentru mentinerea consistentei cunostintelor, sînt urmatoarele:

(1) *Adauga premisa.* Aceasta operatie creeaza un nod cu starea IN si justificarea asociata cu lista IN si lista OUT vide.

(2) *Adauga presupunere*. Aceasta operatie creeaza un nod cu stare OUT si fara nici o justificare.

(3) *Adauga justificare*. Aceasta operatie adauga o justificare unui nod astfel: creeaza acea justificare si o asociaza nodului, actualizeaza corespunzator referintele nodurilor implicate, evalueaza justificarea si propaga consecintele mentinînd consistenta retelei de dependente.

Evident, operatiile (1) si (2) nu mai creeaza un nod daca el exista deja în reteaua de noduri. Aceste operatii sînt simplu de implementat. Operatia mai subtila este (3), adica adaugarea unei justificari la un nod. În continuare se prezinta algoritmul de efectuare a acestei operatii. În algoritm se folosesc structurile de date discutate anterior, selectia unui slot al unei structuri facîndu-se cu ajutorul operatorului **.** . De exemplu, N **.** Valoare reprezinta faptul asociat nodului N. Adaugarea unei justificari poate avea o influenta asupra starii nodului, deci a starii generale a bazei de cunostinte, numai în cazul în care acel nod are starea OUT (prin adaugarea justificarii el trece în starea IN). Daca nodul la care se adauga justificarea are deja starea IN, adaugarea unei justificari suplimentare nu îi modifica starea, deci nu are implicatii asupra starii generale a bazei de cunostinte.

Algoritm: Adauga justificare unui nod

1. Creeaza justificare J pentru nodul N
2. Actualizeaza referintele nodurilor implicate pentru conectarea lui J în retea
3. **daca** nodul justificat N are starea IN
   atunci STOP /* justificarea nu modifica starea nodului*/
4. **daca** nodul justificat N are starea OUT
   atunci
   4.1. Evalueaza justificare J
   4.2. **daca** J este valida
   atunci
   4.2.1. Modifica starea nodului N în IN /* nodul trece din OUT în IN */
   4.2.2. Propaga(N)

sfîrsit.

Observatie. În pasul 3 al algoritmului, daca nodul justificat N este deja în starea IN, i.e. daca J **.** Consecinþã**.** Etichetã = IN, adaugarea unei justificari suplimentare nu va modifica starea lui N, deci algoritmul se opreste.

Propaga(N)

1. **pentru** toate nodurile N_i din N . ConsecIN **executa**

   1.1. **daca** N_i are starea OUT

   atunci

   1.1.1. Evalueaza noua stare a lui N_i /* luînd în considerare noua stare IN a nodului N */

   1.1.2. **daca** noua stare a lui N_i este IN

   atunci Propaga(N_i)

2. **pentru** nodurile N_i din N . ConsecOUT **executa**

   2.1. **daca** N_i are starea IN

   atunci

   2.1.1. Construieste lista LN cu toate nodurile justificate recursiv de N

   2.1.2. **pentru** fiecare nod N_j din LN **executa**

   i. N_j . Etichetã ← OUT /* marcheaza temporar nodul N_j cu starea OUT */

   2.1.3. **pentru** fiecare nod N_j din LN **executa**

   i. Evalueaza starea nodului N_j, SN_j /* SN_j poate fi IN sau OUT */

   îi. N_j . Etichetã ← SN_j /* actualizeaza starea nodului N_j */

sfîrsit.

Observatii:

- În pasul 1, respectiv 2 al subprogramului Propaga(N), nodurile N_i din lista N.ConsecIN sînt nodurile pentru care N este antecedent cu "+" al justificarii, iar nodurile N_i din lista N . ConsecOUT sînt nodurile pentru care N este antecedent cu "–" al justificarii.

- Evaluarea starii nodului N_i din pasul 1.1.1 se face pe baza justificarilor din lista N_i . Justificări.

- La pasul 1.1 al subprogramului Propaga(N), daca nodul N_i are deja starea IN, faptul ca unul dintre nodurile care îl justifica într-o lista IN a devenit IN, nu modifica starea nodului.

- La pasul 2.1 al subprogramului Propaga(N), daca nodul N_i este deja în starea OUT, adaugarea unui nod IN suplimentar în lista OUT asociata lui N nu modifica starea nodului N_i.

- La pasul 2.1.2 se marcheaza temporar nodul N_j ca OUT. Marcarea si reevaluarea se face o singura data si nu se reia procesul pentru fiecare nod N_i.

- Evaluarea starii nodurilor N_j la pasul 2.1.3 se face pe baza justificarilor din Ni . Justificări.

- Pasii 2.1 si 2.1.2 pot fi executati împreuna: se colecteaza (eventual cu o procedura recursiva) nodurile care depind atît direct cît si indirect de nodul N în lista LN si, pe masura colectarii, se marcheaza temporar aceste noduri cu OUT. Aceasta marcare temporara a nodurilor cu starea OUT si reexaminare a starii lor este necesara pentru a evita prezenta nodurilor IN pe baza justificarilor circulare.

Exemplul urmator justifica ultima observatie si arata de ce trebuie calculata lista tuturor nodurilor dependente de nodul N si transformate temporar în noduri cu starea OUT, apoi reevaluate, în cazul în care nodul N devine OUT. Se considera o portiune a unei retele de dependente cu structura prezentata în Figura 5.17.

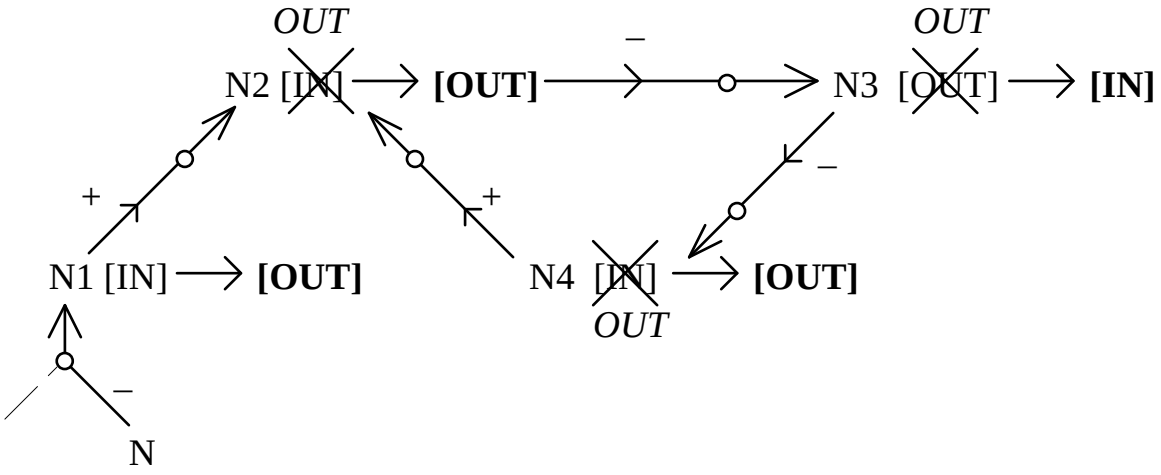


Figura 5.17 Actualizarea retelei de dependente la schimbarea starii unui nod în OUT

Sa presupunem ca nodul N1 trece din starea IN în starea OUT, datorita adaugarii unei justificari valide nodului N. Daca nu s-ar trece întîi în starea OUT toate nodurile care depind recursiv de nodul N1, atunci rezultatul ar fi urmatorul. La evaluarea starii nodului N2 rezulta pentru acesta starea IN, datorita nodului N4 care este IN. Cum nodul N2 era tot în starea IN, nimic nu se schimba în retea. Dar în acest caz nodul N2 ramîne în starea IN numai pe baza nodului N4, justificat de nodul N3, justificat de nodul N2. Apare deci o justificare circulara, deci inconsistenta.

Daca dupa ce nodul N1 trece din starea IN în starea OUT, se trec temporar în starea OUT toate nodurile care depind de el, apoi se reevalueaza fiecare nod, rezultatul final este cel indicat în Figura 5.17. Starile scrise cu caractere îngrosate în figura reprezinta starile finale ale nodurilor dupa ce se termina operatia de propagare a efectului adaugarii unei justificari nodului N.

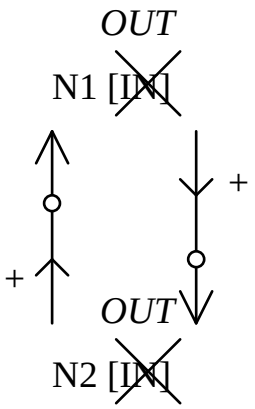
Cunoscatorii limbajului Lisp si-au dat seama pîna în acest moment ca ideea de marcare si reevaluare a nodurilor este similara cu una din tehnicile de colectare a spatiului disponibil din Lisp. Dintre cele doua mecanisme mai raspîndite de colectare a spatiului neutilizat: numararea

referintelor si urmarirea referintelor, prima nu reuseste detectarea spatiului disponibil circular. Urmarirea referintelor detecteaza acest caz deoarece demarcheaza toate celulele si, începînd de la radacini, marcheaza fiecare celula utila la care se poate ajunge, urmarind referintele. Dupa cum se poate vedea, ideea algoritmului de propagare a schimbarii starilor a nodurilor din sistemul de mentinere a consistentei datelor este asemanatoare.

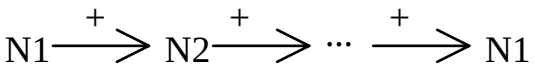
Algoritmul lasa pe baza implementarii rezolvarea unor probleme de circularitate mai simple, cum ar fi:

(1) În cazul construirii listei LN, trebuie sa se excluda listele circulare. În exemplul dat, nodurile justificate de N1 ce trebuie depuse în LN sînt N1, N2, N3, N4 si nu secventa infinita N1,N2,N3,N4,N2,N3,N4,...

(2) Trebuie sa se detecteze justificarile circulare construite numai din legaturi pozitive (legaturi formate pe lista IN), de exemplu

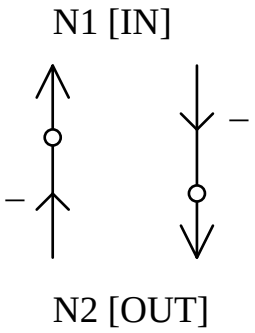


sau



Aceste justificari sînt inconsistente, cu exceptia cazului în care cel putin unul dintre noduri are o alta justificare valida, iar nodurile trebuie considerate cu starea OUT. În cazul în care unul dintre noduri are o alta justificare valida, nu poate apare pericolul ciclarii pe ramura corespunzatoare pasului 1 al algoritmului, deoarece sînt inspectate numai nodurile OUT.

În schimb, sînt perfect posibile structuri de dependente recursive cu legaturi formate pe lista OUT, cum ar fi



5.5.4 Sisteme de mentinere a consistentei datelor bazate pe presupuneri

O varianta extinsa si eficienta a sistemelor de mentinere a consistentei datelor sînt *sistemele de mentinere a consistentei datelor bazate pe presupuneri*, cu prescurtarea din limba engleza ATMS, propuse de deKleer [1986a,b,c]. În sistemele de mentinere a consistentei datelor se urmareste o singura cale de rationament (cale de cautare) si algoritmul de backtracking condus de dependente intervine de fiecare data cînd trebuie sa se schimbe presupunerile sistemului. Acest mecanism poate fi asimilat cu o cautare în adîncime.

În sistemele de mentinere a consistentei datelor bazate pe presupuneri, pe masura rezolvarii problemei, sînt dezvoltate si mentinute diverse cai de rationament. Ideea de baza a acestor sisteme este utilizarea unor *contexte* care corespund unor multimi de presupuneri consistente. Algoritmul de backtraking este eliminat prin mentinerea acestor contexte multiple. Acest mecanism poate fi asimilat cu o cautare pe nivel. Pe masura dezvoltarii diverselor cai de rationament, multimea de contexte este redusa prin eliminarea contextelor contradictorii, adica a contextelor care contin contradictii.

Contextele consistente sînt utilizate pentru a *eticheta presupunerile*, indicînd astfel contextele în care presupunerile au o justificare valida. Sînt eliminate asertiunile care nu au o justificare valida în nici unul dintre contextele considerate. Pe masura ce multimea de contexte se reduce, se reduce si numarul se asertiuni considerate de sistem.

Sistemele de mentinere a consistentei datelor bazate pe presupuneri, la fel ca si sistemele de tip TMS, trebuie sa functioneze împreuna cu motorul de inferenta. Rolul motorului de inferenta în acest caz este:

- transmiterea de asertiuni pentru care sistemul de mentinere a consistentei datelor bazat pe presupuneri trebuie sa creeze noduri;
- transmiterea justificarilor nodurilor, fiecare justificare reprezentînd modalitatea de deducere a asertiunii asociate nodului;
- semnalarea contextelor inconsistente.

Se observa ca rolul motorului de inferenta cuplat cu un sistem de mentinere a consistentei datelor bazat pe presupuneri este similar cu acela al unui motor de inferenta cuplat cu un sistem simplu de mentinere a consistentei, cu exceptia faptului ca motorul de inferenta nu trebuie sa ia decizii asupra caii de rationament urmata.

Rolul sistemului de mentinere a consistentei datelor bazat pe presupuneri este:

- construirea nodurilor si asocierea justificarilor nodurilor;
- propagarea inconsistentelor care are ca efect eliminarea contextelor inconsistente (contradictorii);
- etichetarea fiecarui nod cu multimea contextelor în care nodul are o justificare valida.

De exemplu, fiind data justificarea $A_1 \wedge A_2 \wedge ... \wedge A_n \rightarrow C$, se va eticheta nodul C cu reuniunea contextelor nodurilor $A_1, A_2, ..., A_n$. Contextele se elimina ca rezultat al semnalarii contextelor inconsistente de catre motorul de inferenta si ca rezultat al propagarii inconsistentelor de catre sistemul de mentinere a consistentei datelor bazat pe presupuneri. Daca toate contextele unui nod sînt eliminate ca fiind inconsistente, nodul poate fi de asemenea eliminat.

Se observa ca aceasta comportare a sistemului de mentinere a consistentei bazat pe presupuneri favorizeaza utilizarea lui în situatiile în care se cere gasirea tuturor solutiilor unei probleme, în timp ce sistemul simplu de mentinere a consistentei este mai potrivit pentru cazul în care se cere gasirea unei singure solutii.

O problema care se poate pune în cazul sistemului de mentinere a consistentei datelor bazat pe presupuneri este urmatoarea: pentru o multime de N presupuneri, numarul posibil de contexte ce pot fi considerate este 2^N, deci complex computational. Din fericire, în cele mai multe cazuri, multe din aceste contexte pot fi eliminate fara sa mai fie luate în considerare. În plus, sistemul foloseste o modalitate simpla de înregistrare a contextelor în etichete, bazata pe ideea de *latice de contexte*.

Pentru a ilustra aceasta idee se considera urmatorul exemplu. Pentru o multime de presupuneri $\{A_1, A_2, A_3, A_4\}$ se formeaza laticea de contexte specificata în Figura 5.18. Parcurgerea laticei în sus este echivalenta cu relatia de incluziune a multimilor.

Reprezentarea sub forma de latice a contextelor faciliteaza doua mecanisme. În primul rînd aceasta reprezentare permite eliminarea contextelor inconsistente. Daca, de exemplu, contextul $\{A_2, A_3\}$ este inconsistent, atît el cît si toate contextele ce îl include sînt eliminate deoarece sînt inconsistente. În acest fel, contradictiile, i.e. contextele inconsistente, pot fi propagate si portiuni semnificative ale spatiului de 2^N contexte pot fi eliminate. În al doilea rînd, reprezentarea sub forma de latice a contextelor asigura minimalitatea etichetelor. Fiecare nod va fi etichetat cu cea mai mare limita inferioara a contextelor în care acel nod trebuie crezut.

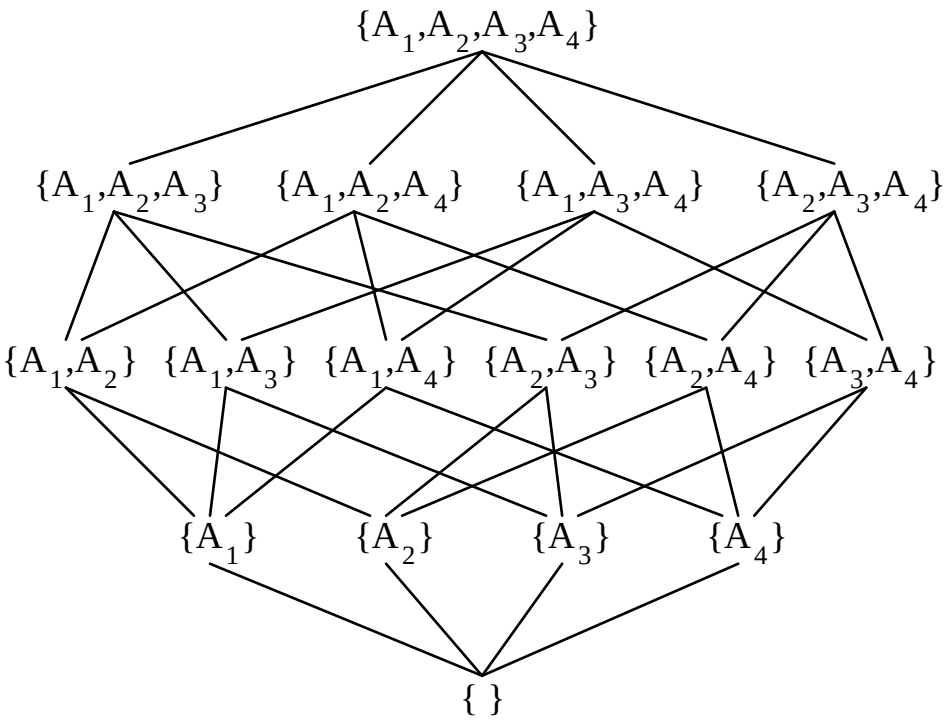


Figura 5.18 Latice de contexte

Fie un nod etichetat cu toate contextele în care are o justificare valida. Sa presupunem ca aceasta justificare depinde de presupunerea A_1. Atunci contextul $\{A_1\}$ si toate contextele ce includ A_1 vor fi contexte în care nodul ar putea avea o justificare valida. Dar acest lucru poate fi indicat numai prin eticheta $\{A_1\}$ si nu mai este necesar sa se enumere toate multimile care includ A_1. Evident, laticea de contexte nu este construita explicit, ci este utilizata doar ca o structura implicita în functionarea sistemului.

Pentru a exemplifica modul de functionare a unui sistem de mentinere a consistentei bazat pe presupuneri, se considera din nou problema crimei ABC. Cititorul stie deja ca scopul problemei este gasirea unui suspect. Se considera urmatoarele presupuneri:

A1 Registru hotel falsificat.
A2 Registru hotel nefalsificat.
A3 Cumnat Barbu minte.
A4 Cumnat Barbu nu minte.
A5 Cezar minte.
A6 Cezar nu minte.
A7 Alecu, Barbu si Cezar sînt singurii suspecti posibili.
A8 Alecu, Barbu si Cezar nu sînt singurii suspecti posibili.

Motorul de inferenta genereaza nodurile si justificarile asociate nodurilor prezentate în primele doua coloane ale Figurii 5.19. Sistemul de mentinere a consistentei datelor va înregistra pentru un nod presupunere o justificare bazata chiar pe acea presupunere. Deci se înregistreaza justificarea corespunzatoare deciziei de a face o anumita presupunere.

Justificarile asociate nodurilor care corespund rezultatului aplicarii unei reguli de inferenta sînt reprezentate chiar de aceste reguli. Sistemul poate asocia etichete nodurilor, asa cum se prezinta în ultimele doua coloane din Figura 5.19.

Observatii:

- În cazul în care un nod are mai multe justificari se înregistreaza toate aceste justificari. Acesta este cazul nodurilor N12, N13 si N14 din Figura 5.19.
- În modelul sistemelor simple de mentinere a consistentei datelor, un nod valid era un nod cu starea IN. În modelul sistemelor de mentinere a consistentei bazate pe presupuneri, un nod valid este etichtat cu cel putin un context, ceea ce revine la a spune ca exista cel putin un context în care nodul are o justificare valida.
- Procesul de calcul al etichetelor nu este banal. Acest proces este descris în continuare.

Se presupune ca motorul de inferenta indica sistemului de mentinere a consistentei datelor bazat pe presupuneri crearea nodurilor N1÷N4 din Figura 5.19, cu justificarile asociate din a doua coloana. În acest fel se stabilesc diverse dependente între asertiunile problemei, dar nu se ia nici o decizie asupra asertiunilor care trebuie crezute sau nu. Motorul de inferenta indica apoi contradictiile, prin semnalarea unui context inconsistent. Fie acest context:

{Alecu, Barbu si Cezar sînt singurii suspecti; Alecu, Barbu si Cezar nu sînt singurii suspecti}

deci contextul $\{A_7, A_8\}$.

Sistemul de mentinere a consistentei datelor bazat pe presupuneri va asocia etichetele aratate în Figura 5.19. Prima coloana de etichete arata etichetele care ar fi generate daca s-ar considera o singura justificare pentru fiecare nod. A doua coloana de etichete arata etichetele reale generate de sistem, etichete care pot contine mai multe contexte, considerînd toate justificarile posibile pentru un nod.

Cele doua coloane de etichete sînt identice pentru cazul în care exista justificari unice, dar diferite pentru cazuri mai complicate, cum ar fi de exemplu nodurile N12, N13 si N14.

Noduri		**Justificari**	**Etichete**	
N1	Registru nu fals	$\{A_2\}$	$\{A_2\}$	$\{A_2\}$*
N2	Alecu la hotel	$N1 \rightarrow N2$	$\{A_2\}$	$\{A_2\}$*
N3	Cumnat nu minte	$\{A_4\}$	$\{A_4\}$	$\{A_4\}$
N4	Barbu la cumnat	$N3 \rightarrow N4$	$\{A_4\}$	$\{A_4\}$
N5	Cezar nu minte	$\{A_6\}$	$\{A_6\}$	$\{A_6\}$
N6	Cezar la schi	$N5 \rightarrow N6$	$\{A_6\}$	$\{A_6\}$
N7	Alecu, Barbu si Cezar sînt singurii suspecti	$\{A_7\}$	$\{A_7\}$	$\{A_7\}$
N8	Suspect Alecu	$N7 \wedge N13 \wedge N14 \rightarrow N8$	$\{A_7, A_4, A_6\}$	$\{A_7, A_4, A_6\}$
N9	Suspect Barbu	$N7 \wedge N12 \wedge N14 \rightarrow N9$	$\{A_7, A_2, A_6\}$	$\{A_7, A_2, A_6\}$*
N10	Suspect Cezar	$N7 \wedge N13 \wedge N13 \rightarrow N10$	$\{A_7, A_2, A_4\}$	$\{A_7, A_2, A_4\}$*
N11	Alecu, Barbu si Cezar nu sînt singurii suspecti	$\{A_8\}$	$\{A_8\}$	$\{A_8\}$
N12	Alecu nu este suspect	$N2 \rightarrow N12$	$\{A_2\}$	$\{A_2\}$*
		$N11 \rightarrow N12$	$\{A_8\}$	$\{A_8\}$
		$N9 \rightarrow N12$	$\{A_7, A_2, A_6\}$	
		$N10 \rightarrow N12$	$\{A_7, A_2, A_4\}$	
N13	Barbu nu este suspect	$N4 \rightarrow N13$	$\{A_4\}$	$\{A_4\}$
		$N11 \rightarrow N13$	$\{A_8\}$	$\{A_8\}$
		$N8 \rightarrow N13$	$\{A_7, A_4, A_6\}$	
		$N10 \rightarrow N13$	$\{A_7, A_2, A_4\}$	
N14	Cezar nu este suspect	$N6 \rightarrow N14$	$\{A_6\}$	$\{A_6\}$
		$N11 \rightarrow N14$	$\{A_8\}$	$\{A_8\}$
		$N8 \rightarrow N14$	$\{A_7, A_4, A_6\}$	
		$N9 \rightarrow N14$	$\{A_7, A_2, A_6\}$	

Figura 5.19 Noduri cu justificari si etichete în reteaua de dependente a unui sistem ATMS

Urmatoarele doua exemple de calcul al etichetelor pun în evidenta procesul de etichetare. Fie nodul N12. Pentru acest nod se genereaza 4 contexte posibile, un context pentru fiecare justificare. Dar se doreste asocierea unei etichete care reprezinta contextele minimale în care nodul este valid. Eticheta $\{A_2\}$ este cea mai mare limita inferioara pentru $\{A_2\}$, $\{A_7, A_4, A_6\}$ si $\{A_7, A_2, A_4\}$, iar $\{A_8\}$ pentru $\{A_8\}$. Deci eticheta nodului N12 va contine doua contexte $\{A_2\}$ si $\{A_8\}$. Fie nodul N8. Acest nod se eticheteaza cu reuniunea contextelor nodurilor N7, N13 si N14. Deoarece nodurile N13 si N14 sînt etichetate cu mai multe contexte, trebuie considerate toate combinatiile posibile dintre aceste contexte. Din fericire, se stie ca $\{A_7, A_8\}$ este un context inconsistent, deci contextul $\{A_7, A_8\}$ plus toate contextele care îl contin vor fi eliminate. Ramîne deci pentru nodul N8 o eticheta cu un singur context $\{A_7, A_4, A_6\}$.

Se presupune acum ca motorul de inferenta semnaleaza faptul ca $\{A_2\}$ este un context inconsistent, ceea ce înseamna ca presupunerea "Registru hotel nefalsificat" este în contradictie cu ceea ce se stie. În acest caz, se elimina toate contextele incosistente care includ $\{A_2\}$ si contextul $\{A_2\}$, deci toate contextele marcate cu * în Figura 5.19.

Singurul nod care corespunde unui suspect, deci unui context cu eticheta nevida, este nodul N8, "Suspect Alecu", deci se poate presupune ca Alecu este suspect. Dar nodul N12, "Alecu nu este suspect", are si el o eticheta nevida, datorita existentei presupunerii A8: "Alecu, Barbu si Cezar nu sînt singurii suspecti posibili". În momentul în care aceasta presupunere s-ar invalida, Alecu devine clar suspect. Atît timp cît presupunerea este valida Alecu poate fi suspect, pe o cale de rationament, dar poate sa nu fie suspect, pe o alta cale, deoarece s-ar putea sa fie suspect altcineva.

Un sistem de mentinere a consistentei bazat pe presupuneri poate raspunde în mod natural la întrebari de tipul "În ce context este asertiunea A adevarata (valida)?". Dar, deoarece contextele inconsistente sînt eliminate, pentru o problema a carei descriere este suprarestrictionata, toate nodurile vor avea în final etichete vide. Nu va exista nici o urma a încercarilor de rezolvare pe baza carora s-ar putea relaxa una sau mai multe restrictii. Aceasta este una din limitarile sistemelor prezentate.

5.6 Multimi de lumi posibile

Modelul lumilor posibile a aparut ca o alta varianta de implementare a rationamentului nemonoton, în particular a rationamentului implicit. Acest model ofera solutii pentru rezolvarea problemelor clasice ale rationamentului nemonoton: problema cadrului, problema calificarii si problema ramificarii. Desi este un model mai slab decît cel al sistemelor de mentinere a consistentei datelor din punct de vedere al eficientei, lumile posibile pot rezolva (partial) si problema mentinerii consistentei datelor din baza de cunostinte a unui sistem nemonoton. Definite formal de Ginsberg [1986], lumile posibile au fost aplicate la formalizarea si implementarea rationamentului despre actiuni, [Ginsberg,1988], asa cum se va prezenta în Capitolul 6. De asemenea, modelul a fost folosit în implementarea unui sistem bazat pe limbajul Prolog cu rationament nemonoton, si anume sistemul MRS [Russell,1985], cît si în sistemul KEE [Filman,1988] în care lumile posibile au fost combinate cu un sistem de mentinere a consistentei datelor bazat pe presupuneri (ATMS).

5.6.1 Modelul lumilor posibile

Multimile de lumi posibile implementeaza o forma de rationament implicit care permite considerarea diverselor extensii posibile ale bazei de cunostinte pe baza inferentelor nemonotone. Acest model reprezinta o solutie în spiritul logicii implicite a lui Reiter, în care nu exista un criteriu de a prefera o extensie sau alta a bazei de cunostinte.

În modelul lumilor posibile, multimea convingerilor sistemului la un moment dat formeaza lumea curenta. Daca se poate face fie o presupunere P, fie o presupunere Q, cu efecte diferite, se vor crea doua lumi posibile noi, pe baza lumii curente, o lume în care exista toate convingerile din lumea curenta plus presupunerea P, si o alta lume în care exista toate convingerile din lumea curenta si în plus presupunerea Q.

Se considera urmatorul exemplu dat de R.C. Moore [Norvig,1992]. Fie trei tari din Europa de Est D, E si C, situate ca în Figura 5.20. Se cunoaste ca tara D are un regim democrat, C un regim comunist, iar E este în ajunul alegerilor, deci nu se poate sti daca va alege un regim democrat sau unul comunist. Întrebarea care se pune este: dupa alegeri, va exista o tara cu regim democrat lînga o tara cu regim comunist? Raspunsul este evident da, dar acest raspuns implica un rationament bazat pe presupuneri.

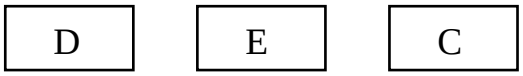


Figura 5.20 Pozitia relativa a tarilor D, E si C

Abordînd modelul lumilor posibile, lumea curenta va fi D-democrata, C-comunista si se vor construi doua lumi posibile din lumea curenta, una în care E va fi democrata si o alta lume în care E va fi comunista. În fiecare din aceste lumi este usor sa se demonstreze ca exista o tara cu regim democrat lînga o tara cu regim comunist. De fapt cele doua lumi formeaza o partitie, deci raspunsul este valabil si în lumea initiala, deoarece s-au creat toate multimile de lumi posibile derivate din lumea curenta.

Modelul discutat si exemplul anterior reprezinta o simplificare a abordarii lumilor posibile. În cazul general, în lumea noua L_1, obtinuta din lumea curenta L, pot exista fapte care trebuie retractate pe baza inferentelor executate în L_1, deci lumea curenta nu este întotdeauna o submultime a lumilor noi create din ea. În acest fel, modelul lumilor posibile implementeaza efectiv o forma de rationament nemonoton.

Complicatii suplimentare apar în cazul în care diversele lumi posibile contin fapte diferite si trebuie preferata o lume sau alta, pe baza unui anumit criteriu. Aceasta problema este de fapt, aceeasi cu cea pusa de modelul formal al logicii implicite a lui Reiter. Spre deosebire de acest model, un sistem care utilizeaza multimi de lumi posibile în solutionarea unei probleme trebuie sa stabileasca o strategie computationala de selectie a lumii celei mai bune. Posibile solutii propuse pentru aceasta problema sînt: utilizarea cunostintelor specifice domeniului si preferarea lumilor care contin fapte considerate ca importante, înregistrarea dependentelor si eliminarea lumilor care contin presupuneri nejustificate sau lucrul cu toate lumile, facilitat de un sistem de mentinere a consistentei datelor bazat pe presupuneri.

Definitia formala a modelului general al lumilor posibile si aplicarea acestui model la efectuarea rationamentului despre actiuni vor fi prezentate în Capitolul 6. În sectiunea de fata si în

cea urmatoare, discutia se orienteaza spre aspectele practice de implementare a modelului, pornind de la ipoteza simplificatoare prezentata la început, ipoteza conform careia lumile noi sînt superseturi ale lumii curente.

Implementarea rationamentului nemonoton utilizînd multimi de lumi posibile poate fi combinata cu diverse modele de reprezentare a cunostintelor: logica, reguli de productie, unitati (retele semantice). Utilizarea modelului lumilor posibile necesita, evident, combinarea modelului cu o strategie de cautare. Strategia de cautare trebuie sa poata executa:

- inferente în lumea curenta;
- crearea unor lumi posibile urmatoare, în functie de diverse presupuneri facute;
- decizii asupra selectiei lumii posibile, care va fi considerata ca lume actuala si comutarea dintr-o lume în alta.

Una dintre cele mai noi solutii de implementare a rationamentului nemonoton este sistemul bazat pe cunostinte KEE [Filman,1988;Filman,s.a.,1992]. Acest sistem este de fapt un mediu de dezvoltare al programelor de inteligenta artificiala, fiind unul dintre cele mai de succes produse comerciale de inteligenta artificiala. Sistemul KEE foloseste unitatile ca forma fundamentala de reprezentare a cunostintelor, la care se adauga reguli de productie care refera aceste unitati. Sistemul implementeaza o forma de rationament nemonoton utilizînd o solutie de combinare a sistemelor de mentinere a consistentei datelor bazate pe presupuneri cu modelul lumilor posibile. Regulile de productie din KEE sînt de doua tipuri: reguli de deductie, care exprima inferente general valabile în sistem, si reguli de actiune, care creeaza lumi noi si modifica presupunerile într-o lume. Utilizarea retelei de dependente bazata pe presupuneri (prezentata în Sectiunea 5.5.4) permite sistemului KEE implementarea eficienta a modelului lumilor posibile, în particular determinarea faptelor adevarate într-o lume si identificarea lumilor care contin contradictii. În acelasi timp, se ofera posibilitatea de a lucra cu mai multe lumi posibile în paralel.

5.6.2 Solutii de implementare

Se prezinta în continuare o implementare în limbajul Lisp a modelului (simplificat) al lumilor posibile discutat anterior, utilizînd ca suport inferential si de reprezentare a cunostintelor regulile de productie nemonotone discutate în Sectiunea 5.4.2. În acest fel cititorul are posibilitatea sa cunoasca atît o solutie de implementare a extinderii nemonotone a regulilor, cît si a functionarii modelului lumilor posibile.

Intr-o astfel de abordare, implementarea modelului se bazeaza de obicei pe indexarea faptelor dupa lumile în care acestea sînt adevarate. Exista întotdeauna o lume curenta activa fata de care se considera adevarul unui fapt sau inferentele curente. În momentul crearii unei lumi noi, toate faptele valide în lumea curenta se pastreaza si în lumea noua, existînd numai posibilitatea adaugarii de noi fapte în momentul crearii noii lumi. Cu toate acestea, datorita posibilitatii de

revenire la o lume anterioara sau de comutare într-o lume alternativa, rationamentul este nemonoton. Datorita caracteristicii mentionate anterior, reprezentarea eficienta a lumilor posibile se poate face cu ajutorul unei ierarhii de lumi posibile.

Fie o lume L_2, derivata dintr-o lume L_1, aceasta fiind derivata la rîndul ei din lumea L. Pentru a putea determina ce fapte sînt adevarate în lumea L_2 se stie ca trebuie sa se gaseasca faptele adevarate în lumea L_2, apoi pe cele adevarate în lumea L_1, apoi pe cele adevarate în lumea L. În acest fel indexarea faptelor se poate face dupa o singura lume posibila: lumea în care ele au aparut pentru prima oara.

Un mecanism de gestiune a lumilor posibile are urmatoarele actiuni de baza:

- obtine lumea curenta;
- creeaza o noua lume;
- adauga presupuneri la lumea noua;
- stabileste o lume ca fiind lumea curenta.

Solutiile adoptate pentru definirea unei ierarhii de lumi posibile si pentru reprezentarea faptelor si regulilor se bazeaza pe functiile Lisp de lucru cu unitati (clase), prezentate în Sectiunea 5.4.1 si mecanismul de unificare prezentat în Sectiunea 3.4.3. Se vor folosi numai fatetele VALOARE ale sloturilor. Astfel, o lume este reprezentata printr-o unitate cu urmatoarele sloturi: *PARINTE*, *COPII*, *FAPTE* si *CURENTA*. Informatia memorata în sloturile unei lumi L reprezinta: lumea din care a provenit lumea L, lumile care provin din L, faptele adevarate în lumea L si atributul lumii L (lumea L este lume curenta sau nu). Exceptînd slotul PARINTE, celelalte sloturi sînt prezente doar cînd informatia memorata este diferita de nil. Variabilele speciale **LUME-CURENTA** si **REGULI** memoreaza lumea curenta si respectiv regulile problemei. O regula este o unitate avînd sloturile: *DACA*, *DACANU* si *ATUNCI*. Atît preconditiile memorate în sloturile DACA si DACANU, cît si concluziile memorate în slotul ATUNCI sînt de forma (relaţie $termen_1$ $termen_2$ ... $termen_m$), unde $termen_i$ poate fi o constanta sau o variabila, structura acestora respectînd cele aratate în Sectiunea 3.4.3.

Functiile si predicatele definite realizeaza urmatoarele actiuni si au urmatoarea semnificatie. Functia *citeste-raspuns* afiseaza un mesaj si citeste raspunsul utilizatorului. Functia este apelata pentru a citi fapte de catre functia *citeste-fapt*, functie care, la rîndul ei, este folosita de functiile *citeste-fapte* si *citeste-regula*, care adauga fapte în lumea curenta si, respectiv, citeste regulile problemei. Semipredicatul *lume-p* verifica daca obiectul primit ca parametru este o lume (o unitate). Functia *creeaza-lume* creeaza o lume noua, care va avea ca parinte lumea curenta, daca aceasta exista, actualizînd ierarhia de lumi. Functia *schimba-lume-curenta* stabileste o lume ca fiind lumea curenta, realizînd actualizarea corespunzatoare a atributelor. Functia *obtine-stramosi* obtine lista tuturor lumilor ce preced în ierarhie lumea primita ca parametru si este folosita de

catre functia *fapte-valide*, care colecteaza toate faptele adevarate într-o anumita lume, inspectînd toti stramosii acesteia. Functia *verifica-regula* verifica preconditiile unei reguli în interiorul unei lumi si întoarce, în caz de succes, lista concluziilor în care variabilele sînt substituite conform unificarilor facute la verificarea preconditiilor. Verificarea fiecarei preconditii în parte este realizata de functia *verifica-preconditie*, care întoarce, în caz de succes, doua valori: t si lista de instantieri generate. Datorita modului în care se efectueaza verificarea preconditiilor, ordinea acestora în cadrul regulii este importanta doar daca preconditiile contin variabile. Acest lucru se poate observa daca în exemplul prezentat se modifica ordinea preconditiilor regulii r.

```
(defvar *LUME-CURENTA* nil)

(defun citeste-raspuns (mesaj) (format t mesaj) (read))

(defun citeste-fapt ( )
  (let ((fapt (citeste-raspuns "~&:- ")))
    (when (listp fapt) fapt)))

(defun citeste-fapte ( )
  (do ((fapt (citeste-fapt) (citeste-fapt))
       ((null fapt))
    (u-adauga *LUME-CURENTA* 'FAPTE 'VALOARE fapt))))

(defun citeste-regula ()
  (let ((regula (gensym "REGULA")))
    (format t "~&DACA:")
    (do ((fapt (citeste-fapt) (citeste-fapt)))
        ((null fapt))
      (u-adauga regula 'DACA 'VALOARE fapt))
    (format t "~&DACANU:")
    (do ((fapt (citeste-fapt) (citeste-fapt)))
        ((null fapt))
      (u-adauga regula 'DACANU 'VALOARE fapt))
    (format t "~&ATUNCI:")
    (do ((fapt (citeste-fapt) (citeste-fapt)))
        ((null fapt))
      (u-adauga regula 'ATUNCI 'VALOARE fapt))
    (push regula *REGULI*)))

(defun lume-p (obiect) (get obiect 'UNITATE))

(defun creeaza-lume ( )
```

```
  (let ((lume (gensym "LUME")))
    (u-adauga lume 'PARINTE 'VALOARE *LUME-CURENTA*)
    (if (lume-p *LUME-CURENTA*)
      (u-adauga *LUME-CURENTA* 'COPII 'VALOARE lume)
      (setf *LUME-CURENTA* lume))))

(defun schimba-lume-curenta (lume)
  (when (lume-p lume)
    (u-sterge *LUME-CURENTA* 'CURENTA 'VALOARE t)
    (setf *LUME-CURENTA* lume)
    (u-adauga lume 'CURENTA 'VALOARE t)))

(defun adauga-fapt-în-lume (lume fapt)
  (when (lume-p lume) (u-adauga lume 'FAPTE 'VALOARE fapt)))

(defun obtine-stramosi (lume)
  (let ((parinte (first (u-obtine lume 'PARINTE 'VALOARE))))
    (when parinte (cons parinte (obtine-stramosi parinte)))))

(defun fapte-valide (lume)
  (do* ((stramosi (obtine-stramosi lume) (rest stramosi))
        (stramos (first stramosi) (first stramosi))
        (fapte (u-obtine stramos 'FAPTE 'VALOARE)
               (append (u-obtine stramos 'FAPTE 'VALOARE) fapte)))
       ((null stramosi) (delete nil (append fapte (u-obtine lume 'FAPTE 'VALOARE))))))

(defun verifica-regula (regula lume)
  (let ((fapte (fapte-valide lume))
        succes instantieri)
    (multiple-value-setq
      (succes instantieri)
      (do* ((preconditii (u-obtine regula 'DACA 'VALOARE) (rest preconditii))
            (preconditie (first preconditii) (first preconditii))
            (flag t)
            noi-instantieri)
           ((or (null flag) (null preconditii))
            (values flag noi-instantieri))
        (multiple-value-setq
          (flag noi-instantieri)
          (verifica-preconditie preconditie fapte noi-instantieri))))
```

```
   (when succes                                   ;; sint toate preconditiiile adevarate?
     (do* ((nonpreconditii
            (mapcar #'(lambda (nonpreconditie)
                        (substituie-variabile nonpreconditie instantieri))
                    (u-obtine regula 'DACANU 'VALOARE))
            (rest nonpreconditii))
           (nonpreconditie (first nonpreconditii) (first nonpreconditii))
           flag)
          ((or flag (null nonpreconditii))
           (unless flag                           ;; sint toate nonpreconditiile false?
             (mapcar #'(lambda (concluzie)
                         (substituie-variabile concluzie instantieri))
                     (u-obtine regula 'ATUNCI 'VALOARE))))
       (setf flag (verifica-preconditie nonpreconditie fapte instantieri))))))

(defun verifica-preconditie (preconditie fapte instantieri)
  (do* ((lista-fapte fapte (rest lista-fapte))
        (fapt (first lista-fapte) (first lista-fapte))
        succes noi-instantieri)
       ((or succes (null lista-fapte))
        (when succes (values t noi-instantieri)))
    (multiple-value-setq
      (succes noi-instantieri)
      (unifica preconditie fapt instantieri))))
#|
(setf l1 (creeaza-lume))
(adauga-fapt-în-lume l1 '(femeie tatiana))
(adauga-fapt-în-lume l1 '(barbat dan))
(adauga-fapt-în-lume l1 '(iubeste dan tatiana))
(setf l2 (creeaza-lume))
(adauga-fapt-în-lume l2 '(barbat radu))
(adauga-fapt-în-lume l2 '(iubeste tatiana radu))
(schimba-lume-curenta l2)
(setf l3 (creeaza-lume))
(adauga-fapt-în-lume l3 '(iubeste radu tatiana))
(setf r (gensym "REGULA"))
(u-adauga r 'DACA 'VALOARE '(iubeste ?z ?x))
(u-adauga r 'DACA 'VALOARE '(iubeste ?y ?x))
(u-adauga r 'DACA 'VALOARE '(barbat ?y))
(u-adauga r 'DACA 'VALOARE '(iubeste ?x ?y))
```

```
(u-adauga r 'DACA 'VALOARE '(femeie ?x))
(u-adauga r 'DACANU 'VALOARE '( ))
(u-adauga r 'ATUNCI 'VALOARE '(gelos ?z ?y))
(verifica-regula r l3)
|#
```

5.7 Exercitii si probleme

1. Sa se formuleze problema crimei ABC utilizînd întîi logica nemonotona a lui McDermott si Doyle, apoi logica implicita a lui Reiter.

2. Se considera urmatoarea problema clasica de rationament nemonoton, specificata de faptele:

   - Cele mai multe fiinte nu zboara.
   - Cele mai multe pasari zboara, cu exceptia cazului în care sînt pui, sau au o aripa rupta.
   - Pinguinii si strutii nu zboara.
   - Strutii magici zboara.
   - Tuti este pasare.
   - Cirpi este fie pinguin fie strut.
   - Pepe este un strut magic.

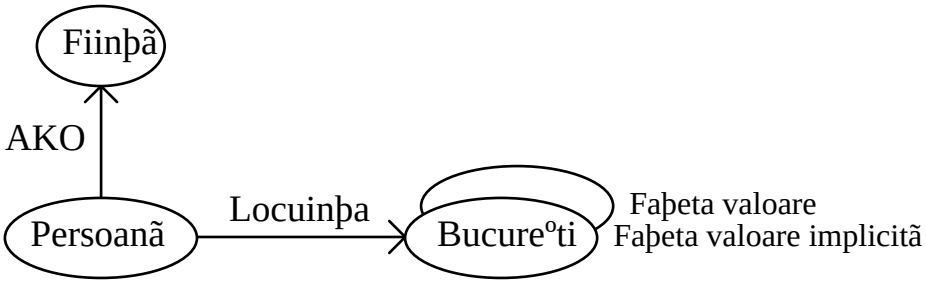
   Sa se exprime aceste fapte în unul dintre sistemele logice nemonotone discutate si sa se arate ce raspunde sistemul la întrebarile:

   - Zboara Tuti?
   - Zboara Cirpi?
   - Zboara Pepe?
   - Zboara Paul?

3. Sa se formuleze abductia în cele doua sisteme de logici implicite prezentate.

4. Sa se exprime mostenirea proprietatilor în logica nemonotona a lui McDermott si Doyle, utilizînd eventual exemplul ierarhiei de obiecte din Sectiunea 5.4.1.

5. Sa se exprime urmatoarea retea semantica în logica implicita a lui Reiter.



6. Fie urmatoarea axioma a numerelor naturale, potrivita pentru cazul în care se presupune ca numerele naturale nu sînt singurele obiecte existente.

$$\text{Este} - \text{nr} - \text{natural}(0) \wedge (\forall x)\,(\text{Este} - \text{nr} - \text{natural}(0) \rightarrow \text{Este} - \text{nr} - \text{natural}(\text{Succ}(x)))$$

Sa se circumscrie predicatul Este−nr−natural si sa se interpreteze rezultatul.

7. În lumea blocurilor, se considera predicatul $\text{Peste}(x,y,s)$ care indica faptul ca blocul x este peste blocul y în situatia s. Exista, de asemenea, predicatul $\text{Deasupra}(x,y,s)$ care indica faptul ca blocul x este deasupra blocului y în situatia s. Se considera urmatoarele doua axiome:

$$(\forall x)\,(\forall y)\,(\forall s)\,(\text{Peste}(x,y,s) \rightarrow \text{Deasupra}(x,y,s)) \qquad (\text{i})$$

$$(\forall x)\,(\forall y)\,(\forall z)\,(\forall s)\,(\text{Deasupra}(x,y,s) \wedge \text{Deasupra}(y,z,s) \rightarrow \text{Deasupra}(x,z,s)) \qquad (\text{ii})$$

Axioma (ii) indica faptul ca Deasupra este o relatie tranzitiva. Sa se circumscrie predicatul deasupra în (i) si (ii) si sa se interpreteze rezultatul.

8. O fateta des utilizata, mai ales în modelul de reprezentare bazat pe unitati, este fateta demon. Un demon este o functie sau o procedura atasata slotului, care este invocata ori de cîte ori se face acces sau se modifica valoarea slotului. În cadrul modelului general al unitatilor (cadrelor), demonii, numiti si valori active, pot fi generalizati astfel încît sa cumuleze si rolul procedurii necesare. În cazul de fata, se cere sa se extinda programul prezentat în Sectiunea 5.4.1 prin adaugarea a doua fatete de tip demon: un demon care se executa la adaugarea de valori sloturilor, si un alt demon care se executa la stergerea sloturilor.

9. Sa se scrie un program pentru aplicarea regulilor de productie folosind înlantuirea înainte a regulilor, considerînd reguli nemonotone. Se va pleca de la urmatoarea ipoteza simplificatoare: premisele si concluziile regulilor nu contin variabile si sînt de forma

< obiect >=< valoare >

Strategia de control va aplica prima regula din multimea de conflicte a regulilor aplicabile.

10. Sa se exprime problema din exercitiul 2 sub forma de reguli de productie nemonotone si sa se construiasca reteaua de justificari asociata acestor reguli. Sa se explice functionarea sistemului si sa se indice raspunsurile date pentru întrebarile din exercitiul 2.

11. Fie o problema de alegere a hainelor specificata de urmatoarele conditii:

    - Port jacheta daca este frig.
    - De obicei iarna este frig.
    - Port sandale daca este cald.
    - De obicei vara este cald.
    - Port ghete daca este frig.
    - Nu pot sa port si sandale si ghete simultan.

    (a) Sa se scrie regulile nemonotone asociate si sa se construiasca o retea de dependente TMS care sa modeleze aceasta problema.

    (b) Sa se arate cum poate fi rezolvata problema si cum se modifica starea nodurilor retelei de dependente pe masura ce se fac diverse presupuneri despre anotimp si timp normal sau anormal.

12. Sa se scrie algoritmul de calcul al etichetelor nodurilor într-un sistem de mentinere a consistentei datelor bazat pe presupuneri.

13. Sa se arate cum s-ar putea utiliza un sistem de mentinere a consistentei datelor (TMS), respectiv un sistem de mentinere a consistentei bazat pe presupuneri (ATMS) pentru a putea rezolva eficient probleme de satisfacere a restrictiilor. Pentru exemplificare se va considera problema criptogramei prezentata în Sectiunea 4.4.

5.8 Rezolvari

5. $(\forall x)\ (\text{Persoanã}(x) \rightarrow \text{Fiinþã}(x))$

   $(\forall x)\ (\text{Persoanã}(x) : \text{Locuinþa}(x, \text{Bucureºti}) / \text{Locuinþa}(x, \text{Bucureºti}))$

6. Circumscrierea predicatului Este–nr–natural în axioma data duce la urmatoarea expresie

   $\phi(0) \wedge (\forall x)\ (\phi(x) \rightarrow \phi(\text{Succ}(x))) \wedge (\forall x)\ (\phi(x) \rightarrow \text{Este} - \text{nr} - \text{natural}(x)) \rightarrow$
   $(\forall x)\ (\text{Este} - \text{nr} - \text{natural}(x) \rightarrow \phi(x))$

Interpretarea expresiei obtinute este aceea ca singurele numere naturale sînt acele obiecte pe care axioma initiala le forteaza sa fie numere naturale, ceea ce este esential schema axiomatica uzuala a inductiei.

7. Circumscrierea predicatului Deasupra în (i) si (îi) produce urmatorul rezultat.

$$(\forall x)(\forall y)(\forall s)\ (\text{Peste}(x,y,s) \rightarrow \phi(x,y,s)) \wedge (\forall x)(\forall y)(\forall z)(\forall s)\ (\phi(x,y,s) \wedge \phi(y,z,s) \rightarrow$$
$$(\forall x)(\forall y)(\forall s)\ (\phi(x,y,s) \rightarrow \text{Deasupra}(x,y,s)) \rightarrow \phi(x,z,s)) \wedge$$
$$(\forall x)(\forall y)(\forall s)\ (\text{Deasupra}(x,y,s) \rightarrow \phi(x,y,s))$$

Rezultatul obtinut indica faptul ca relatia Deasupra este închiderea tranzitiva a relatiei Peste.

8. În solutia prezentata se considera ca actiunile ce se executa la adaugarea unei valori într-un slot sînt disjuncte de cele executate la stergerea unei valori dintr-un slot. Functia *u-adauga+* adauga valori într-o unitate cu inspectarea fatetei demon *ADAUGARE?* în timp ce functia *u-sterge+* sterge valori dintr-o unitate cu inspectarea fatetei demon *STERGERE?*.

```
(defun u-adauga+ (unitate slot fateta valoare)
  (when (u-adauga unitate slot fateta valoare)
    (let ((*SLOT* slot))
      (mapc #'(lambda (stramos)
                (let ((*UNITATE* stramos))
                  (mapc #'(lambda (demon) (funcall demon))
                        (u-obtine stramos slot 'ADAUGARE?))))
            (ierarhie unitate)))
    valoare))

(defun u-sterge+ (unitate slot fateta valoare)
  (when (u-sterge unitate slot fateta valoare)
    (let ((*SLOT* slot))
      (mapc #'(lambda (stramos)
                (let ((*UNITATE* stramos))
                  (mapc #'(lambda (demon) (funcall demon))
                        (u-obtine stramos slot 'STERGERE?))))
            (ierarhie unitate)))
    valoare))
```

11. a) Regulile nemonotone asociate conditiilor sînt:

R1: **daca** este frig
atunci port jacheta

R2: **daca** este iarna
si dacanu este anormal
atunci este frig

R3: **daca** este cald
atunci port sandale

R4: **daca** este vara
si dacanu este anormal
atunci este cald

R5: **daca** este frig
atunci port ghete

R6: **daca** port ghete
atunci nu port sandale

R7: **daca** port sandale
atunci nu port ghete

Reteaua de dependente care modeleaza problema este prezentata în Figura 5.21.

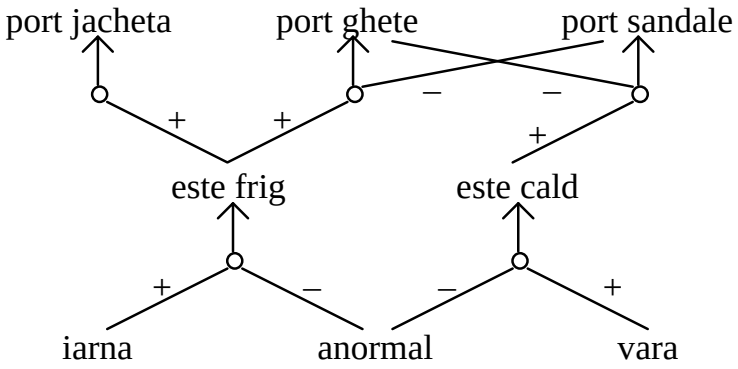


Figura 5.21 Retea de dependente pentru problema anotimpurilor