STUDIA UNIVERSITATIS
Revistă științifică a Universității de Stat din Moldova, 2008, nr.3(13)

ASPECTE TEORETICE ALE PROGRAMĂRII LOGICE

Nicolae PELIN, Serghei PELIN*

Universitatea de Stat Tiraspol, Chişinău
**Universitatea „Alexandru Ioan Cuza", Iaşi*

The present article focuses on some events that contributed to the appearance and evolution of formal logic, logic programming and Prolog programming language. The elements of logic programming theory that brought together let us comprehend the basic concepts are given - from alphabet and syntax to Horn clause. In some sections of the article, the material is structured and presented in the graphical form or in the form convenient for representation in a, so-called, "matrix of knowledge elements" - the environment for the problem description in structured form and tool for knowledge bases construction, used in programmed training.

Introducere

Programarea logică se bazează pe principiile logicii formale. Cercetarile în domeniul logicii formale au fost inițiate de Aristotel în secolul al IV-lea î.e.n. El a creat o operă preştiințifică în domeniul logicii, a elaborat teoria raționii şi formele ei, învățătura despre silogisme, a formulat legile logicii. Logica lui Aristotel, cunoscută şi sub denumirea de logică tradițională, este strâns legată de limbajul natural. În secolul al XIX-lea a apărut primul limbaj formal al logicii – logica propozițională, iar puțin mai târziu un limbaj mai avansat – logica predicatelor.

În limitele acestor formalisme ideile promovate de J.Robinson, R.Kowalski şi alții, predominant din Marea Britanie, în anii '60 ai secolului trecut au pus bazele teoriei programării logice. Însă, prima versiune a limbajului de programare logică Prolog a fost elaborată de francezul A.Colmerauer, lingvist de specialitate. În anul 1971 în Marsilia a fost demonstrat primul interpretator al limbajului de programare Prolog. Denumirea limbajului Prolog este formată din prefixele cuvintelor „PROgramare LOGică". Dar să nu confundăm noțiunile „limbajul Prolog" şi „programare logică". Prologul este doar o mică parte a teoriei programării logice, care în aşa formă şi-a găsit întrebuințarea practică.

Aşadar, ce reprezintă programarea logică ca teorie? La baza limbajului de programare logică stau elementele logicii. Procesul deducerii unei concluzii în baza unor condiții prealabile se numeşte *deducție logică*. În deducția logică, mulțimea predicatelor unei concluzii se pun în concordanță cu mulțimea subiectelor altei concluzii. În procesul deducerii, pe baza subiectului unei concluzii şi a predicatului altei concluzii, se formează o concluzie nouă. În aşa fel, deducția logică poate fi determinată şi ca un proces de raționament.

În articolul dat ne limităm numai la analiza elementelor teoriei limbajului de programare logică. Programarea logică este un limbaj universal, abstract, destinat pentru prezentarea cunoştintelor şi rezolvarea problemelor. Este un mod de tratare a informaticii, prin care în calitate de limbaj de nivel înalt este utilizată logica predicatelor de ordinul întâi (ramură a logicii formale) în formă de fraze Horn [1, p.17].

Prologul este mai adaptat pentru rezolvarea problemelor inteligenței artificiale. Însă, a-l considera unic pentru acest domeniu ar fi incorect. LISP-ul este nu mai puțin cunoscut în domeniul inteligenței artificiale. Totuşi, eleganța lui manifestată în construirea sistemelor de inteligență artificială nu au putut reține autorii monografiilor [2,3] de la dorința de a atrage atenția cititorilor, chiar şi în titlurile de cărți, că Prologul este limbajul inteligenței artificiale.

Prin intermediul programelor în logică, de asemenea pot fi descrise şi realizate bazele de date relaționale, problemele ingineriei programării şi reprezentarea cunoştințelor. Limbajul Prolog este destul de bine adaptat şi la soluționarea altor probleme, fapt care îl determină să fie destul de atragător şi de perspectivă. În Prolog pot fi realizate diverse probleme. Întrebarea ar fi: cât de efectiv şi util este pentru fiecare din ea? O analiză reuşită în acest sens este facută în [6, p.27]. Aprecierea perspectivelor limbajului Prolog este bine formulată în însăşi denumirea monografiei [19].

Limbajul de programare Prolog ne atrage prin simplitatea sa. Un program scris în Prolog este uşor de citit. Datorită formalismelor universale ale frazelor lui Horn, textul programului scris în Prolog, în comparație cu programele scrise în alte limbaje, este mai puțin supus influenței particularităților ce țin de gestionarea cal-

culelor în calculator [1, p.18]. La utilizarea limbajului de programare logică o atenţie deosebită se acordă descrierii structurii problemei aplicative şi nu elaborării indicaţiilor pentru computer, adică la ceea ce el (computerul) trebuie să realizeze pas cu pas.

În prezent Prologul este un limbaj bine cunoscut. La dezvoltarea lui lucrează diverse centre ştiinţifice şi companii, printre care Asociaţia Programării logice (Londra) [16], Centrul dezvoltării Prologului (Copenhaga) [17] şi altele [18].

Un limbaj de programare impune utilizatorului o viziune despre lume. Aceasta se manifestă prin faptul că programatorul, utilizând timp îndelungat un limbaj de programare şi însuşind conceptul acestui limbaj, va tinde să găsească noi sfere de utilizare pentru acele tipuri de calcule, la care acest limbaj este adaptat şi va ignora acele sarcini pentru a căror soluţionare acest limbaj este nepotrivit. Este oportună aprecierea viziunii impuse de diverse limbaje de programare pentru a determina concordanţa lor vizavi de scopurile puse. Aceste raţionamente se referă la ingineria ce ţine de metaprogramare: dacă este pus scopul programării, urmează să fie determinate limbajele şi mijloacele de programare care satisfac acest scop [1, p.13].

Unele noţiuni despre programe logice şi interpretatorul Prolog

Programarea logică este bazată pe ideea că nu omul trebuie instruit să gândească în termenii de operare a calculatorului, dar calculatorul trebuie să îndeplinească instrucţiunile, similar cum le-ar face omul [12, p.10]. În forma sa pură, programarea logică presupune că înseşi instrucţiunile nici nu se includ, iar în locul lor, evident, în formă de axiome logice, sunt formulate date (cunoştinţe) despre problemă şi presupuneri (ipoteze), suficiente pentru soluţionarea ei. Mulţimea de axiome este o alternativă pentru un program obişnuit şi poate fi îndeplinită în acel caz, dacă va fi pusă o afirmaţie-scop formulată în formă de afirmaţie logică.

Lumea reală în programul logic realizat la calculator. În logică, o situaţie reală poate fi descrisă prin propoziţiile (disjuncţiile) lui Horn, adică prin mulţimea de fapte, reguli şi întrebări. La prelucrarea computaţională aceste propoziţii (disjuncţi), conform unei strategii de dirijare stabilite iniţial (de exemplu, căutare în adâncime) sunt supuse influenţei de dirijare (gestionare), folosind aşa noţiuni ca unificarea şi metoda rezoluţiilor. Ordonarea mulţimilor faptelor şi regulilor, la fel şi consecutivitatea stabilită a afirmaţiilor-scop (propoziţia lui Horn, scrisă în formă de între bare), formează aşa-numitul program computaţional aplicativ. Interpretatorul logic este un program care „ştie" cum să deducă concluzii din mulţimile de propoziţii Horn. Reieşind din cele sus-menţionate, apare întrebarea despre interpretarea adecvată a noţiunii de algoritm şi a noţiunii de program vizavi de prezentarea tradiţională.

Noţiunea de algoritm în filosofia programării logice. Conform definiţiei [10, p.55], algoritmul (în prezentare tradiţională) este totalitatea de reguli ce determină procedura efectivă de soluţionare a oricărei probleme dintr-o mulţime de probleme date. Limbajele algoritmice pot fi considerate ca o concretizare a noţiunii de algoritm. În acest caz, algoritmul este tratat ca un text, scris în limbajul algoritmic. Semantica limbajului determină pentru fiecare algoritm (program), scris în acest limbaj, o totalitate de calcule, care se realizează în dependenţă de starea informaţiei, prelucrată prin acest algoritm. Reieşind din această definiţie a algoritmului, problema şi procesul de soluţionare a ei este un tot unitar.

În filosofia programării logice ideea prezentării algoritmilor prin identificarea separată a componentelor „logica" şi „gestiune" joaca rolul principal [5, p.127-128]. R.Kowalski a prezentat schematic această idee în felul următor:

$$\textit{algoritm} = \textit{logica} + \textit{gestionare.}$$

Separarea logicii de gestiune (deducţie logică ca mecanism de calcul) simplifică problema analizei posibilităţilor logice ale algoritmului şi eficacitatea lui în procesul executării.

Programul în limbajele algoritmice (tradiţionale) de programare. Scopul fiecarui calcul constă în transformarea stării iniţiale într-o altă stare finală, „obligând" calculatorul să urmeze un şir de paşi succesivi, stabiliţi iniţial [5, p.299]. Programul este necesar pentru a stabili exact această continuitate şi el (programul) însuşi trebuie pastrat în memoria maşinii. Prin urmare, programul trebuie să fie alcătuit din instrucţiuni discrete ce descriu detaliat atât paşii stărilor, cât şi consecutivitatea lor. În aşa mod, putem ajunge la conştientizarea noţiunii „limbaj de programare von Neumann" ca un limbaj de instrucţiuni pentru maşină. Toate limbajele algoritmice (de la Fortran şi Algol până la Modula şi Java) sunt limbaje de tipul von Neumann. Reieşind

din aceasta, deducerile de mai departe prezentate în [5, p.127-128] de asemenea sunt evidente. Dacă un program este descrierea algoritmului, atunci schematic poate fi reprezentat în felul urmator:

program = descrierea algoritmului.

Luând în considerație schema lui R.Kowalski, schema de mai sus poate fi reprezentată ca

program = descrierea (logica + gestionare).

A separa în aşa mod scrierea programelor în limbajele algoritmice, în cele mai dese cazuri, este imposibil. În limbajele algoritmice procesul de executare a programului este reglat de stările în care se află variabilele. Însă, prin aceste stări se determină ce atribuiri pot avea loc şi în ce ordine. Adică, a descrie (logică + gestionare) în progamul algoritmic se poate numai ca un tot. În acest caz, legăturile logice dintre variabile nu pot fi analizate şi explicate fără a face trimitere la executarea programului. Însă, această circumstanță este fatală pentru practica programării.

Sarcina ce stă în fața programatorului având la dispoziție un interpretator logic. Programând în logică, programatorul dispune deja de o descriere a procesului de executare a calculului, ca parte componentă a interpretatorului logic (un motor de inferență). Procesul de executare a calculului nu depinde de problema în cauză. Scopul programatorului constă în descrierea logicii soluționării problemei. Programatorul trebuie doar să înțeleagă corect comportamentul interpretatorului ce-i stă la dispoziție. Alegând logica corespunzătoare a programului inițial, el capătă posibilități mari în gestionarea comportamentului programului său.

Aşadar, în programele logice practic nu se face descrierea gestionării cu procesul de calcul, deoarece deja se află în competența interpretatorului. Din cele sus-menționate reiese că formula se reduce la expresia:

program = descrierea logicii,

adică, pentru identificarea mulțimii de afirmații logice poate fi folosit termenul „program”. Din aceste considerente, am convenit ca în articolul de față să fie prezentată teoria programei scrise în logică, iar în articolul care va urma se va face descrierea calculului executat în mod automat de către interpretatorul logic. Considerăm că, din punct de vedere metodologic, această separare permite, într-o oarecare măsură, o altă viziune asupra aceleiaşi materii ce ține de programarea logică şi poate trezi interes suplimentar la studierea problemei.

Pentru conştientizarea diferenței dintre programarea logică şi cea algoritmică, schemele prezentate mai sus sunt de o importanță majoră. Dar, totuşi, trebuie să tinem cont de unele constrângeri puse in limbajul practic de programare Prolog. Predicatul ***cut*** [28], admiterea dreptului de a nu aplica legea comutativă ce ține de conjuncții ale lui Horn [4,25], forma conjunctiv-normală utilizata în Prolog, alte aspecte asigură mari posibilități programatorului, dar fac limbajul Prolog mai puțin logic. Posibil, probmele ce țin de curățenia limbajului, din punctul de vedere al logicii, vor fi soluționate în noile versiuni ale limbajelor logice de programare care vor urma; posibil, aceste probleme parțial sunt deja soluționate în limbajul logic Mercury.

Caracterul nondeterminist al programelor logice. Programele scrise în logică pot da o mulțime de rezultate de alternativă. Dacă programul concret permite mai mult decât un rezultat, atunci el se consideră nondeterminist. Caracterul nondeterminist presupune lipsa factorilor care determină exact mersul executării programului. Prologul este un limbaj de programare nondeterminist, deoarece în afirmațiile programelor logice lipsesc factorii ce determină mersul executării ei.

Interpretatorul limbajului Prolog. Noțiunea „interpretare” în traducere din latină înseamnă explicare, traducere. Interpretatorul este un procesor al limbajului, care rând cu rând analizează programul la ieşire şi concomitent execută unele acțiuni prescrise, dar nu formează în limbajul maşinii un program compilat, care să fie realizat ulterior [11, p.245].

Interpretatorul limbajului Prolog este un mecanism al deducției logice, acea „forță activă” care execută *de facto* programele scrise în Prolog. Despre interpretatorul limbajul Prolog se va scrie în unul din articolele următoare.

Programarea logică ca teorie

După cum s-a menționat mai sus, programarea logică a luat naştere în adâncurile logicii formale şi nu ne putem lipsi de cunoaşterea unor elemente importante din logica propozițională şi logica predicatelor. Aceste fomalisme sunt bine descrise şi pe larg în mai multe surse [1,4,6,7,8,14]. În ce ne priveşte, vom menționa,

succint, numai momente din logica propoziţională şi logica predicatelor necesare pentru descrierea aspectelor principale care ne aduc la (şi ţin de) teoria programării logice.

Logica propoziţională este o teorie simplă, dar utilizată pe larg în diverse domenii. Practic, stă la baza oricărei teorii logico-matematice. Simplitatea ei nu este un obstacol pentru a o considera plină de conţinut şi utilizată pe larg pentru diverse probleme teoretice şi aplicative. Informaticienii ar trebui să manifeste străduinţă faţă de studierea logicii, ca astfel să se poată descurca mai uşor în mai multe probleme, inclusiv în programarea logică, în principiul de lucru al interpretatorului limbajului de programare logică, în versiunile practice ale limbajului Prolog.

Dupa cum ştim, logica propoziţională este un limbaj ce studiază propoziţii atomice, adică propoziţii ce nu pot fi descompuse. Ele pot fi desemnate prin simbolurile ***p, q, r*** şi interpretate ca ***adevăr*** ori ***fals [A, F].***

Logica predicatelor permite de a pătrunde în structura internă a propoziţiei, având o formă de prezentare, de exemplu, ***p(a,b),*** dar poate fi interpretat ca şi în logica propoziţională, numai împreună – ca ***[A, F].***

Propoziţiile compuse se formează cu ajutorul legăturilor, analogice conjuncţiei «şi», desemnate în formule prin simbolul $\wedge$, «ori »$\vee$, «dacă….., atunci» $\supset$, «atunci şi numai atunci, când» $\equiv$ şi negaţia „not", desemnata prin $\neg$. Prioritatea legăturilor în propoziţiile compuse se stabileşte ca în matematică cu ajutorul parantezelor. Pentru a reprezenta totalitatea propoziţiilor, care formează un text integru, rareori se foloseşte semnul «,». În logica predicatelor se operează şi cu cuantificatori universali şi existenţiali, noţiuni de constante şi variabile. Regulile de formare a formulelor se reduc la: a) stabilirea bazelor, că propoziţiile elementare ***p*** şi ***q*** în logica propoziţională (ori ***p(a, b)*** şi ***q(x, y)*** în logica predicatelor), sunt formule (formule elementare, atomice), iar b) propoziţiile compuse se formează prin utilizarea succesivă a pasului inductiv $\neg p$, $p \wedge q$, $p \vee q$, $p \supset q$ şi $p \equiv q$ de câte ori este necesar.

Veridicitatea formulelor se determină cu ajutorul tabelelor de adevăr, a formulelor mai complicate – prin aplicarea metodelor algoritmice şi a tabelelor de adevăr. Interpretarea prin care valoarea veridicităţii formulei este ***A*** se numeşte ***modelul*** acestei formule. Formula ce admite unul sau mai multe modele se numeşte *executabilă*, iar cea care nu admite nici un model – *neexecutabilă*. Dacă formula admite un model indiferent de valoarea formulelor atomice din care constă, atunci formula este numită *tautologie*.

Pentru interpretarea formulelor cu ajutorul metodelor algoritmice pot fi utilizate transformări echivalente cu scopul de a le aduce la o forma normală.

În programarea logică se operează cu forme normale conjunctive sau normale disjunctive. În prezentul studiu ne limităm doar la examinarea formelor normale conjunctive. De aceea, este tocmai timpul să iniţiem noţiunea de disjunct.

Disjunctul. Disjunctul (clauses) se numeşte disjuncţia numarului finit al literalilor, adică a propoziţiilor elementare sau negaţia lor.

Fie $p_1, p_2, \ldots, p_n$ – mulţime de literali, atunci prezentarea formală a disjunctului poate fi următoarea:

$$p_1, \vee p_2 \vee \ldots \vee p_n,$$

Disjunctul poate să conţină numai un literal. În cazul acesta prezentarea formală a disjunctului va fi, de exemplu, ***p***, adică indentică cu prezentarea unei propoziţii (formule) elementare. Disjunctul poate să nu conţină nici un literal. În acest caz îl vom indentifica prin □**.**

Noţiunea de disjunct are o însemnătate practică majoră. Descrierea problemelor şi algoritmilor în termeni disjuncţi constituie baza programării logice şi a limbajului Prolog în particular. De fapt, Prologul operează numai cu o submulţime de disjuncţi, numiţi *disjuncţii lui Horn*, care vor fi explicaţi mai jos.

Un disjunct vid, adică un disjunct care nu are nici un literal, este unicul din disjuncţii care nu sunt executabili şi este desemnat prin ***F***. Determinarea noţiunii de disjunct vid este importantă în programarea logică. Odată cu introducerea ei apare posibilitatea determinării scopului în deductorul automat al problemelor.

Forma normală conjunctivă (FNC). Explicând noţiunea de disjunct, aducem definiţia FNC. Forma normală conjunctivă este caracterizată prin conjuncţia numarului final de disjuncţi. Formal poate fi reprezentată astfel:

$$(p_1, \vee p_2 \vee \ldots \vee p_n) \wedge (\neg q_1 \vee \neg q_2, \ldots, \neg q_m) \wedge \ldots \wedge (r_1 \vee r_2 \vee \ldots \vee r_l)$$

sau

$$S=\{(p_1, \vee p_2 \vee \ldots \vee p_n), (\neg q_1 \vee \neg q_2, \ldots, \neg q_m), \ldots, (r_1 \vee r_2 \vee \ldots \vee r_l)\}.$$

Orice formulă logică poate fi transformată în FNC. Orice FNC poate fi retransformată în FNC care conține numai disjuncți Horn. Dovada corectitudinii acestor afirmații se poate găsi, de exemplu, în [4].

Definiția disjuncților Horn. În articolul dat analizăm doar FNC care conține numai disjuncții Horn (în sursele bibliografice mai pot fi numiți *propoziții, clauze, fraze Horn*), în care se admite cel mult câte un literal pozitiv.

În corespundere cu definiția dată, disjunctul $\neg p \vee \neg q \vee \neg r \vee s$ este disjunct Horn şi este echivalent implicației $(p \wedge q \wedge r) \supset s$. Dacă acest disjunct îl vom scrie în forma ***s: - p, q, r.*** , atunci el (disjunctul Horn) poate asemui regula de structură a frazei tipice pentru gramatica liberă de context [8, p.49]. În acest disjunct propozițiile şi simbolul ***F*** pot fi considerate simboluri ale acestei gramatici. Aşa o formă de prezentare este apropiată de cea acteptată în limbajul Prolog, dacă literalii sunt considerați ca proceduri.

Disjunctul Horn ***unar*** conține numai un literal pozitiv şi nici unul negativ. Exemplu de formă scrisă: ***s:-.*** Acest disjunct serveşte pentru reprezentarea faptelor.

Disjunctul Horn ***fix*** conține un literal pozitiv şi unul sau mai mulți literali negativi. Disjunctul ***fix*** exprimă regula în care literalii negativi sunt ipoteze, reprezentate de propoziții corespunzătoare, iar literalii pozitivi- încheiere. Exemplu de formă scrisă*:* ***s:-p,q,r.*** Această propoziție corespunde înscrierii pentru un disjunct ***fix***.

Disjunctul Horn ***negativ*** poate fi reprezentat astfel: ***F :- p,q,r.*** Aceasta corespunde noțiunii «scop, întrebare» sau, mai exact, «negarea scopului, întrebării». În exemplul dat scopul (disjunctul Horn ***negativ***) este compus. Exemplu al unui scop simplu: ***F:-p.***

Formatul înscrierii scopului şi faptelor în Turbo-Prolog (versiune bine cunoscută a limbajului de programare în legatură cu numeroasele publicații în mai multe limbi [8,22,23,24,25,26,27]) se deosebeşte de cel al înscrierilor prezentate mai sus, având la bază acelaşi conținut. De exemplu, înscrierea unui scop simplu în Turbo-Prolog va fi: ***p.*** Înscrierea scopului compus în acest limbaj: ***p,q,r.*** Înscrierea faptului**:** ***p.*** Pentru a nu confunda, spre exemplu, faptele ***p*** cu întrebările **p**, faptele şi regulile se înscriu untr-un compartiment al programului, numit ***CLAUSES*****,** iar scopul se înscrie în alt compartiment al programului, numit ***GOAL*****.**

Logica predicatelor. Până în prezent, pentru descrierea unui şir de elemente, acceptate în programarea logică, ne-am limitat la formalismul logicii propozițiilor. Pentru îndeplinirea următorilor paşi în programarea logică necesităm formalismul logicii predicatelor.

Cum s-a menționat mai sus, în logica predicatelor propozițiile elementare, la fel ca în logica propozițiilor, sunt considerate ca un tot unitar, înterpretat cu semnificația ori ***Adevăr,*** ori ***Fals***. Cu toate acestea, în logica predicatelor este posibilă formalizarea structurii interne. Să demonstrăm aceasta printr-un exemplu. Să analizăm o propoziție elementară:

«Limbajul Prolog este mai eficient decât limbajul C».

În logica propozițiilor putem stabili numai adevărul acestei propoziții. În logica predicatelor putem prezenta şi structura ei internă. În acest context, relația ***«este mai eficient decât»*** vom nota-o prin ***P,*** iar obiectele legate cu această relație ***«Limbajul Prolog»*** şi ***«limbajul C»*** le vom nota prin q_1 şi q_2**.** Atunci, prezentarea formală a formulei atomice pentru

Limbajul Prolog este mai eficient decât limbajul C»
obiect relație obiect
$q_1 P q_2$

va fi expresia: $P(q_1, q_2)$**.** Pentru a opera cu noțiunile logicii predicatelor, este important de a avea cunoştințe mai clare despre formula atomică pentru a nu confunda mulțimea definițiilor fiecăruia din elementele ei. Definițiile elementelor respective sunt bine formulate în diverse izvoare [1,4,7,13,14]. Totuşi, după cum arată practica, aceste definiții sunt însuşite cu greu de către studenți în procesul de instruire.

În viziunea noastră, este eficientă prezentarea formulei atomice în formă de arbore (Fig.1), propusă în [14] pentru perceperea mai uşoară a tuturor elementelor în interconexiune.

În calitate de identificator simbolic ***P*** pentru relație în formula atomică în general şi în formula $P(q_1, q_2)$ se foloseşte termenul ***„constantă predicativă”.*** Obiectele q_1, q_2 sunt termeni, care au rolul de argumente în formulă**,** iar însuşi $P(q_1, q_2)$ se numeşte *formă predicativă*. Numărul argumentelor în formă predicativă determină aritatea ei. De exemplu, forma predicativă $P(q_1, q_2)$ este de aritatea ***doi;*** forma predicativă ***P*** – de aritatea ***zero***; forma predicativă ***P(a,b,c)*** **–** de aritatea ***trei.***

Toate componentele posibile ale formulei atomice în logica predicatelor poate fi identificată şi prezentată demonstrând în formă grafică legăturile dintre ele, dupa cum să vede din Figura 1.

Deci, formula atomică, ori atomul, este forma predicativă (de exemplu, ***p(a,b)*** ori ***p(a,b,x)***), prezentată în forma prefix, ori o relaţie (de exemplu, s = t), prezentată în formă infixă. Forma infixă poate fi înlocuită cu cea prefixă, dacă se acceptă un acord în acest caz. Atunci, în cazul exemplului nostru, formula atomică va fi prezentă în forma: = (s,t). Restul componenţilor formulei atomice pot fi formulaţi interpretând elementul corespunzator din Figura 1.

Obiectele logicii predicatelor sunt exprimate prin variabile şi constante funcţionale de aritatea ***zero***. În limbajul logicii predicatelor este inclus limbajul logicii propoziţionale. Propoziţiile în logica predicatelor sunt considerate ca constante predicative fără argumente. Ea se mai numeşte constanta predicativă cu aritatea ***zero***, ori constantă de aritatea ***zero***.

În limbajele naturale se întâlnesc frecvent expresii de tipul «pentru toţi bărbaţii», «pentru unii studenţi» şi multe alte expresii analogice. Expresiile universale de tipul «pentru toţi…» în logica predicatelor sunt identificate prin cuantificatorul universal $\forall$, iar «pentru unii…» – prin cuantificatorul existenţial $\exists$.

Înscrierea $\forall xP(x)$ semnifică «pentru toate valorile lui ***x P(x)*** **–** propoziţie adevărată**».** Exemplu: $\forall x(x^2+1>0)$ – pentru fiecare valoare ***x*** inegalitatea $(x^2+1>0)$ este adevărată.

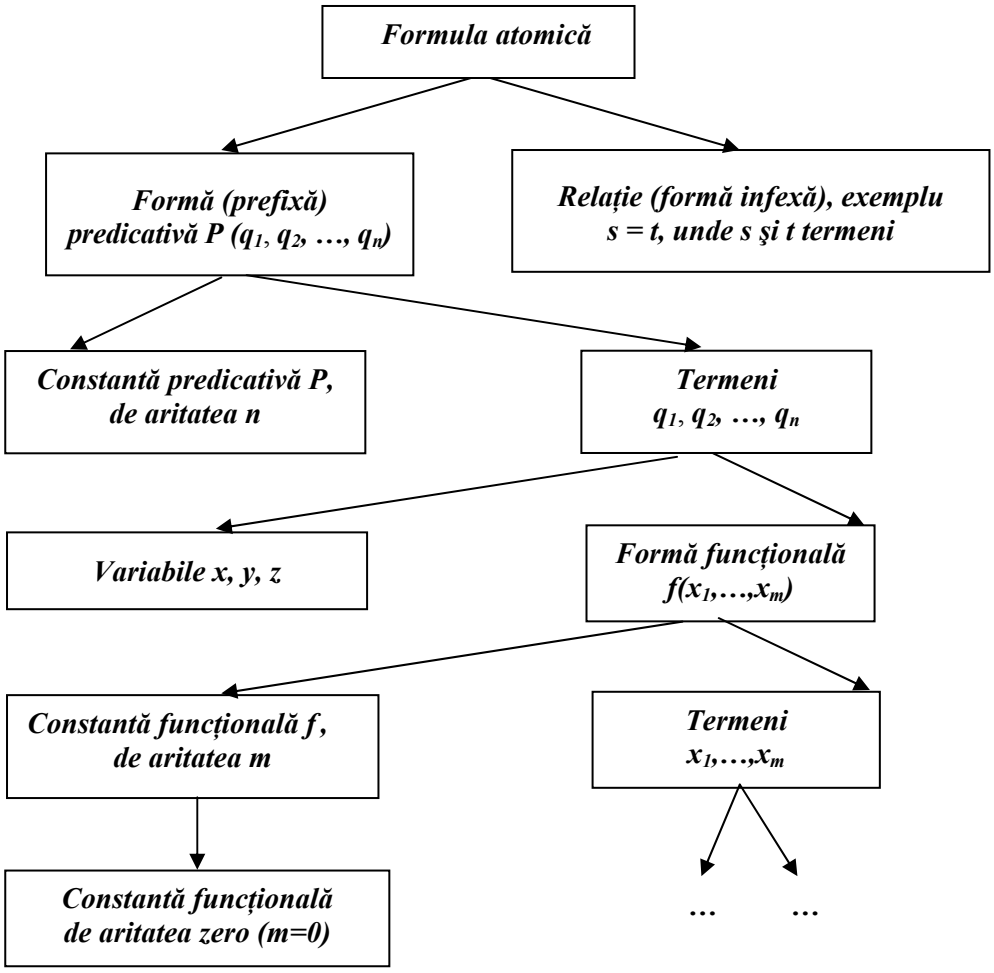

Fig.1. Prezentare grafică a tuturor componenţilor posibili în formula atomară a logicii predicatelor.
Înscrierea $\exists xP(x)$ denotă «pentru unele valori ale lui ***x P(x)*** – propozitie adevărată».
Exemplu: $\exists x(3+x=3)$ – dacă $x=0$ egalitatea $(3+x=3)$ este adevarată.

Să reţinem că dacă în componenţa formulei este o variabilă, atunci inainte de a i se atribui o valoare de adevar acestei formule, variabila urmeaza a fi cuantificată.

Simbolurile de bază ale limbajului logicii predicatelor sunt: variabile, constante de aritatea 0, constante predicative, conectori (negația, conjuncția, disjuncția, implicația şi echivalența) logici, cuantificatorii universal şi existențial.

Variabilele şi rolul lor. În matematică variabilele constată, în ctructura obiectului matematic, acele locuri, care prin utilizarea acestui obiect vor fi ocupate de alte obiecte. Variabilele se divizează în *variabile libere* şi *variabile legate*. De exemplu, în formula

$$\mathbf{S} = \sum_{i=1}^{n} \boldsymbol{6}\, \boldsymbol{x},$$

$\boldsymbol{x}$ – variabilă liberă, $\boldsymbol{i}$ – variabilă legată.

În logică, din înțelegerea intuitivă a cuantificării reiese ca există legături între intrările variabilei ce se conține în cuantificator. Examinăm formula $\forall \boldsymbol{x}\, \boldsymbol{p}\, (\boldsymbol{x})$**.** Aici variabila $\boldsymbol{x}$ este «legată».

Domeniul de acțiune a unui cuantificator este formula în care se aplică această cuantificare, iar variabila $\boldsymbol{x}$ în cuantificatorii $\forall \boldsymbol{x}$ sau $\exists \boldsymbol{x}$ se numeste *cuantificată*. Fiecare intrare a variabilei $\boldsymbol{x}$ în aria de acțiune a acestei cuantificări este legată. Examinăm altă formulă: $\forall \boldsymbol{x}\, \boldsymbol{p}\, (\boldsymbol{x})\, \wedge \boldsymbol{q}(\boldsymbol{x})$**.** Aici prima intrare a variabilei $\boldsymbol{x}$ este «legată», a doua – «liberă».

Despre interpretarea formulelor în logica predicatelor. La fel ca în logica propozițională, formulele logicii predicatelor pot fi identificate prin semnificația: ***[A, F]***. Totodată, în logica predicatelor formulele constau nu numai din subformule şi literali, dar şi din termeni, adică variabile şi forme funcționale. Ultimele, la rândul lor, constău din constante funcționale, legate corespunzător cu un număr potrivit de termeni. Intuitiv, noțiunea de termen înseamnă obiect. Prin urmare, interpretarea trebuie să specifice mulțimea de obiecte, care constituie domeniu de interpretare.

Să examinăm o metodă de construcție a semanticii logicii predicatelor, care se prezintă astfel:

1) se evidențiază o mulțime nevidă, numită domeniu de interpretare;
2) se determină funcția ce pune în comparație expresiile limbajului cu obiecte, mulțimi de obiecte sau relația dintre obiecte din domeniul de interpretare;
3) în baza acestor condiții se determină noțiuni importante ce țin de executabilitatea, veridicitatea, tautologia formulelor limbajului cercetat.

O informație mai amplă referitor la interpretarea limbajului logicii predicatelor o putem găsi în [4, p.62-64] şi în [21, p.23-24].

În logica predicatelor, suplimentar la transformările echivalente din logica propozițională, există, la rândul lor, formule suplimentare, care descriu corelația dintre cuantificări şi conectori [4, 9,13,14].

Forma prenex şi legătura ei cu formulele logicii propoziționale. Analogic cu formele disjunctiv şi conjunctiv normale ale formulelor din logica propozițională, se aduc la forma normală şi formulele logicii predicatelor. Dar aici operăm cu cuantificatorii şi domeniile lor de acțiune pot fi destul de complicate. Problema se simplifică pentru cazul formelor prenex. Dupa cum se ştie, forma prenex este o formulă ce constă din matrice, în fața căreia stau un şir de cuantificatori finiți. Aceşti cuantificatori se referă la diverse variabile şi ordinea lor contează. În cazul când cuantificarea se repetă pentru una şi aceeaşi variabilă, putem accepta numai primul cuantificator: aceasta se coordonează cu regula generală de interpretare a formulelor cuantificate [4, p.69-70]. Aşadar, formula se prezintă astfel: $Q_1x_1, Q_2x_2, \ldots, Q_nx_nM$, unde simbolul Q denotă ori $\forall$, ori $\exists$, pentru $i= 1,2,\ldots, n$ şi M – formulă (matrice) care nu conține cuantificatori.

Interesul față de formele prenex este legat de teorema: **„**Pentru orice formulă logică există o formă prenex, logic echivalentă ei". Aici nu vom analiza algoritmul deducerii formei prenex. Etapele deducerii formei prenex sunt aduse, de exemplu, în [4, p.70-71].

FCN, deduse în logica propozițională, se extind şi asupra logicii predicatelor. FCN este formă prenex, a cărei matrice este conjuncția disjuncților.

Probleme ce apar în operarea cu formele prenex. Orice formulă a logicii predicatelor permite forma prenex echivalentă. Pe de altă parte, forma prenex permite forma echivalentă, normală, uneori mai voluminoasă decât cea inițială. Formele prenex prezintă interes şi prin faptul că păstrează puterea expresivă a cal-

culului predicatelor şi cer utilizarea „disciplinată" a mecanismului de cuantificare. Dar anume aceasta este foarte greu de realizat.

Pot fi fixate limite mai stricte de utilizare a mecanismului de cuantificare, cu toate că prin aceasta puterea expresivă a mecanismului de cuantificare se diminuează.

Pentru aceasta, fiecare formulă A este comparată cu oricare formulă S_A – simplă în construcţie şi care satisface doar condiţia că şi S_A, şi A sunt concomitent executabile ori sunt imposibil de executat. Formula S_A este cunoscută sub denumirea *forma Skolem*. În Figura 3 este reprezentat algoritmul de aducere în forma Skolem.

Legătura dintre A şi S_A este strict mai slabă decât echivalenţa logică. Şi totuşi, dovada neexecutabilităţii devine mai efectivă, dacă ne limităm la formulele reprezentate în forma Skolem. Operând cu forme Skolem, se analizează numai forme prenex ale formulelor logicii predicatelor, şi aceste formule trebuie sa fie închise.

După cum cunoaştem, formule închise sunt acele formule care nu au variabile libere.

Exemple:

1. $\forall x\,[Q(x) \wedge \forall y\,\exists z\,P(a,y,z)] \wedge R(x)$ – formulă deschisă.
2. $\forall x\,[Q(x) \wedge \forall y\,\exists z\,P(a,y,z) \wedge R(x)]$ – formulă închisă.

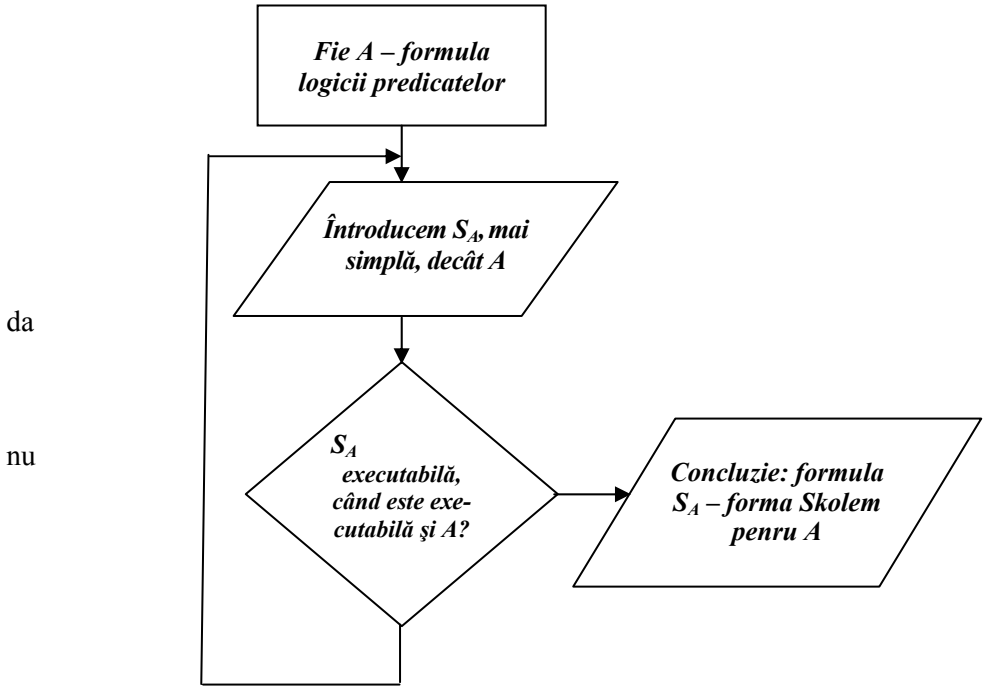

Fig.2. Bloc-schema algoritmului de determinare a formei Skolem.

Operaţiile preventive de retransformare a unei formule a logicii predicatelor în forma Skolem ne aduc la forma prenex închisă. Procesul de retransformare a formulei logicii predicatelor în forma Skolem presupune reprezentarea cuantificatorului $\exists$ prin $\forall$.

Forma clauzală. Formă clauzală se numeşte aşa o forma Skolem S_A a formulei logicii predicatelor A, a cărei matrice se găseşte în FNC, adică

$$S_A = <\textbf{consecutivitatea cuantificatorilor universali} > [\textbf{matricea în FNC}].$$

Oricare formă Skolem admite formă clauzală echivalentă.

Nu există un algoritm care ar permite să stabilim că o formulă este tautologie, neutră ori neexecutabilă, deoarece ea poate să dispună de o mulţime infinită de interpretări. În situaţia dată prezintă interes rezultatele lui Herbrand. Ele duc la controlul simplificat al executabilităţii formulelor.

Universul Herbrand. Ideea de bază care ne aduce la noțiunea „universul Herbrand" constă în următoarele [4, p.7]. Forma clauzală este neexecutabilă atunci şi numai atunci când primeşte valoarea ***F*** în toate interpretările. Analizarea tuturor interpretărilor posibile este irealizabilă, dar ar fi bine dacă ar exista posibilitatea de a determina o arie specială strâns legată de forma clauzală studiată, şi forma nominalizată să fie neexecutabilă atunci şi numai atunci când ea primeşte valoarea ***F*** în toate interpretările acestei arii. Aşa o arie există şi se numeşte *universul Herbrand.*

Forma frazală a logicii predicatelor. Forma frazală a logicii predicatelor este o metodă de scriere a formulelor, în care se folosesc numai conectorii $\neg$, $\wedge$, $\vee$. FNC determinată mai sus este reprezentată în forma frazală. Literalul este o formulă atomică pozitivă ori negativă. Fiecare frază în parte reprezintă o mulțime de literali, conectați prin $\vee$. Aceasta nu este altceva decât un disjunct, determinat mai sus. Literalii negativi sunt repartizați la sfârşitul fiecărei fraze, iar cei pozitivi la început [1, c.53-54]. Schematic forma frazei este:

$$p_1 \vee p_2 \vee \ldots \vee p_n \vee \neg q_1 \vee \neg q_2 \vee \ldots \vee \neg q_m .$$

După cum se ştie, o frază poate fi examinată ca o noțiune generalizată a implicației. De exemplu, dacă ***p*** şi ***q*** sunt formule atomice, atunci formula $p \vee \neg q$ în acelaşi timp poate fi reprezentată şi ca $q \supset p$.

Termenii «frază» şi «formă frazală» de asemenea sunt utilizați şi la fel se folosesc ca şi termenii «disjunct» şi «FNC» .

Cuantificarea variabilelor în formă frazală. Cuantificare evidentă a variabilelor în formă frazală nu se face. Dar se subînțelege că toate variabilele sunt cuantificate. Astfel, în fraza $P(x,y,z) \vee Q(x,y,z)$ se subînțelege prezența cuantificatorilor, adică această frază se interpretează (se subînțelege) în felul în care este scrisă mai jos:

$$\forall x \forall y \forall z \, [P(x,y,z) \vee Q(x,y,z)].$$

Structura programului logic standard. Un program logic standard constituie o afirmație scop (întrebare) şi un număr arbitrar de proceduri (fapte şi reguli). Numărul şi lungimea afirmațiilor nu este limitat. În programe reale, constanta predicativă şi termenii formulei atomice sunt identificați, pentru a fi mai uşor citit în limbajul natural, nu numai prin charactere (litere latine, cifre etc.), dar şi prin cuvinte ori grupe de cuvinte, divizate unele de altele (de exemplu, în limbajul Turbo-Prolog) prin simbolul subliniere. Constantele în limbajul Turbo-Prolog, dacă nu sunt luate în „", se încep cu literă latină minusculă ori cu cifre arabe, variabilele – prin majusculă ori simbolul subliniere. Exemplu de program scis în limbajul Turbo-Prolog:

```
PREDICATES
 parinte(symbol,symbol)
CLAUSES
 părinte(ion,nicolae).
 părinte(nicolae,sergiu).
 părinte(sergiu, diana).
 bunei(X,Z):-
 părinte(X, Y),
 părinte(Y, Z).
 GOAL
 bunei(X, diana).
```

Textul progamului conține toate trei tipuri de propoziții Horn: trei fapte (disjuncți unari), o regulă (disjunct fix) şi o întrebare ori un scop (disjunct negativ). În urma coordonării scopului ***bunei(X, diana) X*** se leagă cu valoarea ***nicolae.***

Ordinea în care procedurile (fapte şi reguli) urmează în program nu are semnificație logică, deoarece fiecare procedură constată un fapt ce ține de problema preconizată pentru soluționare. Totuşi, procedurile cu una şi aceeaşi constantă predicativă se adună într-o grupă.

Concluzii

În articol sunt analizate succint unele aspecte teoretice ale elementelor programării logice.

Din punct de vedere metodologic şi restricții în volum, s-a atras atenția doar la aspectele principale. În afara studiului au rămas mai multe direcții alternative ori de importanță secundară. Dar şi astfel prezentată,

materia poate fi interesantă pentru cei care doresc să se informeze în linii generale despre teoria programării logice. Informaţia din prezentul studiu este structurată în aşa fel pentru a fi mai uşor prezentată în «matricea elementelor de cunoştinţe» [15] şi utilizată în sisteme automatizate de educaţie, inclusiv la distanţă prin Internet.

Referinţe:

1. Малпас Дж. Реляционный язык Пролог и его применение / Пер. с англ. Под. ред. В.Н. Соболева. - Москва: Наука, 1990. - 464 с.
2. Макаллистер Дж. Искусственный интеллект и Пролог на микроЭВМ / Пер. с англ.- Москва: Машиностроение, 1990. - 240 с.
3. Братко И. Программирование на языке Пролог для искусственного интеллекта / Пер. с англ. - Москва: Мир, 1990. - 560 с.
4. Тей А., Грибомон П., Луи Ж. и др. Логический подход к искусственному интеллекту: от классической логики к логическому программированию / Пер. с. франц. Пермякова П.П. Под. ред. Гаврилова Г.П. - Москва: Мир, 1990. - 432 с.
5. Хоггер К. Введение в логическое программирование / Пер. с англ. - Москва: Мир, 1988.- 348 с.
6. Cotelea V. Programarea în logică. - Chişinău: Nestor, 2000. - 394 p.
7. Ковальски Р. Логика в решении проблем // Проблемы искусственного интеллекта / Пер. с англ. - Москва: Наука, 1990. - 280 с.
8. Masalagiu C., Ibănescu L., Andrei Ş. Practica programarii în TURBO PROLOG. - Iaşi, 1998.
9. Кузнецов О.П., Адельсон-Вельский Г.М. Дискретная математика для инженера. 2–е изд., перераб. и доп. - Москва: Энергоатомиздат, 1988. - 480 с.
10. Словарь по кибернетике / Под ред. В.С. Михалевича. 2-е изд. - Киев: Гл. ред.УСЭ, 1989. - 751 с.
11. Толковый словарь по вычислительным системам / Под ред. В. Иллингуорта и др. Пер. с англ. - Москва: Машиностроение, 1989. - 568 с.
12. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. - Москва: Мир, 1987. - 235 с.
13. Pelin S. Prezentarea realităţii cu ajutorul logicii. Raport prezentat în doctorantura Universităţii „Alexandu Ioan Cuza". - Iaşi, 2007.
14. Пелин Н. Элементы логического программирования. - Кишинев: Nestor, 2000. - 171 с.
15. Pelin N., Pelin S., Pelin A. The matrix of knowledge elements effective tehnology for knowledge management. Proceedings of the International Conference "Knowledge management: Projects, Systems and Tehnologies". Volume I, november 9-10, 2006, Bucharest. The 2nd supliment of the review INFORMATICA ECONOMICA. Volume X, ISSN 1453-1305, p.63-71.
16. http://www.cs.kuleuven.ac.be/~dtai/projects/ALP/
17. www.pdc.dk
18. www.prolog.md
19. Доорс Дж., Рейблейн А., Вадера С. Пролог – язык программирования будущего. – Москва: Финансы и статистика, 1990. - 144 с.
20. Лорьер Ж.-Л. Системы искусственного интеллекта / Пер. с франц. - Москва: Мир, 1991. -568 с.
21. Логика и компьютер. Моделирование рассуждений и проверка правильности программ / Н.А. Алешина, А.М. Анисов, П.И. Быстров и др. - Москва: Наука, 1990. - 240 с.
22. Янсон А. Турбо-Пролог в сжатом изложении. - Москва: Мир, 1991. - 94 с.
23. Ин Ц., Соломон Д. Использование Турбо-Пролога. - Москва: Мир, 1993.- 608 с.
24. Shildt R. Turbo-Prolog. V.1.0 - Borland, 1988. - 457 p.
25. Tatar D. Inteligenţă Artificială: demonstrarea automată a teoremelor, prelucrarea limbajului natural.- Cluj-Napoca: Editura Albastră, 2001. - 230 p.
26. Language Tutotial: Prolog 3.31. - Copenhagen: PDC, 1996. - 465 p.
27. Pelin S., Orlovschi I. Turbo-Prolog. - Chişinau: UŞAM, 2001. - 17 p.
28. Robinson J. Logic programming – past, present and future. - New Generation Computing, 1, 1983, p.107-121.

Prezentat la 18.03.2008