STUDIA UNIVERSITATIS
Revistă științifică a Universității de Stat din Moldova, 2008, nr.8(18)

DEDUCȚIA LOGICĂ – MECANISM DE CALCUL AL LIMBAJULUI DE PROGRAMARE LOGICĂ

Serghei PELIN*[*]*, *Nicolae PELIN

Universitatea de Stat Tiraspol, Chişinău
[*]Universitatea „Al.I. Cuza”, Iaşi

The present article focuses on analysis and synthesis of elements which are the base of abstract interpreter of programs written in logic. Logic programming and Prolog language significance in studying computer science is appreciating. Some methodological aspects of logic programming training are analyzed, are described some core functions of SPprolog – a developing environment that represents means for developing applications in Prolog supported by the intellectual training system which provides training for working with the system and logic programming language.

Rolul limbajului Prolog în procesul de studiere a cursului de informatică

Alen Colmerauer, creatorul primului interpretator al limbajului Prolog, scria că limbajului Prolog îi este predestinat să joace unul dintre principalele roluri în informatică. Pentru predarea corectă a cursului de informatică este necesară utilizarea unui limbaj, care ar ajuta să structureze gândirea. Acesta ar putea să fie Lisp, dar şi mai bine Prolog [5, p.29].

După cum este menționat în [8], în procesul de formare a cursului de informatică pot fi deosebite trei etape: algoritmică, tehnologică şi conceptuală. La etapa ***algoritmică***, conținutul cursului de informatică se axa pe studierea algoritmilor şi programarea în limbajul algoritmic; în următoarea etapă, cea ***tehnologica,*** accentul era pus pe însuşirea diverselor programe şi mijloace tehnice. În perioada modernă, aspectul ***conceptual*** al cursului de informatică prezintă un raport de interacțiune a ideilor filosofice, lingvisticii, formalismelor şi a programării. În unele țari, sistemul metodic de studiere a informaticii suferă schimbări, legate de trecerea de la cursul integru de informatică în clasele superioare la structurarea lui în mai multe etape. Dacă vorbim despre cursul de informatică cu profil umanitar, atunci, desigur, el tinde spre informatica conceptuală, şi cursului dat îi corespunde, cum nu se poate mai bine, paradigma de programare logică şi limbajul Prolog. De fapt, Colmerauer, creatorul limbajului Prolog, este de specialitate lingvist. Şi aceasta vorbeşte deja despre multe.

Reieşind din experiența de mai mulți ani de predare a cursurilor respective de teorie şi practică a programării logice şi observând succesele audienților acestor cursuri în activitatea lor ulterioară, concludem că cursul de programare logică trebuie să fie predat atât audienților de la profil umanitar, cât şi celor de la profil real. Trebuie să fie predat acest curs şi în instituțiile de învățământ superior. Despre necesitatea studierii bazelor programării logice şi introducerea unui compartiment ce ține de tema dată în cursul şcolar de informatică, predat în şcolile din Republica Moldova, prin revizuirea lui capitală, s-a discutat la diverse niveluri, începând cu anul 1990 [13]. Sunt unele succese în acest sens. În prezent, programarea logică, limbajul Prolog este predat în mai multe instituții de învățământ superior din Republica Moldova, dar la nivelul studiilor liceale rămâne până în prezent fără schimbări.

În [4], teoriei programării logice, programelor scrise în logică li s-a acordat atenție suficientă. Programul logic, care reprezintă o mulțime de clause Horn şi prin intermediul căruia pot fi descrise situații din lumea reală, este interpretat cu ajutorul interpretatorului logic, a cărui destinație este asigurarea funcționării mecanismului de inferențe.

În continuare, vom analiza elementele teoriei, în baza cărora sunt elaborate asemenea mecanisme.

Elemente de teorie

Noțiuni respective despre forme normale conjuctive (FNC) au fost date în [9]. Verificarea executabilității FNC (adică, dacă cel puțin o dată formula este interpretată ca ***adevăr***, în continuare ***A***) este posibilă, dar nu şi de fiecare dată este efectivă. Totuşi, există o metodă convenabilă pentru depistarea neexecutabilității mulțimilor de disjuncți. Se ştie că mulțimea de disjuncți sunt neexecutabili atunci şi numai atunci, cînd un disjunct vid este concluzia logică din el. Neexecutabilitatea mulțimii **S** poate fi verificată, generând concluzii logice din **S** până nu primim un disjunct vid. Acest mecanism este realizat efectiv cu ajutorul aşa-numitei reguli a rezoluției – unul dintre cele mai importante aspecte ale mecanismului de deducție în programare logică.

Regula rezoluţiei. Să analizăm următoarea schemă. Avem formulele

$$(p \vee q) \text{ şi } (\neg q \vee r).$$

Presupunem că ele sunt adevărate. Atunci, dacă ***q*** este adevărat, şi ***r*** este tot adevărat. Dacă ***q*** este fals, atunci ***p*** este adevărat. Pentru orice valoare a lui ***q*** formula $(p \vee r)$ este adevărată. Să prezentăm regula expusă ca tautologie, semnificată |=, iar formulele prezentate mai sus – în forma de mulţime. Atunci:

$$\{ p \vee q , \neg q \vee r \} \models p \vee r .$$

În cazul când ***q*** este formulă atomară, dar **p** şi **r** sunt disjuncţi, această regulă se numeşte regula rezoluţiei. Expresia $p \vee r$ se numeşte rezolventa formulelor $(p \vee q)$ şi $(\neg q \vee r)$. Metodele de demonstrare a neexecutabilităţii formulelor bazate pe metoda rezoluţiei dau posibilitatea de a utiliza metode de demonstrare automată, aplicate în programarea logică.

Proprietăţile de finalizare a metodei rezoluţie. Mulţimea finită ***S*** este neexecutabilă atunci şi numai atunci când un disjunct vid poate fi dedus din ***S,*** aplicând metoda rezoluţiei. Disjunctul vid este neexecutabil. El nu poate fi concluzie logică din mulţimea executabilă de disjuncţi. Dacă mulţimea de disjuncţi ***S*** este neexecutabilă şi conţine rezolvente ale elementelor ei, atunci ea conţine obligatoriu un disjunct vid, unicul interpretat prin ***fals***, în continuare – ***F***. Faptul neexecutabilităţii formulei poate fi oricând determinat prin metoda rezoluţie.

Nu există un criteriu suficient de efectiv pentru verificarea executabilităţii FNC. Însă, posibilitatea de constatare că mulţimea de disjuncţi ***S*** este neexecutabilă atunci şi numai atunci când disjunctul vid, indentificat prin ***F,*** este concluzie logică din mulţimea dată, totuşi permite verificarea neexecutabilităţii ***S***, generând concluzii logice din ***S*** până când nu va fi primit disjunctul vid. În aşa mod am ajuns la metoda rezoluţie.

Nu în toate cazurile această metodă este efectivă. Dar există o mulţime specială de disjuncţi, pentru care aplicarea rezoluţiei este destul de efectivă: în cazuri generale – atunci când în fiecare din disjuncţi ai mulţimii se conţine cel mult un literal pozitiv (sau negativ), dar toţi ceilalţi literali în fiecare din disjuncţi conţin semnul opus. Aşa disjunct este numit disjunctul Horn.

Algoritmul de verificare a exacutabilităţii mulţimilor de disjuncţi Horn. Dacă mulţimea de disjuncţi Horn ***S*** nu conţine tautologii, atunci algoritmul de verificare este analogic algoritmului de verificare pentru FNC. Dacă $F \notin S$ şi ***p*** este disjunct unitar pozitiv din ***S***, dar ***c*** – disjunct din ***S***, care conţine $\neg p$, atunci calculăm rezolventa ***r*** şi înlocuim ***S*** cu $(S \setminus \{c\}) \cup \{r\}$.

Şi totuşi diferenţă există. Aici, la fiecare etapă, un anumit literal este eliminat dintr-un disjunct [3, p.46-47]. Rezolventa **r** nu este altceva decât disjunctul ***c***, din care a fost eliminat literalul $\neg p$. Prin urmare, executarea algoritmului se finalizează în toate cazurile, independent de strategia acceptată la selectarea **p** şi **c**. Dacă ***N*** este numărul literalilor, prezente iniţial în ***S*** (ţinînd cont de repetari), atunci ciclul va fi repetat nu mai mult de N ori.

Algoritmul poate fi finalizat cu generarea disjunctului vid ori cu obţinerea mulţimii ***S,*** care deja nu conţine disjuncţi identici cu ***p*** şi ***c.*** Prima finalizare înseamnă neexecutabilitatea mulţimii ***S***, a doua – mulţimea ***S*** obligatoriu este executabilă. Oricare mulţime finită şi executabilă de disjuncţi Horn permite unul şi numai un model minimal [3, p.48-49]. După cum ştim, o formulă dispune de un model în cazul când această formulă poate fi interpretată prin valoarea ***A***. Determinarea modelului minimal înseamnă că se numesc adevărate numai faptele (propoziţiile) formulate evident şi concluziile logice, primite din ele conform regulilor. Această situaţie corespunde ipotezei „lumii închise", care este utilizată pe larg în baze de date.

Regula rezoluţiei în logica predicatelor. Regula rezoluţiei în logica predicatelor lucrează în modul urmator [1, p. 55-56]. Din doi disjuncţi poate fi dedusă rezolventa, dacă unul din ei conţine un literal pozitiv, iar altul – acelaşi literal, dar negativ. Numele constantei predicative, precum şi numărul de termeni (argumente) ai literalului negativ corespund întocmai celui pozitiv şi argumentele ambilor literali pot fi unificate unele cu altele.

Să analizăm două fraze:

$$P(a) \vee \neg Q(b,c)$$

$$Q(b,c) \vee R(b,c)$$

În prima frază se conţine literalul negativ $\neg Q(b,c)$, iar în fraza a doua – literalul pozitiv $Q(b,c)$. Argumentele ambilor literali pot fi unificate, adică ***b*** se unifică cu ***b*** şi ***c*** se unifică cu ***c***. Prin urmare, din aceşti disjuncţi poate fi dedusă rezolventa. În rezultat primim al treilea disjunct: $P(a) \vee R(b,c)$. Acest disjunct se numeşte rezolventă şi se include în mulţimea de disjuncţi, în care până la momentul dat intrau numai primii doi disjuncţi. Pentru executarea următoarelor rezoluţii ne putem folosi de oricare din aceşti disjuncţi.

Unificarea. Esența unificării constă în următoarele. Doi atomi pot fi unificați dacă:

- sunt formați dintr-o constantă predicativă, aplicată la perechile de termeni unificați (aici este suficientă examinarea unificării a doi termeni);
- unul din ei este variabilă, în acest caz are loc unificarea imediată;
- ambii termeni sunt construiți (alcătuiți) din una şi aceeaşi constantă funcțională, aplicată la termenii unificați în pereche.

Primim algoritmul (recursiv) de unificare – este foarte simplu şi de dificultate liniară în ce priveşte numărul de termeni.

Dacă unul din termeni este variabila ***x***, iar altul conține ***x***, dar nu se rezumă (reduce) la ***x***, atunci în acest caz concret unificarea este imposibilă. Verificarea sistematică a neintrării unei variabile în termen, ce urmează a fi supus unificării cu această variabilă, aduce la un algoritm neefectiv. Totuşi, se pot găsi mijloace pentru soluționarea problemei în cauză. În programarea logică verificarea neintrării deseori se omite.

Folosind unificarea, algoritmul rezoluției poate fi răspândit (răsfrâns) şi la calculul predicatelor.

Unificarea unei variabile cu o constantă. Dacă în una din fraze, în calitate de argument avem o variabilă, atunci ea este unificabilă cu constanta respectivă a altei fraze, dar rezolventa va conține această constantă pe locul unde variabila analizată se află în fraza inițială. De exemplu, frazele:

$$P(a,b) \vee \neg Q(b,c)$$

$$Q(x,y) \vee R(x,y)$$

sunt rezolvente, deoarece constanta predicativă ***Q*** este legată cu un număr egal de argumente în ambele fraze. Totodată, variabila ***x*** se unifică cu constanta ***b***, iar variabila ***y*** se unifică cu constanta ***c***. În rezolventa primită

$$P(a,b) \vee R(b,c)$$

variabilele, care servesc drept argumente pentru **R**, sunt substituite de constante.

Principiul programării logice. Dacă folosim o strategie potrivită de alegere, atunci metoda rezoluției devine un mijloc bun pentru verificarea executabilității mulțimilor de disjuncți Horn în logica predicatelor. Principiul programării logice constă în faptul că calitățile (proprietățile) algoritmice ale unei funcții pot fi prezentate ca o mulțime de disjuncți şi poate fi folosită metoda rezoluției pentru calcularea valorilor acestei funcții. Deducția automată a soluțiilor ce se bazează pe principiul rezoluției poate executa algoritmul descris prin mulțimile de disjuncți Horn. Principiul rezoluției este baza funcționării limbajului de programare Prolog.

Strategia descendentă de rezolvare a problemelor prin utilizarea regulii rezoluției

La utilizarea regulii rezoluției se selectează diverse strategii de rezolvare a problemelor. Să examinăm una din ele, numită descendentă, sau strategie reversă [1, p.57-59]. În această strategie se stabileşte scopul de a identifica dacă fraza unică ***P*** este consecința mulțimii existente de fraze ***Q.*** Această strategie a primit denumirea de descendentă, din cauza ca procesul de rezolvare a problemei (prima rezoluție) se începe cu negarea concluziei (disjunctul scopului) şi apoi această negație se unifică (se compară) cu faptul sau regula (disjunctul Horn unitar sau exact). Rezolventa primită trebuie să participe în următoarea rezoluție, şi aceasta se repetă până la deducerea disjunctului vid. Această strategie este numită „căutare în adâncime", din cauza că rezultatul ultimei rezoluții se utilizează permanent în rezoluția ce urmează.

Exemplu de rezoluție descedentă. Fie o mulțime de fraze (disjuncți) [1, p. 58-59]:

1) $p(a) \vee \neg q(a, b)$;
2) $q(x, y) \vee \neg r(x, y)$;
3) $s(b)$;
4) $r(a, b)$.

Se cere să determinăm dacă fraza ***p(a)*** este consecința mulțimilor de fraze existente. Aşadar, adăugăm la primele patru fraze negația frazei ***p(a),*** care în gramatica contextual-liberă se înscrie în urmatorul mod:

5) $\neg p(a)$.

În continuare, executam prima rezoluție cu utilizarea obligatorie a frazei nou-adăugate cu numărul 5. Se unifică iarăşi prima frază. Primim rezolventa:

6) $\neg q(a, b)$.

Apoi, unificăm rezolventa 6 cu fraza sub numărul 2.

Primim următoarea rezolventă:

7) ***¬ r (a, b).***

Unficăm 7 cu 4 (fraza 3 nu se unifică). Obţinem o frază vidă, ceea ce înseamnă că a fost depistată o contrazicere. Deoarece fraza ***¬ p(a)*** adăugată la mulţimea de fraze aduce la contrazicere, atunci ***p(a)*** este consecinţa mulţimilor de fraze.

Eficienţa strategiei de rezolvare. Principiul rezoluţiei reprezintă un interes deosebit, deoarece se află în legătură reciprocă nemijlocit cu problema de „demonstrare automată". Totuşi, căutarea strategiei eficiente de rezolvare pentru toate domeniile de aplicare aduce la concluzia că: pentru un set dat de fraze, demonstrarea la calculator se face ori foarte repede, ori nu se poate obţine în general [8, p.173-174]. Eficienţa programelor se micşorează odată cu creşterea numărului de fraze. Problema se reduce la determinarea momentului când trebuie să fie stopat programul, deoarece în acest caz apar două riscuri:

1) creşterea enormă a numărului de fraze;
2) riscul de întrerupere pretimpurie a executării programului, nemijlocit în momentul obţinerii soluţiei.

Interpretarea programelor logice

Interpretatorul abstract al programelor logice. Interpretatorul abstract, descris în [7, p.21-23], execută calcule ce conţin răspunsurile *[A,F]*. El primeşte la intrare programul ***P*** şi întrebarea de bază ***Q,*** apoi oferă răspunsul ***A***, dacă ***Q*** se deduce din ***P***, şi răspunsul ***F*** în caz contrar. Dacă scopul ***Q*** nu se deduce, atunci interpretatorul nu va finaliza lucrul şi nu va oferi nici un răspuns.

Demonstrarea lucrului interpretatorului:

La întrare: ***Întrebarea de bază Q şi programul P.***
Rezultat: ***Adevăr, dacă întrebarea (afirmaţie-scop) Q să deduce din programul P.***
Fals în caz contrar.
Algoritm: ***Fie rezolventa este egală cu Q.***
Dacă Q nu este vidă, atunci din Q şi elementul din P se deduce rezolventa Q_1.
Dacă rezolventa obţinută Q_1 nu este vidă, atunci din Q_1 si
următorul element din P se deduce rezolventa Q_2 şi aşa mai departe,
până nu se va analiza fiecare din elementele existente în P.
Dacă rezolventa Q_k este vidă, atunci rezultatul este A (adevăr),
în caz contrar rezultatul este F (fals).

Mecanisme de gestionare, implementate în interpretatorul programelor logice. Conducerea executării programelor logice se realizează prin mecanisme de gestionare, implementate în interpretator. Programatorul poate influenţa mersul executării programului, stabilind, spre exemplu, consecutivitatea calculelor.

Programele logice sunt nedeterminate, adică lipsesc faptele care determină exact procesul executării programului. Citcumstanţele date obligă interpretatorul să realizeze căutarea pentru a nu omite nici o soluţie. Este necesară o strategie standard de căutare – arborele de căutare, pe care interpretatorul îl formează pentru ca, având programul, să execute calculări concrete. Arborele de căutare se conturează cu metoda „căutare în profunzime". Mecanismul de căutare este prezentat în figura de mai jos.

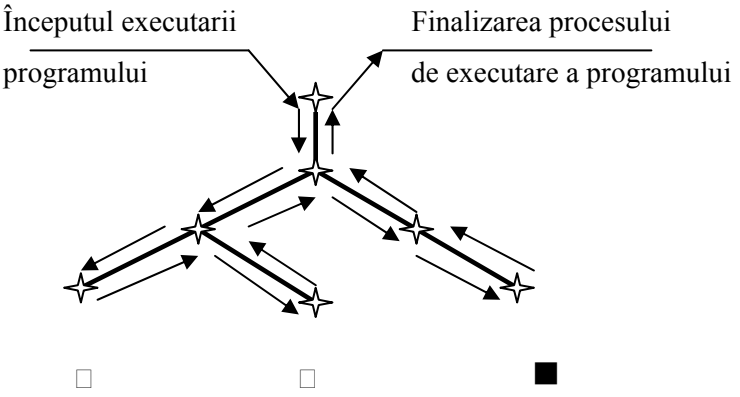

Fig.1. Căutare în profunzime.

Interpretatorul formează calcule, începând cu nodul rădăcinii arborelui, apoi avansează în jos de la stânga la dreapta, pe ramura ce creşte din rădăcină, până când nu atinge următorul nod marginal (până la obținerea disjunctului vid □ sau a nodului final ■). Dacă se atinge un anumit (nod) □, atunci interpretatorul poate extrage răspunsul şi îl poate comunica dispozitivului de imprimare a informației. Despre interpretatorii care sunt gestionați de o aşa strategie standard se spune că ei realizează cautarea „de la stângă la dreapta în profunzime cu revenire".

Operatorul de tăiere completează strategia standard pentru evitarea calculelor nedorite, motiv din care interpretatorul nu poate folosi pe deplin rezoluția [4, p.73].

În unii interpretatori există mijloace pentru „verificarea blocărilor", care încearcă să recunoască calculele infinite.

Executarea programelor de către interpretatorul logic. Interpretatorul este un program, care este capabil să formeze concluzii rezolutive, de regulă, prin metoda „de sus în jos, de la stânga la dreapta". Executarea programului logic începe după introducerea lui la intrarea interpretatorului logic [4, p.55]. Primind programul logic la intrare, interpretatorul inițiază paşi generali, necesari pentru executarea programului în regimul de interpretare:

1) verifică sintaxa afirmațiilor de intrare;

2) le păstrează în memoria centrală în formă simplificată, compactă şi accesibilă;

3) traduce afirmația-scop de intrare în reprezentare internă şi apoi începe procesul de formare a deducției prin aplicarea consecutivă a regulii rezoluției la afirmația-scop curentă (întrebare) şi la o procedură părintească (la fapt sau regulă), selectată din programul de intrare ce se păstrează în memorie.

Dacă interpretatorul reuşeşte să deducă negație vidă, care înseamnă că soluția este obținută, atunci el emite un mesaj despre aceasta împreuna cu valorile obținute ale variabilelor-scop.

Executarea programelor în regim de interpretare. Executarea programelor în regim de interpretare se inițializează şi se gestionează de interpretator [4, p.227]. Programul-interpretator realizează accesul la două domenii de date importante, amplasate în memoria calculatorului:

1) Domeniu de date static (sau mulțimea de date de intrare), în care se conține versiunea codificată a programului logic de intrare, care nu se modifică pe întreg parcursul de executare;

2) Domeniu dinamic (sau steckul de executare), care se foloseşte de interpretator pentru întocmirea protocolului operațiunilor personale.

În steck sunt prezentate starea de gestionare cu executarea programului de catre interpretator şi starea datelor (căror variabile şi ce valori au fost atribuite la momentul dat).

Ipoteza lumii închise. Ipoteza lumii închise este implementată în mecanismul de interpretare a programului de intrare, scrise în limbajul Prolog [1, p.70-71]. Prologul acționează ca şi cum mulțimea de fraze ale programului curent este unica sursă de cunoştințe. Dacă apelul este un insucces, aceasta înseamnă că interpretatorul nu a putut realiza demonstrarea lui prin mulțimea de fraze ce formează programul curent.

Interpretatorul limbajului Prolog reiese din ipoteza că dacă respectarea unui caz concret de relație nu poate fi demonstrat, atunci acest caz de relație este fals. Aici nu se fac deosebiri între relații necunoscute şi relația a cărei falsitate poate fi demonstrată.

«Prologul pur». Programul scris în limbajul «Prolog pur» este un program logic, în care este definită ordinea frazelor şi afirmațiilor-scop. Interpretatorul abstract este format astfel ca să folosească informația despre ordinele predefinite [7, p.80-81].

Programele logice, executate prin intermediul modelului de calcul Prolog, se numesc programe scrise în limbajul «Prologul pur». «Prologul pur» este o realizare aproximativă a modelului de calcul în programarea logică prin intermediul maşinii consecutive.

Aspecte metodologice de studiere a programării logice

După cum putem constata din cele expuse mai sus, programarea logică ca teorie este destul de simplă. Însă, după cum demonstrează practica de predare a cursului de programare logică în majoritatea universităților din Republica Moldova şi România, dar şi predarea lui la cursurile corespunzătoare, în afara programelor universitare [13], înțelegerea materiei este lentă, deprinderile dobândite pentru programarea practică în limbajul Prolog nu sunt destul de mari. După părerea noastră, cauza este, în primul rând, metoda incorectă de predare

a cursului de informatica în şcoli. Cum se menţionează în [5], cel mai bun rezultat se obţine prin studierea iniţială a bazelor şi tehnicii programării logice (declarativă), şi numai după aceasta să se urmeze studierea programării algoritmice (procedurale). Însă, din punct de vedere istoric, s-a creat o altă situaţie, şi acum apare necesitatea de a găsi metode efective de predare, precum şi variante de realizare tehnică, care ar contribui la diminuarea «efectului algoritmic» în gândirea studentului, ar spori succesele la începători, ar oferi noi posibilităţi atât în studierea tradiţională, individuală, cât şi în studierea la distanţă prin Internet.

Am realizat un şir de metode şi mijloace care permit eficientizarea metodei de predare, în primul rând a cursurilor ce ţin de teoria şi practica programării logice. Este vorba, în primul rând, de sistemul SPprolog, descris mai pe larg în [9], metoda de structurizare a cunoştinţelor cu ajutorul aşa-numitei «matrice a elementelor de cunoştinţe» [11,12], şi de sistemul UŞAM SOFT «QUEST», destinat pentru instruire şi autoinstruire, controlul şi autocontrolul cunoştinţelor. Aici vom enumera doar unele date despre sistemului SPprolog.

SPprolog este un sistem complex, care permite realizarea aplicaţiilor soft în limbajul de programare logică Prolog, însoţit de un sistem intelectual de instruire ce asigură atât însuşirea lucrului cu sistemul, cât şi a limbajului de programare logică.

Sistemul intelectual de instruire este un complex de mijloace şi măsuri, care includ:

- Un interpretator multimedia-interactiv, intelectual, al limbajului de programare logică, care permite să se efectueze, simplu şi accesibil, interpretarea vizuală şi detaliată a programului, să se evidenţieze în mod grafic fragmente corespunzătoare din program, ceea ce asigură realizarea procedurilor concrete şi a mecanismelor (backtraking, reguli recursive, aritmetice, logice şi alte operaţii ale limbajului de programare logică), în acelaşi rând permite înzestrarea fragmentelor selectate cu explicări textuale necesare despre esenţa proceselor care au loc în ele, utilizând efectele audiovizuale pentru ridicarea nivelului de percepere a materialului.
- regimul automatizat «Sfatul Expertului», întemeiat pe cunoştinţe expert despre programarea «culturală», efectivă şi optimală, de asemenea pe cunoştinţe despre limbaj şi teoria pe care se bazează acest limbaj de programare logică, despre problemele principale ce apar la studierea limbajului. În timpul scrierii programului, regimul dat va «informa» programatorul despre erorile apărute sau va indica variantele de scriere a unui cod mai efectiv; va «propune» lista predicatelor accesibile ce pot fi utilizate în blocurile corespunzătoare ale programului; va «propune» o variantă tipică de rezolvare a unei probleme şi va «explica» principul de lucru al programului realizat în baza acestei variante. În cazul unor dificultăţi, regimul «Sfatul Expertului» va identifica procedura sau mecanismul corespunzător, conform cărora programatorul nu dispune de destulă claritate şi va «propune» pentru consultare compartimentul respectiv din manualul electronic;
- cursul electronic al limbajului de programare Prolog, prezentat în formă de «matrice a elementelor de cunoştinţe» [12] şi înzestrat cu un sistem de reproducere succesivă (programată de profesor) a cunoştinţelor pentru studierea individuală şi control;
- un curs avansat de studiere a limbajului şi sistemului SPprolog, bazat pe o culegere de acţiuni succesive (actions), programată, ce permite, prin utilizarea a celor două mecanisme precedente ale sistemului, să explice simplu şi clar principiile de lucru al sistemului şi al limbajului Prolog.

Una dintre particularităţile de bază ale sistemului SPprolog este un mediu funcţional lărgit (spre deosebire de sistemele existente) de elaborări Soft în limbajul Prolog cu un redactor de cod multifuncţional şi un sistem automatizat de organizare a elementelor personale de programare (toolbox).

Sistemul SPprolog poate fi utilizat de cei ce manifestă interes faţă de programarea logică: în instruire şi autoinstruire, în instruirea tradiţională şi la distanţă, de asemenea în programarea de zi cu zi, ca formă profesională de activitate. Sistemul poate fi utilizat în diverse instituţii de învăţământ (şcoli, universităţi, cursuri specializate): ca un mediu de elaborare a aplicaţiilor şi înţelegere evidentă a rezultatelor programului creat, ca un mediu experimental la studierea unora din mecanismele limbajului; ca mijloc tehnic de instruire – pentru profesori în procesul de explicare a diverselor tipuri de mecanisme ori în procesul de explicare a principiului de lucru al unui program.

Se presupune că succesul de bază al sistemului în cauză trebuie să se manifeste anume în procesul de autoinstruire a limbajului de programare logică, deoarece sistemul de sine stătător va „descrie" şi va „explica" tema nouă în baza planului introdus (acţiunile programate); va oferi exemplele necesare, va explica detaliat şi evident principiile lor de lucru, iar în procesul de alcătuire a programului va înlocui „tutela" profesorul cu sfaturi preţioase şi moderne extrase din sistemul expert.

Concluzii

În articol se face analiza elementelor teoriei, în baza căreia este construit mecanismul de deducţie logică implementat în interpretatorii limbajului de programare Prolog. În baza studierii literaturii şi a rezultatelor cercetărilor efectuate se conturează opinia despre necesitatea de revizuire a programei de studii la cursul de informatică în licee şi instituţii de învăţământ superior în scopul introducerii unui compartiment respectiv, consacrat programării declarative şi programării în logică, în particular. Se descriu caracteristicile de bază ale SPprolog, ale unui sistem complex elaborat de unul autori [10], care permite proiectarea programelor soft în limbajul de programare Prolog la asistarea sistemului intelectual de instruire ce asigură instruirea şi auto-instruirea atât a lucrului cu sistemul, cât şi a limbajului de programare logică.

Referinţe:

1. Малпас Дж. Реляционный язык Пролог и его применение / Пер. с англ. Под. ред. В.Н. Соболева. - Москва: Наука, 1990. - 464 с.
2. Ковальски Р. Логика в решении проблем // Проблемы искусственного интеллекта / Пер. с англ. - Москва: Наука, 1990. - 280 с.
3. 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.
4. Хоггер К. Введение в логическое программирование / Пер. с англ. - Москва: Мир, 1998. - 348 с.
5. Колмероэ А., Кануи А., М. ван Канегем. Пролог – теоретические основы и современное развитие // Логическое программирование. - Москва: Мир, 1988, с.27-133.
6. Лорьер Ж.-Л. Системы искусственного интеллекта / Пер. с франц. - Москва: Мир, 1991. - 568 с.
7. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. - Москва: Мир, 1987.
8. Алексеев М. Совершенствование методики построения образовательного ВЕБ-сайта: Автореферат диссертации на соискание ученой степени кандидата педагогических наук. - Москва, 2001.
9. Pelin N., Pelin S. Aspecte teoretice ale programării logice // Studia Universitatis. - 2008. - Nr.3(13). Seria „Ştiinţe exacte şi economice”.
10. Pelin S. Work optimisator and intellectual assistant for Prolog development environment. 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.185-190.
11. 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.
12. Pelin N., Pelin S., Pelin A. The problems of knowledge structure defining, activation an usage. The Proceedings of the EICHT International Conference on Informatics in Economy. Informatics in Knowledge Society. Bucharest, may 17-18, 2007, ISBN 978-973-594-921-1, p.749-755.
13. Pelin N. Studierea programării logice în Moldova. Analele ştiinţifice ale doctoranzilor şi competitorilor: Probleme actuale ale ştiinţelor umanistice. Vol. I. - Chişinău: Universitatea Pedagogica de Stat „Ion Creanga”, 1996, p.69-73.

Prezentat la 18.03.2008