

METALIMBAJUL ÎN SISTEMUL LIMBAJELOR DE PROGRAMARE

Olimpia VARGA
Universitatea „Ovidius” din Constanța
olimpia_varga@yahoo.com

Abstract: In this paper we try to clarify the mechanisms of programming languages starting from the general concepts, the alphabet and vocabulary, syntax and the semantics of artificial languages compared to natural languages. Within programming languages we analyze metalanguage elements, the combinations of letters, numbers, word categories, the set of symbols and the commands used in the program.

Keywords: programming language, semantics, syntax, symbol, metalanguage

Un specialist în informatică sau un filolog specializat în lingvistică computațională are imaginea mecanismelor de funcționare a limbajelor de programare, după modelul limbilor naturale și după cum le este foarte familiar și metalimbajul aferent.

Pentru nespecialiști, cele două tipuri de sisteme¹ – natural și artificial –, precum și terminologia „gramaticii” după care se organizează fiecare nu își dezvăluie specificul și modelele de autoreglare chiar atât de ușor. Ceea ce obturează suplimentar imaginea reluată a limbajelor, de la natural la artificial, este accepția oarecum particulară a terminologiei de specialitate.

Dacă termeni precum „semantică”, „sintaxă” etc. sunt relativ unitari în interpretarea ambelor tipuri de sisteme, există alți termeni, precum „dicționar”, „bibliotecă”, „limbaje de interogare”² etc. cu înțelesuri speciale în limbajele generate de inteligența artificială. În plus, aceasta din urmă își dezvoltă limbajele într-un ritm extrem de rapid, procesul de simplificare a procedurilor de analiză fiind invers proporțional cu complexitatea operațiilor logice și comunicaționale pe care sunt capabile să le realizeze.

Limbajul artificial este un sistem de semne nonlingvistice, clare, care pot fi înțelese pe baza unor convenții, la fel ca sistemul de semne dintr-o limbă naturală. Acesta este motivul pentru care a fost „creat în mod

¹ Sistemul este definit ca fiind compus dintr-o sintaxă și o semantică, sintaxa derivând formele de suprafață produse de regulile de bună formare, iar semantica - o formă logică dedusă prin intermediul regulilor de compunere. Ansamblul constituit dintr-o formă de suprafață și o formă logică constituie *semnificația frazei* (prin opoziție cu *sensul enunțului*). Cf. Jacques Moeschler, Arme Reboul, 1999, *Dicționar Enciclopedic de Pragmatică*. Coordonarea traducerii și prefața: Carmen Vlad, Liana Pop, Cluj-Napoca: Editura Echinox, p. 21.

² Interogarea este operația prin care se obțin datele dorite dintr-o baza de date, selectate conform unui anumit criteriu. Întrucât operația de interogare este cea mai importantă operație de manevrare a datelor, de multe ori limbajele de manevrare a datelor sunt denumite ‘limbaje de interogare’. Deci, limbajul de interogare este un limbaj de programare specific pentru manipularea datelor în cadrul bazelor de date relaționale.

intenționat, de unul sau mai mulți indivizi pentru a servi la realizarea comunicării într-unul sau altul din domeniile de activitate umană”³, după anumite reguli, în vederea fixării, prelucrării și transmiterii de informații. După cum se vede, această definiție a limbajului este valabilă atât pentru *limbajele naturale*, adică limbile formate în procesul comunicării sociale, cât și pentru *limbajele artificiale*, construite de om în procesul cunoașterii științifice.

Sistemul de semne este guvernat de trei tipuri de reguli:

a) *Reguli sintactice*, care vizează legăturile dintre semne.

Exemplu: *regulile legăturilor dintre diversele părți de propoziție*.

b) *Reguli semantice*, care vizează legătura dintre semne și semnificațiile lor.

Exemplu: *regulile de traducere*.

c) *Reguli pragmatice*, care stabilesc normele de utilizare a semnelor de către oameni.

Limbajul formal este „limbajul artificial cu o descriere riguroasă, matematică, bazat pe un sistem formal de tip gramatică sau automat, ce pot fi folosite ca modele ale limbajelor de programare.”⁴

Limbajul mașină este „limbajul programelor în format direct executabil al unui calculator, constituind cel mai de jos nivel pe care poate fi programat un calculator. Un program în limbajul mașină este o *secvență de instrucțiuni* și *zone de date*, care pot fi imediat executate de unitatea centrală de prelucrare. Pentru reprezentările externe ale calculatorului, un program în limbajul mașină poate fi dat ca șir de cifre binare, organizate pe locații ale memoriei, folosind denumiri simbolice ale instrucțiunilor.”⁵

Limbile artificiale sunt limbi a căror vocabular și gramatică au fost inventate de către un om sau un grup, spre deosebire de limbile naturale, apărute în decursul istoriei.

Limbile artificiale pot fi clasificate în:

Limbi auxiliare inventate pentru a fi folosite în comunicarea dintre oameni pe tot globul. Exemplu: *limba Esperanto*;

Limbi fictive plătuite în opere artistice, întâlnite în literatură sau film cu personaje imaginare;

Limbi experimentale create de lingviști ca metodă de cercetare;

Limbi secrete folosite pentru codificarea datelor;

Limbi inventate doar pentru amuzament. Exemplu: *limba „pășărească”*.

Noțiuni generale privind limbajele de programare

Calculatoarele electronice și „limbajele de programare” joacă un rol foarte important în dialogul om-calculator.

³ Cf. *Dicționar de Informatică*, 1981, București: Editura Științifică și Enciclopedică, p. 193.

⁴ *Ibidem*, p. 195.

⁵ *Ibidem*.

Limbașele de programare fac parte din setul de limbașe artificiale create de om, care servesc la exprimarea, cu ajutorul calculatorului, a algoritmului de rezolvare a unei probleme, folosind o serie de comenzi executabile. Algoritmul indică modul de prelucrare, pas cu pas, a datelor primare până la obținerea rezultatelor finale.

Un *program* este reprezentarea unui algoritm într-un limbaj de programare și poate fi privit ca un *transformator de aserțiuni* care descrie proprietățile datelor de intrare și a rezultatelor.⁶

Un *limbaj de programare* este un set bine definit de caractere și simboluri, de expresii și reguli sintactico-semantice privind comenzile folosite pentru scrierea unui program pe un computer. Limbajul de programare este un instrument de dialog, interfața între om și calculator.

Limbașele de programare servesc la transformarea modului de rezolvare a unei probleme într-un format accesibil calculatorului. Folosind un limbaj de programare, specialistul în informatică va crea un *soft*, care va descrie, în liniile de comandă, *calea de urmat* pentru rezolvarea unei probleme.

Limbașele de programare prin *biblioteca de comenzi*⁷, pe care o oferă, dau posibilitate programatorului să specifice în programul sursă, procedurile pe care trebuie să le execute calculatorul în derularea *soft*-ului.

Toate limbașele de programare se bazează pe un set de simboluri elementare, numit *alfabetul limbajului* cu literele mari și mici ale alfabetului latin, cu cifrele sistemului zecimal, cu anumite caractere speciale: + - * /, %....

Aceste simboluri sunt asamblate în cuvinte-cheie sau expresii care formează *vocabularul limbajului* (variabile, constante, instrucțiuni, funcții, comenzi). Ansamblul regulilor prin care se construiesc instrucțiunile constituie *gramatica limbajului*.

Exprimarea regulilor gramaticale din limbașele de programare se realizează cu ajutorul unui *metalimbaj*. Elementele de metalimbaj care apar în manualul de utilizare al produsului-program sunt:

- Cuvintele rezervate,⁸ scrise cu majuscule și folosite sub numele de *comenzi, clauze, funcții*.

Exemplu: *IF(...), FOR, COPY, PRINT*.

- Cuvintele-utilizator, scrise cu minuscule și folosite de programatori în declarațiile de variabile și în stabilirea câmpurilor din structura unică a unui fișier.

Exemplu: *numepren (nume prenume), cnp (codul numeric personal), tel (telefon)*.

- Parantezele drepte „[]” încadrează o declarație opțională.

⁶ *Ibidem*, p. 266.

⁷ Reamintim că în metalimbajul programatic, termenul de „bibliotecă” nu coincide cu cel din lexicul uzual, însemnând „corpus”, „inventar” de termeni, de texte etc.

⁸ Prin „cuvânt rezervat” se înțelege un șir de caractere alfanumerice, care constituie un simbol cu semnificație particulară într-un limbaj de programare și exprimă o comandă, o clauză etc., fără a putea fi folosit ca identificator al datelor. Cf. DI, 1981, p. 109.

Exemplu:

Comanda *SORT TO* permite sortarea, ordonarea datelor într-un tabel.

```
SORT TO <fișier> ON <câmp1> [/A /D] [/C]  
[, <câmp2>[/A /D] [/C] ...]
```

În sintaxa comenzii de sortare, datele din tabelul numit <fișier> vor fi aranjate după primul câmp <câmp1>, apoi după cel de-al doilea câmp <câmp2>, din structura de fișier, în ordine ascendentă/descendentă [/A/D], sau în ordine alfabetică [/C], după literele mari sau mici ale alfabetului.

- Acoladele „{ }” sau linia verticală „|”, în liniile de comandă din cadrul fișierelor program sunt alte elemente ale limbajului. Pentru o cât mai mare claritate, vom folosi un singur element: sau „{ }”, sau “|”.

Exemplu: *Ordonare ascendentă sau descendentă a datelor:*

```
[ASCENDING / DESCENDING]
```

Programul ⁹ pentru calculator reprezintă un ansamblu de instrucțiuni scrise cu ajutorul unui limbaj de programare. Succesiunea de instrucțiuni de pe liniile de comandă, dintr-un fișier de tip program, descriu pașii pe care trebuie să-i execute calculatorul în scopul rezolvării unei probleme.

*Instrucțiunea*¹⁰ reprezintă descrierea unei acțiuni codificate, care se transmite calculatorului cu ajutorul unui limbaj de programare. Execuția instrucțiunii, într-un interval de timp finit și cu un efect bine definit, este realizată de către un procesor. Fiecare limbaj de programare conține anumite tipuri de instrucțiuni: de memorare a valorilor, de atribuire, de ciclare, de control, de salt, de transfer de date de intrare/ieșire etc.

În programare, instrucțiunea de transfer prezintă forma de transfer condiționat și de transfer necondiționat, folosind cuvintele-cheie IF și THEN.

Exemplu: „IF condiție, THEN secvență, ELSE secvență”.

„IF condiție, THEN secvență”.

Ansamblul activităților de documentare, proiectare, dezvoltare, implementare și întreținere a programelor în elaborarea unui produs-program poartă numele de *programare*. Programul scris de un analist-programator se numește program-sursă. Fișierul program-sursă (notat *nume-fișier.prg*) trebuie *compilat*, moment în care calculatorul citește liniile de comandă și *rulat*, transformat în format executabil (notat *nume-fișier.exe*). Formatul de tip executabil rezultă prin traducerea sursei, cu ajutorul programelor speciale numite *interpretoare* sau *compilatoare*.

⁹ Cf. *Dicționar de matematică și cibernetică în economie*, București: Editura Științifică și Enciclopedică, 1979, s.v. *Program - Plan al procesului de prelucrare automată a informației la calculatorul electronic*.

¹⁰ Comandă dată unui calculator pentru îndeplinirea unor operații. Cf. Mircea Regneală, 2001, *Dicționar explicativ de biblioteconomie și știința informării*, vol. I. București: FABR, p. 317.

În cazul problemelor complexe, activitatea de creare a programelor presupune mai mult timp și implicarea unui număr mai mare de specialiști, iar rezultatul activității este produsul-program.

Un produs-program-procedural cuprinde ansamblul de programe-procedurale cu fișierele sursă, fișierele executabile, fișierele bază de date și *Manualul de utilizare* cu documentația pentru implementarea și folosirea programului.

Manualul de utilizare accesat cu tasta F1-Help, cuprinde fișierele document cu descrierea algoritmului de rezolvare, modul de implementare și de exploatare al *soft*-ului.

În programul-sursă printre *liniile de comandă* sunt inserate, cu ajutorul semnelor „#”, * (sau alte simboluri), *liniile comentariu*, care nu influențează descrierea structurii datelor de intrare/ieșire, a funcțiilor sau execuția programului.

La primele limbaje de programare, trecerea de la programele sursă la programele executabile se realiza prin comenzi distincte, în care specificam ordinea de efectuare a operațiilor. În momentul actual, programatorii se orientează către *medii de programare*.

Mediile de programare reprezintă *pachete de programe*, care asigură integrarea următoarelor funcții: introducerea și editarea programului sursă, interpretarea sau compilarea, editarea de legături, încărcarea și lansarea în execuție, plus depanarea programului. Astăzi, majoritatea limbajelor de programare sunt integrate în medii de programare având implicit instalate un editor de texte, un interpretor, un depanator de programe, care oferă fișierelor sursă o gestionare completă și o informare rapidă prin sistemul HELP.

Alfabetul și vocabularul unui limbaj de programare

Orice limbaj de programare are la baza două elemente:

a) Un alfabet, alcătuit din literele alfabetului englezesc, de la A la Z, majuscule și minuscule, cu un total de 52 caractere.

b) Un set de caractere, format din cifrele arabe (de la 0 la 9, în total 10 caractere) și caracterele speciale (. , ; = < > # \$ % + - * / " ' ()).

Ordonarea simbolurilor alfabetului se face pe baza codurilor numerice corespunzătoare caracterelor respective. Caracterele din alfabetul limbajului permit compunerea cuvintelor care formează vocabularul limbajului.

În limbajele de programare există următoarele categorii de cuvinte:

a) *Cuvintele cheie*, care au un înțeles clar într-un context precizat și desemnează *nume de variabile, instrucțiuni, funcții-condiție etc.*;

b) *Cuvintele rezervate* ale limbajului sunt folosite doar în scopul pentru care au fost definite. Avantajele utilizării acestei categorii de cuvinte sunt următoarele:

- programul devine mai facil;
- măresc viteza de compilare a programului (citirea liniilor de program);

- erorile sunt depistate mai repede.

c) *Cuvintele definite de utilizator* sunt cuvinte folosite pentru a desemna diverse elemente din program: numele de câmpuri din structura de fișier, numele de funcții, numele de proceduri, variabile, numele de fișiere.

Sintaxa și semantica unui limbaj de programare

Dispozitivele periferice de intrare/ieșire ale unui sistem de calcul permit preluarea, prelucrarea, stocarea, transmiterea, afișarea și tipărirea datelor unui utilizator sau unui grup de utilizatori. Echipamentele periferice folosite pentru introducerea datelor în sistem sunt clasica tastatură, scannerul, creionul electronic etc. Cele mai folosite metode pentru transmiterea datelor sunt: poșta electronică (*e-mail*), *software* de tip *messenger* sau *chat*, *forum*-urile *electronice*, *skype* pentru video-conferințe, rețelele de socializare, programele audio pentru Internet.

Dacă dorim să scriem o aplicație pentru un sistem electronic de calcul trebuie să stabilim limbajul de programare pe care îl vom folosi și, implicit, toate comenzilor din biblioteca de comenzi a limbajului, setul de caractere, modul în care vom realiza analiza sintactică și semantică a textului în fișierul program.

Un program este format din atomi lexicali (*tokens*) și separatori. Un atom lexical este cea mai mică unitate sintactică cu înțeles de sine stătător într-un context precizat.

În programul sursă, analiza lexicală este cea fază a procesului de analiză care are drept scop identificarea atomilor lexicali din care e compus programul. Categoriile de *tokeni* întrebuințați sunt: simbolurile speciale, identificatori, etichete și literali.

Sintaxa unui limbaj de programare

Un limbaj de programare este un set de reguli, simboluri și cuvinte speciale folosite pentru a scrie un program. Regulile sunt valabile atât pentru gramatică (sintaxă), cât și pentru semantică (semnificație).

Cu ajutorul unui limbaj de programare și urmând anumite reguli, vom scrie în fișierul program-sursă liniile de comandă formate din comenzi. O comandă conține numele comenzii cu sintaxa comenzii alcătuită din cuvinte combinate, simboluri și parametri.

Sintaxa unui limbaj de programare reprezintă un set de reguli care guvernează alcătuirea propozițiilor dintr-un limbaj ¹¹. Prin sintaxă determinăm dacă o anumită instrucțiune este scrisă corect sau nu, în privința asocierii enunțurilor sau părților de enunț.

Sintaxa limbajelor de programare este foarte *rigidă*, nu acceptă ambiguități și este alcătuită dintr-un set de reguli care arată exact ce combinații de litere, numere și simboluri pot fi folosite în program. Remarcăm aici o pregnantă diferență față de sintaxa limbilor naturale, unde

¹¹ Cf. DI, 1981, p. 312.

părțile de propoziție sau propozițiile subordonate se pot suprima, permuta etc., desigur, între anumite limite și după anumite reguli, specifice fiecărei limbi.

Exemplu:

Eroarea de sintaxă (*syntax errors*) este generată la orice scriere incorectă și incompletă a unei comenzi, a unui cuvânt, a unei virgule, a unui parametru de comandă.

Sintaxa unui limbaj de programare poate fi descrisă în diverse moduri, unul dintre acestea fiind notația BNF (Backus-Naur Form).

Notația BNF a fost utilizată prima dată la descrierea sintaxei limbajului ALGOL (în cadrul raportului ALGOL60, apărut în 1963). În cadrul BNF, sunt folosite metasimboluri, simboluri terminale și simboluri neterminale.

Metasimbolurile sunt simbolurile care fac parte din mecanismul de descriere a limbajului.

Exemple de metasimboluri: „<”, „>”, „1/2”, „::=”

Simbolul „1/2” semnifică o alternativă.

Simbolul „::=” îl vom citi „se definește astfel”.

Simbolurile terminale sunt cuvintele care desemnează comenzile-condiție în proceduri sau în buclele de program.

Exemplu: *FOR*, *WHILE*, *DO*.

Simbolurile neterminale sunt cuvintele încadrate între semnul mai mic „<” și semnul mai mare „>” fiind definite în program.

Exemplu: <variabila>, <identificator>, <instrucțiune>.

Sintaxa unui identificator în BNF se prezintă în trei moduri alternative, folosind semnele mai mic și mai mare.

- Un identificator este fie o <litera>, fie un <identificator> urmat de o <cifra>, fie un <identificator> urmat de o <litera>.

Exemplu:

<identificator> ::= <litera>1/2<identificator><cifra>1/2<identificator><litera>

Unde: <litera> ::= a1/2b1/2...1/2z1/2A1/2B1/2...1/2Z

<cifra> ::= 01/211/221/2...1/29

- Rezultă că un identificator poate să se identifice cu: o singură literă „a”; o literă urmată de o cifră „b2”; multe litere și cifre „alo2a3”.

- Descrierea sintaxei instrucțiunii condiționale *IF- THEN* în notația BNF.

Exemplu: <instrucțiune IF> ::= IF <condiție > THEN.

Semantica unui limbaj de programare

Semantica s-a dezvoltat ca ramură de sine stătătoare a lingvisticii abia în secolul al XX-lea. Termenul de „semantică”, introdus în literatura de

specialitate de către Michel Bréal¹², în 1883, a fost folosit în înțelesul actual începând cu lucrările lui Charles William Morris¹³. În cadrul semanticii, el a cercetat în special legăturile dintre semnele verbale și ceea ce este înțeles prin ele, deosebind-o de sintaxă, care studiază relațiile semnelor între ele, și deosebind-o, de asemenea, de pragmatică, disciplină care se ocupă cu studiul relațiilor dintre semne și utilizatorul lor. Lingvistica structuralistă a preluat această diferențiere și a dezvoltat mai departe studiul semnificației simbolurilor verbale. În cadrul semanticii, studiul semnificației cuvintelor se numește *semasiologie*, iar studiul denumirilor, *onomasiologie*, termeni care se regăsesc numai parțial în limbajele de programare.

- *Sensul* din aceste limbaje artificiale se referă la legăturile dintre cuvintele, semnele și parametri unei comenzi.
- *Semnificația* cuvintelor se referă la relația dintre semne cu întreg programul.

O preocupare importantă a semanticii, tradițională, de altfel, o constituie transformarea în formule logice a exprimărilor naturale într-o anumită limbă, după o metodă care a fost dezvoltată de Richard Montague. Spre deosebire de fonetică, morfologie și sintaxă, obiectul de studiu al semanticii „... *est l'étude du sens des mots, des phrases et des énoncés*”¹⁴.

Domeniile de cercetare ale semanticii din limbajele de programare coincid, în bună parte, cu cele din lingvistica generală aplicabilă limbilor naturale.

Domeniile de cercetare ale semanticii sunt¹⁵:

- *Semantica lexicală* are ca obiect studiul semnificației cuvintelor și al structurii interne a vocabularului în întregime.
- *Semantica propozițională* cercetează felul în care, din sensul fiecărui cuvânt în parte rezultă unități sintactice mari (fraze) cu semnificație proprie.
- *Semantica textelor* analizează combinația propozițiilor reale sau ipotetice, din care rezultă o descriere, o narațiune sau o argumentație coerentă.
- *Semantica discursivă* are ca obiect studiul exprimărilor diverselor persoane care se găsesc angajate într-o discuție, convorbire banală sau într-o dispută științifică.

Relațiile între limbă și gândire, respectiv între limbă și lumea existentă ne permit să amintim și de alte domenii ale semanticii:

- *Semnificația cognitivă* se referă la relația dintre limbă și gândire. O exprimare verbală poate fi înțeleasă doar în măsura în care ea reconstruiește structura gândirii celui ce o exprimă.

¹² Michel Bréal, 1883/2012, *Essai de Sémantique*, Paris: Hachette.

¹³ Charles William Morris, 1949, *Foundations of the Theory of Signs*, Chicago: University of Chicago Press.

¹⁴ Irène Tamba-Mecz, 1988, *La sémantique*, Paris, PUF, p. 7.

¹⁵ <http://ro.wikipedia.org/wiki/Semantic%C4%83>.

- *Semnificația informațională*, denumită și *teorie referențială*, este disciplina conform căreia rolul principal îl joacă relația dintre limbajul descompus în unități informaționale și persoana referentă.

- *Semnificația pragmatică* studiază relațiile între sensul lingvistic al unei exprimări și un anumit context.

În cazul limbajelor de programare, semantica reprezintă un set de reguli care determină semnificația instrucțiunilor scrise într-un fișier program.

Semantica comenzii, de pildă, arată sensul cuvintelor dintr-o comandă, având ca scop clarificarea sensului și a semnificației noțiunilor complexe, derivate din simbolurile cele mai simple ale limbajului, sprijinindu-se pe regulile sintaxei, fără a se identifica însă cu aceasta.

În logică și informatică se folosesc termenii de „*Semantică formală*” sau „*Semantica limbajului de programare*”. Între semantică și sintaxă există același raport ca între fond și formă.

Analiza textului într-un program sursă

Analiza textului-sursă, adică a liniilor de comandă din programul-sursă, constă în: analiza lexicală, analiza sintactică și analiza semantică a acestuia.

Analiza lexicală este prima operație pe care o facem asupra fișierului program-sursă, considerat un șir de caractere care conține subșiruri de caractere, numite „atomi lexicali”, cu operatori (+, -, <, >), cuvinte-cheie sau cuvinte rezervate, constante (10, 1, 2, 3, 23, A, ANA), identificatori și separatori.

Analizorul lexical execută următoarele operații:

- detectează în programul-sursă subșirurile care respectă regulile de formare a atomilor lexicali;
- clasifică subșirurile, adică identifică clasa de care aparțin aceste subșiruri;
- traduce subșirurile în atomi lexicali;
- memorează atomii în tabela de simboluri.

Analiza sintactică a șirului atomilor lexicali identifică structurile sintactice - expresii, liste, instrucțiuni, proceduri - și generează o descriere structurală a acestuia. În cazul în care șirul de intrare este corect sintactic, apare *arborele sintactic* (de derivare) sau, în caz contrar, un *mesaj de eroare*. Arborele sintactic descrie relațiile de separare ori de incluziune dintre structuri.

Analiza semantică folosește arborele sintactic, creat în faza de analiză sintactică, pentru a extrage informații privind aparițiile în programul-sursă a obiectelor purtătoare de date (tipuri de date, variabile, proceduri, funcții) și pentru a atăta necesitatea utilizării lor. Odată cu parcurgerea arborelui sintactic are loc și generarea codului intermediar. Acesta reprezintă un șir de instrucțiuni simple, cu format fix, în care codurile operațiilor sunt asemănătoare cu codurile mașină corespunzătoare, ordinea operațiilor respectă ordinea execuției (conform apariției lor în

programul sursă), iar operanzii sunt prezentați sub forma variabilelor din programul-sursă (nu sub forma de adrese de memorie).

În practică, analiza semantică are loc în paralel cu cea sintactică, prin asocierea acțiunilor analizorului sintactic cu anumite structuri de date ce reprezintă atribute ale componentelor sintactice.

Generarea codului obiect presupune alocarea locațiilor de memorie și a *registrelor*¹⁶ unității centrale pentru variabilele programului și înlocuirea codurilor de operații din codul intermediar cu codul mașină.

Componentele de bază (lexicală, sintactică, semantică) sunt asistate pe tot parcursul compilării de următoarele două module:

a) *Modulul de tratare a erorilor* prezintă o colecție de proceduri care sunt activate ori de câte ori este detectată o eroare în timpul operațiilor de analiză.

În funcție de *faza de analiză* în care se află programul, erorile pot fi *lexicale, sintactice* sau *de semantică*. Modul de tratare a erorilor afișează mesajele de diagnostic relativ la eroarea depistată și iau decizii privind metoda de continuare a traducerii: fie se continuă compilarea, ignorând elementul ce conține eroarea, fie se încearcă corectarea erorii sau se întrerupe traducerea.

b) *Modulul de gestiune a tabelor* prezintă o colecție de proceduri care creează și actualizează baza de date a compilatorului și conține informații proprii compilatorului (generate la implementare și constituite din mecanismele de descriere a analizei lexicale, sintactice și semantice) și informații ce aparțin programului-sursă, care se traduce (identificatorii, constantele, cuvintele-cheie), memorate în tabela de simboluri.

Gestionarea tabelii de simboluri se constituie în funcție de: modul de reprezentare al acesteia, caracterul general al tabelii, tipul limbajului din care se traduce programul-sursă și convențiile alese pentru reprezentarea atributelor. De obicei, în faza de analiză lexicală, la întâlnirea unui nume nou, acesta este introdus în tabela de simboluri, reținându-se și adresa intrării. Orice referire ulterioară la acest nume actualizează informația din tabela corespunzătoare acestui nume, verificându-se și consistența utilizării acestuia (în cadrul analizei semantice). La generarea codului, atributele numelui determină lungimea zonei de memorie alocată acestuia. Atributele numelui pot servi și în faza de tratare a erorilor.

Translatorul programelor de calculator

Translatorul este programul care traduce fișierele-sursă scrise într-un anumit limbaj de programare, într-un program echivalent dintr-un alt limbaj, acesta numindu-se „program destinație”.

Translatorii se clasifică în:

¹⁶ Registrul este un dispozitiv destinat memorării unui vector binar. Registrul de adrese păstrează adresa curentă folosită pentru efectuarea accesului la o memorie. Cf. DI, 1981, p. 287.

- *Compilatoare* - pentru acestea programul destinație se numește *program obiect* sau *cod obiect*, fiind apropiat de *codul mașină*. În cazul limbajelor de nivel înalt, translatorul poartă denumirea de *compiler sau interpretor*.

- Compilatoarele sunt transatoare care citesc tot programul scris în limbajul cod sursă și îl traduc în *cod mașină*.

- Interpretoarele sunt transatoare care citesc pe rând fiecare instrucțiune din programul sursă, după care o traduc și o execută.

- *Asambleare* – sunt compilatoarele limbajelor de asamblare folosite pentru traducerea mnemonicilor din limbajul de asamblare în limbajul mașină.

În aceste două cazuri, traducerea este urmată, de obicei, de editarea de legături, înainte de execuția propriu-zisă a programului. În această fază, codul executabil se constituie prin legarea codului-obiect rezultat din traducere cu alte module obiect (rezultate ale unor compilări anterioare sau cele existente în bibliotecile de comenzi).

- *Interpretoarele* realizează execuția programului sursă instrucțiune cu instrucțiune.

- *Preprocesoarele* traduc programele-sursă din limbaje de nivel înalt în programe destinație tot în limbaje de nivel înalt.

- *Cross-compilatoarele sau cross-asambleare* generează pe un calculator „gazdă” un cod obiect pentru un alt calculator-obiect (care are memorie mică și nu poate implementa programul de traducere);

- *Compilatoarele incrementale* reprezintă o combinație între compiler și interpretor, care folosește secvențe din programul-sursă cu o anumită independență sintactică și semantică pentru a le executa interpretativ.

Execuția limbajelor de programare

Pentru executarea unui program scris într-un limbaj oarecare, există, în principiu, două abordări: compilare sau interpretare. La compilare, compilerul transformă programul-sursă, în totalitatea sa, într-un program echivalent scris în limbaj mașină, care apoi este executat.

La interpretare, interpretorul ia prima instrucțiune din programul-sursă, o transformă în limbaj mașină și o execută; apoi trece la instrucțiunea a doua și repetă aceleași acțiuni. Unele limbaje se pretează bine la compilare; de exemplu, limbajele clasice: Pascal, C, C++, altele sunt interpretate, de exemplu, SQL, PHP.

Multe limbaje moderne combină compilarea cu interpretarea: codul sursă este *compilat* într-un limbaj binar numit *bytecode*, care la rulare este *interpretat* de către o *mașină virtuală*. De remarcat este faptul că unele interpretoare de limbaje pot folosi compilatoare așa-numite *just-in-time*, care transformă codul în limbaj mașină chiar înaintea executării.

Bibliografie

- BÂRLEA, Petre Gheorghe, 2015, „Pentru o analiză semantică informatizată a textului poetic”, în: Florina Loredana Streinu, *Poezia Anei Blandiana - o analiză semantic-textuală*, București: Editura Muzeul Literaturii Române, pp. 5-10.
- BÂRLEA, Petre Gheorghe, 2014, „Baza logică a structurilor condiționale în limbile naturale”, în: Gabriela Duda (coord.), *Cultura limbii. Omagiu Doamnei Profesor Domnița Tomescu*, Ploiești: Editura Universității de Petrol și Gaze, pp. 53-64.
- BRÉAL, Michel, 1883/2012, *Essai de Sémantique*, Paris: Hachette.
- Dicționar de Informatică (DI)*, 1981, București: Editura Științifică și Enciclopedică.
- Dicționar de matematică și cibernetică în economie*, București: Editura Științifică și Enciclopedică, 1979.
- MOESCHLER, Jacques; Reboul, Arme, 1999, *Dicționar Enciclopedic de Pragmatică*, Cluj-Napoca: Editura Echinox.
- MORRIS, Charles William, 1949, *Foundations of the Theory of Signs*, Chicago: University of Chicago Press.
- REGNEALĂ, Mircea, 2001, *Dicționar explicativ de biblioteconomie și știința informării*, vol. I. București: FABR.
- TAMBA-MECZ, Irène, 1988, *La sémantique*, Paris: PUF.

