ANALIZA LEXICALĂ

Obiectul analizei lexicale

Analiza lexicală este prima fază a procesului de translatare (compilare). Partea din programul translator care realizează funcția de analiză lexicală se numeşte **analizor lexical** (ANLEX).

Analizorul lexical primeşte la intrare textul sursă al programului, sub forma unui şir de caractere, grupează caracterele în **simboluri** sau **atomi lexicali** (**tokens**, în limba engleză), codifică numeric atomii lexicali furnizându-i, unul câte unul sau în ansamblu (toți atomii lexicali corespunzători programului), fazei următoare de translatare: analiza sintactică (fig. 3.1).

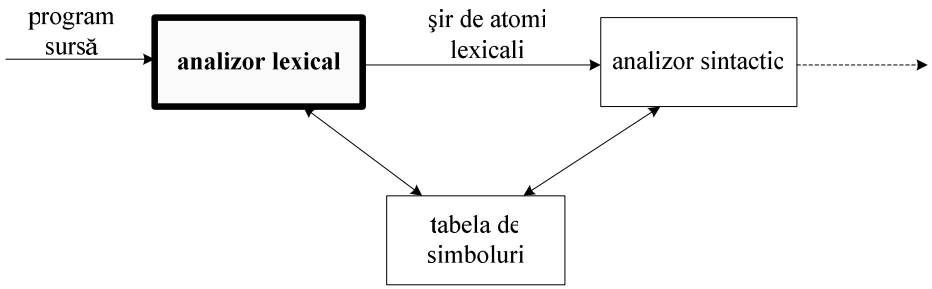


Fig. 3.1. Locul analizorului lexical în cadrul programului translator

Activitatea analizei lexicale poate fi despărțită, la rândul ei, în două faze:

- **Faza de prelucrare** (**scanning**, în limba engleză iar partea corespunzătoare din program se numeşte **scanner**): grupează caracterele consecutive ale textului sursă în unități lexicale distincte (atomi);
- **Faza de triere** (**screening,** în limba engleză): compactifică programul prin eliminarea spațiilor şi comentariilor, precum şi prin identificarea şi codificarea cuvintelor rezervate.

În urma analizei lexicale programul se transformă într-un şir de numere reprezentând codificarea, în ordine, a atomilor lexicali. Atomii lexicali pot fi grupați în clase de şiruri de caractere cu reguli de formare precise.

În majoritatea limbajelor de programare, printre alte clase se întâlnesc:

- identificatorii;
- numerele întregi;
- numerele reale;
- şirurile de caractere;
- operatorii;
- delimitatorii sau separatorii.

Clasele conțin un număr finit de elemente ceea ce ar permite, în principiu, ca apartenența unui atom la o anumită clasă să poată fi decisă prin comparația, pe rând, a atomului respectiv cu elementele clasei. Totuşi anumite clase (exemple: identificatorii, numerele întregi sau reale) conțin un număr foarte mare de elemente şi metoda de mai sus nu poate fi, practic, aplicată. În aceste cazuri, identificarea clasei se poate face pe baza regulilor comune de grupare a caracterelor specifică elementelor acelei clase.

În codificarea numerică a unui atom lexical se marchează clasa de care aparține atomul care, în cele mai multe cazuri, este însoțită de alte informații caracteristice atomului reprezentând **atributele** sale. De exemplu, notând cu 1 clasa identificatorilor şi cu 2 clasa numerelor întregi, identificatorul ”pas” şi numărul 17 pot fi codificați ca în fig. 3.2 a) şi respectiv b).

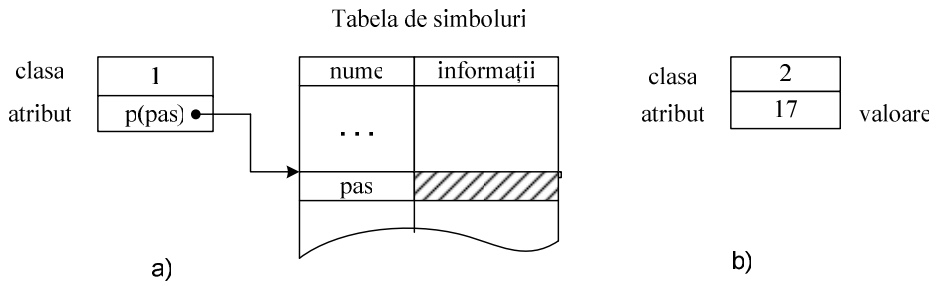


Fig. 3.2. Exemplu de codificare a unor atomi lexicali

Analiza lexicală execută şi unele operații auxiliare: eliminarea spațiilor şi comentariilor din program, gestionarea liniilor sursă în vederea semanlizării exacte a erorilor, listarea textului sursă eventual pregătirea unui text sursă care, înainte de listare, să fie completat cu mesaje de eroare etc.

Relația analizor sintactic - analizor lexical

Rolul analizei lexicale este foarte asemănător cu cel al analizei sintactice: identificarea, conform unor reguli, a unităților distincte în cadrul programului, semnalarea de erori în cazul abaterii de la reguli, clasificarea şi codificarea unităților identificate etc. Din această cauză, funcțiile analizorului lexical pot fi preluate de analizorul sintactic. Cu toate acestea, în marea majoritate a cazurilor, se preferă separarea celor două activități în faze distincte. În favoarea acestei decizii se pot aduce următoarele argumente:

✓ Proiectarea separată este mai simplă, mai clară, permite lucrul în echipă şi corespunde organizării modulare a compilatorului.

✓ Procesul de analiză lexicală deşi este mai simplu, este relativ lung, în timp. El presupune: citirea textului sursă de pe suportul extern, accesul la fiecare caracter, numeroase comparații cu elementele unor mulțimi de caractere sau de grupuri de caractere în vederea identificării şi clasificării atomilor, căutări şi/sau introduceri în tabele etc. Separarea va conduce în acest caz la organizarea mai judicioasă a programului, la creşterea eficienței analizei lexicale permițând chiar scrierea ANLEX într-un limbaj cu facilități speciale pentru aceste operații în timp cu următoarele faze pot fi realizate în alt limbaj de programare. Un alt procedeu pentru creşterea vitezei analizorului lexical este utilizarea unor buffere de intrare, în memoria internă, pentru înmagazinarea textului sursă. În timpul desfăşurării analizei lexicale având la intrare un buffer, poate fi pregătit, prin citirea de pe suportul extern, bufferul următor.

✓ Sintaxa atomilor lexicali este simplă în comparație cu sintaxa instrucțiunilor limbajului şi poate fi exprimată printr-o gramatică regulată (GR). Regulile specifice unei GR şi procesul de analiză aferent pot fi modelate cu ajutorul expresiilor regulate sau a

automatelor finite ceea ce simplifică mult procesul de proiectare-programare al analizorului lexical.

✓ Prin separarea celor două faze se deschid două căi distincte de simplificare a analizorului sintactic (ANSIN):

- o parte din sarcini sunt efectiv preluate de ANLEX;
- în urma prelucrărilor efectuate de ANLEX, textul programului este scurtat şi simpificat din mai multe puncte de vedere: se elimină porțiunile inutile (spații, comentarii etc.); atomii lexicali sunt codificați numeric; se elimină anumite construcții (lexicale) dificil de exprimat în gramatica independentă de context care descrie sintaxa limbajului; se rezolvă şi semantica atomilor lexicali.

✓ Se îmbunătățeşte portabilitatea compilatorului. Particularitățile de alfabet ale limbajului care depind de perifericul de intrare în sistemul de calcul, sunt localizate numei în ANLEX astfel că, la trecerea compilatorului de pe un sistem pe altul, trebuie rescrisă numai această parte. Evident, s-a presupus că ambele sisteme acceptă limbajul în care este scris compilatorul.

În funcție de gradul de separare dintre ANLEX şi ANSIN se disting următoarele situații:

- **ANLEX independent de ANSIN**: analizorul lexical reprezintă un pas (o trecere) al compilatorului, comunicarea cu analizorul sintactic realizându-se printr-un fişier sau zonă de memorie.

- **ANLEX apelat ca funcție din ANSIN**: analizorul sintactic apelează funcția analizor lexical ori de câte ori are nevoie de un nou atom lexical. Este soluția utilizată cel mai des.

- **ANLEX funcționează în regim de corutină cu ANSIN**: cele două faze sunt simultan în execuție şi îşi transferă controlul, una alteia, în momentele necesare.

Noțiuni specifice analizei lexicale

În general există o mulțime de şiruri de intrare pentru care ANLEX produce **simboluri** (**atomi)** din aceeaşi **clasă**. Această mulțime de şiruri este descrisă de o regulă numită **tipar** sau **şablon** al clasei respective.

Lexemele sunt şiruri de caractere în limbajul sursă care corespund tiparului pentru o anumită clasă de atomi. În concluzie, **tiparul reprezintă o regulă pentru descrierea mulțimii de lexeme care pot forma o clasă de simboluri**.

Exemplu:

text sursă: `---------`

```
a:=5;
if b1>3.5 then sir:='eroare';
```

În fig. 3.3 se prezintă atomii (grupați în clase) şi lexemele corespunzătoare fragmentului de program din exemplul de mai sus.

clasa de simboluri (atomi)	lexeme	descrierea tiparului
ident	a, b1, sir, -----	literă urmată, eventual, de litere sau cifre
opatrib	:=	sirul de caractere ':='
delim	;, ,, (,), ----	caracterul ';' sau caracterul ',' sau caracterul '(' sau ----
if	if	şirul de caractere 'if'
intreg	5, -3, 2, ----	orice constantă numerică întreagă, reprezentabilă în calculator
real	3.5, -0.2E-7, ----	orice constantă numerică reală, reprezentabilă în calculator
oprel	<, >, <=, >=, =, <>	caracterul '<' sau caracterul '>' sau sirul de caractere '<=' sau ----
then	then	şirul de caractere 'then'
sircar	'eroare' ---------	orice succesiune de caractere încadrată între ' '

Fig. 3.3. Exemple de clase de atomi, lexeme şi tipare

Unele convenții adoptate în diferite limbaje de programare au ca rezultat creşterea gradului de dificultate în analiza lexicală. Limbaje ca Fortran sau Cobol, de exemplu, impun o anumită structură a liniei sursă pe fişierul de intrare. Limbajele moderne au, în exclusivitate, formatul de intrare liber, aranjarea instrucțiunilor pe linii urmărind doar creşterea clarității şi lizibilității programului.

Tratarea spațiilor este, de asemenea, diferită de la un limbaj la altul. În Fortran, şi Algol 68 spațiile, cu excepția celor din şirurile de caractere, sunt nesemnificative. Acest fapt îngreunează identificarea atomilor din unele instrucțiuni. Pentru exemplificare, să considerăm instrucțiunea Fortran:

```
DO 5 I = 1.25
```

Întrucât spațiile sunt nesemnificative şi se elimină, instrucțiunea de mai sus este o instrucțiune de atribuire (variabilei DO5I i se atribuie valoarea 1.25). În schimb instrucțiunea:

```
DO 5 I = 1,25
```

este binecunoscuta instucțiune de ciclare, marcată prin cuvântul cheie DO. Se observă uşor că, abia la întâlnirea virgulei din dreapta semnului egal, analizorul lexical poate deosebi cele doua cazuri şi poate împărți textul sursă în mod corect în simboluri lexicale. În varianta Fortran 77 s-a adoptat o măsură care poate simplifica analiza lexicală: între etichetă şi indice se poate prevedea, opțional, o virgulă, astfel:

```
DO 5,I = 1,25
```

O altă dificultate importantă apare în limbajele în care cuvintele cheie nu sunt rezervate, urmând ca ANLEX sau ANSIN să deosebească din context cuvintele cheie de cele definite de utilizator. Să considerăm următoarea instrucțiune corectă în limbajul PL/I:

```
IF THEN THEN THEN = ELSE; ELSE ELSE = THEN
```

Cuvinte cheie sunt doar cele subliniate. Celelalte două cuvinte, THEN şi respectiv ELSE, reprezintă două variabile logice.

Atribute ale atomilor

S-a arătat că, pentru fiecare simbol, analizorul lexical furnizează numărul clasei din care face parte. Atunci când unei clase îi corespund mai multe lexeme, numărul clasei trebuie să fie însoțit de informații suplimentare care să caracterizeze complet atomul în cauză. Aceste informații se furnizează sub formă de *atribute ale atomilor*. În cele mai multe cazuri este suficient un singur atribut care să fie ori un pointer spre o tabelă cu informții (de exemplu tabela de simboluri) ori un număr cu altă semnificație (de exemplu valoarea atomului).

Exemplu: Să considerăm următoarea instrucțiune:

```
a:=2*b-c;
```

Pentru atomii conținuți în instrucțiune, ANLEX va furniza următoarea secvență de coduri:

< **ident**, p(a) >
< **opatrib** >
< **intreg**, 2 >
< **operator**, inm >
< **ident**, p(b) >
< **operator**, dif >
< **ident**, p(c) >
< **delim**, punctvir >

p(x) reprezintă atributul identificatorului *x*. Pe baza codului lexical se poate intra în tabela de simboluri, în rândul corespunzător identificatorului respectiv. *inm*, *dif* şi *punctvir* sunt constante care desemnează operatori de inmulțire şi scădere şi respectiv delimitatorul ";". Valoarea constantei 2 s-a transmis direct. În alte variante, pentru constante se poate creea o tabelă asemănătoare cu tabela de simboluri, caz în care se furnizează ca atribut un pointer de acces la tabelă.

Erori lexicale

Deoarece analiza lexicală tratează relativ puține aspecte legate de verificarea programului, viziunea asupra corectitudinii programului este limitată în această fază şi se semnalează un număr redus de erori.

Pot apărea însă situații în care analiza lexicală nu poate continua pentru că nici unul dintre tiparele cunoscute analizorului lexical nu corespunde cu un prefix al textului sursă care urmează în fişierul de intrare. În astfel de cazuri se semnalează erori lexicale.

Exemplu de erori lexicale tipice:

- caracter interzis în fişierul de intrare;
- ortografierea greşită a unei constante reale;
- constantă întreagă sau reală nereprezentabilă în calculator;
- şir de caractere prea lung;
- şirul de caractere conține sfârşitul de linie; { pentru acele limbaje sau implementări care nu admit trecerea unui şir de pe o linie pe alta} etc.

În cazul apariției unei erori se poate opri compilarea sau, dacă ea se continuă, ANLEX trebuie să ia unele măsuri pentru ca şi analiza lexicală să poată continua. **Totalitatea acestor măsuri reprezintă revenirea în caz de eroare.** Cea mai simplă strategie de revenire este cea cunoscută sub numele de **modul de panică**: se şterg succesiv caractere din textul sursă furnizat la intrare, începând cu caracterul curent, până când ANLEX poate să formeze un atom corect. Aceată tehnică poate determina confuzii în analiza sintactică. Alte acțiuni posibile pentru revenirea din erori lexicale sunt:

- ştergerea unui caracter considerat în plus;
- inserarea unui caracter lipsă;
- înlocuirea unui caracter incorect cu unul corect;
- inversarea a două caractere adiacente.

Aceste acțiuni pot fi utilizate combinat, în cadrul aceleiaşi strategii.

Modalități de specificare a simbolurilor în vederea implementării analizorului lexical

Cel mai simplu mod de a construi un analizor lexical îl reprezintă scrierea programului propriu-zis pe baza unor **diagrame**, realizate în prealabil, care precizează structura atomilor din limbajul sursă. Această tehnică numită **manuală** asigură, de obicei, analizoare lexicale eficiente.

O altă modalitate principial diferită este definirea unui limbaj de specificare a structurii atomilor şi a eventualelor acțiuni asociate cu analiza lexicală şi realizarea unui program de traducere (”compilator”) pentru acest limbaj. Rezultatul unei astfel de ”compilări” este un analizor lexical. Această metodă reprezintă **generarea automată** a analizorului lexical şi permite ca tehnicile de analiză lexicală implementate în programul generator să fie aplicabile asupra unei clase mai largi de limbaje de programare.

Limbajele pentru descrierea structurii atomilor lexicali sunt **limbaje regulate**. Pentru specificarea acestui tip de limbaje se pot utiliza fie **gramatici regulate** (**GR**) sau gramatici independente de context (**GIC**) reductibile la gramatici regulate, fie **expresii regulate**.

Odată stabilit limbajul atomilor lexicali şi având o gramatică pentru acest limbaj, problema de principiu a analizei lexicale apare, în primul rând, ca o problemă de decizie: să stabilească dacă un şir de caractere este sau nu simbol lexical. Decizia trebuie în continuare completată de codificarea atomilor lexicali şi de semnalarea şi tratarea erorilor.

În cazul unui limbaj regulat, modelul formal care poate fi utilizat în acceptarea cuvintelor din limbaj este automatul finit care poate fi deci considerat modelul de bază al analizorului lexical.

Gramatica atomilor lexicali

S-a precizat anterior că sintaxa atomilor lexicali poate fi descrisă printr-o GR sau o GIC care generează un limbaj regulat. În acest din urmă caz există metode simple de a transforma gramatica independentă de context într-o gramatică regulată echivalentă.

O gramatică a atomilor lexicali reprezentativă pentru multe limbaje de programare este următoarea:

(G_0) 1. <program> → <şir atomi>
2. <şir atomi> → <atom> | <şir atomi> <atom>
3. <atom> → <ident> | <const> | <operator> | <delim> | <com>
4. <ident> → <litera> | <ident> <litera> | <ident> <cifra>
5. <const> → <intreg> | <real> | <şircar>
6. <întreg> → <cifra> | <întreg> <cifra>
7. <real> → <întreg> **.** <întreg> | <mantisă> E<exponent>
8. <mantisă> → <întreg> | <întreg> **.**<întreg>
9. <exponent> → <semn> <întreg>
10. <semn> → + | - | ε
11. <operator> → + | - | ∗ | / | < | <= | > | >= | = | <> | :=
12. <delim> → spațiu | ; | , | (|) | [|] | **:** | **..** | **.**
13. <şircar> → ‘ <orice succesiune de caractere> ’
14. <com> → { <orice succesiune de caractere ce nu conține }> }
15. <litera> → A | B | . . . | Z | a | b | . . . | z
16. <cifra> → 0 | 1 | 2 | . . . | 9

Gramatica G_0 nu este regulată dar poate fi transformată într-o gramatică regulată mărind numărul producțiilor şi al neterminalelor. De exemplu producția 6 se poate scrie:

<întreg> → 0 | 1 | 2 | . . . | 9 | <întreg>0 | <întreg>1 | . . . | <întreg>9

Dacă vom proceda aşa procesul de proiectare-programare al analizorului lexical se lungeşte şi se complică substanțial. De aceea se preferă o simplificare a proiectării prin stratificarea gramaticii G_0 într-o ierarhie de gramatici mai simple, regulate şi care ulterior se asamblează astfel încât limbajul final să rămînă acelaşi.

În gramatica pentru descrierea sintaxei limbajului sursă, atomii se tratează ca simboluri terminale (componente ale alfabetului gramaticii)

Exemplu:

<întreg> → <**cifra**> | <întreg> <**cifra**>

Diagrame de tranziție

Diagrama de tranziție este un pas intermediar în construcția analizorului lexical. Ea se reprezintă ca o ordinogramă stilizată (un graf) şi descrie acțiunile care au loc atunci când ANLEX e apelat de ANSIN pentru a furniza următorul atom. Diagrama de tranziție ține evidența informațiilor obținute din şirul de intrare pe măsură ce se avansează pointerul din fişierul de intrare (numit *pointer de anticipare*) şi se analizează noi caractere din fişierul de intrare.

Diagrama de tranziție se compune din cercuri care indică poziția (starea curentă) şi segmente orientate (arce) etichetate care unesc stările. Arcele care ies dintr-o stare au ca etichete caracterele de intrare la apariția cărora sunt definite, în diagrama respectivă, transferuri între stări. Pentru a indica orice alt caracter care nu apare explicit ca etichetă a arcelor care ies din aceeaşi stare, se utilizează o etichetă specială numită **altceva**.

Se presupune că diagramele de tranziție sunt **deterministe** adică nu există simboluri care pot să apară ca etichete pe două sau mai multe arce ieşind din aceeaşi stare. Starea inițială, în care se află controlul când începe recunoaşterea unui atom se numeşte **start**. Unor stări li se pot asocia acțiuni care trebuie să se execute atunci când fluxul de comandă ajunge la starea respectivă. La ieşirea dintr-o stare se citeşte următorul caracter şi, dacă se va găsi un arc etichetat cu acest caracter, se continuă deplasarea în diagramă; în caz contrar se indică *eşec*.

Exemplu:
Se consideră relația: <oprel> → < | <= | > | >= | = | <>

Diagrama de tranziție pentru tiparele >= şi > este (fig. 3.4):

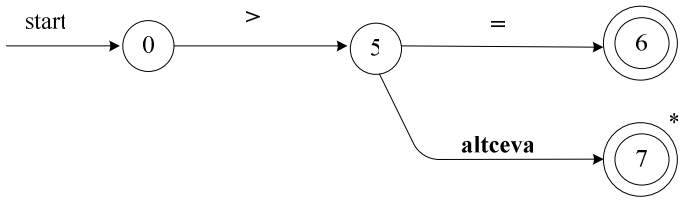


Fig. 3.4. Diagrama pentru tiparele >= şi >

Din starea **start**, se intră în această diagramă dacă următorul caracter citit este “>”. Acesta conduce fluxul de comandă în starea 5, în care se citeşte următorul caracter din intrare. Dacă noul caracter este “=”, s-a recunoscut atomul “>=”, diagrama ajungând în starea 6 (finală). Pentru orice alt caracter citit în starea 5, se ajunge în starea finală 7. În acest caz, atomul recunoscut este “>”, dar pointerul de anticipare este avansat cu o poziție şi trebuie retractat. Acest fapt s-a marcat în diagramă prin semnul “∗”.

În general, analizorul lexical constă din mai multe diagrame de tranziție ce pornesc din starea 0 şi specifică grupuri de atomi. Dacă, la parcurgerea unei diagrame, se semnalează *eşec*, se retractează pointerul de anticipare la poziția pe care o avea în starea de **start** a diagramei şi se activează următoarea diagramă de tranziție. Dacă, în toate diagramele, se semnalează *eşec*, înseamnă că s-a detectat o eroare lexicală şi se apelează rutina corespunzătoare pentru tratare şi revenire.

Ordinea de plasare a diagramelor parțiale în diagrama completă a analizorului lexical poate fi importantă atât pentru corectitudinea cât şi pentru viteza analizei.

În proiectarea analizorului lexical, diagramele de tranziție au un rol orientativ. Ele arată care sunt sarcinile analizorului în identificarea tiparelor corespunzătoare atomilor lexicali. Pentru rezolvarea celorlalte probleme: semnalarea erorilor şi generarea codului lexical, în practica proiectării analizoarelor lexicale se utilizează **proceduri semantice** asociate tranzițiilor din diagramă. Procedurile lucrează asupra unor structuri de date pe baza cărora stabilesc deciziile şi pe care, eventual, le modifică. Aceste structuri de date alcătuiesc baza de date a analizorului lexical.

Exemple :

a) **Diagrama completă pentru toți operatorii relaționali**. Este prezentată în fig. 3.5.

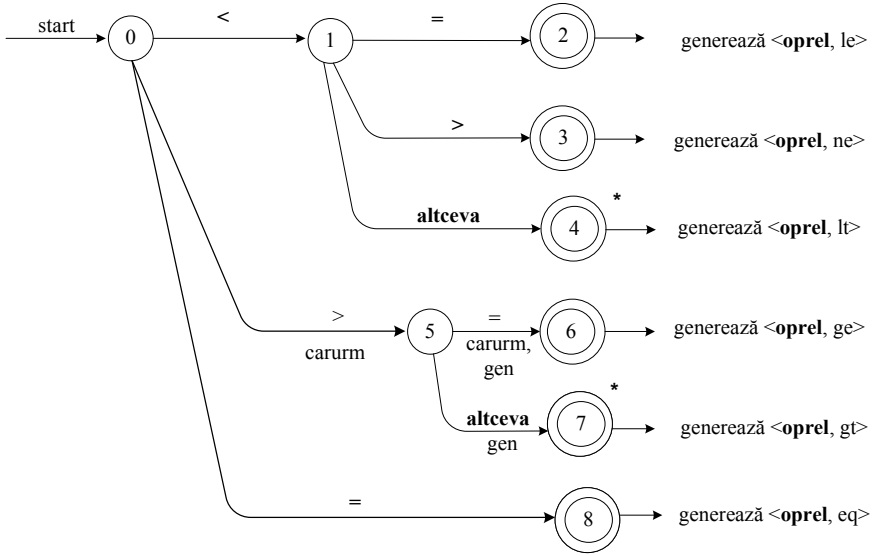


Fig. 3.5. Diagrama de tranziții completă pentru operatorii relaționali

b) **Tratarea identificatorilor şi a cuvintelor cheie**. O problemă importantă este aceea a diferențierii între cuvintele cheie şi identificatorii utilizator, dat fiind faptul că din punct de vedere lexical au aceeaşi structură: o succesiune de litere şi/sau cifre care începe cu o literă. O primă modalitate este construirea de diagrame separate pentru cuvintele cheie. Această variantă are dezavantajul că măreşte mult numărul de stări din analizorul lexical.

O variantă practică des întâlnită este aceea în care cuvintele cheie şi ceilalți identificatori se tratează în mod uniform (fig. 3.6). Cuvintele cheie sunt prezente în tabela de simboluri de la începutul compilării, prin inițializare.

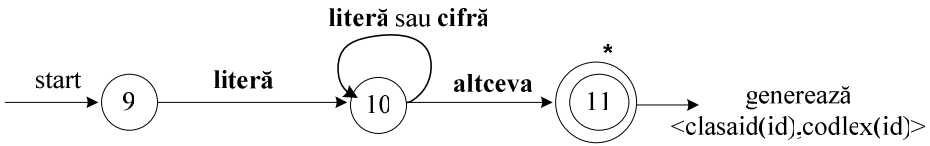


Fig. 3.6. Diagrama de tranziții pentru identificatori

Funcția *clasaid* caută cuvântul *id*, transmis ca parametru, în tabela de simboluri şi returnează clasa sa (identificator utilizator sau cuvânt cheie).

Funcția *codlex* are, de asemenea, acces la tabela de simboluri în care caută dacă identificatorul *id*, transmis ca parametru, este prezent sau nu. În cazul în care identificatorul este găsit în tabelă, se returnează codul lui lexical sub forma, de exemplu, a indicelui sau pointerului de intrare în tabelă. Dacă lexema respectivă nu este prezentă în tabela de simboluri, ea se va introduce, returnându-se indicele(pointerul) intrării nou create.

Procedând în acest fel, diagrama de tranziții şi implicit programul ANLEX, nu se va modifica la extinderea limbajului prin introducerea de noi cuvinte cheie. Este necesară doar reinițializarea tabelei de simboluri.

c) **Tratarea numerelor fără semn în Pascal**

Numerelor fără semn din limbajul de programare Pascal le corespund definiția regulată:

număr → **cifră** +(.**cifră**+)?(E(+ | -)?**cifră**+)?

(vezi cap.2)

În fig. 3.7 se prezintă trei diagrame de tranziții corespunzătoare celor trei forme în care se poate găsi numărul în fişierul de intrare:

- număr zecimal în format ştiințific (cu mantisă şi exponent);
- număr zecimal cu punct (parte întragă şi parte zecimală);
- număr întreg.

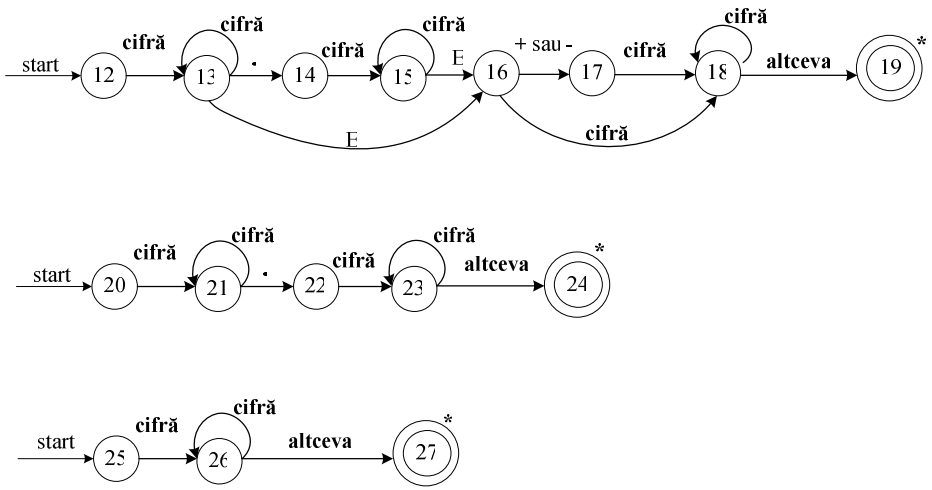


Fig. 3.7. Diagramele de tranziții pentru numerele fără semn în Pascal

Întrucât forma 3 este inclusă, ca prefix, în forma 2 şi, analog, forma 2 în forma 1, aceste diagrame parțiale trebuie plasate în diagrama completă a analizorului lexical exact în această ordine. Altfel se poate ajunge, de exemplu, la recunoaşterea eronată a unui număr întreg care reprezintă de fapt partea întreagă a unui număr zecimal.

d) **Tratarea spațiului alb.** Spațiul alb reprezintă reuniunea acelor lexeme din fişierul de intrare care sunt ignorate de analizorul lexical. El poate fi definit astfel:

ignor → spațiu | newline | com
spalb → ignor+

La întâlnirea unei lexeme corespunzătoare tiparului spațiului alb, nu se revine în analizorul sintactic ci se continuă cu detectarea de atomi în fişierul de intrare. Diagrama de tranziții corespunzătoare este prezentată în fig. 3.8 se continuă cu starea de start a primei diagrame de tranziții, pentru a căuta un alt tipar.

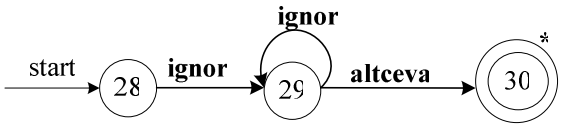


Fig. 3.8. Diagrama de tranziții pentru recunoaşterea şi ignorarea spațiului alb

Pentru a mări viteza de căutare şi recunoaştere a tiparelor este necesar ca, la stabilirea ordinei de plasare a diagramelor parțiale în diagrama completă a ANLEX, să se pună în față diagramele corespunzătoare atomilor care apar mai frecvent. Din acest punct de vedere este bine ca diagrama corespunzătoare spațiului alb să fie tratată chiar la începutul ANLEX.

În principiu, implementarea în program a unei secvențe de diagrame de tranziții, se poate realiza în felul următor:

Fiecărei stări i se va asocia o porțiune de program distinctă. Dacă există arce care ies din starea curentă, atunci se citeşte următorul caracter din fişierul de intrare şi se selectează arcul corespunzător (dacă există un astfel de arc). Citirea caracterului următor se poate realiza într-o procedură *carurm* care operează asupra tamponului de intrare şi returnează caracterul. Dacă există un arc etichetat cu caracterul citit sau cu o clasă de caractere din care face parte şi caracterul citit, se transferă controlul la secvența de program pentru noua stare, printr-o instrucțiune selectivă de tipul `if` sau `case`. Dacă nu există arc etichetat cu caracterul citit şi starea curentă nu este acceptoare (finală) se va apela o procedură *eşec*, care retractează pointerul de anticipare la începutul de lexemei şi face trecerea la următoarea diagramă. Dacă nu mai există diagrame, tot *eşec* poate apela o procedură specială de revenire din erori.

În cazul în care limbajul de programare utilizat la implementare nu are o instrucțiune de tip `case`, se poate crea un tablou pentru fiecare stare, indexat prin codul caractererelor din intrare. Fiecare element din tablou va corespunde unei stări noi, fiind un pointer la o secvență de program care trebuie executată atunci când caracterul curent are valoarea corespunzătoare acelui indice. Porțiunea de program asociată unei stări se poate incheia cu o instrucțiune de salt la starea următoare din diagrama de tranziții (se simulează, de fapt, un `case`).

Proiectarea şi implementarea unui analizor lexical complet

Elementul inițial de proiectare îl constituie gramatica atomilor lexicali, exprimată în BNF (§3.6) sau setul de definiții regulate echivalent cu această gramatică (cap.2). Să optăm pentru această a doua variantă. Ca etapă intermediară care va facilita scrierea programului se va realiza diagrama de tranziții generalizată a automatului de recunoaștere corespunzător analizorului lexical. ANLEX va funcționa ca o funcție în serviciul analizorului sintactic, fiind apelat ori de cîte ori se solicită un atom lexical. El va furniza codul lexical corespunzător clasei atomului şi, eventual, unul sau mai multe argumente.

Pentru fiecare definiție regulată se va realiza câte o diagramă de tranziții distinctă. Toate diagramele vor avea aceeaşi stare inițială. Stările finale pot să fie, de asemenea, grupate într-o stare finală unică, corespunzătoare ieşirii din procedura ANLEX, sau pot fi lăsate distincte. Diagrama de tranziții completă (generalizată), va rezulta din legarea în paralel a acestor diagrame parțiale.

În cazul în care expresia regulată din membrul drept al unei definiții este complicată, diagrama de tranziții corespunzătoare acestei definiții poate să fie realizată şi ea din părți, pe baza unui procedeu similar cu cel descris pentru diagrama completă. De asemenea, dacă expresiile regulate din membrul drept au acelaşi prefix sau conțin părți comune, diagramele de tranziții corespunzătoare pot să fie unificate pe porțiunile comune, pentru a evita descrierile de acțiuni redondante din codul programului ANLEX. Această ultimă măsură rezolvă şi nedeterminismul atât pentru automatele parțiale cât şi pentru cel final, ceea ce constituie o condiție necesară la scrierea corectă a programului. O altă condiție de aceeaşi natură este următoarea: orice simbol primit la intrarea unui automat şi care nu activează nici o tranziție, trebuie să conducă la o situație de neacceptare sau de eroare.

O altă situație specială care trebuie analizată este diferențierea stărilor finale în funcție de poziția în care rămâne pointerul de anticipare (*pa*) la recunoaşterea unui tipar. Această diferențiere s-a realizat în

exemplele din paragrafele anterioare prin marcarea cu '*' a acelor stări finale pentru care *pa* punctează deja primul caracter din atomul următor. Există două soluții pentru uniformizarea acestor situații:

— revenirea în şirul de intrare cu un simbol (retractarea *pa* cu o poziție);
— generalizarea avansului pentru toate stările finale prin citirea caracterului următor şi pentru stările nemarcate.

În continuare se va adopta această ultimă soluție.

Realizarea diagramelor de tranziții

Diagramele de tranziții prezentate în continuare sunt completate şi cu procedurile semantice aferente. Pentru aceasta, arcele care reprezintă tranzițiile vor fi marcate, deasupra, cu simbolul de intrare ce provoacă tranziția şi, dedesubt, cu procedurile semantice apelate. Există două proceduri semantice cu un mare grad de generalizate, fiind apelate în majoritatea situațiilor:

- **carurm**: se activează ori de câte ori este nevoie de un nou caracter din fişierul de intrare furnizând, în variabila globală *ch*, caracterul următor;
- **gen**: generează informațiile de ieşire ale automatului final (codul lexical şi eventualele argumente, cu precizarea că argumentele pot să fie pregătite în forma finală sau aproape finală de proceduri semantice apelabile anterior.

În continuare se prezintă diagramele de tranziții parțiale şi procedurile semantice specifice care apar în fiecare caz:

a) **Eliminarea spațiilor** (fig. 3.9). Atâta timp cât procedura *carurm* furnizează spații se rămâne în starea de **start** (0).

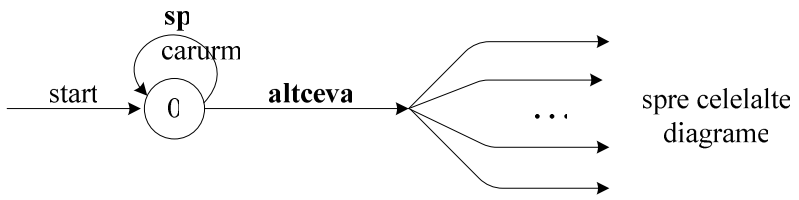


Fig. 3.9. Eliminarea spațiilor

b) **Tratarea identificatorilor** (fig. 3.10). În prima parte (starea 1) cuvintele utilizator şi cuvintele cheie sunt tratate în mod unitar. Procedura semantică *ad* adaugă (literă sau cifră) în zona reprezentată prin variabila globală *id*.

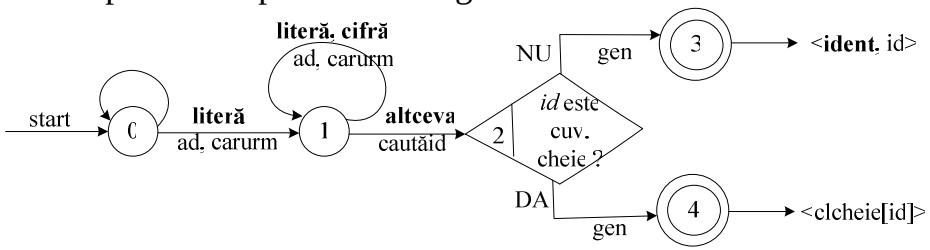


Fig. 3.10. Tratarea identificatorilor

La terminarea unui identificator (ieşirea din starea 1), se apelează procedura *cautaid* care, inspectând tabela cuvintelor cheie, stabileşte dacă identificatorul *id* este sau nu este cuvânt cheie. Din starea 2, pe baza răspunsului furnizat de *cautaid*, se merge într-una din stările acceptoare 3 sau 4, generând codul lexical corespunzător. Pentru cazul când *id* este cuvânt cheie, codul lexical se extrage dintr-un tabel special denumit *clcheie*. Pentru a nu mări în mod excesiv numărul de stări, s-a adoptat varianta unei singure ieşiri pentru toate cuvintele cheie (vezi şi §3.7, exemplul b)). Tabela cuvintelor cheie precum şi *clcheie* trebuie completate la iniţializarea compilatorului. Spre deosebire de varianta prezentată în exemplul b) din §3.7, tabela cuvintelor cheie este distinctă de tabela de simboluri propriu-zisă a compilatorului iar căutarea şi, dacă este cazul, introducerea unui identificator utilizator în tabela de simboluri este lăsată pe seama analizorului sintactic.

c) **Tratarea numerelor** (fig. 3.11). Spre deosebire de varianta prezentată în fig. 3.7 (exemplul c) din §3.7), numerele întregi şi reale se tratează în mod unitar, într-o singură diagramă.

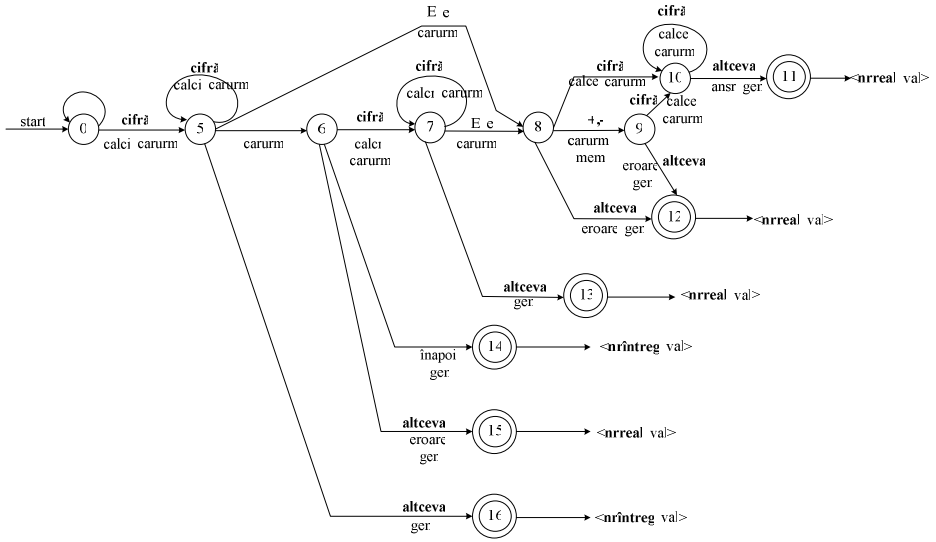


Fig. 3.11. Tratarea numerelor fără semn

Procedura *calci* construieşte numărul întreg furnizat în fişierul de intrare care poate să fie chiar valoarea atomului recunoscut (în stările acceptoare 14 sau 16) sau poate să reprezinte partea întreagă a unui număr real (eventual a mantisei lui).

Procedura *calcr* continuă calculul început de *calci* pentru cazul când, în fişierul de intrare, apare un număr real (zecimal) cu parte întreagă şi parte zecimală.

Pentru varianta în care numărul real este exprimat în format ştiinţific, procedura *calce* colectează exponentul iar procedura *mem* înregistrează, dacă este cazul, semnul ”–” al acestuia. În final, procedura semantică *ansr* asamblează numărul real complet furnizat în starea acceptoare 11.

Procedura semantică *inapoi* retractează pointerul de anticipare în cazul când, după un număr întreg, urmează tiparul ”**..**” iar procedura *eroare* semnalează toate eventualele greşeli.

d) **Tratarea tiparelor având ca prefix caracterul ”:”** (fig. 3. 12). Diagrama are două ieşiri (stări acceptoare), corespunzătoare operatorului de atribuire şi respectiv caracterului ”**:**”, izolat.

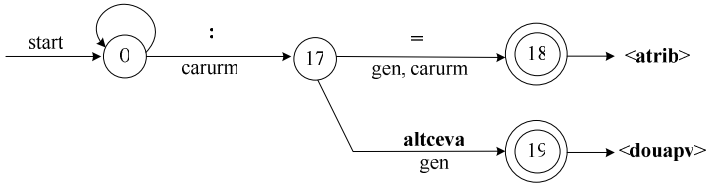


Fig. 3.12. Tratarea tiparelor ”**:**” şi ”**:=**”.

e) **Tratarea operatorilor relaționali** (fig. 3.13). Nu diferă, în esență, de cea prezentată în §3.7, exemplul a). Spre deosebire de aceasta, nu se tratează caracterul ”=” izolat, stările finale sunt uniformizate din punctul de vedere al poziției pointerului de anticipare şi se generează coduri lexicale distincte pentru toți operatorii relaționali.

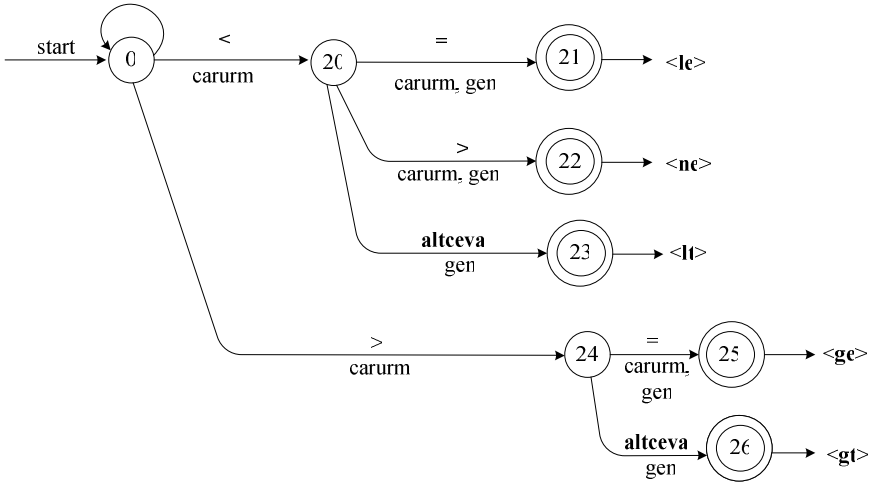


Fig. 3.13. Tratarea operatorilor relaționali

f) **Tratarea tiparelor având ca prefix caracterul ”.”.** (fig. 3.14). Diagrama are două ieşiri, corespunzătoare tiparului ”**.** **.**” şi respectiv caracterului ”**.**” izolat.

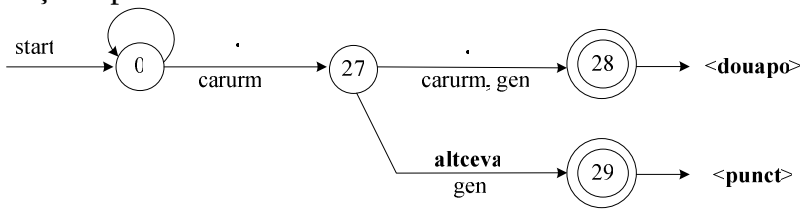


Fig. 3.14. Tratarea tiparelor ”**.** ” şi respectiv caracterului ”**.** **.**”

g) **Tratarea şirurilor de caractere** (fig. 3.15). Se tratează în mod unitar atât şirurile de caractere propriu-zise cât şi caracterele izolate. Procedura *constrsir* colectează şirul într-un tabel furnizând, în final, pointerul *p* de unde începe şirul în tabel precum şi lungimea sa, *lungsir*. Pentru caractere izolate, se furnizează ca atribut, valoarea efectivă a caracterului. Din diagramă se observă, de asemenea, faptul că apostroful trebuie dublat, dacă apare ca şi caracter în interiorul şirului.

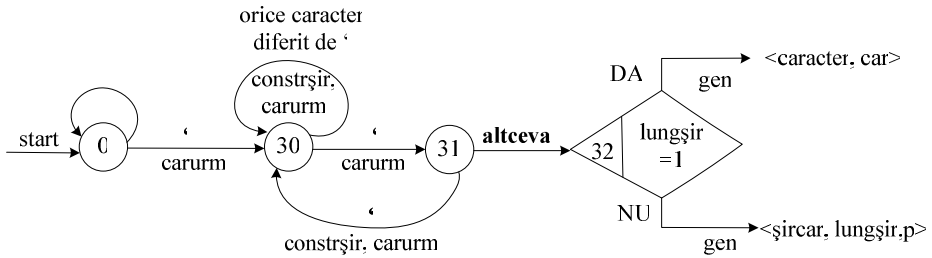


Fig. 3.15. Tratarea şirurilor de caractere

h) **Tratarea comentariilor** (fig. 3.16). Comentariile se ignoră ceea ce, în diagramă, înseamnă revenirea în starea 0.

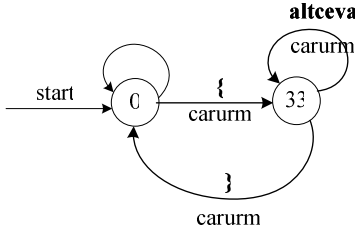
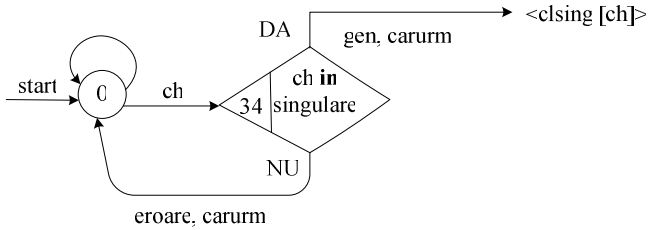


Fig. 3.16. Tratarea comentariilor

i) **Tratarea tiparelor formate din caractere izolate** (fig. 3.17). Aceste caractere sunt tratare în mod unitar, prin cuprinderea lor în mulțimea *singulare*. Codul lexical corepunzător se extrage din tabloul *clsing*. Fiind ultima diagramă a analizorului, pentru cazul în care caracterul curent nu este prezent în mulțimea *singulare*, se semnalează eroare şi se revine în starea 0.



singulare := [‘+’, ‘-‘, ‘*’, ‘/’, ‘(‘, ‘)’, ‘=’, ‘,’, ‘|‘, ‘]’, ‘;’]

Fig. 3.17. Tratarea tiparelor formate din caractere izolate