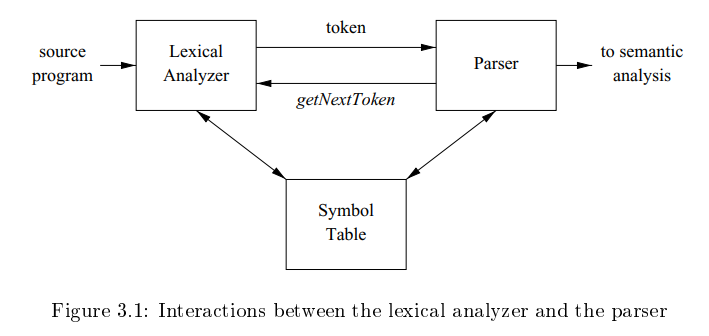
Ca primă fază a unui compilator, sarcina principală a analizatorului lexical este să citească caracterele codului sursă, să le grupeze în lexeme și să producă o secvență de tokeni pentru fiecare lexem. Tokenii sunt trimiși către parsator pentru analiza de sintaxă. Analizatorul lexical interacționeză cu tabela de simboluri. Atunci când analizatorul lexical descoperă un lexem care constituie un identificator, introduce acel lexem în tabelă.[[1]](#footnote-1)



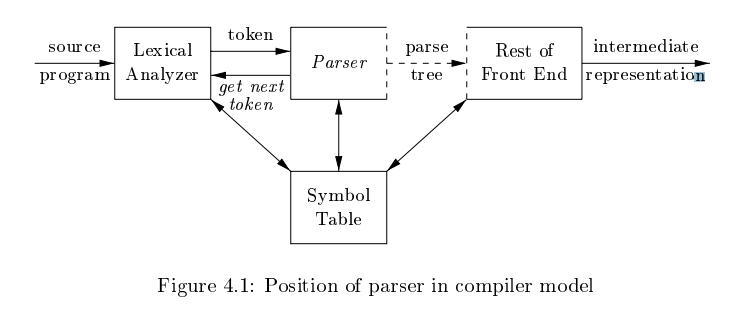
Analizatorii lexicali se divid în două procese:

a) Scanarea constă din procesele simple care nu necesită tokenizarea intrării, cum ar fi ștergerea comentariilor și compactarea spațiilor consecutive într-unul singur.

b) Analiza lexicală propriu-zisă este porțiunea mai complexă, care produce tokeni din codul procesat de scanner.[[2]](#footnote-2)

---

Parsatorul obține un șir de tokeni de la analizatorul lexical, așa cum se arată în Fig. 4.1, și verifică că șirul de tokeni poate fi generat de gramatica limbajului. Parsatorul va raporta orice erori de sintaxă într-un mod inteligibil. Pentru programele bine formate, analizatorul construiește un arbore de parsare și îl transmite restului compilatorului pentru procesare ulterioară. Parsatorul și restul părții de frontend pot fi implementate într-un singur modul.[[3]](#footnote-3)



Rezultatul parsatorului este o reprezentare a arborelui de parsare pentru șirul de tokeni care provine de la analizatorul lexical. Există o serie de sarcini care pot fi efectuate în timpul parsării, cum ar fi colectarea de informații despre diverse simboluri în tabela de simboluri, efectuarea verificării tipului variabilelor și a altor tipuri de analiză semantică și generarea de cod intermediar.[[4]](#footnote-4)

---

fazele ulterioare ale compilatorului trebuie să analizeze rezultatul parsatorului, pentru a asigura conformitatea programului cu regulile care nu sunt verificate de parsator[[5]](#footnote-5)

O gramatică pentru C sau Java nu distinge între identificatorii care sunt șiruri de caractere diferite. În schimb, toți identificatorii sunt reprezentați printr-un simbol precum id în gramatică. Într-un compilator pentru un astfel de limbaj, faza de analiză semantică verifică dacă identificatorii sunt declarați înainte de a fi utilizați.[[6]](#footnote-6)

---

GARBAGE

Informațiile sunt asociate cu un element constitutiv al unui limbaj prin atașarea de atribute la simbolului gramatical care reprezintă elementul. O definiție direcționată de sintaxă specifică valorile atributelor prin asocierea regulilor semantice regulilor de producție. De exemplu, o translație în formă postfixă ar putea avea o producție și o regulă

PRODUCTION E->E1 + T SEMANTIC RULE E ->code = E1->code || T ->code || ’+’

Această regulă de producție are două neterminale, E și T; indicele din E1 distinge apariția lui E în corpul regulii de producție de apariția lui E de la începutul regulii. Atât E cât și T au un atribut code de tip șir de caractere. Regula semantică specifică că șirul E->code este format prin concatenarea E1->code, T ->code și caracterul ’+’ .[[7]](#footnote-7)

Cea mai uzuală abordare a traducerii direcționate de sintaxă este de a construi un arbore de sintaxă și de a calcula valorile atributelor pentru nodurile arborelui, traversându-l.[[8]](#footnote-8)

Regulile unei definiții direcționate de sintaxă sunt aplicate prin construirea unui arbore de parsare și evaluarea tuturor atributelor pentru fiecare dintre nodurile arborelui de parsare. Un arbore de parsare, care prezintă valoarea (valorile) atributelor sale, se numește arbore de parsare adnotat.

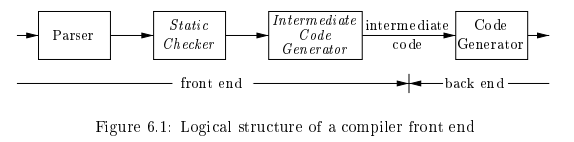
Atributele sintetizate pot fi evaluate în orice ordine de jos în sus, cum ar fi cea a unei traversări postordine a arborelui de parsare.[[9]](#footnote-9)

Tehnicile de translatare direcționată de sintaxă sunt aplicate în verificarea tipurilor și generarea intermediară de cod.[[10]](#footnote-10)

Fiecare nod dintr-un arbore de sintaxă reprezintă un element constitutiv. Copiii nodului reprezintă componente semnificative ale elementului.[[11]](#footnote-11)

----

Partea de frontend a compilatorului este organizată ca în Fig. 6.1, unde parsarea, verificarea statică și generarea codului intermediar se fac secvenţial. Vom folosi formalismele direcționate de sintaxă pentru a specifica verificarea și traducerea. Schemele de translatare pot fi implementate în timpul parsării de jos în sus folosind tehnicile din Capitolul 5. Schemele pot fi implementate prin crearea unui arbore de sintaxă și apoi parcurgerea acestuia.



Verificarea statică include verificarea tipului variabilelor declarate pentru a se asigura că operatorii sunt aplicați operanzilor compatibili. Include, de asemenea, orice verificări sintactice care rămân după parsare.[[12]](#footnote-12)

O acțiune semantică asociată unei reguli de producție poate fi reprezentată de crearea unei intrări în tabela de simboluri prin executarea unui fragment de cod.[[13]](#footnote-13)

Altele

ANALIZA LEXICALA

Prima fază a unui compilator se numește analiză lexicală sau scanare. Analizatorul lexical citește fluxul de caractere care formează programul sursă și grupează personajele în secvențe semnificative numite lexeme. Pentru fiecare lexemă, analizatorul lexical produce ca ieșire un simbol al formei

token-name; attribute-value

că trece la faza ulterioară, analiza sintaxei. În token, prima componentă nume-jeton este un simbol abstract care este utilizat în timpul sintaxanalizei, iar a doua componentă atribut-valoare indică o intrare în tabelul de simboluri pentru acest simbol. Informațiile din tabelul de simboluri sunt necesare pentru analiza semantică și generarea de cod. De exemplu, să presupunem că un program sursă conține instrucțiunea de atribuire

poziție = inițială + rata \* 60

Caracterele din această sarcină ar putea fi grupate în următoarele lexeme și mapate în următoarele simboluri transmise analizorului de sintaxă: 1. poziție este un lexem care ar fi mapat într-un token hid; 1i, unde este un simbol abstract care reprezintă identi er și 1 indică la intrarea tabelului de simboluri pentru poziție. Intrarea din tabelul de simboluri pentru un identificator conține informații despre identificator, cum ar fi numele și tipul acestuia. 2. Simbolul de atribuire = este un lexem care este mapat în simbolul h=i. Deoarece acest simbol nu are nevoie de valoare-atribut, am omis a doua componentă. Am fi putut folosi orice simbol abstract, cum ar fi atribuirea pentru numele simbolului, dar pentru comoditate notațională am ales să folosim lexemul în sine ca nume al simbolului abstract. 3. initiala este un lexem care este mapat in token hid; 2i, unde 2 indică intrarea din tabelul de simboluri pentru inițială. 4. + este un lexem care este mapat în simbolul h+i. 5. rate este un lexem care este mapat în token hid; 3i, unde 3 indică intrarea în tabelul de simboluri pentru rata. 6. \* este un lexem care este mapat în simbolul hi. 7. 60 este un lexem care este mapat în simbolul h60i.1 Spațiile libere care separă lexemele ar fi eliminate de analizorul lexical. Figura 1.7 prezintă reprezentarea instrucțiunii de atribuire (1.1) după analiza lexicală ca succesiune de jetoane

hid; 1i h=i hid; 2i h+i hid; 3i hi h60i[[14]](#footnote-14)

---

ANALIZA DE SINTAXĂ

A doua fază a compilatorului este analiza sintaxei sau analizarea. Analizatorul folosește primele componente ale jetoanelor produse de analizorul lexical pentru a crea o reprezentare intermediară sub formă de arbore care descrie structura gramaticală a fluxului de jetoane. O reprezentare tipică este un arbore de sintaxă în care fiecare nod interior reprezintă o operație, iar copiii nodului reprezintă argumentele operației. Un arbore de sintaxă pentru fluxul de simboluri (1.2) este prezentat ca rezultat al analizorului sintactic în Fig. 1.7. Acest arbore arată ordinea în care operațiunile din atribuire

position = initial + rate \* 60

urmează a fi efectuate. Arborele are un nod interior etichetat cu hid; 3i ca copil stâng și întregul 60 ca copil drept. Nodul s-a ascuns; 3i reprezintă rata de identificare. Nodul etichetat explică faptul că mai întâi trebuie să înmulțim valoarea ratei cu 60. Nodul etichetat + indică faptul că trebuie să adăugăm rezultatul acestei înmulțiri la valoarea inițialei. Rădăcina arborelui, etichetată =, indică faptul că trebuie să stocăm rezultatul acestei adunări în locația pentru poziția identi er. Această ordonare a operațiilor este în concordanță cu convențiile obișnuite ale aritmeticii care ne spun că înmulțirea are o prioritate mai mare decât adunarea și, prin urmare, înmulțirea trebuie efectuată înainte de adunare.[[15]](#footnote-15)

---

ANALIZA SEMANTICĂ

Analizatorul semantic folosește arborele de sintaxă și informațiile din tabelul de simboluri pentru a verifica programul sursă pentru coerența semantică cu definiția limbajului. De asemenea, adună informații de tip și le salvează fie în arborele de sintaxă, fie în tabelul de simboluri, pentru utilizare ulterioară în timpul generării codului intermediar.[[16]](#footnote-16)

---

MANAGEMENT-UL TABELEI DE SIMBOLURI

O funcție esențială a unui compilator este de a înregistra numele variabilelor utilizate în programul sursă și de a colecta informații despre diferitele atribute ale fiecărui nume. Aceste atribute pot oferi informații despre spațiul de stocare alocat pentru un nume, tipul său, domeniul său de aplicare (unde în program poate fi utilizată valoarea sa) și, în cazul numelor de proceduri, cum ar fi numărul și tipurile argumentelor sale, metoda de trecând fiecare argument (de exemplu, după valoare sau prin referință) și tipul returnat. Tabelul de simboluri este o structură de date care conține o înregistrare pentru fiecare nume de variabilă, cu câmpuri pentru atributele numelui. Structura datelor ar trebui să fie concepută pentru a permite compilatorului să găsească rapid înregistrarea pentru fiecare nume și să stocheze sau să recupereze rapid datele din acea înregistrare. Tabelele cu simboluri sunt discutate în Capitolul 2.

1. P. 109 [↑](#footnote-ref-1)
2. P. 110 [↑](#footnote-ref-2)
3. P. 192 [↑](#footnote-ref-3)
4. P. 193 [↑](#footnote-ref-4)
5. P. 209 [↑](#footnote-ref-5)
6. P. 216 [↑](#footnote-ref-6)
7. P. 304 [↑](#footnote-ref-7)
8. P. 305 [↑](#footnote-ref-8)
9. P. 307 [↑](#footnote-ref-9)
10. P. 318 [↑](#footnote-ref-10)
11. P. 353 [↑](#footnote-ref-11)
12. P. 357 [↑](#footnote-ref-12)
13. P. 376 [↑](#footnote-ref-13)
14. Inceput analiza lexicala [↑](#footnote-ref-14)
15. Inceput analiza de sintaxa [↑](#footnote-ref-15)
16. Inceput analiza semantica [↑](#footnote-ref-16)