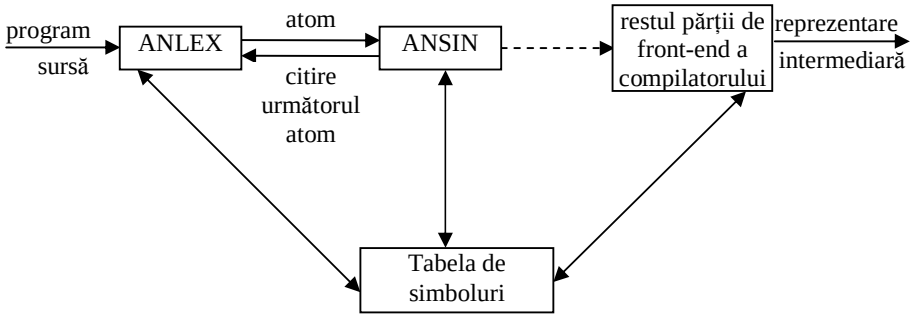
ANALIZA SINTACTICĂ

Rolul Analizorului sintactic



Poziția analizorului sintactic în modelul compilatorului

Există trei tipuri generale de analizoare sintactice bazate pe gramatici:

1) **AS universale**: Pot analiza orice gramatică dar sunt ineficiente din punct de vedere practic.
2) **AS descendente** ("top-down")
3) **AS ascendente** ("bottom-up")

Tratarea erorilor sintactice

Scopurile urmărite la tratarea erorilor sintactice sunt:

- **raportarea precisă și clară** a prezenței erorilor;
- **revenirea rapidă din eroare**, necesară pentru detectarea următoarei erori;
- **nu trebuie să se reducă** semnificativ **viteza** de prelucrare a programelor corecte.

Există patru **strategii** diferite **de revenire din eroare**:

a) **modul de panică** (simplu de implementat, deci frecventă)
b) revenire la nivel de **propoziție**
c) utilizarea **producții de eroare**
d) efectuarea de **corecții globale**

a) După detectarea unei erori se elimină din intrarea unul sau mai multe simboluri până când se ajunge la un atom special de sincronizare. Din acest punct, analiza va fi reluată normal.

b) La descoperirea unei erori, AS va efectua corecții locale în șirul de intrare. Adică, va înlocui un prefix al intrării rămase cu un şir care să permită analizorului să-şi continue funcționarea.

c) Dacă se cunosc suficient de precis tipurile de erori care pot fi întâlnite, se poate suplimenta gramatica limbajului cu producții care să genereze construcțiile eronate.

d) Modalitate, mai mult teoretică, de analiză a erorilor, deoarece tratează un program eronat x în comparație cu acel program corect y, care se poate obține din x printr-un număr minim de transformări.

Factorizarea la stânga

- Este o transformare utilă pentru a obține gramatici adecvate analizoarelor predictive.
- Necesitatea acestei transformări apare atunci când nu este evident care alternativă să fie folosită pentru expandarea unui neterminal A.

Exemplu:
Considerăm cele două alternative pentru **if**:
<instr>→**if** <expr> **then** <instr> **else** <instr> |
if <expr> **then** <instr>

Când în intrare apare atomul **if** nu ştim încă dacă trebuie să folosim prima sau a doua alternativă pentru instr. Cu notația generală:

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$, α=şir nevid, nu se poate face selecția între cele două alternative: $\alpha\beta_1$ sau $\alpha\beta_2$. Pentru a amâna decizia, se introduce un nou neterminal, A', şi se transformă producțiile astfel:

$A \rightarrow \alpha A'$
$A' \rightarrow \beta_1 \mid \beta_2$

Algoritmul de factorizare general este următorul:
Intrare: G
Ieşire: G' echivalent cu factorizarea la stânga
Metoda: Pentru fiecare neterminal A se caută cel mai lung prefix α comun pentru două sau mai multe din alternativele lui A. Dacă $\alpha \neq \varepsilon$, înlocuim toate producțiile lui $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \Upsilon$, unde Υ reprezintă toate alternativele care nu încep cu α, prin:

$A \rightarrow \alpha A' \mid \Upsilon$
$A' \rightarrow \beta_1 \mid \beta_2 \ldots \mid \beta_n$

Se repetă acest procedeu până când nu mai există neterminale având prefix comun.

Reluarea exemplului:
$S \rightarrow iEtS \mid iEtSeS \mid a$; (gramatica pentru **if**)
$S \rightarrow iEtSS' \mid a$
$S' \rightarrow eS \mid \varepsilon$
$E \rightarrow b$
b – număr boolean.

Eliminarea recursivității la stânga

O gramatică este recursivă la stânga dacă are cel puțin un neterminal A pentru care există o derivare de forma:

$$A \overset{+}{\Rightarrow} A\alpha \qquad (\alpha \text{ - şir oarecare})$$

Cazul în care gramatica are producții de forma $A \rightarrow A\alpha$ constituie **recursivitate de stânga imediată**. El se rezolvă simplu prin introducerea unui neterminal nou A' şi prin mărirea numărului producțiilor.

Exemplu: $A \rightarrow A\alpha \mid \beta \Rightarrow A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \varepsilon$

Exemplu: gramatica pentru expresii aritmetice, recursivă la stânga pentru neterminalele E şi T.

$$\begin{array}{lll} E \rightarrow E+T \mid T & E \rightarrow TE' & E' \rightarrow +TE' \mid \varepsilon \\ T \rightarrow T*F \mid F & \Rightarrow T \rightarrow FT' & T' \rightarrow *FT' \mid \varepsilon \\ F \rightarrow (E) \mid \mathbf{id} & F \rightarrow (E) \mid \mathbf{id} & \end{array}$$

.....

Recursivitatea de stânga imediată se poate elimina indiferent de numărul producțiilor pentru A, prin următoarea metodă:

– Se grupează producțiile pentru A în următoarea formă:
  $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

unde nici un β_i nu începe cu A.

– Se înlocuiesc producțiile pentru A prin următoarele producții echivalente:
  $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$
  $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$

Această metodă elimină orice recursivitate de stânga imediată cu condiția ca nici un α_i să nu fie ε.

Exemplu de gramatică în care recursivitatea la stânga nu este imediată:
$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid \varepsilon$

Algoritmul de mai jos elimină complet recursivitatea la stânga dintr-o gramatică dacă gramatica nu are cicluri (derivări de forma $A \overset{+}{\Rightarrow} A$) şi nici producții ε (de forma $A \rightarrow \varepsilon$). Ambele aceste restricții pot fi eliminate prin metode specifice, aplicate anterior.

Algoritm: Eliminarea recursivității de stânga.

Intrare: o gramatică G fără cicluri şi producții ε.
Ieşire: o gramatică G', echivalentă cu G, fără recursivitate de stânga.
Metoda:

1. Se aranjează neterminalele gramaticii într-o anumită ordine şi se numerotează:
   $A_1, A_2, ..., A_n$
2. **for** i:=1 **to** n **do begin**
   for j:=1 **to** i-1 **do**
   ∗ înlocuieşte fiecare producție de forma $A_i \rightarrow A_j\gamma$ prin producțiile $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid ... \mid \delta_k\gamma$, unde $A_j \rightarrow \delta_1 \mid \delta_2 \mid ... \mid \delta_k$ sunt toate producții A_j curente
   ∗ elimină recursivitatea la stânga imediată pentru producția A_i
   end

Se remarcă faptul că gramatica nerecursivă la stânga rezultată poate avea producții ε.

Exemplu:

– gramatica anterioară

Algoritmul nu este garantat datorită producției $A \rightarrow \varepsilon$ (care totuşi nu dăunează în acest caz).

Ordonăm neterminalele S, A.

Nu există recursivitate de stânga imediată între producțiile S. Rezultă că nu se întâmplă nimic pentru i=1. Pentru i=2, se înlocuiesc producțiile **S** în **A→Sd**

$\Rightarrow A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$

Eliminând recursivitatea de stânga imediată între producțiile A, rezultă:

$S \rightarrow Aa \mid b$
$A \rightarrow bdA' \mid A'$
$A' \rightarrow cA' \mid adA' \mid \varepsilon$

Analiza sintactică cu descendenți recursivi

Analiza sintactică descendentă poate fi considerată ca o încercare de a găsi o derivare stânga a unui şir de intrare dat. Ea este echivalentă cu o încercare de construire a arborelui sintactic pentru şirul de intrare respectiv începând de la rădăcină spre frunze (de sus în jos), construind nodurile în preordine.

Mecanismul de analiză

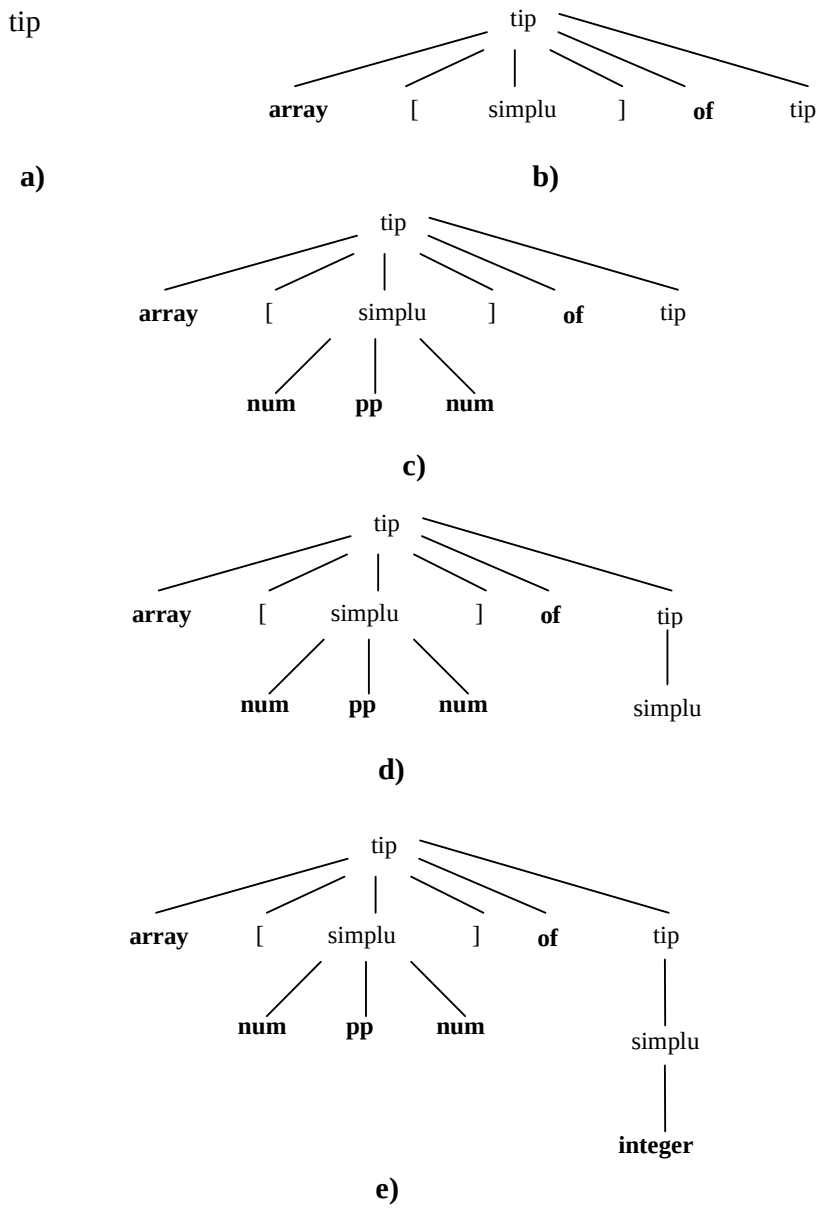
Să considerăm în continuare o gramatică reprezentând o submulțime a tipurilor din Pascal care este potrivită pentru AS descendentă:

tip→simplu | ↑**id** | **array** [simplu] **of** tip
simplu → **integer** | **char** | **num pp num** (**pp**='..')

Construcția descendentă a arborelui sintactic se face începând cu rădăcina care va fi etichetată cu neterminalul de start şi apoi executând următorii doi paşi, până când toate nodurile fii sunt etichetate cu terminale (noduri frunze):

1. La nodul curent **n** etichetat cu neterminalul A, se selectează una din producțiile pentru A şi se construiesc fiii lui **n** pentru simbolurile din partea dreaptă a producției.

2. Se caută următorul nod la care se construieşte un subarbore.



Paşi în construcția unui arbore sintactic

Baleierea şirului: **array** [**num pp num**] **of integer**

Intrare	Arbore sintactic
array [**num pp num**] **of integer** ↑	tip ↑
	tip: **array** [simplu] **of** tip ↑
array [**num pp num**] **of integer** ↑	tip: **array** [simplu] **of** tip ↑

AS descendentă în timpul baleierii intrării de la stânga la dreapta

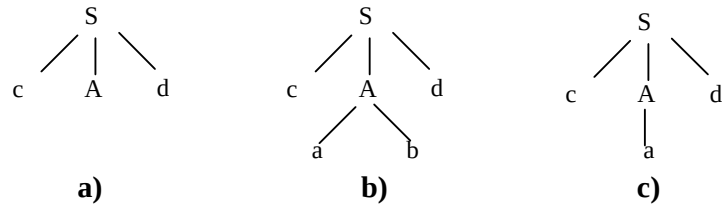
Exemplu: Să considerăm gramatica:

$S \rightarrow cAd$

$A \rightarrow ab \mid a$

şi şirul de intrare: w=cad

Inițial, arborele are un singur nod, etichetat S, şi pointerul de intrare indică spre c. Apoi se expandează arborele, conform primei producții, ca în figură:



Paşi în analiza sintactică cu revenire

<u>Observație</u>. O gramatică recursivă la stânga poate determina ca un analizor sintactic descendent recursiv, chiar şi pentru varianta cu reveniri, să intre în ciclu infinit. Aceasta se întâmplă în situația în care, expandând neterminalul recursiv, ajungem din nou la expandarea lui, fără a avansa în şirul de intrare.

Analizorul sintactic predictiv

Analizorul sintactic descendent recursiv de tip predictiv **este format dintr-un set de proceduri recursive. Se asociază câte o astfel de procedură cu fiecare neterminal al gramaticii**. Pe baza SA, se poate selecta în mod univoc procedura care trebuie apelată. Secvența de proceduri apelate pentru prelucrarea şirului de intrare defineşte implicit arborele sintactic concret corespondent intrării.

```
    procedure coresp(t:atom);
    begin
        if sa=t then
            sa:=următorul
                else eroare
    end;
procedure tip;
    begin
        if sa in [integer, char, num] then simplu
            else if sa='↑' then begin
                coresp('↑'); coresp(id)
            end else if sa=array then begin
                    coresp(array); coresp('['); simplu;
                    coresp(']'); coresp (of); tip
                end else eroare
end;
procedure simplu;
    begin
        if sa=integer then coresp (integer)
            else if sa=char then coresp(char)
                else if sa=num then begin
                    coresp(num); coresp(pp); coresp(num);
                end else eroare
end;
```

Cod pentru un analizor sintactic predictiv

- AS predictivă se bazează pe informații despre simbolurile care pot fi primele în şirurile generate de partea dreaptă a unei producții.

- Notând cu α partea dreaptă a unei producții pentru neterminalul A, se defineşte PRIM(α) ca fiind mulțimea de atomi ce apar ca prime simboluri pentru unul sau mai multe simboluri generate din α. Dacă α este şirul vid, sau poate genera şirul vid, atunci şi ε se include în PRIM(α).

  Exemple: PRIM(simplu)=[**integer**, **char**, **num**]
  PRIM(↑**id**)=[↑]
  PRIM(**array** [simplu] **of** tip=[**array**]

- În practică, multe părți drepte ale unor producții încep cu un atom, ceea ce simplifică determinarea lui PRIM. Mulțimile PRIM sunt necesare întotdeauna când avem cel puțin două producții pentru un neterminal.
  Exemplu: $A \rightarrow \alpha$ $A \rightarrow \beta$

<u>Observație</u>. Analiza sintactică descendentă recursivă fără reveniri cere ca PRIM(α) şi PRIM(β) să fie disjuncte.

Proiectarea unui ASP

Un ASP este **un program constând din câte o procedură pentru fiecare neterminal**. Fiecare procedură execută câte două acțiuni:

1) Pe baza SA, se decide care producție se va utiliza, şi anume, se va utiliza producția cu partea dreaptă α dacă SA este în PRIM(α). Dacă există un conflict (ambiguitate) între două părți drepte pentru acelaşi SA atunci gramatica nu poate fi analizată prin această metodă. Dacă SA nu aparține lui PRIM(α) pentru nici o parte dreaptă, se va utiliza producția cu ε în partea dreaptă (dacă există).

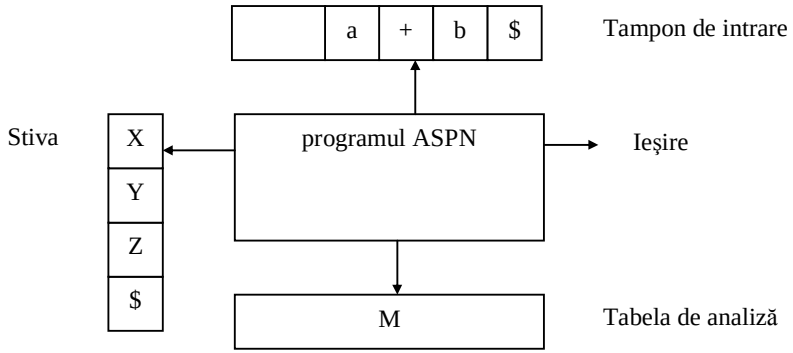
2) Activitatea procedurii va "mima" (simula) partea dreaptă a producției alese. Un neterminal în partea dreaptă va determina apelul procedurii pentru acel neterminal iar un atom ce corespunde cu SA determină citirea următorului atom. În cazul în care atomul din producție nu corespunde cu SA, se semnalează eroare.

Analizorul sintactic predictiv, nerecursiv

Analiza sintactică predictivă poate fi implementată şi într-o variantă nerecursivă (ASPN). **Eliminarea apelurilor recursive se obţine prin gestiunea în mod explicit a unei stive de lucru**.

Structura ASPN

Schematic, ASPN are structura din figura de mai jos:



- **Tamponul de intrare** conţine şirul de analizat, terminat cu un marcaj special de sfârşit ($).

- **Stiva** conţine o secvenţă de simboluri gramaticale (neterminale şi terminale) care se modifică pe parcursul analizei. Capătul inferior al stivei este, de asemenea, marcat prin $. La pornirea analizei (starea iniţială), stiva conţine doar simbolul de start al gramaticii, urmat de $.

- **Tabela de analiză** (M) este o matrice ale cărei linii corespund neterminalelor gramaticii iar coloanele corespund simbolurilor terminale, între care se include şi $. În cele ce urmează, un element oarecare al matricii va fi notat M[A,a], unde A este neterminal iar a este terminal sau simbolul $. Un astfel de element fie conţine producţia ce trebuie aplicată fie nu conţine nimic (intrare de eroare).

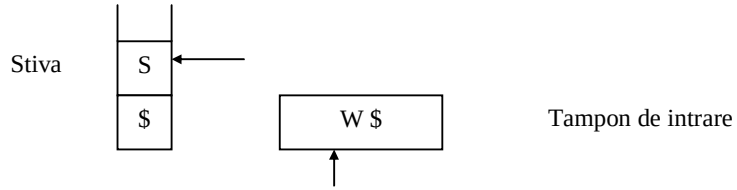
- **Ieşirea** poate fi reprezentată prin mesaje de eroare sau prin acţiuni ce se execută atunci când se aplică o anumită producţie şi care pot avea ca obiect construirea arborelui sintactic, generarea de cod intermediar etc.

Funcţionarea ASPN

Notând cu *X* simbolul din vârful stivei şi cu *a* simbolul curent de intrare, funcţionarea ASPN poate fi rezumată astfel:

1) Dacă X=a=$, analiza se încheie cu succes.
2) Dacă X=a≠$, se extrage X din stivă şi se avansează pointerul în şirul de intrare (în această situaţie s-a analizat cu succes un simbol al şirului de intrare dat).
3) Dacă X este neterminal, se cercetează elementul M[X,a] al tabelei de analiză. În funcţie de conţinutul acestui element se disting două situaţii:
   a) M[X,a]={X→UVW}, se înlocuieşte X în vârful stivei cu WVU (cu U în vârf)
   b) M[X,a]={} - eroare, se apelează o procedură de tratare a erorii şi revenire

Starea iniţială a stivei şi a tamponului de intrare:



Algoritmul propriu-zis:

∗ se poziţionează pointerul în intrare pe primul simbol din *w* ;
repeat
  ∗ fie *X* simbolul din vârful stivei şi *a* simbolul curent de intrare;
  if ∗ *X* este un terminal sau $ **then**
    if X=a **then**
      ∗ extrage *X* din stivă şi avansează pointerul de intrare
    else eroare()
  else {*X* - neterminal}
    if M[X,a]={X→$Y_1Y_2...Y_k$} **then begin**
      ∗ extrage *X* din stivă;
      ∗ introduce în stivă $Y_k,Y_{k-1},...,Y_1$ (cu Y_1 în vârf) ;
      ∗ tipăreşte producţia X→$Y_1Y_2...Y_k$
    end else eroare()
until X=$; {stiva s-a golit}
if ∗ simbolul curent de intrare ≠$ **then** eroare()

Exemplu: Se consideră gramatica:

$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to (E) \mid \mathbf{id}$

Tabela M este următoarea:

Neterminal	Simbol de intrare					
	id	**+**	*****	**(**	**)**	**$**
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to$ **id**			$F \to (E)$		

Operațiile pe care le efectuează analizorul de intrare pentru şirul de intrare **id+id*id** sunt următoarele:

Stiva	Intrare	Ieşire
$E	**id+id*id$**	
$E' T	**id+id*id$**	$E \to TE'$
$E' T'F	**id+id*id$**	$T \to FT'$
$E' T'**id**	**id+id*id$**	$F \to$ **id**
$E' T'	**+id*id$**	
$E'	**+id*id$**	$T' \to \varepsilon$
$E'T+	**+id*id$**	$E' \to +TE'$
$E'T	**id*id$**	
$E' T'F	**id*id$**	$T \to FT'$
$E' T'**id**	**id*id$**	$F \to$ **id**
$E' T'	***id$**	
$E' T'F*	***id$**	$T' \to *FT'$
$E' T'F	**id$**	
$E' T'**id**	**id$**	$E \to$ **id**
$E' T'	$	
$E'	$	$T' \to \varepsilon$
$	$	$E' \to \varepsilon$
		succes

Funcțiile PRIM şi URM

Funcția PRIM se defineşte pentru orice şir α de simboluri gramaticale, astfel: PRIM(α) este mulțimea terminalelor care pot să fie primele în orice şir derivat din α. Dacă $\alpha \Rightarrow \varepsilon$, atunci şi ε se include în PRIM(α).

Analog, **funcția URM** este definită doar pe mulțimea neterminalelor, astfel: URM(A) este mulțimea terminalelor care pot să urmeze imediat după A într-o formă propozițională. Un terminal oarecare $a \in$ URM(A) dacă şi numai dacă există o derivare de forma $S \overset{*}{\Rightarrow} \alpha A a \beta$, pentru α şi β -oarecari. Dacă A poate fi simbolul cel mai din dreapta într-o formă propozițională, atunci se va include în URM(A) şi $.

Calculul mulțimilor PRIM(α) pentru şiruri de simboluri trebuie precedat de determinarea mulțimilor PRIM(X) pentru orice simbol gramatical (terminal şi neterminal) *X*. Aceasta se realizează prin aplicarea în mod repetat a următoarelor reguli până nu se mai pot adăuga noi terminale sau ε la nici o mulțime PRIM:

1) Dacă X este terminal, atunci PRIM(X)={X}.
2) Dacă $X \to \varepsilon$ este o producție a gramaticii, atunci se va include ε la PRIM(X).
3) Dacă X este neterminal şi gramatica include producția $X \to Y_1 Y_2 ... Y_k$, atunci se include *a* în PRIM(X) dacă pentru un *i* oarecare, *a* este în PRIM(Yi) iar PRIM(Y_1)....PRIM(Y_{i-1}) includ, toate, pe ε ($Y_1 Y_2 ... Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$).
   În plus, dacă $\varepsilon \in$ PRIM(Y_j), $\forall j=1, 2, ... , k$ atunci se adaugă ε la PRIM(X).

Modul de calcul sugerat de această a treia regulă se poate rezuma astfel: PRIM(Y_1) se include cu certitudine în PRIM(X), mai puțin, eventual, ε. Dacă $\varepsilon \in$ PRIM(Y_1) atunci se include în PRIM(X) şi PRIM(Y_2), altfel calculul se opreşte, ş.a.m.d.

Calculul mulțimilor URM(A), pentru toate neterminalele A ale gramaticii se realizează prin aplicarea în mod repetat a următoarelor reguli, până când nu se mai pot adăuga elemente noi la nici o mulțime URM:

1) Marcajul de sfârşit al şirului de intrare, $, se include în URM(S), unde *S* este simbolul de start al gramaticii.
2) Dacă în gramatică există o producție $A \to \alpha B \beta$, atunci elementele din PRIM(β), mai puțin ε, se includ în URM(B).
3) Dacă în gramatică există o producție $A \to \alpha B$ sau $A \to \alpha B \beta$ cu $\varepsilon \in$ PRIM(β), atunci toate elementele din URM(A) sunt prezente şi în URM(B).

Exemplu: Aceeaşi gramatică din exemplul anterior.

PRIM(F)={**(**,**id**} prin regula 3.
PRIM(T')={**∗**,**ε**} prin regulile 3 şi 2.
PRIM(T)=PRIM(F)={**(**,**id**} prin regula 3.
PRIM(E')= {**+**,**ε**} prin regulile 3 şi 2.
PRIM(E)=PRIM(T)={**(**,**id**} prin regula 3.

Calculul mulțimilor URM porneşte de la:
URM(E)={**$**} - regula 1.
URM(T)=PRIM(E')-{**ε**}={+} - regula 2.
URM(F)=PRIM(T')-{**ε**}={**∗**} - regula 2.
) se include în URM(E) – regula 2 ⇒ URM(E)={**$**,**)**}

Aplicăm acum în mod repetat regula 3:
URM(E')=URM(E) ⇒ URM(E')={**$**,**)**}
Din prima producție, ținând cont că E'→ε ⇒ elementele din URM(E) se includ în URM(T) ⇒ URM(T)={+,**$**,**)**}
Din cea de a treia producție ⇒ URM(T')=URM(T) ⇒ URM(T')={+,**$**,**)**}
În plus, deoarece T'=ε, elementele din URM(T) se includ în URM(F) ⇒ URM(F)={**∗**,+,**$**,**)**}
Din producția a patra, deoarece T'→ε, şi elementele lui URM(T') se includ în URM(F), ceea ce însă nu mai adaugă elemente noi.

Construirea tabelelor pentru ASPN

Algoritmul de construcție

Intrare: Gramatica G.
Ieşire: Tabela de analiză M.

Algoritmul propriu-zis:

1. Pentru fiecare producție A→α a gramaticii se execută paşii 2 şi 3.
2. Dacă a∈PRIM(α), se adaugă A→α la M[A,a].
3. Dacă ε∈PRIM(α), se adaugă A→α la M[A,b] pentru fiecare terminal *b* din URM(A). Dacă ε∈PRIM(α) şi $∈URM(A), se adaugă A→α la M[A,$].
4. Toate intrările necompletate în paşii 2 şi 3 se consideră intrări de **eroare**.

Exemplu: Gramatica anterioară.

Deoarece PRIM(TE')=PRIM(T)={**(**,**id**} ⇒ M[E,**(**]=M[E,**id**]={E→TE'}
Analog, M[E',+]={E'→+TE'}
Deoarece E'→ε şi URM(E')={**)**,$} ⇒ M[E',**)**]=M[E',**$**]={E'→ε}
ş.a.m.d.

⇒ tabela M prezentată în exemplul anterior (**pag. 13**).

Gramatici LL(1)

Exemplul 1: $E \to E+T \mid T$ PRIM(E)=PRIM(T)=PRIM(F)={**(**,**id**}
$T \to T*F \mid F$ URM(E)={+,**)**,**$**}
$F \to (E) \mid$ **id** URM(T)=URM(E)∪{∗}={∗,+,**)**,**$**}
URM(F)=URM(T)={∗,+,**)**,**$**}

Neterminal	Simbol de intrare					
	id	**+**	**∗**	**(**	**)**	**$**
E	E→E+T E→T			E→E+T E→T		
T	T→T∗F T→F			T→T∗F T→F		
F	F→**id**			F→(E)		

Exemplul 2: <instr>→**if** <expr> **then** <instr> <altern> | a
<altern>→**else** <instr> | ε
<expr>→b
PRIM(<instr>)={**if**,**a**}
PRIM(<altern>)={**else**,**ε**}
PRIM(<expr>)={**b**}
URM(<expr>)={then}
URM(<instr>)=PRIM(<altern>)\{**ε**}∪{**$**}={**else**,**$**}
URM(<altern>)=URM(<instr>)={**else**,**$**}

Tabela de analiză pentru această gramatică este următoarea:

Neterminal	Simbol de intrare					
	a	**b**	**else**	**if**	**then**	**$**
<instr>	<instr>→a			<instr>→**if** <expr>**then**..		
<altern>			<altern>→ε <altern>→ **else** <instr>			<altern>→ε
<expr>		<expr>→b				

GIC ce permit analiza descendentă fără reveniri, bazată pe inspectarea următoarelor *k* simboluri din şirul de intrare, formează clasa gramaticilor LL(k).

- L - baleierea intrării de la stânga la dreapta.
- L - reconstituirea unei derivări stânga.

În practică, apare frecvent cazul k=1, ceea ce înseamnă că analizorul decide ce acțiune să execute utilizând un singur simbol de intrare (cel curent). **Unei gramatici LL(1) îi corespunde o tabelă ASPN fără intrări multiple. Nici o gramatică ambiguă sau recursivă la stânga nu poate fi LL(1).**

Se poate arăta că o gramatică G este LL(1) dacă şi numai dacă, pentru fiecare neterminal *A*, producțiile sale $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_k$ îndeplinesc următoarele două condiții:

1. $PRIM(\alpha_i) \cap PRIM(\alpha_j) = \Phi$, $\forall i,j \in \overline{1,k}$, $i \neq j$ adică, luând în considerare oricare două părți drepte, nu pot deriva din amândouă şiruri care încep cu acelaşi terminal sau, eventual, şirul vid.

2. dacă $\alpha_i \Rightarrow \varepsilon$ atunci $PRIM(\alpha_j) \cap URM(A) = \Phi$, $\forall$ $i,j \in \overline{1,k}$, $i \neq j$ adică, dacă dintr-o anumită parte dreaptă derivă şirul vid atunci din oricare celelalte părți drepte nu pot deriva şiruri care să înceapă cu un terminal din URM(A).

Aplicând aceste două condiții la cele trei gramatici din exemplele de până acum rezultă că gramatica pentru expresii, cu recursivitatea de stânga eliminată, este LL(1) iar cele două gramatici din exemplele anterioare nu sunt LL(1).