FUNDAMENTELE PROIECTĂRII COMPILATOARELOR

CURS 4

Gianina Georgescu

CUPRINSUL CURSULUI 4

- Automate push-down
- Proprietăți ale gramaticilor independente de context și ale automatelor push down. Gramatici în forma normală Chomsky
- Translatoare stivă
- Metode generale de analiză sintactică: TOP DOWN și BOTTOM UP
- Metoda TOP DOWN: algoritm breadth first pentru derivări stângi
- Metoda TOP DOWN: algoritm depth first
- Metoda TOP-DOWN: algoritmul de parsare recursiv descendent general

ANALIZA SINTACTICĂ – PARSER, AUTOMAT PUSH-DOWN

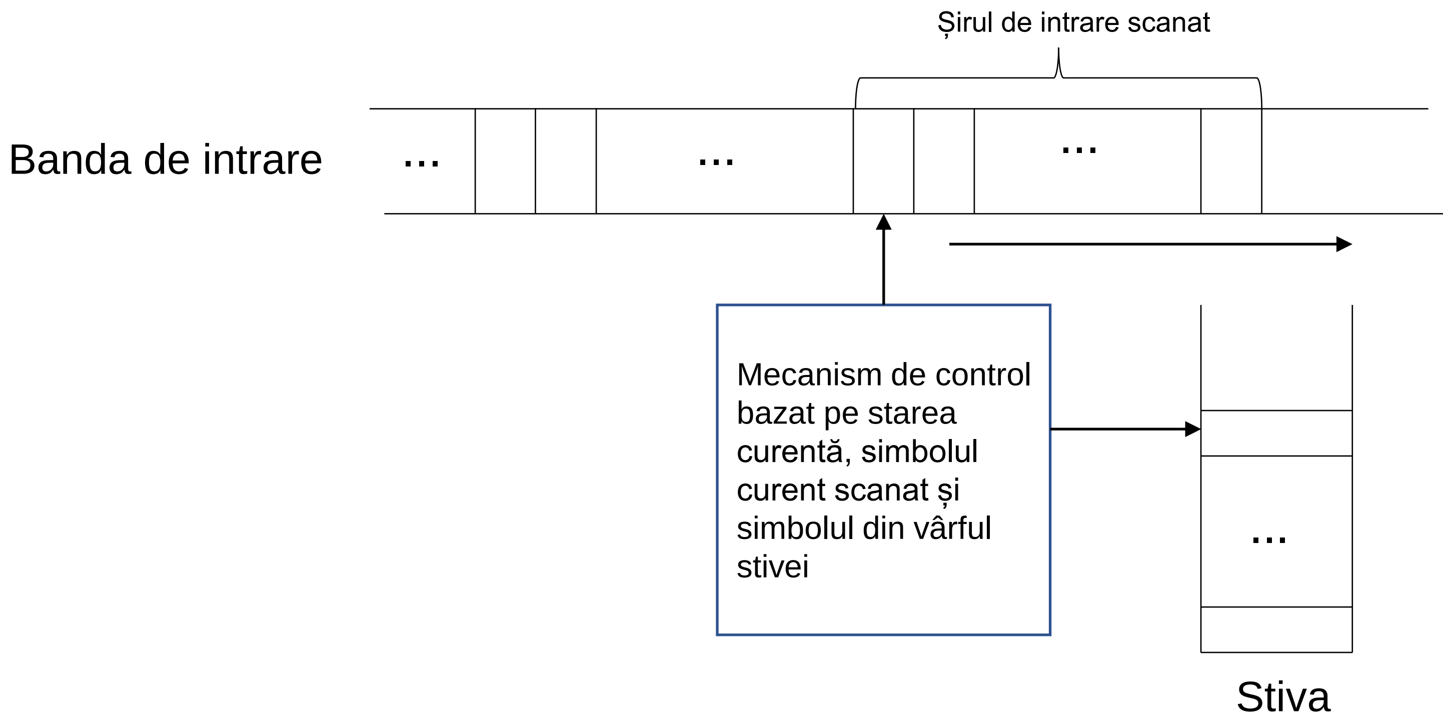
Analizorul sintactic (parserul) este de fapt implementarea unui **automat push-down.**

Definiție. Un **automat push down** are o structură de forma:

$$A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)\text{, unde:}$$

- Q este mulțimea stărilor
- Σ este alfabetul automatului
- Γ este alfabetul stivei
- $\delta: Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \mathcal{P}_{fin}(Q \times \Gamma^*)$
- q_0 este starea inițială a automatului
- $Z_0 \in \Gamma$ simbolul inițial al stivei
- $F \subseteq Q$ mulțimea stărilor finale

ANALIZA SINTACTICĂ – PARSER, AUTOMATUL PUSH-DOWN



ANALIZA SINTACTICĂ –AUTOMATUL PUSH-DOWN

- **Descriere instantanee** (instanță) a lui A

  $(p, x, \alpha), p \in Q$ starea curentă a lui A

  $x \in \Sigma^*$ șirul curent scanat din intrare

  $\alpha \in \Gamma^*$ conținutul stivei

- **Mișcare a lui A:**

  $(p, ax, Z\alpha) \vdash (q, x, \beta\alpha)$ ddacă $(q, \beta) \in \delta(p, a, Z)$

  pentru $p, q \in Q, a \in \Sigma \cup \{\lambda\}, x \in \Sigma^*, \mathrm{Z} \in \Gamma, \alpha, \beta \in \Gamma^*$

- **Închiderea reflexivă și tranzitivă** a relației $\vdash$ este notată cu $\vdash^*$
- **Limbajul acceptat de A cu stări finale:**

  $$L(A) = \{w \in \Sigma^* | (q_0, w, Z_0) \vdash^* (\mathrm{q}, \lambda, \alpha), q \in F, \alpha \in \Gamma^*\}$$

- **Limbajul acceptat de A cu vidarea stivei:**

  $$L_\lambda(A) = \{w \in \Sigma^* | (q_0, w, Z_0) \vdash^* (\mathrm{q}, \lambda, \lambda), q \in Q\}$$

ANALIZA SINTACTICĂ – AUTOMATUL PUSH-DOWN

- **Propoziție:** Pentru un automat push-down cu stări finale există un automat push-down cu vidarea stivei echivalent și reciproc.

- **Automat push-down determinist:**

  Spunem că $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ este **determinist** dacă:

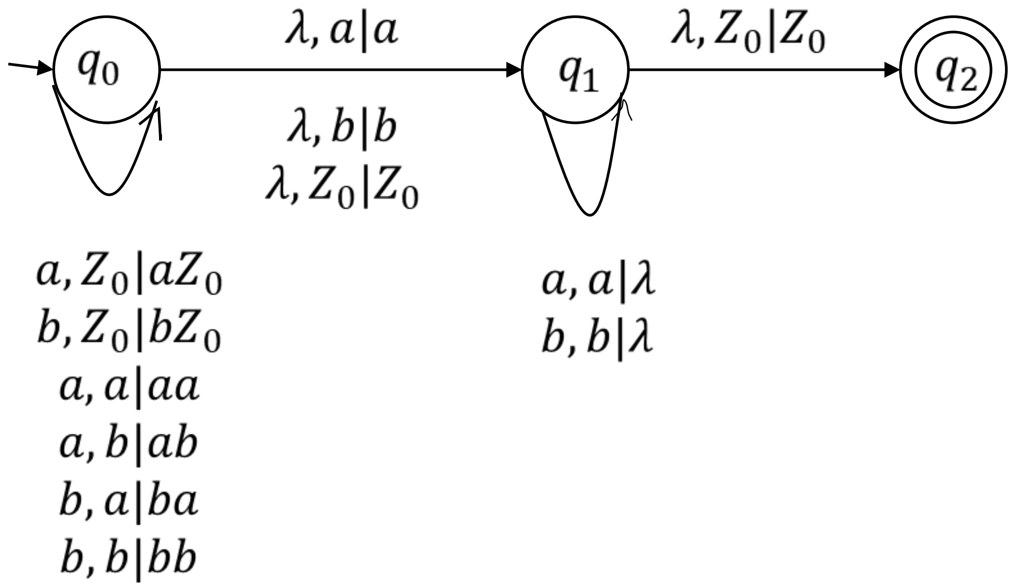
  $$|\delta(p, a, Z)| + |\delta(p, \lambda, Z)| \leq 1, \forall p \in Q, \forall a \in \Sigma, \forall Z \in \Gamma$$

- **În cazul limbajelor de programare, este important ca parserul să fie construit pe baza unui automat determinist ce corespunde gramaticii (neambigue) a sintaxei limbajului.**
- **Algoritmii bazați pe automate deterministe sunt liniari**

ANALIZA SINTACTICĂ – AUTOMATUL PUSH-DOWN

Exemplu: Automatul nedeterminist

$A = (\{q_0, q_1, q_2\}, \{\mathrm{a}, \mathrm{b}\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_2\})$ cu stări finale



recunoaște limbajul $L(A) = \{ww^R | w \in \{a, b\}^*\}$

$(q_0, abba, Z_0) \vdash (q_0, bba, aZ_0) \vdash (q_0, ba, baZ_0) \vdash (q_1, ba, baZ_0) \vdash$
$(q_1, a, aZ_0) \vdash (q_1, \lambda, Z_0) \vdash (q_2, \lambda, Z_0) \vdash \text{accept}$

Pentru automatul A de mai sus putem scrie funcția de tranziție, δ, redată mai sus prin graf, astfel:

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$; $\delta(q_0, b, Z_0) = \{(q_0, bZ_0)\}$; $\delta(q_0, a, a) = \{(q_0, aa)\}$; $\delta(q_0, a, b) = \{(q_0, ab)\}$; $\delta(q_0, b, a) = \{(q_0, ba)\}$; $\delta(q_0, b, b) = \{(q_0, bb)\}$; $\delta(q_0, \lambda, Z_0) = \{(q_1, Z_0)\}$;

$\delta(q_0, \lambda, a) = \{(q_1, a)\}$; $\delta(q_0, \lambda, b) = \{(q_1, b)\}$; $\delta(q_1, a, a) = \{(q_1, \lambda)\}$; $\delta(q_1, b, b) = \{(q_1, \lambda)\}$; $\delta(q_1, \lambda, Z_0) = \{(q_2, Z_0)\}$.

Pentru șirul bba avem tranzițiile:

$(q_0, bba, Z_0) \vdash (q_0, ba, bZ_0) \vdash (q_0, a, bbZ_0) \vdash (q_0, \lambda, abbZ_0) \vdash (q_1, \lambda, abbZ_0)$, blocare în q_1 nefinală

$(q_0, bba, Z_0) \vdash (q_0, ba, bZ_0) \vdash (q_0, a, bbZ_0) \vdash (q_1, a, bbZ_0) \vdash (q_1, \lambda, bZ_0)$, blocare în q_1 nefinală

$(q_0, bba, Z_0) \vdash (q_0, ba, bZ_0) \vdash (q_1, ba, bZ_0)$ blocare în q_1 nefinală

$(q_0, bba, Z_0) \vdash (q_1, bba, Z_0) \vdash (q_2, bba, Z_0)$. Deși q_2 este finală, șirul de intrare nu a fost complet citit, deci nici această tranziție nu conduce la acceptarea șirului.

Acestea au fost toate tranzițiile posibile cu bba la intrare.

În concluzie, bba nu este acceptat de automatul A.

IMPLEMENTAREA UNUI AUTOMAT PUSH DOWN

Pentru implementarea unui algoritm care să redea funcționarea unui automat push down pentru un anumit șir de intrare se poate utiliza backtracking.

Proprietăți ale gramaticilor independente de context și ale automatelor push down

Definiție. Spunem că două gramatici G, G' sunt echivalente dacă generează același limbaj, $L(G) = L(G')$.

Definiție. Spunem că două automate push down A, A' sunt echivalente dacă recunosc același limbaj, $L(A) = L(A')$.

Definiție. Spunem că un limbaj este determinist dacă există un automat push down determinist care să îl accepte.

Propoziția 1. Limbajul $\{a^n b^n | n \geq 0\}$ este linear dar nu este regulat.

Propoziția 2. Limbajul $\{a^n b^n a^m b^m | n, m \geq 0\}$ este independent de context, dar nu este linear.

Propoziția 3. Limbajul $\{a^n b^n c^n | n \geq 0\}$ este dependent de context, dar nu este independent de context.

Proprietăți ale gramaticilor independente de context și ale automatelor push down

Propoziția 4. Pentru orice gramatică independentă de context cu λ-producții există o gramatică echivalentă fără λ-producții cu excepția producției $S \to \lambda$, unde S neterminalul de start al noii gramatici, iar S nu mai apare în membrul drept al niciunei producții.

Propoziția 5. (Forma normală Chomsky) Pentru orice gramatică independentă de context G există o gramatică independentă de context $G' = (\mathrm{N}, \Sigma, S, P)$ cu producții de forma $A \to BC, A \to a, A \in N, B, C \in N - \{S\}, a \in \Sigma$, astfel încât $L(G') = L(G\ ) - \{\lambda\}$. Dacă vrem ca $L(G)$ să includă λ, atunci adăugăm producția $S \to \lambda$.

Proprietăți ale gramaticilor independente de context și ale automatelor push down

Propoziția 6. Pentru orice automat push down cu stări finale există un automat push down cu vidarea stivei echivalent.

Propoziția 7. Pentru orice automat push down cu vidarea stivei există un automat push down cu stări finale echivalent.

Propoziția 8. Limbajul $\{a^n b^n | n \geq 0\} \cup \{a^n b^{2n} | n \geq 0\}$ nu poate fi generat de niciun automat push down determinist.

Propoziția 9. Pentru orice automat push down A există o gramatică independentă de context G cu $L(G) = L(A)$.

Propoziția 10. Pentru orice gramatică independentă de context G există un automat push down A cu $L(A) = L(G)$.

Proprietăți ale gramaticilor independente de context și ale automatelor push down

Propoziția 11. Familia limbajelor recunoscute de automatele push down coincide cu familia limbajelor generate de gramaticile independente de context, $\mathcal{L}_2$ sau $\mathcal{CF}$.

Observație. Sintaxa limbajelor de programare poate fi formalizată cu gramatici independente de context, iar parserul corespondent poate fi implementat ca un automat push down.

ANALIZA SINTACTICĂ – TRANSLATORUL STIVĂ

Ca și în cazul translatoarelor finite, translatoarele stivă pot produce ieșiri pentru fiecare tranziție.

Definiție. Un **translator stivă** are o structură de forma:

$$T = (Q, V_i, V_e, \Gamma, \delta, q_0, Z_0, F), \text{unde:}$$

- Q este mulțimea stărilor
- V_i este alfabetul de intrare
- V_e este alfabetul de ieșire
- Γ este alfabetul stivei
- $\delta: Q \times (V_i \cup \{\lambda\}) \times \Gamma \to \mathcal{P}_{fin}(Q \times \Gamma^* \times V_e^*)$
- $q_0 \in Q$ starea inițială
- $Z_0 \in \Gamma$ simbolul inițial al stivei
- $F \subseteq Q$ mulțimea stărilor finale (F poate fi mulțimea vidă)

ANALIZA SINTACTICĂ –TRANSLATORUL STIVĂ

- **Descriere instantanee** (instanță) a lui T

  $(p, x, \alpha, y), p \in Q$ starea curentă a lui T

  $x \in V_i^*$ șirul curent scanat din intrare

  $\alpha \in \Gamma^*$ conținutul stivei

  $y \in V_e^*$ șirul curent din ieșire

- **Mișcare a lui $\boldsymbol{T}$:**

  $(p, ax, Z\alpha, y) \vdash (q, x, \beta\alpha, yy')$ ddacă $(q, \beta, y') \in \delta(p, a, Z)$

  pentru $p, q \in Q, a \in \Sigma \cup \{\lambda\}, x \in V_i^*, \mathrm{Z} \in \Gamma, \alpha, \beta \in \Gamma^*,$

  $y, y' \in V_e^*$

- **Închiderea reflexivă și tranzitivă** a relației $\vdash$ este notată cu $\vdash^*$

ANALIZA SINTACTICĂ –TRANSLATORUL STIVĂ

Translatarea definită de T

Cu stări finale

- pentru un șir de intrare $x \in V_i^*$:

  $T(x) = \{y \in V_e^* | \ (q_0, x, Z_0, \lambda) \vdash^* (q, \lambda, \alpha, y), q \in F, \alpha \in \Gamma^*\}$
- pentru un limbaj $L \subseteq V_i^*$

  $T(L) = \bigcup_{x \in L} T(x)$
- translatarea definită de T cu stări finale (global)

  $\tau(T) = \{(x, y) | x \in V_i^*, y \in V_e^*, y \in T(x)\}$

Cu vidarea stivei

- pentru un șir de intrare $x \in V_i^*$:

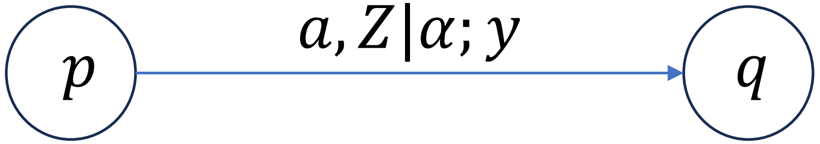
  $T_\lambda(x) = \{y \in V_e^* | \ (q_0, x, Z_0, \lambda) \vdash^* (q, \lambda, \lambda, y), q \in Q\}$
- pentru un limbaj $L \subseteq V_i^*$

  $T_\lambda(L) = \bigcup_{x \in L} T_\lambda(x)$
- translatarea definită de T cu vidarea stivei (global)

  $\tau_\lambda(T) = \{(x, y) | x \in V_i^*, y \in V_e^*, y \in T_\lambda(x)\}$

TRANSLATOR STIVĂ - EXEMPLU

Vom figura grafic $(q, \beta, y\ ) \in \delta(p, a, Z)$ (unde p, q sunt stări, a este un simbol din alfabetul de intrare sau este λ, Z este simbolul din vârful stivei, β este un șir peste alfabetul stivei care îl va înlocui pe Z, iar y este un <u>șir</u> peste alfabetul de ieșire):



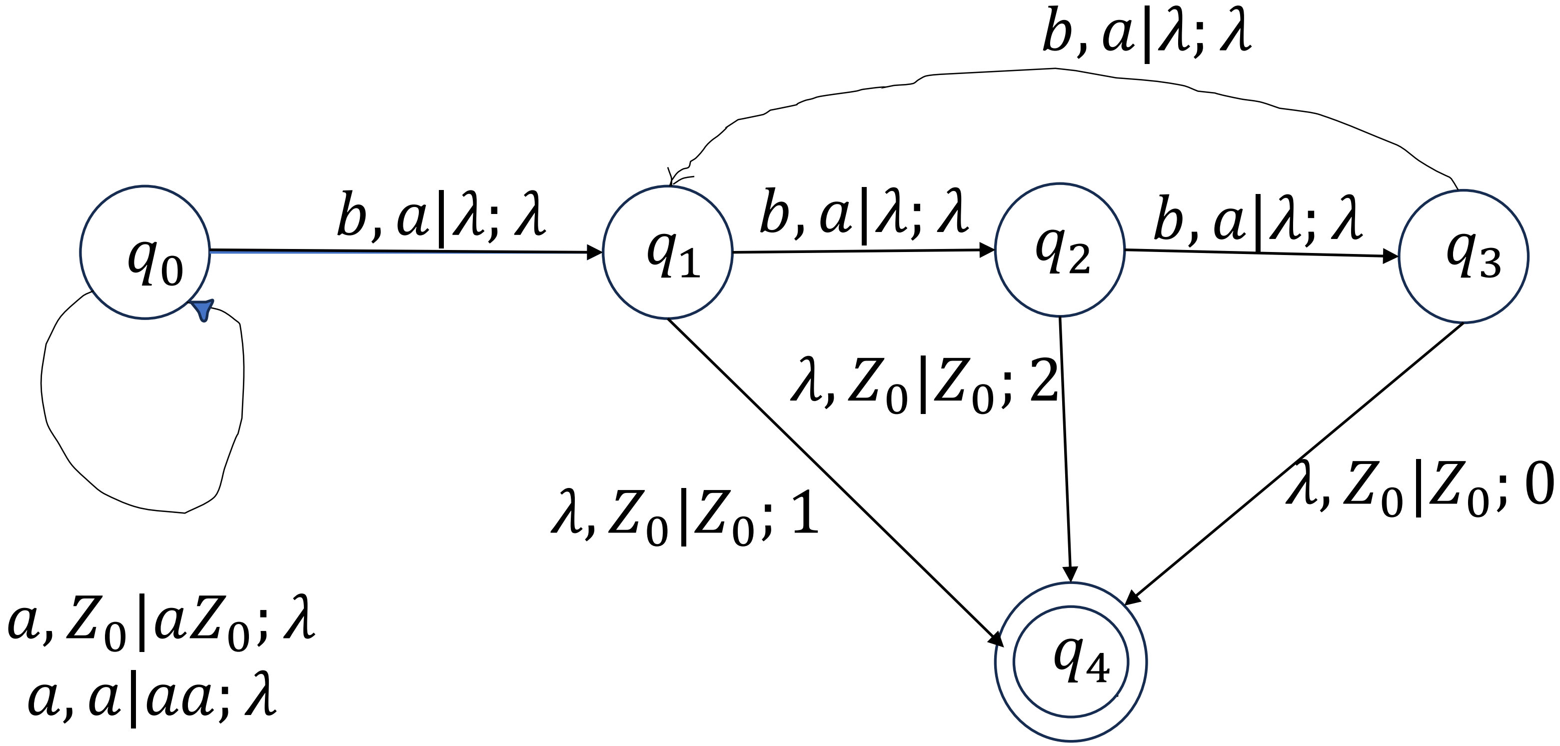
Mai jos avem un exemplu de translator

$$T = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1,2\}, \{a, Z_0\}, \delta, q_0, Z_0, \{q_4\})$$

a cărui translatare este

$$\{(a^n b^n, x) | n \geq 0, x \equiv n(modulo\ 3), x \in \{0,1,2\}\}$$

TRANSLATOR STIVĂ – EXEMPLUL 1



Pentru șirul de intrare $aaaabbbb$ obținem mișcările:
$(q_0, aaaabbbb, Z_0, \lambda) \vdash (q_0, aaabbbb, aZ_0, \lambda) \vdash (q_0, aabbbb, aaZ_0, \lambda) \vdash$
$(q_0, abbbb, aaaZ_0, \lambda) \vdash (q_0, bbbb, aaaaZ_0, \lambda) \vdash (q_1, bbb, aaaZ_0, \lambda) \vdash$
$(q_2, bb, aaZ_0, \lambda) \vdash (q_3, b, aZ_0, \lambda) \vdash (q_1, \lambda, Z_0, \lambda) \vdash (q_4, \lambda, Z_0, 1)$.
Rezultă că pentru șirul de intrare a^4b^4 ieșirea (unică) este $1 = 4$ ($modulo$ 3).

- Translatorul de mai sus este determinist. În cazul translatoarelor nedeterministe se folosește backtracking pentru a identifica toate drumurile etichetate cu șirul de intrare de la starea inițială la o stare finală (când se lucrează cu stări finale) sau de la starea inițială până la vidarea stivei (când se lucrează cu vidarea stivei). Pentru un astfel de drum, se obține o ieșire (o translatare a intrării).
- Este posibil ca pentru anumite șiruri să nu obținem nicio translatare. De exemplu, în cazul translatorului de mai sus, șirul $aaaabb$ nu are nicio ieșire (translatare):
- $(q_0, aaaabb, Z_0, \lambda) \vdash (q_0, aaabb, aZ_0, \lambda) \vdash (q_0, aabb, aaZ_0, \lambda) \vdash (q_0, abb, aaaZ_0, \lambda) \vdash (q_0, bb, aaaaZ_0, \lambda) \vdash (q_1, b, aaaZ_0, \lambda) \vdash (q_2, \lambda, aaZ_0, \lambda)$. Din această configurație nu mai avem nicio mișcare validă pentru translator. Cum q_2 nu este finală, iar mișcările au fost unice, rezultă că translatarea lui $aaaabb$ este $\emptyset$.

ANALIZA SINTACTICĂ –TRANSLATORUL STIVĂ

Exemplu:

Fie gramatica $G = (N, \Sigma, S, P)$ independentă de context, cu producțiile numerotate $1, \cdots, |P|$.

Translatorul următor:

$$T_G = (\{q\}, \Sigma, \{1, \dots, |P|\}, N \cup \Sigma, \delta, q, S, \emptyset),$$

unde $\delta(q, \lambda, A) = \{(q, \alpha, i) | i: A \to \alpha \in P\}$

$$\delta(q, a, a) = \{(q, \lambda, \lambda)\} \, \forall a \in \Sigma$$

are ca translatare

$$\tau_\lambda(T_G) = \{(w, \pi) | S \overset{\pi}{\Rightarrow}_s w, w \in L(G) \subseteq \Sigma^*, \pi \in \{1, \dots, |P|\}^*\}$$

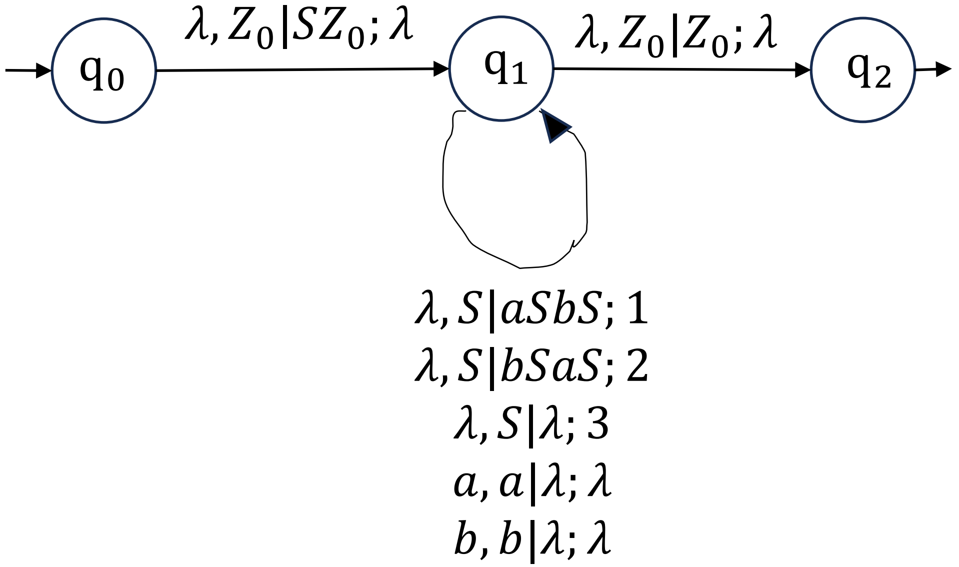
Observații.

— Translatorul de mai sus simulează (într-un mod ”foarte nedeterminist”) derivările stângi din G

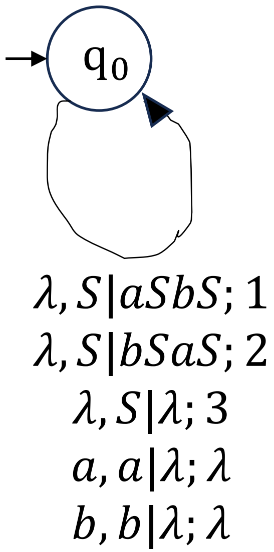
— i reprezintă numărul de ordine al producției aplicate în ordinea dată, iar π succesiunea de producții aplicate pentru derivarea lui w din simbolul de start S într-o derivare stângă

Exemplul 2: translatorul stivă pentru gramatica independentă de context:
$G = (\{S\}, \{a, b\}, S, \{S \to aSbS, S \to bSaS, S \to \lambda\})$

Numerotăm producțiile gramaticii: $1{:}\, S \to aSbS,\ \ 2{:}\, S \to bSaS,\ \ 3{:}\, S \to \lambda$



Translator cu stare finală



Translator cu vidarea stivei

Cele 2 translatoare de mai sus au translatarea globală: $\{(w, \pi) | w \in \{a, b\}^*, |w|_a = |w|_b, S \overset{\pi}{\Rightarrow} w, \pi \in \{1,2,3\}^*\}$

Exerciții:

Să se scrie un translator stivă a cărui translatare este:

$M_1 = \{(a^n b^n, x) | n \geq 0, x \equiv n(modulo\ 3), x \in \{0,1,2\}\}$ (un alt mod de a obține această translatare decât în exemplul 1)

$M_2 = \{(a^m b^n, c^{n-2m+2}) | n \geq 2m \geq 0\}$

$M_3 = \{(w, c^n) | w \in \{a,b\}^*, ||w|_a \geq |w|_b, n = |w|_a - |w|_b\}$

$M_4 = \{(a^i b^j c^k d^t, e^{i+t}) | i, j, k, t \geq 0, i + k = j + t\}$

$M_5 = \{(a^m b^n, c^{m-2n+2}) | m \geq 2n \geq 0\}$

METODE DE ANALIZĂ SINTACTICĂ

A. METODA TOP-DOWN

Fie $G = (N, \Sigma, S, P)$ gic și $w \in \Sigma^*$. În cazul metodei top-down

- Se pornește de la simbolul de start S
- În forma sentențială curentă (inițal S) se alege un neterminal A și o producție $A \to \alpha$, se înlocuiește A cu α (alegerea producției se face conform unor criterii prestabilite sau pur și simplu în ordinea în care sunt listate producțiile gramaticii)
- Dacă nu mai există nicio alternativă pentru A(nicio producție $A \to \alpha$ care să nu fi fost analizată), atunci se revine la un pas anterior
- Dacă la final nici pentru S nu mai există alternative, atunci $w \notin L(G)$
- Altfel, dacă forma sentențială curentă este egală cu w, atunci $w \in L(G)$ și se furnizează o derivare stângă a sa
- Există gramatici pentru care metoda descrisă mai sus nu funcționează
- Denumirea top-down provine de la faptul că w este obținut ca și cum s-ar construi arborele lui de derivare de sus în jos, dacă $w \in L(G)$

METODE DE ANALIZĂ SINTACTICĂ

B. METODA BOTTOM-UP

Fie $G = (N, \Sigma, S, P)$ gic și $w \in \Sigma^*$. În cazul metodei bottom-up:

- Se pornește de la șirul analizat, w
- În forma sentențială curentă $\alpha \in N \cup \Sigma)^*$(inițal $\alpha = w$) se identifică un subșir β, astfel ca $\alpha = \alpha'\beta\alpha''$, $\beta \in (N \cup \Sigma)^*$, pentru care există $A \to \beta \in P$; se înlocuiește β cu A (operație care se numește reducere, inversă derivării), obținându-se șirul curent $\alpha' A \alpha''$ (identificarea lui β se poate face în conformitate cu anumite criterii, sau este considerat arbitrar)
- Se repetă secvența de mai sus până când:

  - se ajunge la forma sentențială S, caz în care $w \in L(G)$, STOP

  - nu mai există în forma sentențială curentă niciun subșir care să poată fi redus (înlocuit de un neterminal); în acest caz se încearcă revenirea la un pas anterior

  - dacă nu mai există alternative, atunci $w \notin L(G)$, STOP
- În cazul $w \in L(G)$, se furnizează o derivare dreaptă a sa
- Există gramatici pentru care metoda generală descrisă mai sus nu funcționează
- Denumirea bottom-up provine de la faptul că dacă $w \in L(G)$, prin aplicarea acestei metode este ca și cum arborele sintactic pentru w s-ar parcurge de jos în sus

METODE DE ANALIZĂ SINTACTICĂ

Pentru cele două metode descrise mai sus la modul general, există mai multe variante de algoritmi:

- exponențiali; nu se folosesc în practică
- polinomiali (cum este algoritmul CYK, de ordin $O(n^3)$)
- liniari; aceștia sunt cei mai utilizați de compilatoare

  - metode liniare de tip top-down: algoritmi de tip LL

  - metode liniare de tip bottom-up: algoritmi de tip LR

METODA DE ANALIZĂ SINTACTICĂ TOP-DOWN

Fie $G = (N, \Sigma, S, P)$ gramatică independentă de context și $w \in \Sigma^*$ un șir peste Σ. Pentru a verifica dacă $w \in L(G)$ se pleacă de la simbolul de start, se generează toate formele sentențiale, eventual după anumite criterii, până când eventual se obține w. Există mai multe abordări:

1) Metoda **breadth-first.** Este cel mai general algoritm de analiză sintactică care poate fi utilizat pentru orice gramatică independentă de context. Se folosește o coadă, care inițial conține pe S, în care se pun formele sentențiele (șiruri derivate din simbolul de start, S) curente, până se ajunge eventual la w. Este foarte lent, nu se folosește în practică.

2) Metoda **breadth-first pentru derivări stângi**. În acest caz se reduce din volumul de calcul, dar funcționează doar pentru gramatici nerecursive la stînga.

ALGORITM TOP DOWN CU BREADTH FIRST PENTRU GRAMATICI INDEPENDENTE DE CONTEXT, NERECURSIVE LA STÂNGA

INPUT: gramatica independentă de context $G = (N, \Sigma, S, P)$ nerecursivă la stânga, $w \in \Sigma^*, n = |w|$

Top_Down_Breadth_First_Left (){

$S \Longrightarrow Q$; // Q este o coada; initial introducem in coada simbolul de start

while (Q este nevida)

{ $z \Leftarrow Q$; // extragem z din coada. Fie $z = xAu, x \in \Sigma^*, A \in N, u \in (N \cup \Sigma)^*$

if (x este sufix al lui w) // daca $x = \lambda$, atunci consideram x sufix

{ **for** (fiecare $A \to v \in P$)

{ **if** ($w == xvu$) $accept(\,)$;

if ($|xvu|_N \geq 1$ && $|xvu|_\Sigma \leq n$) $xvu \Longrightarrow Q$ // introducem xvu in coada doar daca are cel

// putin un neterminal si maxim n terminali

} // end_for

} // end_if

} // end_while

$reject(\,)$; // coada este vidă

} // end_ Top_Down_Breadth_First_Left

3) Algoritm Deapth-First pentru derivări stângi. Algoritmul folosește backtracking și inițial pune pe stivă simbolul \$. getNextToken() este o funcție care furnizează următorul token. În cazul în care lucrăm cu o gramatică cu simboluri terminale abstracte și facem analiza pentru un șir *w* de lungime *n*, atunci această funcție ne furnizează următorul simbol al lui *w* (adică este o banală incrementare a unui indice; când acest indice devine n+1, atunci s-au citit toate simbolurile lui w).

Algoritmul de mai jos se termină întotdeauna atunci când *G* <u>nu</u> este recursivă la stânga. Algoritmul se încheie atunci când în interiorul lui while se apelează funcția reject(), caz în care $w \notin L(G)$, sau atunci când se apelează funcția accept(), caz în care $w \in L(G)$.

INPUT: gramatica independentă de context $G=(N, \Sigma, S, P)$ și $w=z\#$, unde $z\in\Sigma^*$, $\#=EOF$ (dacă *G* formalizează sintaxa unui limbaj de programare, atunci Σ reprezintă mulțimea categoriilor de atomi lexicali (token-ii) acceptați de limbajul respectiv, iar *EOF* marchează sfârșitul fișerului care conține programul sursă, program reprezentat de *z*)

ALGORITM TOP-DOWN CU DEPTH-FIRST (BACKTRACKING)

```
// initializare
push($);
X← S;
a ← getNextToken();
void Top_Down_Depth_First_Left(){
while(true)
  {if (X este neterminal)
      {
         if ( mai exista productie X→ Y1 ⋯ Yk din P neanalizata)
            {
             push Yk; ⋯ ;  push Y1;
             X ← pop();
            }
         else if ( X == S ) reject();  // am analizat toate alternativele pentru S
      }
   else   // X este terminal sau $
         if (X== $ && a == EOF) accept();
         else
               if (X== a)
                   { a  ← getNextToken();
                     X ← pop();
                   }
                else  //daca token-ul curent este diferit de simbolul din varful stivei
                      // revenim la pasul anterior
                     Top_Down_Depth_First_Left();
  }  //  end_while
 } // end_ Top_Down_Depth_First_Left
```

METODA DE ANALIZĂ SINTACTICĂ TOP-DOWN
PARSER RECURSIV DESCENDENT GENERAL

Un parser recursiv descendent pentru o gramatică independentă de context $G = (N, \Sigma, S, P)$ constă din implementarea unei funcții pentru fiecare neterminal.

Execuția începe cu apelul funcției corespunzătoare simbolului de start.

Parserul nu funcționează pentru gramatici recursive la stânga.

Fie $w = z\#$, unde $z \in \Sigma^*$ șirul analizat, iar # este un simbol nou (echivalentul lui EOF). tok este o variabila globală ce conține token-ul curent. Funcția getNextToken furnizează token-ul următor. Pentru un neterminal $A \in N$, funcția este:

```
void A() {
    while (true)
      {
        if ( mai exista productie A→ X_1 ⋯ X_k din P neanalizata)
          {
            for (i = 1, … , k) {
            if ( X_i este neterminal )
                        X_i() // se apeleaza functia corespunzatoare
            else if ( X_i== tok ) // daca simbolul curent al productiei coincide cu
                                  //  tokenul curent
                          tok = getNextToken(); // avansam la token-ul urmator
                 else  A(); // a aparut o eroare; incercam o alta productie pentru A
            }
        else  if ( A == S ) reject();  // am analizat toate alternativele pentru S
              else;
      } // end_while
} // end_A()
```

Programul principal va consta din:

void main() {
// initializari; daca se lucreaza cu un fisier care contine programul sursa pentru un limbaj,
// atunci se initializeaza pointerul curent pe primul caracter al fisierului; altfel $k = 0$;
$tok\ =\ getNextToken()$;
$S()$;
if $(tok \neq \#)$ reject(); // sau if $(tok \neq EOF)$
}

Observăm că acest algoritm folosește backtracking. Însă, pentru anumite tipuri de gramatici, și anume gramaticile de tip $LL(1)$, pe care le vom studia în continuare, alegerea A-producției se poate realiza în mod unic. Dacă lucrăm cu o gramatică simbolică și $|z| = n$, $z = a_1 a_2 \dots a_n$, $a_{n+1} = \#$, atunci getNextToken constă din incrementarea unui indice k, inițializat cu 0, care indică caracterul curent din z.

Totuși, metoda backtracking este foarte rar folosită pentru parsarea limbajelor de programare, deoarece nu este eficientă.