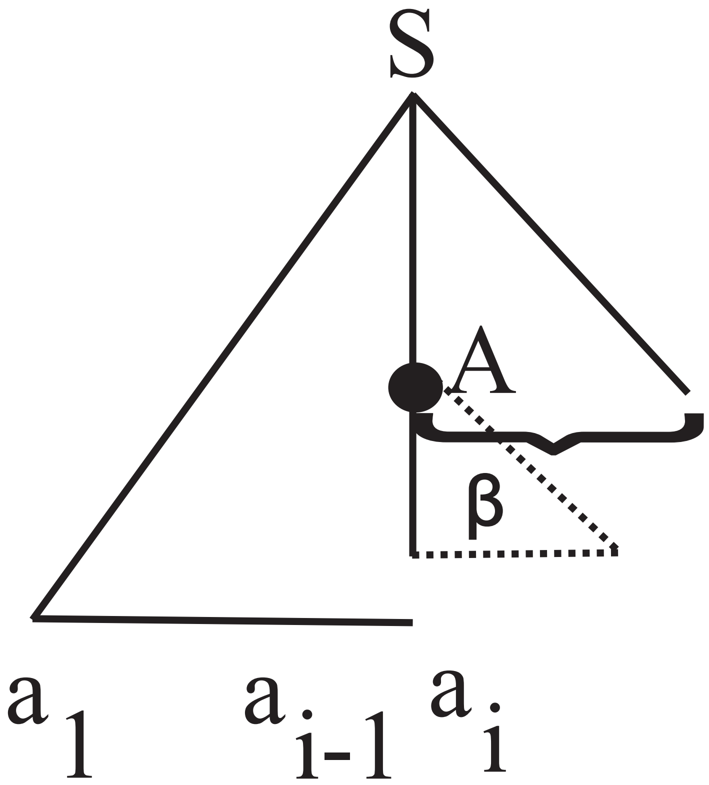
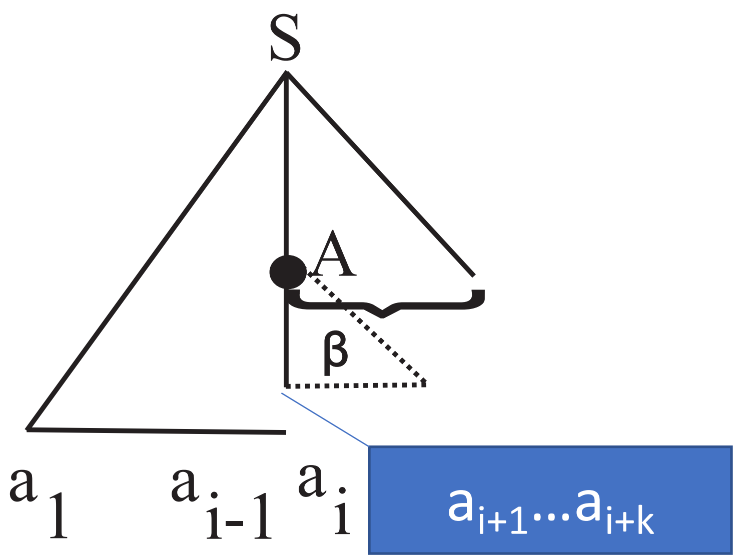
LL(1) Parser



Linear algorithm

LL(k)

- L = left (sequence is read from left to right)
- L = left (use leftmost derivation)
- Prediction of length **k**



LL(k) Principle

- In any moment of parsing, <u>acțion is uniquely determinde </u>by:
- Closed part ($a_1...a_i$)
- Current symbol A
- Prediction $a_{i+1}...a_{i+k}$ (length k)

FIRST$_k$

- ≈ first k terminal symbols that can be generated from α
- **Definition**:

$$FIRST_k : (N \cup \Sigma)^* \rightarrow \mathcal{P}(\Sigma^k)$$
$$FIRST_k(\alpha) = \{u | u \in \Sigma^k, \alpha \stackrel{*}{\Rightarrow} ux, |u| = k \text{ sau } \alpha \stackrel{*}{\Rightarrow} u, |u| \leq k\}$$

Definition

- *A cfg is LL(k) if for any 2 leftmost derivation we have:*

1. $S \overset{*}{\Rightarrow}_{st} wA\alpha \Rightarrow_{st} w\beta\alpha \overset{*}{\Rightarrow}_{st} wx;$

2. $S \overset{*}{\Rightarrow}_{st} wA\alpha \Rightarrow_{st} w\gamma\alpha \overset{*}{\Rightarrow}_{st} wy;$

such that $FIRST_k(x) = FIRST_k(y)$ *then* $\beta = \gamma.$

Theorem

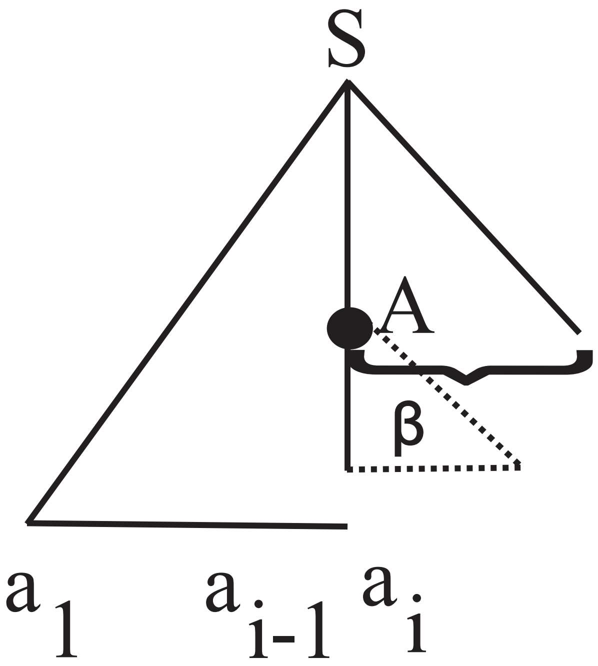
The necessary and sufficient condition for a grammar to be LL (k) is that for any pair of distinct productions of a nonterminal ($A\rightarrow \beta$, $A\rightarrow\gamma,\beta\neq\gamma$) the condition holds:

$$\mathrm{FIRST}_k(\beta\alpha) \cap \mathrm{FIRST}_k(\gamma\alpha)= \Phi,\forall\alpha \quad \text{such that} \quad S \overset{*}{=>} uA\alpha$$

Theorem: A grammar is LL(1) if and only if for any nonterminal A with productions $A \rightarrow\alpha_1 | \alpha_2|\ldots| \alpha_n$, FIRST(α_i) ∩ FIRST(α_j) = ∅ and if $\alpha_i\Rightarrow\varepsilon$, FIRST($\alpha_i$) ∩ FOLLOW(A)= ∅, ∀i,j = 1,n,i≠j

FOLLOW

$A \rightarrow \varepsilon$



- $FOLLOW_k(A) \approx$ next k symbols generated after/ following A

$$FOLLOW : (N \cup \Sigma)^* \rightarrow \mathcal{P}(\Sigma)$$
$$FOLLOW(\beta) = \{w \in \Sigma | S \stackrel{*}{\Rightarrow} \alpha\beta\gamma, w \in FIRST(\gamma)\}$$

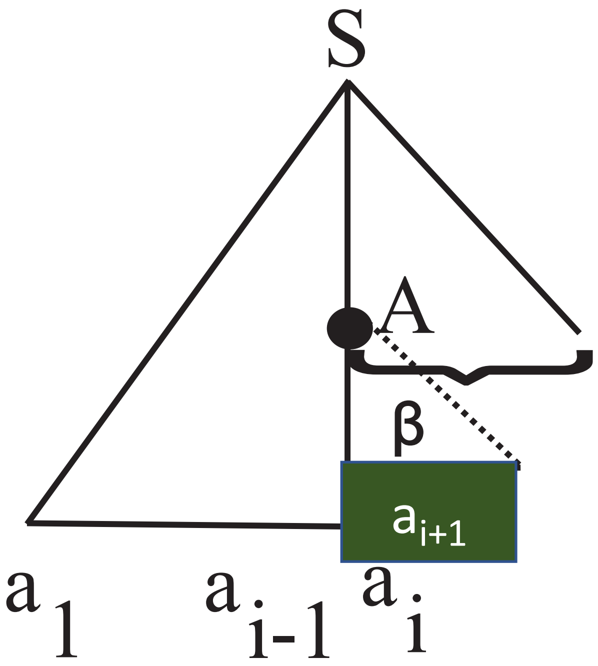
LL(1) Parser

- Prediction of length 1

- Steps:
  1) construct FIRST, FOLLOW
  2) Construct LL(1) table
  3) Analyse sequence based on moves between configurations

Executed 1 time

Construct FIRST



if **A -> aBb** then FIRST(A) = {a}

if **A->BC** and **B->b** and **B->cA** then FIRST(A) = {b,c}

if **A->BC** and **B-> eps** and **C->a** then FIRST(A) = {a}

Construct FIRST

Concatenation of length 1

- $FIRST_1$ denoted FIRST
- Remarks:
  - If L_1, L_2 are 2 languages over alphabet Σ, then : $L_1 \oplus L_2 = \{w|x \in L_1, y \in L_2, xy = w, |w| \leq 1 \text{ sau } xy = wz, |w| = 1\}$ and
  - $FIRST(\alpha\beta) = FIRST(\alpha) \oplus FIRST(\beta)$

    $FIRST(X_1 \dots X_n) = FIRST(X_1) \oplus \dots \oplus FIRST(X_n)$

$L1 = \{aa,ab,ba\}$

$L2 = \{00,01\}$

then $L1L2 = \{aa00,aa01,ab00,ab01,ba00,ba01\}$

$L1 \oplus L2 = \{a,b\}$

$L1 = \{a,b\}$

$L2 = \{0,1\}$

then $L1 \oplus L2 = \{a,b\}$

$L1 = \{a, \boldsymbol{\varepsilon}\}$

$L2 = \{0,1\}$

then $L1 \oplus L2 = \{a,0,1\}$

Algoritmul 3.3 FIRST

INPUT: G
OUTPUT: $FIRST(X), \forall X \in N \cup \Sigma$
for $\forall a \in \Sigma$ **do**
  $F_i(a) = \{a\}, \forall i \geq 0$
end for
i := 0;
$F_0(A) = \{x | x \in \Sigma, A \to x\alpha \text{ sau } A \to x \in P\}$; $\{\textit{inițializare}\}$
repeat
  i := i+1;
A **for** $\forall \cancel{X} \in N$ **do**
    if F_{i-1} au fost calculate $\forall X \in N \cup \Sigma$ **then**
      $\{\textit{dacă } \exists Y_j, F_{i-1}(Y_j) = \emptyset \textit{ atunci nu se poate aplica}\}$
      $F_i(A) = F_{i-1}(A) \cup$
      $\{x | A \to Y_1 \dots Y_n \in P, x \in F_{i-1}(Y_1) \oplus \dots \oplus F_{i-1}(Y_n)\}$
    end if
  end for
until $F_{i-1}(A) = F_i(A)$
$FIRST(X) := F_i(X), \forall X \in N \cup \Sigma$

Algorithm FOLLOW

INPUT: G, FIRST(X), $\forall X \in N \cup \Sigma$
OUTPUT: FOLLOW(A), $\forall A \in N$

for $A \in N - \{S\}$ **do** {init}
    $L_0(A) = \Phi$;
endFor;
$L_0(S) = \{\varepsilon\}$; {init}
i = 0;
repeat
    i = i+1;
    for $B \in N$ **do**
        for $A \to \alpha B y \in P$ **do**
            for $\forall a \in FIRST(y)$ **do**
                if $a = \varepsilon$ **then** $F_i(B) = F_i(B) \cup F_{i-1}(A)$
                    else $F_i(B) = F_{i-1}(B) \cup First(y)$
                endif
            endFor
        endFor
    endfor
until $Fi(X) = Fi\text{-}1(X), \forall X \in N$
$FOLLOW(X) = Fi(X), \forall X \in N$

$S \Rightarrow^0 S$ // ε after S

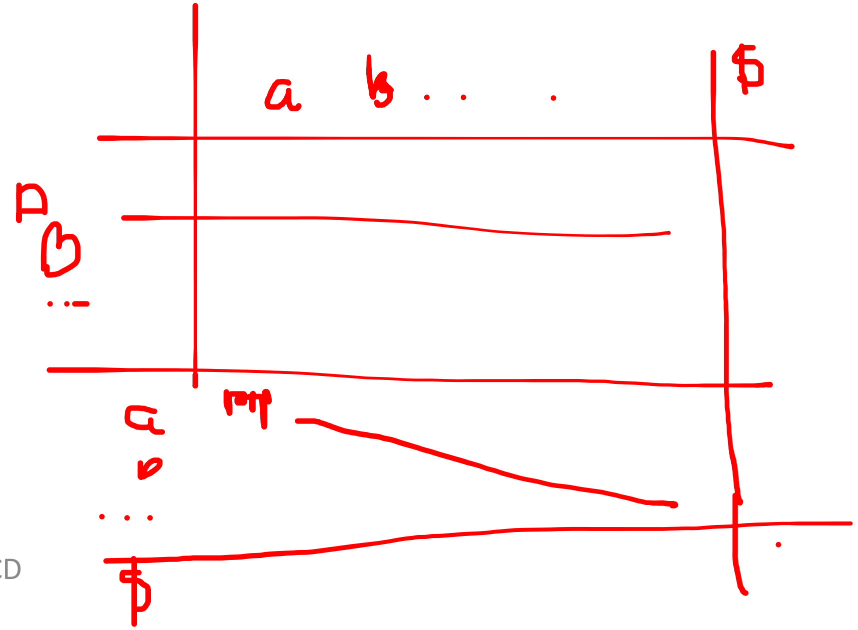
S => aAc=> abBc
A -> bB

Step 2: Construct LL(1) table

- Possible action depend on:
  - Current symbol ∈ $\mathbf{N} \cup \mathbf{\Sigma}$
  - Possible prediction ∈ $\mathbf{\Sigma}$
- Add a special character “\$” (∉ $\mathbf{N} \cup \mathbf{\Sigma}$) – marking for “empty stack”

= > table:

- One line for each symbol ∈ $\mathbf{N} \cup \mathbf{\Sigma} \cup \{\$\}$
- One column for each symbol ∈ $\mathbf{\Sigma} \cup \{\$\}$



Rules LL(1) table

1. $M(A,a) = (\alpha, i), \forall a \in FIRST(\alpha), a \neq \epsilon, A \rightarrow \alpha$ production in P with number i
   $M(A,b) = (\alpha, i),$ if $\epsilon \in FIRST(\alpha), \forall b \in FOLLOW(A), A \rightarrow \alpha$ production in P with number i

2. $M(a,a) = pop, \forall a \in \Sigma;$

3. $M(\$,\$) = acc;$

4. M(x,a)=err (error) otherwise i.

Remark

A <u>grammar is LL(1)</u> if the LL(1) parse table does NOT contain conflicts – there exists at most one value in each cell of the table M(A,a)

Step 3: Definire configurations and moves

- INPUT:
  - Language grammar G = (N, $\boldsymbol{\Sigma}$, P,S)
  - LL(1) parse table
  - Sequence to be parsed w =$a_1 \ldots a_n$
- OUTPUT:

If (w ∈L(G)) *then* **string of productions**

else **error** & **location of error**

LL(1) configurations

$$(\alpha, \beta, \pi)$$

where:

- α = input stack
- β = working stack
- π = output (result)

Initial configuration:
$(w\$,S\$,\varepsilon)$

Final configuration:
$(\$, \$, \pi)$

Moves

1. Push – put in stack

$$(ux, A\alpha\$, \pi) \vdash (ux, \beta\alpha\$, \pi i), \quad \text{if} \quad M(A,u) = (\beta, i);$$

(pop A and push symbols of β)

2. Pop – take off from stack (from both stacks)

$$(ux, a\alpha\$, \pi) \vdash (x, \alpha\$, \pi), \quad \text{if} \quad \mathrm{M(a,u)=pop}$$

3. Accept

$$(\$, \$, \pi) \vdash acc$$

4. Error - otherwise

Algorithm LL(1) parsing

- INPUT:
  - LL(1) table with NO conflicts;
  - G –grammar (productions)
  - Input sequence $w = a_1a_2 \ldots a_n$
- OUTPUT:
  - sequence accepted or not?
  - If yes then string of productions

Algorithm LL(1) parsing (cont)

alfa := w$;beta := S$;pi := ε;
go := true;

```
while go do
      if M(head(beta),head(alfa))=(b,i) then
                  pop(beta); push(beta,b); push(pi,i)
      else
            if M(head(beta),head(alfa))=pop then
                  pop(beta); pop(alfa);
            else
                  if M(head(beta),head(alfa))=acc then
                        go:=false; s:="acc";
                  else  go:=false; s:="err";
                  end if
            end if
      end if
end while
```

```
if  s="'acc"' then
          write("Sequence accepted");
          write(pi)
   else
          write(" Sequence not accepted")
```

Remarks

1) LL(1) parser provides location of the error

2) Grammars can be transformed to be LL(1)

   example:

   I -> if C then S | if C then S else S // is not LL(1)

   I -> if C then S T

   T -> ε | else S // is LL(1)

Play time!!!

- Menti.com cod: 41 95 54 9