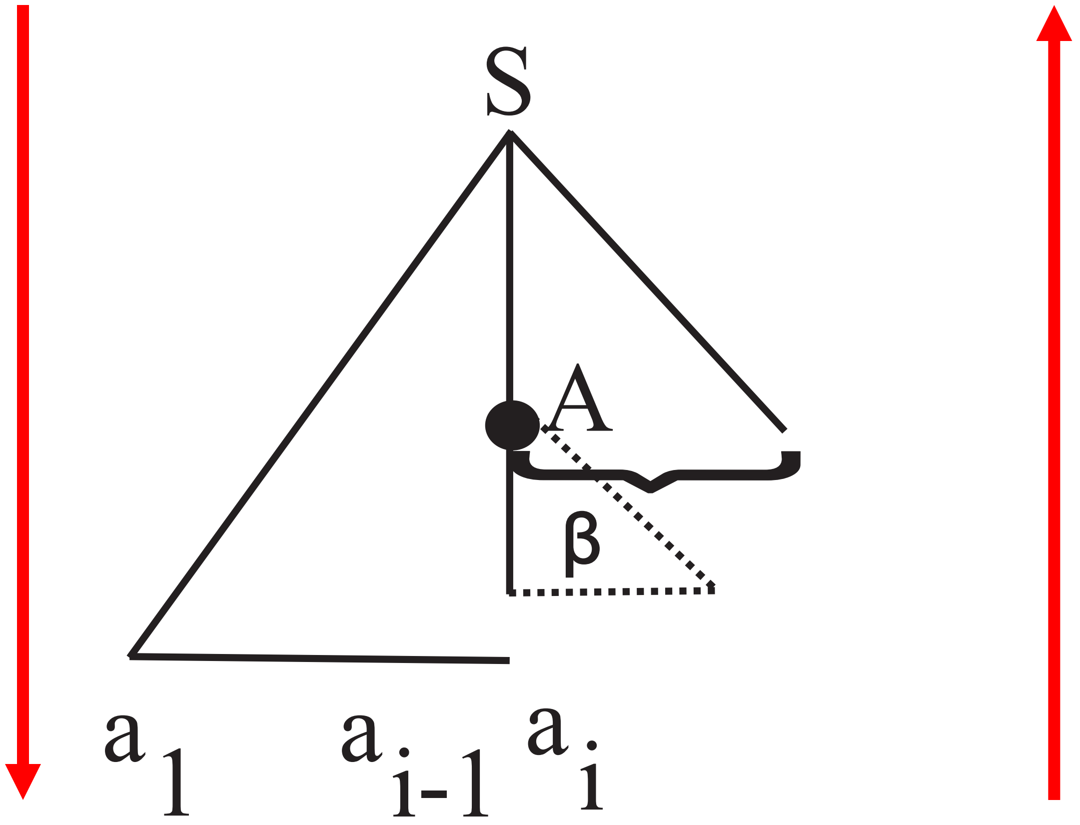
Course 6

Parsing

- Cfg G = (N, **Σ**, P,S) check if w ∈ L(G)
- Construct parse tree
- How:
  1. Top-down vs. Bottom-up
  2. Recursive vs. linear



	Descendent	Ascendent
Recursive	Descendent recursive parser	Ascendent recursive parser
Linear	LL(k): $\boxed{\text{LL(1)}}$	LR(k): LR(0), SLR, LR(1), $\boxed{\text{LALR}}$

Result – parse tree -representation

- Arbitrary tree – child sybling representation (left child right sybling)
- Sequence of derivations S => $\boldsymbol{\alpha}_1$ => $\boldsymbol{\alpha}_2$ =>… => $\boldsymbol{\alpha}_n$ = w
- String of production – index associated to prod – which prod is used at each derivation step

Descendent recursive parser

- Example

Formal model

- Configuration

$$(\mathbf{s}, \mathbf{i}, \alpha, \beta)$$

where:

- s = state of the parsing, can be:
  - q = normal state
  - b = back state
  - f = final state - corresponding to success: $w \in L(G)$
  - e = error state – corresponding to insuccess: $w \notin L(G)$
- i – position of current symbol in input sequence

  $w = a_1 a_2 \ldots a_n$, $i \in \{1, \ldots, n+1\}$
- α = working stack, stores the way the parse is built
- β = input stack, part of the tree to be built

Initial configuration:
$(q, 1, \varepsilon, S)$

Define moves between configurations

Final configuration:
$(f, n+1, \alpha, \varepsilon)$

Expand

WHEN: head of input stack is a nonterminal

$(q,i,\boldsymbol{\alpha}, A\boldsymbol{\beta}) \vdash (q,i,\boldsymbol{\alpha}A_1, \boldsymbol{\gamma}_1\boldsymbol{\beta})$

where:

$A \rightarrow \boldsymbol{\gamma}_1 \mid \boldsymbol{\gamma}_2 \mid \ldots$ represents the productions corresponding to A

1 = first prod of A

Advance

WHEN: head of input stack is a terminal = current symbol from input

$(q,i, \boldsymbol{\alpha}, a_i\boldsymbol{\beta}) \vdash (q,i+1, \boldsymbol{\alpha}a_i, \boldsymbol{\beta})$

Momentary insuccess

WHEN: head of input stack is a terminal ≠ current symbol from input

$(q,i, \boldsymbol{\alpha}, a_i\boldsymbol{\beta}) \vdash (b,i, \boldsymbol{\alpha}, \boldsymbol{\beta})$

Back

WHEN: head of working stack is a terminal

$(b, i, \boldsymbol{\alpha} a, \boldsymbol{\beta}) \vdash (b, i-1, \boldsymbol{\alpha}, a\boldsymbol{\beta})$

Another try

WHEN: head of working stack is a nonterminal

$(b,i,\boldsymbol{\alpha} A_j, \boldsymbol{\gamma}_j \boldsymbol{\beta}) \vdash$ $(q,i,\boldsymbol{\alpha} A_{j+1}, \boldsymbol{\gamma}_{j+1}\boldsymbol{\beta})$, if $\exists\ A \rightarrow \boldsymbol{\gamma}_{j+1}$

$(b,i,\boldsymbol{\alpha}, A\,\boldsymbol{\beta})$, otherwise with the exception

$(e,i,\boldsymbol{\alpha},\boldsymbol{\beta})$, if $i=1$, $A=S$, **ERROR**

Success

$(q, n+1, \boldsymbol{\alpha}, \varepsilon) \vdash (f, n+1, \boldsymbol{\alpha}, \varepsilon)$

Algorithm

$w \in L(G)$ - HOW

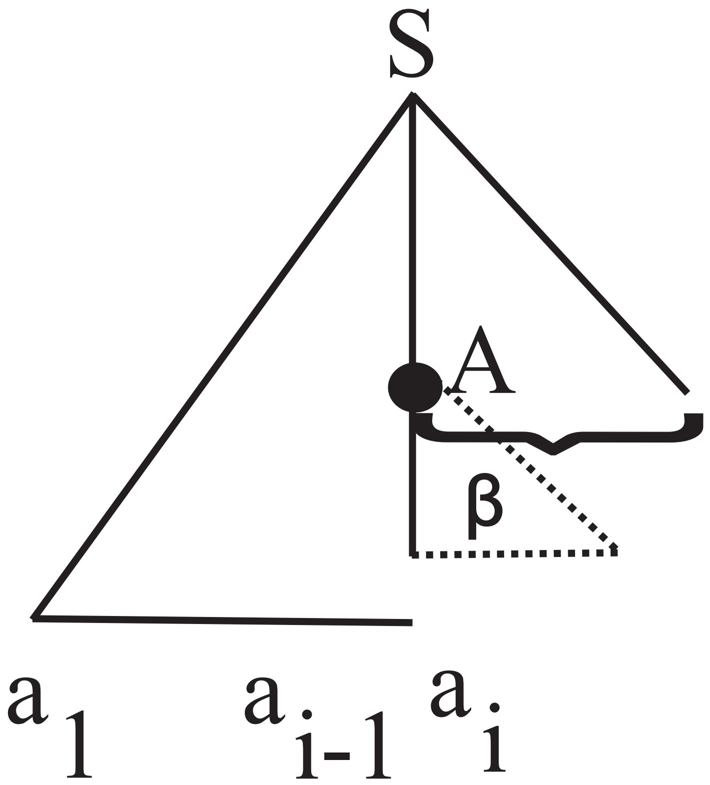
- Process $\boldsymbol{\alpha}$:
  - From left to right (reverse if stored as stack)
  - Skip terminal symbols
  - Nonterminals – index of prod

- Example: $\boldsymbol{\alpha} = S_1\ a\ S_2\ a\ S_3\ c\ b\ S_3\ c$

When the algorithm never stops?

- S->S**α** – expand infinitely (left recursive)

LL(1) Parser



Linear algorithm

FIRST_k

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

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

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
  - $FIRST(\alpha\beta) = FIRST(\alpha) \oplus FIRST(\beta)$

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

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

$L2 = \{00,01\}$

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

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

$L1 =\{a,b\}$

$L2 =\{0,1\}$

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

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

$L2=\{0,1\}$

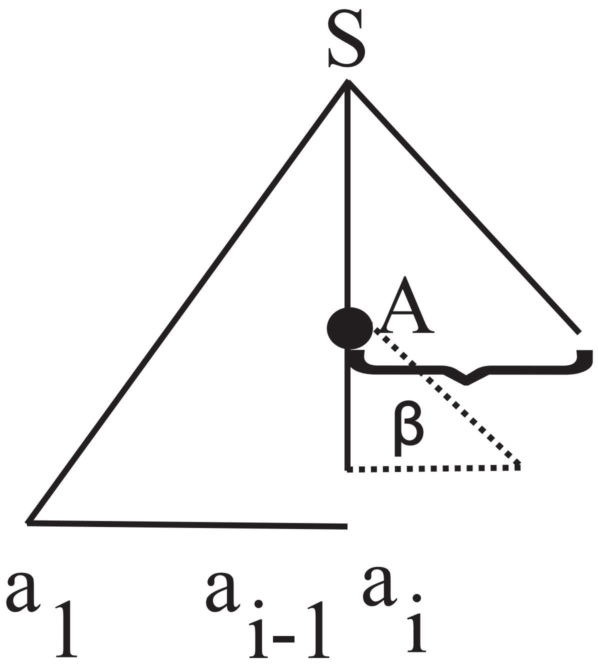
$L1 \bigoplus L2 =\{a,0,1\}$

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;
$F_0(A) = \{x | x \in \Sigma, A \rightarrow x\alpha$ sau $A \rightarrow x \in P\}$; $\{iniţializare\}$
repeat
    i := i+1;
$\boxed{A}$ **for** $\forall \cancel{X} \in N$ **do**
        if F_{i-1} au fost calculate $\forall X \in N \cup \Sigma$ **then**
            $\{dacă\ \exists Y_j, F_{i-1}(Y_j) = \emptyset$ *atunci nu se poate aplica*$\}$
            $F_i(A) = F_{i-1}(A) \cup$
            $\{x | A \rightarrow Y_1 \ldots Y_n \in P, x \in F_{i-1}(Y_1) \oplus \ldots \oplus F_{i-1}(Y_n)\}$
        end if
    end for
until $F_{i-1}(A) = F_i(A)$
$FIRST(X) := F_i(X), \forall X \in N \cup \Sigma$

FOLLOW

$A \rightarrow \boldsymbol{\varepsilon}$



- FOLLOW$_k$(A)≈ next k symbols generated after/ following A

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

Algoritmul 3.4 FOLLOW

INPUT: G, FIRST(X), $\forall X \in N \cup \Sigma$
OUTPUT: FOLLOW(X), $\forall X \in N \cup \Sigma$
$F(X) = \emptyset, \forall X \in N - \{S\}$; {*inițializare*}
$F(S) = \{\epsilon\}$; {*corespunzător simbolului \$ folosit în analiză*}
repeat
  for $B \in N$ **do**
    for $A \rightarrow \alpha B y \in P$ **do**
      if $\epsilon \in FIRST(y)$ **then**
        $F'(B) = F(B) \cup F(A)$;
      else
        $F'(B) = F(B) \cup FIRST(y)$
      end if
    end for
  end for
until $F'(X) = F(X), \forall X \in N$
$FOLLOW(X) = F(X), \forall X \in N.$

S =>^{0} S // ε after S

Correction:
In order for the algorithm to function correctly, consider:
For ∀a ∈ FIRST(y) do

S => aAc=> abBc
A -> bB