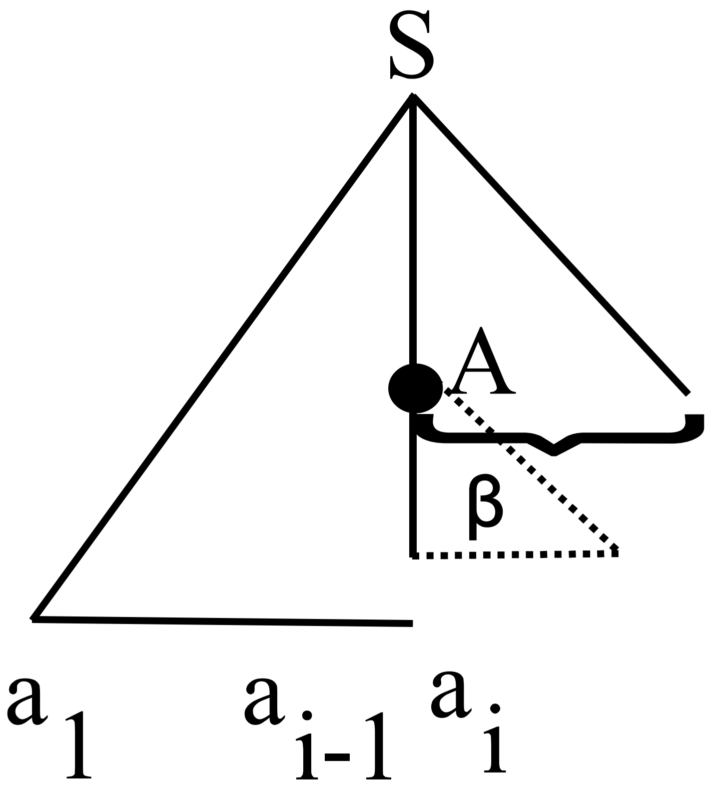
LL(1) Parser



Linear algorithm

Operation: $\oplus$ = concatenation of length 1

L1 = {aa,ab,ba}

L2 = {00,01}

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

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

L2={0,1}

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

FIRST_k

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

$$FIRST_k : (N \cup \Sigma)^* \to \mathcal{P}(\Sigma^k)$$

$$FIRST_k(\alpha) = \{u | u \in \Sigma^k, \alpha \stackrel{*}{\Rightarrow} ux, |u| = k \text{ sau } \alpha \stackrel{*}{\Rightarrow} u, |u| \leq k\}$$

FIRST_k

- Which are the first k terminal symbols that can be generated from A?

- https://forms.office.com/r/kNHNGW7XtC

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

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

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 $A \rightarrow x \in P\}$; {*inițializare*}

repeat

  i := i+1;

A **for** $\forall X \in N$ **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 $F_{i-1}(A) = F_i(A)$

$FIRST(X) := F_i(X), \forall X \in N \cup \Sigma$

A -> BC

B -> DA

D -> a

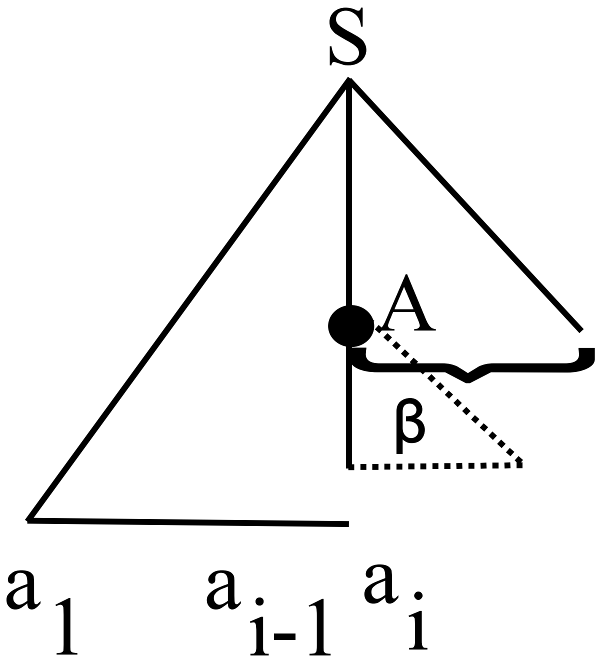
$F_0(A)=F_0(B)=\emptyset$; $F_0(D)=\{a\}$

$F_1(A) = F_0(A) \cup \{\ldots | A\text{->}BC\ F_0(B) \oplus F_0(D)\} = \emptyset$

$F_1(B) = \{a\}$

FOLLOW

$A \to \varepsilon$



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

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

Follow(A)
S=>* xBy => xaAy
What if B->uA

Algorithm FOLLOW

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

```
for A ∈ N – {S} do                          {init}
        L0(A) = Φ;
endFor;
L0(S) = {ε};                                {init}
i = 0;
repeat
   i = i+1;
   for B ∈ N do
        for A → αBy ∈ P do
           for ∀a ∈ FIRST(y) do
              if a = ε then Fi(B) = Fi (B) U Fi-1(A)
                  else Fi(B) = Fi-1(B) U First(y)
              endif
           endFor
        endFor
   endfor
until Fi(X) = Fi-1(X), ∀X ∈ N
FOLLOW(X) = Fi(X), ∀X ∈ N
```

$S \Rightarrow^0 S$ // ε after S

S => aAc=> abBc
A -> bB

FIRST

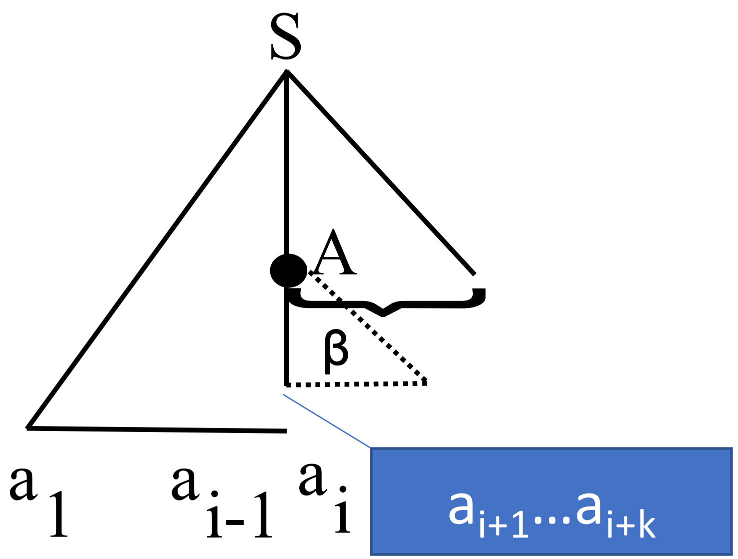
- ≈ first terminal symbols that can be generated from α

FOLLOW

- ≈ next symbol generated after/ following A

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>action is uniquely determinde</u> by:
- Closed part ($a_1 \ldots a_i$)
- Current symbol A
- Prediction $a_{i+1} \ldots a_{i+k}$ (length 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 $A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$, $\mathrm{FIRST}(\alpha_i) \cap \mathrm{FIRST}(\alpha_j) = \emptyset$ and if $\alpha_i \Rightarrow \varepsilon$, $\mathrm{FIRST}(\alpha_i) \cap \mathrm{FOLLOW}(A) = \emptyset$, $\forall i,j = 1,n, i \neq j$

LL(1) Parser

- Prediction of length 1

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

Executed 1 time

Step 2: Construct LL(1) parse table

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

= > table:

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

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 grammar is LL(1) 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: Define configurations and moves

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

  If ($w \in L(G)$) *then* **string of productions**

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

LL(1) configurations

$$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\pi})$$

where:

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

Initial configuration:
(w\$,S\$,ε)

Final configuration:
(\$, \$, π)

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)

alpha := w$;beta := S$;pi := ε; config =(alpha,beta, pi)
go := true;

```
while go do
      if M(head(beta),head(alpha))=(b,i) then
                  ActionPush(config)
      else
            if M(head(beta),head(alpha))=pop then
                  ActionPop(config)
            else
                  if M(head(beta),head(alpha))=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,
              syntax error at", head(alpha))
```

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)