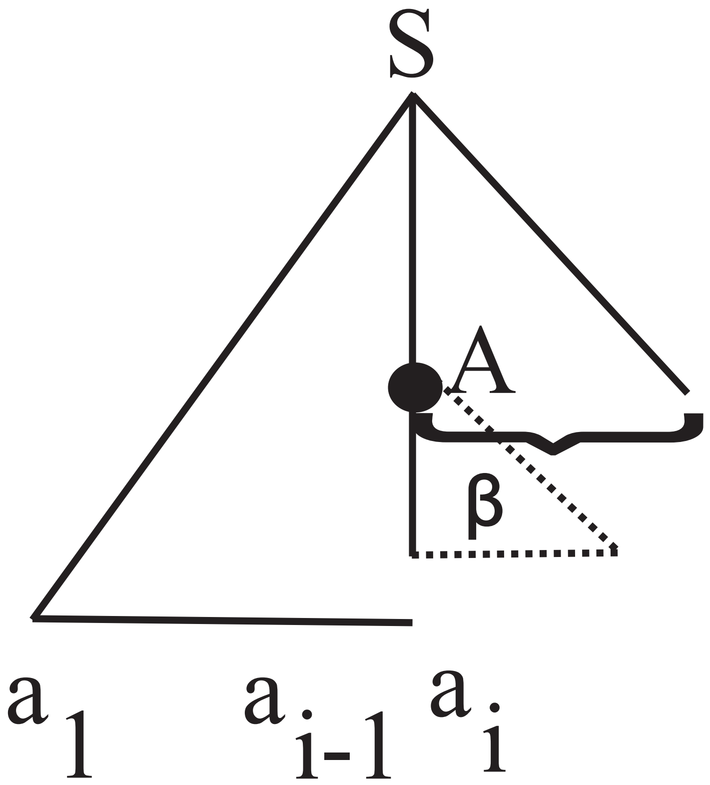
LL(1) Parser



Linear algorithm

FIRST_k

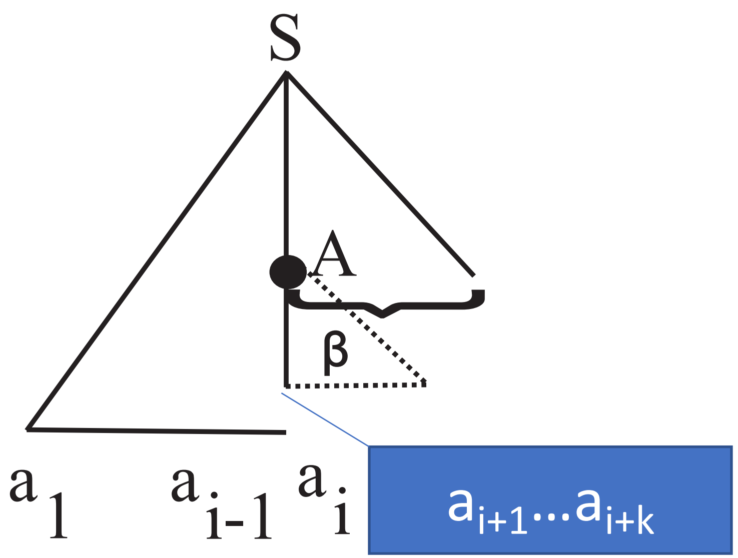
- ≈ first k terminal symbols that can be generated from α

FOLLOW_k

- ≈ next k symbols 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, action is uniquely determinde by:
- Closed part ($a_1...a_i$)
- Current symbol A
- Prediction $a_{i+1}...a_{i+k}$ (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) Analyse sequence based on moves between configurations

Executed 1 time

Step 2: Construct LL(1) parse table

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

= > table:

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

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 $\in$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); add(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!!!

- Teams: 1 – 5
- Each team complete the pattern and then present in front of course

3-2-1 – LL(1) parser

- Make a list with 3 ideas that have been presented at course

1. a
2. A
3. a

- Make a list with 2 examples :

1. A
2. A

- Ask 1 question:

1. ?