Course 6

Problem: Parsing (construct the parsee tree)

if the *source program is sintactically correct*

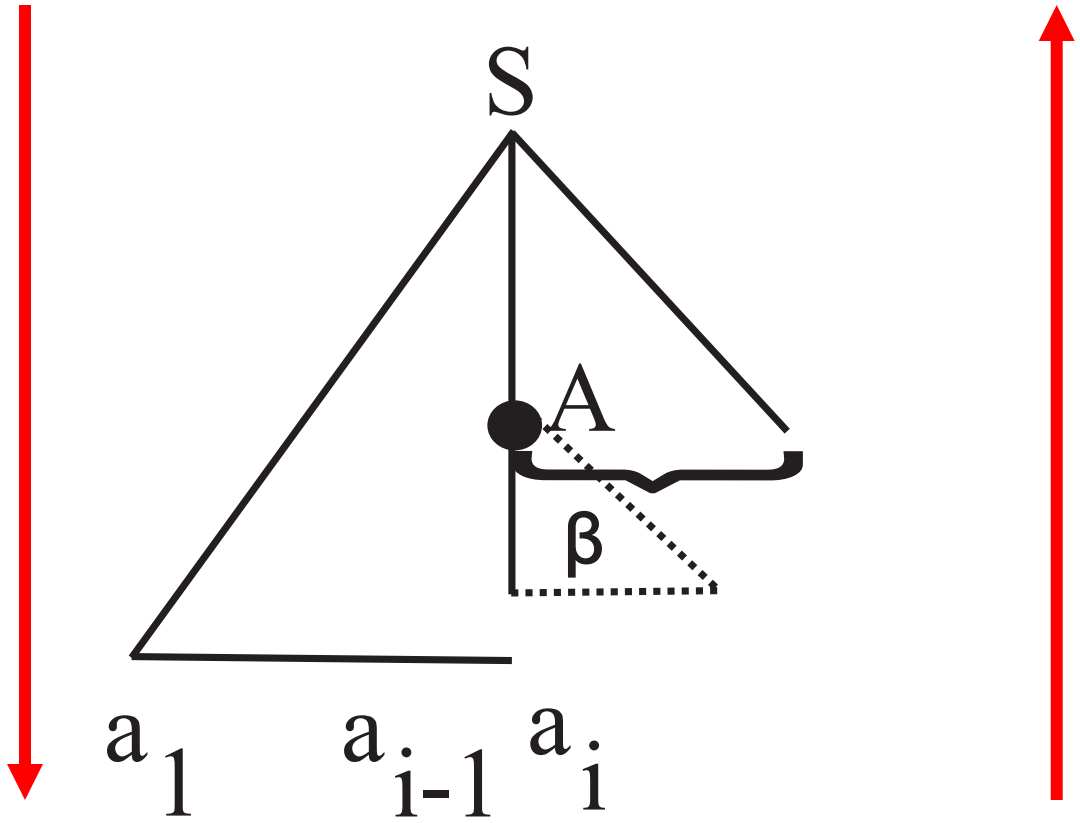
then construct syntax tree

else ”syntax error”

source program is sintactically correct = $w \in L(G) \Leftrightarrow S \overset{*}{\Rightarrow} w$

Parsing

- How:
  1. Top-down vs. Bottom-up
  2. Recursive vs. linear

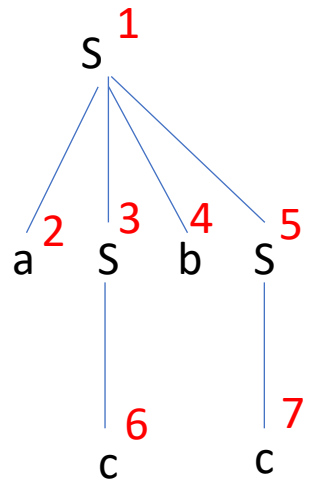


	Descendent	Ascendent
Recursive	Descendent recursive parser	Ascendent recursive parser
Linear	LL(k): LL(1)	LR(k): LR(0), SLR, LR(1), LALR

Result – parse tree -representation

- Arbitrary tree – child sybling representation
- Sequence of derivations S => α_1 => α_2 =>… => α_n = w
- String of production – index associated to prod – which prod is used at each derivation step: 1,4,3,…

index	Info	Parent	Left sibling
1	S	0	0
2	a	1	0
3	S	1	2
4	b	1	3
5	S	1	4
6	c	3	0
7	c	5	0



Descendent recursive parser

- Example

Formal model

- Configuration

$$(s, i, \alpha, \beta)$$

where:

- s = state of the parsing, can be:
  - q = normal state
  - b = back state
  - f = final state - corresponding to success: w ∈ L(G)
  - e = error state – corresponding to insuccess: w ∉ 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

$(q,i, \alpha, A\beta) \vdash (q,i, \alpha A_1, \gamma_1\beta)$

where:

$A \rightarrow \gamma_1 \mid \gamma_2 \mid \ldots$ represents the productions corresponding to A

1 = first prod of A

Advance

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

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

Momentary insuccess

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

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

Back

WHEN: head of working stack is a terminal

(b,i, αa, β) ⊢ (b,i-1, α, aβ)

Another try

WHEN: head of working stack is a nonterminal

$(b,i, \alpha\ A_j, \gamma_j\ \beta) \vdash$ $(q,i, \alpha A_{j+1}, \gamma_{j+1}\beta)$, if $\exists\ A \rightarrow \gamma_{j+1}$

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

(e,i, α, β), if i=1, A =S, **ERROR**

Success

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

Algorithm

Algorithm Descendent Recursive

INPUT: G, w =$a_1a_2...a_n$
OUTPUT: string of productions and message

```
config = (q,1, ε,S);                                   //initial configuration (s,i,α,β)
while (s ≠ f) and (s ≠ e) do
   if s = q
     then
          Success(config)
              else
                   if Head(β) = A
                     then Expand(config)
                     else
                       if  Head(β) = a_i
                           then Advance(config)
                           else MomentaryInsuccess(config)
     else
          if s = b
            then
                 if Head(α) = a
                   then Back(config)
                   else AnotherTry(config)
endWhile
if s = e  then message"Error"
          else message "Sequence accepted";
               BuildStringOfProd(α)
```

w ∈ 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

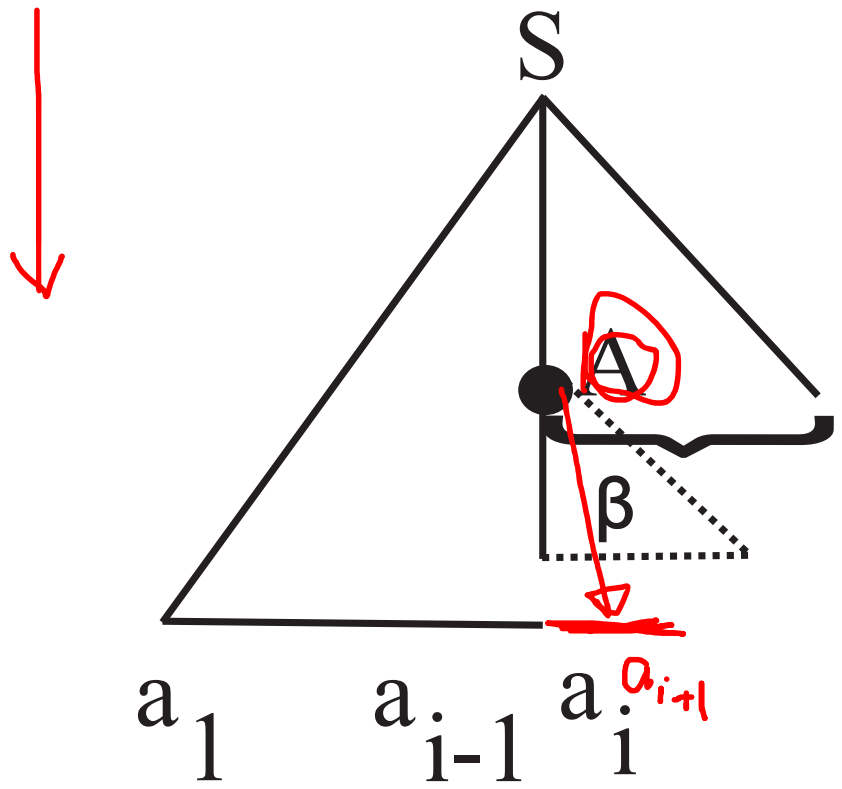
When the algorithm never stops?

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

- Left recursive can be equiv. transformed in a right recursive grammar [Aho]

- A -> Ax transf into A -> yA

LL(1) Parser



Linear algorithm

$FIRST_k$

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

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

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

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

L1 = {aa,ab,ba}

L2 = {00,01}

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

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

L1 ={a,b}

L2 ={0,1}

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

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

L2={0,1}

L1L2 = {a0,a1,0,1}

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