

## Analizador Sintático (AS)

Um AS deve responder a seguinte pergunta:

A entrada é válida? Se for, qual é a sua estrutura (árvore de derivação). A árvore de derivação é então utilizada em outras fases da compilação.

### Dois métodos de AS

a) TOP-DOWN – O AS “descobre” qual é a árvore de derivação correspondente à sequência de tokens iniciando pelo topo da árvore (o símbolo inicial), e então expandindo a árvore de uma forma depth-first (profundidade). É uma análise descendente e um caminhar em pré-ordem da árvore de derivação.

b) BOTTON-UP (ascendente) – O AS “descobre” qual é a árvore de derivação iniciando pelo fundo (as folhas da árvore, que são os símbolos terminais) e determinando as produções usadas para gerar as folhas. Estas produções são usadas para gerar os pais imediatos das folhas descobertas. O AS continua até chegar ao símbolo inicial. O AS é feito em pós-ordem da árvore de derivação.

### AS TOP-DOWN

Há dois tipos de análise:

-Com retrocesso : são realizadas algumas tentativas para reconhecer a entrada.

-Sem retrocesso: não ha tentativas repetidas porque a gramática está adequada a análise sem retrocesso.

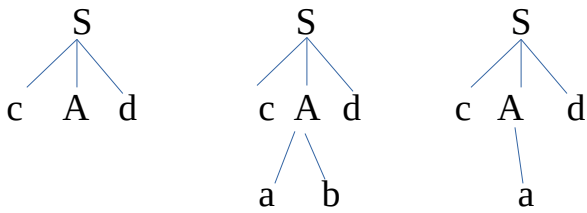
Não há recursividade à esquerda nem necessidade de fatoração à esquerda.

Ex. de AS descendente recursiva com retrocesso

-Gramática

$$\begin{array}{l} S \rightarrow cAd \\ A \rightarrow ab \mid a \end{array} \implies A \rightarrow a(b \mid \epsilon) \quad \text{fatoração}$$

-cadeia cad



## FATORAÇÃO À ESQUERDA

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$$

$$A \rightarrow \alpha (\beta_1 \mid \beta_2 \mid \dots \mid \beta_n) \mid \gamma$$

$$A \rightarrow \alpha A' \mid \gamma$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

Ex.:  $S \rightarrow cAd$   
 $A \rightarrow ab \mid a$



$$A \rightarrow aA'$$
$$A' \rightarrow b \mid \epsilon$$

$$\langle \text{variável} \rangle (\text{id} \mid \text{id} [\text{expressão}]) \Rightarrow \langle \text{variável} \rangle \text{id} \langle X \rangle$$
$$\langle X \rangle \rightarrow \epsilon \mid [\text{expressão}]$$

## ELIMINAÇÃO DE RECURSÃO À ESQUERDA

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$
$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A'$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \varepsilon$$

Ex.:

$$\begin{array}{ccccccc} \langle \text{listaDeIdentificadores} \rangle & \rightarrow & \text{id} & \mid & \langle \text{listaDeIdentificadores} \rangle & , & \text{id} \\ & & A & & \beta & & A & & \alpha \end{array}$$
$$\langle \text{listaDeIdentificadores} \rangle \rightarrow \text{id} \langle F \rangle$$
$$\langle F \rangle \rightarrow , \text{id} \langle F \rangle \mid \varepsilon$$

Exemplo de AS descendente recursivo sem retrocesso.

--Gramática

$$S \rightarrow (L) \mid a$$
$$L \rightarrow L, S \mid S$$

FIRST (X) : é o primeiro símbolo terminal de X

$$\text{First}(S) = \{(, a\} \quad \text{First}(L) = \text{First}(S) = \{(, a\}$$
$$L \rightarrow S L'$$
$$L' \rightarrow , S L' \mid \varepsilon$$
$$\text{First}(L') = \{', ', \varepsilon\}$$

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
char buffer[80], lookahead;
```

```
int i=0;
```

```
void reconhecedor(char t) {
```

```
    if (lookahead ==t)
```

```
        lookahead ==buffer[i++];
```

```
    else
```

```
        erro();
```

```
}
```

```

void erro(){
    printf("\n Erro Sintático");
    exit(1);
}

```

```

void gets(char *s, int size){

    fgets(s, size, stdin);
    s[strlen(s)-1]='\0';
}

```

```

int main(){ //testar para a,a,a,a
    printf("\n Cadeia");
    gets(buffer, 80);
    lookahead = buffer[i++];
    S(); //S → (L) | a
    if (lookahead == '\0')
        printf("\nAnálise Sintática completa com sucesso");
    else
        printf("\nEra esperado fim da cadeia");
    getchar();
}

```

```

void S(){ //S → (L) | a

    if (lookahead=='('){
        reconhecedor('(');
        L();
        reconhecedor(')');
    }
    else if (lookahead == 'a')
        reconhecedor('a');
    else erro();

}

```

```

void L(){ //L → S L'

    S();
    L_();

}

```

```
void L_() { //L' -> ,SL' | ε
    if(lookahead==''){
        reconhecedor(',');
        S();
        L_();
    }
}
```