



Construção de Compiladores

Alterado por: prof. Marlon Oliveira
Elaborado por: Prof^a Christine Vieira

ANÁLISE SINTÁTICA OU PARSER

- É um algoritmo
- Tem por finalidade verificar se:
 - o programa que está sendo compilado possui ou não erros de sintaxe. Verifica se as regras (produções) da gramática que definem a linguagem em questão, foram observadas na construção do programa fonte.

Exemplo:

| | |
|------------|--|
| 1 | SENT::=SUJ PRED |
| 2,3,4 | SUJ::=SUBST ART SUBST ART ADJ SUBST |
| 5 | PRED::=VERBO OBJ |
| 6,7,8,9,10 | SUBST::=joão maria cão livro pão |
| 11,12 | ART::=o a |
| 13,14,15 | ADJ::=pequeno bom bela |
| 16,17,18 | VERBO::=mordeu leu olha |
| 19,20,21 | OBJ::=SUBST ART SUBST ART ADJ SUBST |

Verificar se a sentença “O cão leu Maria” pertence a esta linguagem.

SENT::=SUJ PRED

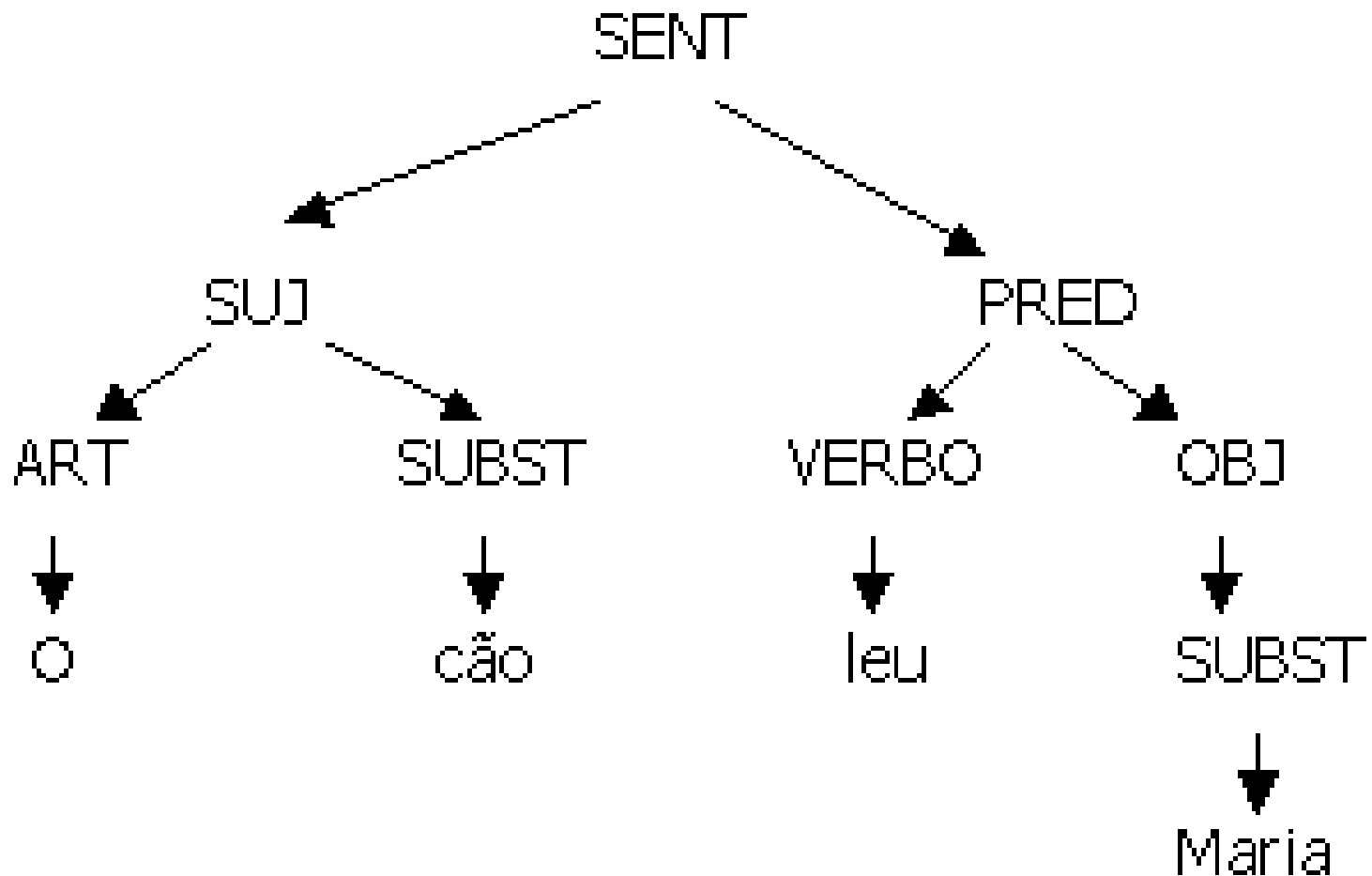
SENT::=ART SUBST VERBO OBJ

SENT::= o cão leu SUBST

SENT::= o cão leu Maria

- Sintaticamente correta
- Semânticamente incorreta.

Árvore de derivação



Classe de analisadores sintáticos

Dividido em 2 classes:

Ascendentes
BOTTON-UP

e

Descentenes
TOP-DOWN



COM back-tracking

SEM back-tracking

Analísadores Ascendentes (BOTTOM-UP)

Procura chegar ao símbolo inicial da gramática a partir da sentença a ser analisada.

Sentença -> Símbolo Inicial

Exemplo:

1 $S ::= aABe$
2,3 $A ::= Abc \mid b$
4 $B ::= d$

A sentença `abbcde` pode ser reduzida a `S` da seguinte forma:

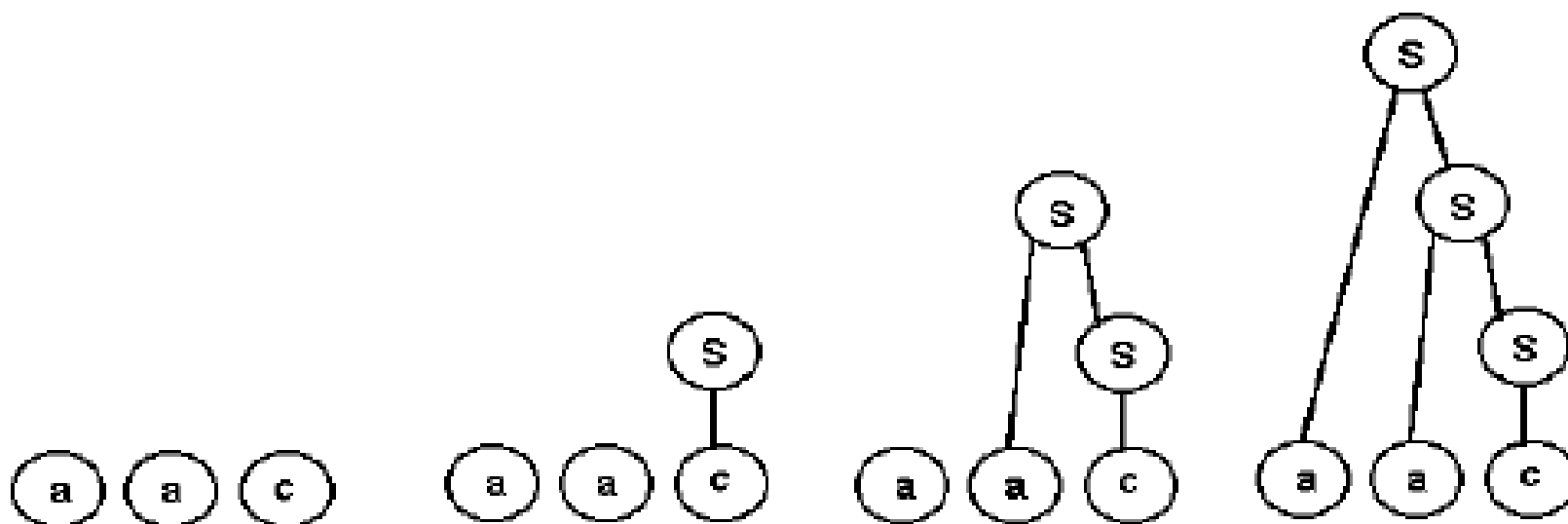
$S ::= a\underline{b}bcde$
 $S ::= a\underline{Abc}de$ 3
 $S ::= aA\underline{d}e$ 2
 $S ::= a\underline{AB}e$ 4
 $S ::= S$ 1

■

Ex:

$S ::= aS \mid c$

$w = aac$



Analísadores Descendentes (TOP-DOWN)

Procura-se chegar a sentença a partir do símbolo inicial da Gramática G.

Símbolo Inicial -> Sentença

Exemplo:

1 $S ::= aABe$
2,3 $A ::= Abc \mid b$
4 $B ::= d$

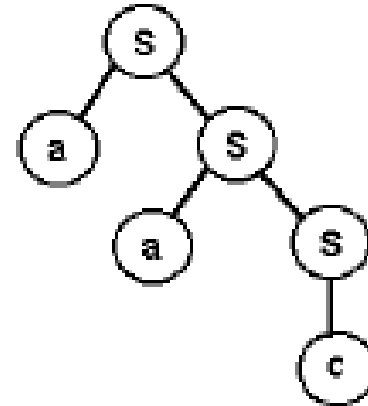
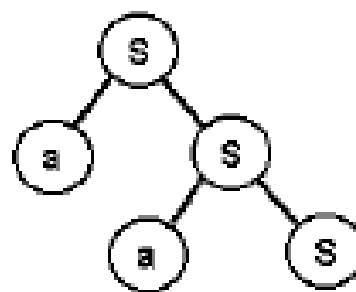
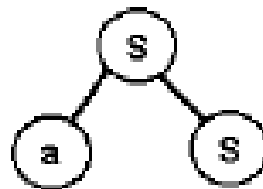
A sentença abbcde pode ser encontrada da seguinte forma:

| | |
|----------------------------|---|
| $S ::= aA\underline{B}e$ | 1 |
| $S ::= aA\underline{d}e$ | 4 |
| $S ::= aA\underline{b}cde$ | 2 |
| $S ::= abbcde$ | 3 |

Ex:

$S ::= aS \mid c$

$w = aac$



ANALISADORES ASCENDENTES (Booton-UP) (Família LR)

Sentença -> Símbolo Inicial

Baseia-se em uma técnica primitiva denominada **Shift-Reduce (Avança-Reduz)**, ou **empilhar e reduzir**.

Importância:

- Analisam praticamente todas as construções sintáticas de linguagens que podem ser produzidas por GLC.
- Detecção imediata de erro sintático.
- Tempo de análise é proporcional ao tamanho da sentença.
- Técnicas determinísticas, em qualquer situação existirá uma ação a ser tomada.

Pontos negativos:

- Complexidade de construção da tabela de parsing.
- Requer muito espaço em disco e memória para o seu armazenamento.
- Requer muito tempo para análise se a sentença for grande;
- Detecta o erro sintático após consumir toda a sentença a ser analisada;
- Pode rejeitar sentenças corretas.

Para implementar é necessário:

- Algoritmo de análise sintática ascendente.
- Tabela de análise sintática ou parser.
- Pilha de estados (ou pilha sintática) que conterà um histórico da análise efetuada.
- Buffer de entrada (contendo o token a ser analisado).
- Uma GLC com produções numeradas de 1 a p.

ANALISADORES DESCENDENTES SLR (LL(1))

Preditivo Tabular

Símbolo Inicial -> Sentença

Resume-se a uma tentativa de construir a derivação mais a esquerda da sentença da entrada.

Pode ser classificado em:

Analizador Sintático Descendente **COM back-tracking**

1 $S ::= cAd$

2,3,4,5 $A ::= ab \mid a \mid aAa \mid \varepsilon$

Sentença a ser analisada: “cad”

$S ::= cAd$ Regra 1

$S ::= cabd$ Regra 2

“Não reconheceu”

$S ::= cAd$ Regra 1

$S ::= cad$ Regra 3

“Reconheceu”

Vantagens:

- Um conjunto maior de GLCs pode ser analisado.

Desvantagens:

- Maior tempo para a análise.
- Dificuldade na recuperação de erros.
- Problemas na análise semântica e geração de código.

*Problemas decorrentes do **não-determinismo**.*

Analizador Sintático Descendente **SEM back-tracking**

1 $S ::= cAd$

2,3,4,5 $A ::= ab \mid cd \mid dAd \mid \varepsilon$

Sentença a ser analisada: "ccdd"

$S ::= cAd$ Regra 1

$S ::= ccdd$ Regra 3

"Reconheceu"

Desvantagem: Limita a classe de gramáticas que podem ser analisadas.

Vantagens:

- determinismo
- superam as deficiências da técnica COM Back-Tracking

ANALISADORES DESCENDENTES SEM BACK-TRACKING

A GLC deve satisfazer as seguintes condições:

- a) Não possuir recursão à esquerda;
- b) Estar fatorada;

Para todo $A \in N$, tal que $A ::=^* \varepsilon$,

$$\text{FIRST}(A) \cap \text{FOLLOW}(A) = \emptyset$$

Técnica de Implementação

- Consiste na construção de um conjunto de procedimentos
- Um para cada símbolo não terminal da gramática em questão.

Desvantagens:

- a. Não é geral, os procedimentos são específicos para cada gramática.
- b. Necessidade de uma linguagem que permita recursividade
- c. Aceita uma classe restrita de linguagens

Vantagens:

- a. Simplicidade na construção da matriz de parsing
- b. Algoritmo simples e de fácil implementação

Implementação

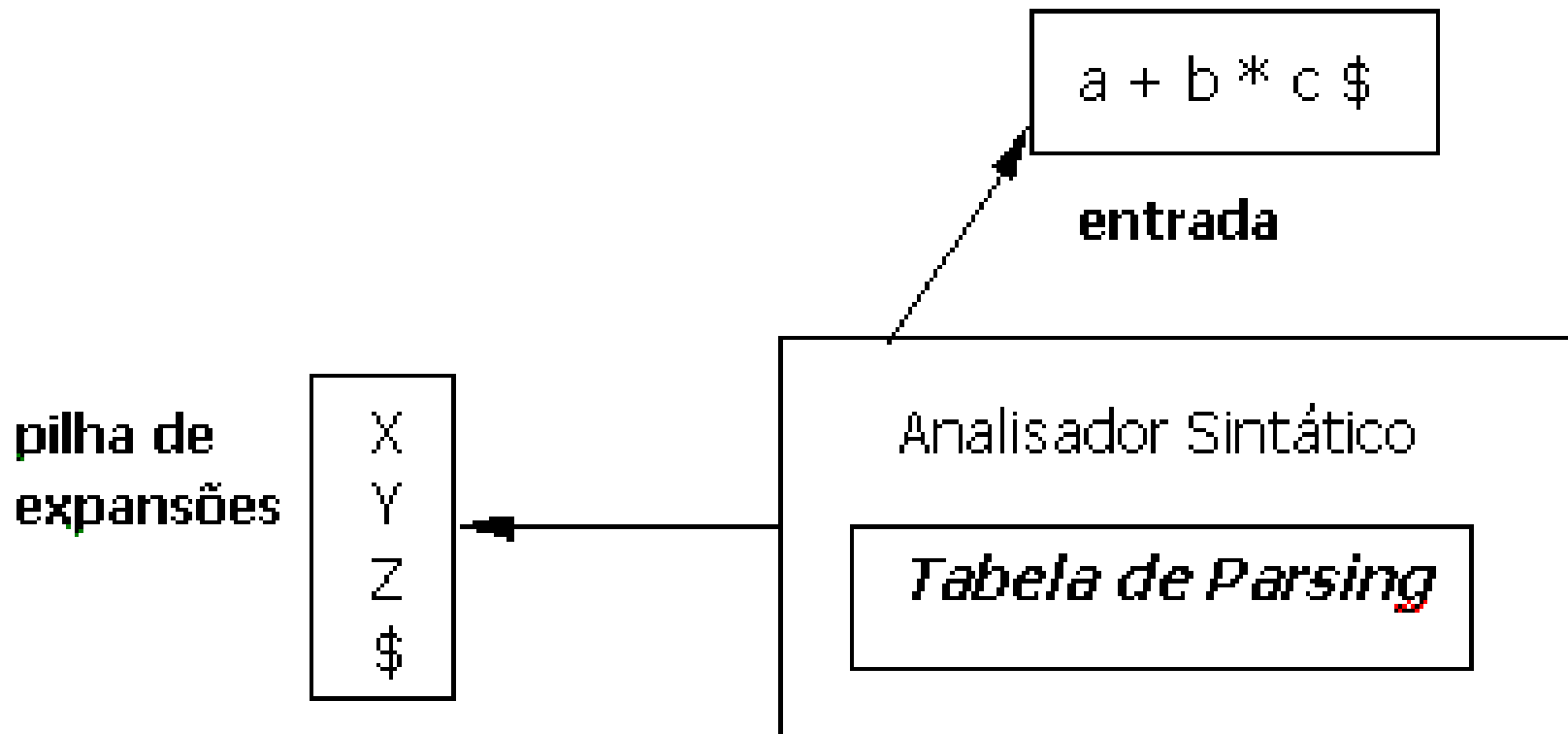
Entrada: contendo o token a ser analisado

Pilha: usada para simular a recursividade, prevê a parte da entrada que está para ser analisada. Ela é inicializada com \$ e o símbolo inicial da gramática em questão.

Tabela de parsing: ou tabela de análise sintática. Contém as ações a serem efetuadas. É uma matriz $M(X,a)$, onde $X \in N$ e $a \in T$.

Algoritmo de análise sintática descendente sem back-tracking

MODELO



Algoritmo

Início

X recebe o topo da pilha

“a” recebe o símbolo da entrada

Repita

Se $X = \hat{}$ então

Retire o elemento do topo da pilha

X recebe o topo da pilha

Senão

Se X é terminal então

Se $X = a$ então

Retire o elemento do topo da pilha

Sai do Repita

Senão

Continua...

Erro

Encerra o programa

Fim Se

Senão (* X é não-terminal*)

Se $M(X,a) \neq \emptyset$ então (existe uma regra)

Retire o elemento do topo da pilha

Coloque o conteúdo da regra na pilha

X recebe o topo da pilha

Senão

Erro

Encerra o programa

Fim Se

Fim Se

Até $X=\$$ (*pilha vazia, análise concluída*)

Fim

TABELA DE PARSING

Condições necessárias para montar a tabela:

- a. A gramática deverá ser uma Gramática Livre de Contexto
- b. A gramática deve estar fatorada;
- c. A gramática não deve possuir recursão à esquerda;
- d. Para todo $A \in N$, tal que $A ::=^* \varepsilon$,
$$\text{FIRST}(A) \cap \text{FOLLOW}(A) = \emptyset$$

Regras do Follow(A)

- 1 – Sempre o FOLLOW do símbolo inicial conterá \$
- 2 – Para $A \rightarrow \alpha B \beta$ então $\text{FIRST}(\beta)$ exceto \hat{a} está em $\text{FOLLOW}(B)$
- 3 – Para $A \rightarrow \alpha B$ ou $A \rightarrow \alpha B \beta$, onde o $\text{FIRST}(\beta)$ contém \hat{a} , então deve-se incluir o $\text{FOLLOW}(A)$ em $\text{FOLLOW}(B)$

- 1 $E ::= TE'$
- 2,3 $E' ::= +TE' | \varepsilon$
- 4 $T ::= FT'$
- 5,6 $T' ::= *FT' | \varepsilon$
- 7,8 $F ::= (E) | id$

| Produções | First (α) | Follow (Δ) |
|---|--------------------|---------------------|
| 1) $E ::= TE'$ | (, id |), \$ |
| 2) $E' ::= +TE'$ 3) $E' ::= \varepsilon$ | + ε |), \$ |
| 4) $T ::= FT'$ | (, id | +,), \$ |
| 5) $T' ::= *FT'$ 6) $T' ::= \varepsilon$ | * ε | +,), \$ |
| 7) $F ::= (E)$ 8) $F ::= id$ | (id | +, *,), \$ |

Para montar a tabela sempre colocar as regras para o conjunto First, quando este tiver o símbolo vazio usa-se o conjunto Follow

TABELA DE PARSING

| | id | + | * | (|) | \$ |
|----|----|---|---|---|---|----|
| E | 1 | | | 1 | | |
| E' | | 2 | | | 3 | 3 |
| T | 4 | | | 4 | | |
| T' | | 6 | 5 | | 6 | 6 |
| F | 8 | | | 7 | | |