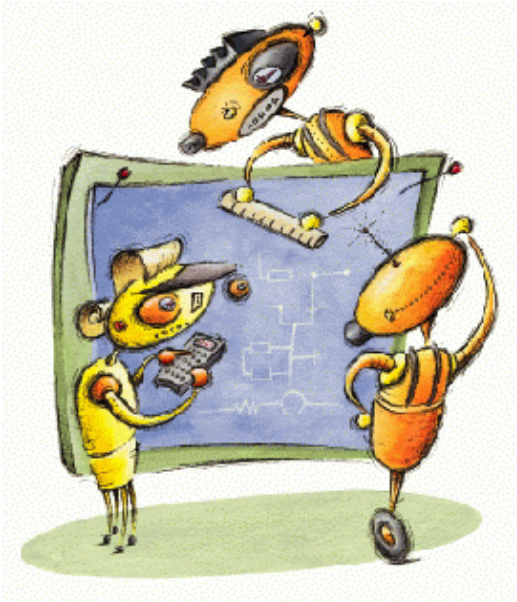
PUCRS | ESCOLA POLITÉCNICA

Análise Sintática

Profa. Dra. Andréa A. Konzen



Linguagens de Programação

Análise Sintática

O Analisador Sintático obtém uma cadeia de tokens proveniente do analisador léxico e verifica se o mesmo pode ser gerado pela gramática da linguagem-fonte.

Espera-se que o analisador sintático relate quaisquer erros de sintaxe e também recupere erros que ocorrem mais comumente, a fim de poder continuar processando o resto de sua entrada.

Análise Sintática

Os métodos de análise sintática mais usados são classificados como ***top-down*** ou ***botton-up***.

Os AS ***top-down,*** constroem uma árvore do topo (raiz) para o fundo (folha)

Os AS ***botton-up,*** começam pelas folhas e trabalha a árvore acima até a raiz

Análise Sintática

- Os analisadores implementados manualmente trabalham freqüentemente com a gramática LL.
- Os analisadores da classe mais ampla das gramáticas LR., são usualmente construídos através de ferramentas automatizadas (Lex & Yacc).
- A sintaxe da construção de uma linguagem de programação pode ser descrita pela GLC (Gramática Livre de Contexo) ou pela notação BNF(Forma de Backus-Naur).
- Vantagens da Utilização da gramática:
  - Precisão (Ver o que se pode ou não fazer)
  - Tirar ambiguidades
  - Não determinismo, Recursividade
  - Adaptabilidade, Tratamento de erro

Tratamento dos erros

- Um bom compilador deveria assistir o programa na identificação e localização dos erros.
- Os erros podem ser:
  - Léxicos, tais como símbolos desconhecidos, erro de grafia de um identificador, palavra-chave ou operador.
  - Sintáticos, tais como uma expressão aritmética com parênteses não balanceados.
  - Semânticos, tais como um operador aplicado a um outro operador incompatível.
  - Lógicos, tais como uma chamada infinitamente recursiva.

Tratamento dos erros

O tratador de erros num analisador sintático possui metas simples a serem estabelecidas:

- Relatar de forma clara apurada a presença de erros
- Recuperar cada erro suficientemente rápido a fim de ser capaz de detectar erros subsequentes.

Uma das estratégicas de recuperação de erros é a Correção global

Árvore para análise sintática

- Como podemos determinar se uma sequência de códigos está sintaticamente correta?
- Utilizando as regras da gramática, a sintaxe está correta se a string for derivada do símbolo inicial.
- Os algoritmos são desenvolvidos para gerar derivações da string na linguagem da gramática. Quando a entrada “string” não fizer parte da linguagem, o processo deve identificar que não existe derivação. O procedimento que faz esta função é chamado de “parsing”.

Analisador sintático ou parsing

- O analisador sintático recebe do analisador léxico, o código-fonte (tokens) e efetua a análise sintática.
- A análise sintática determina os elementos estruturais do programa e seus relacionamentos. Os resultados da análise sintática são geralmente representados como uma árvore sintática

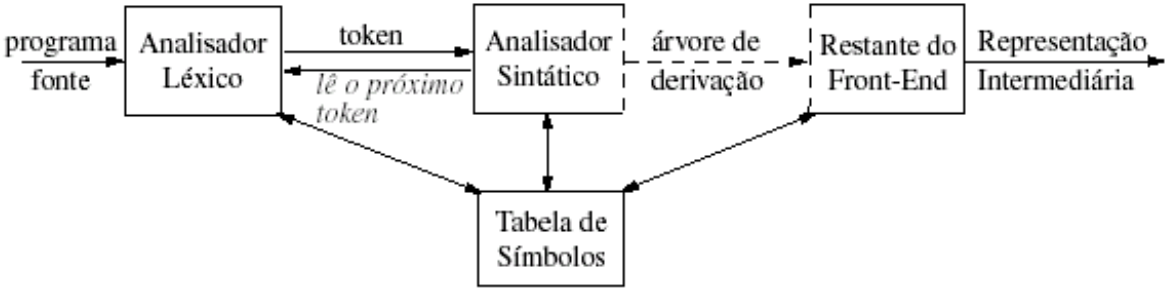
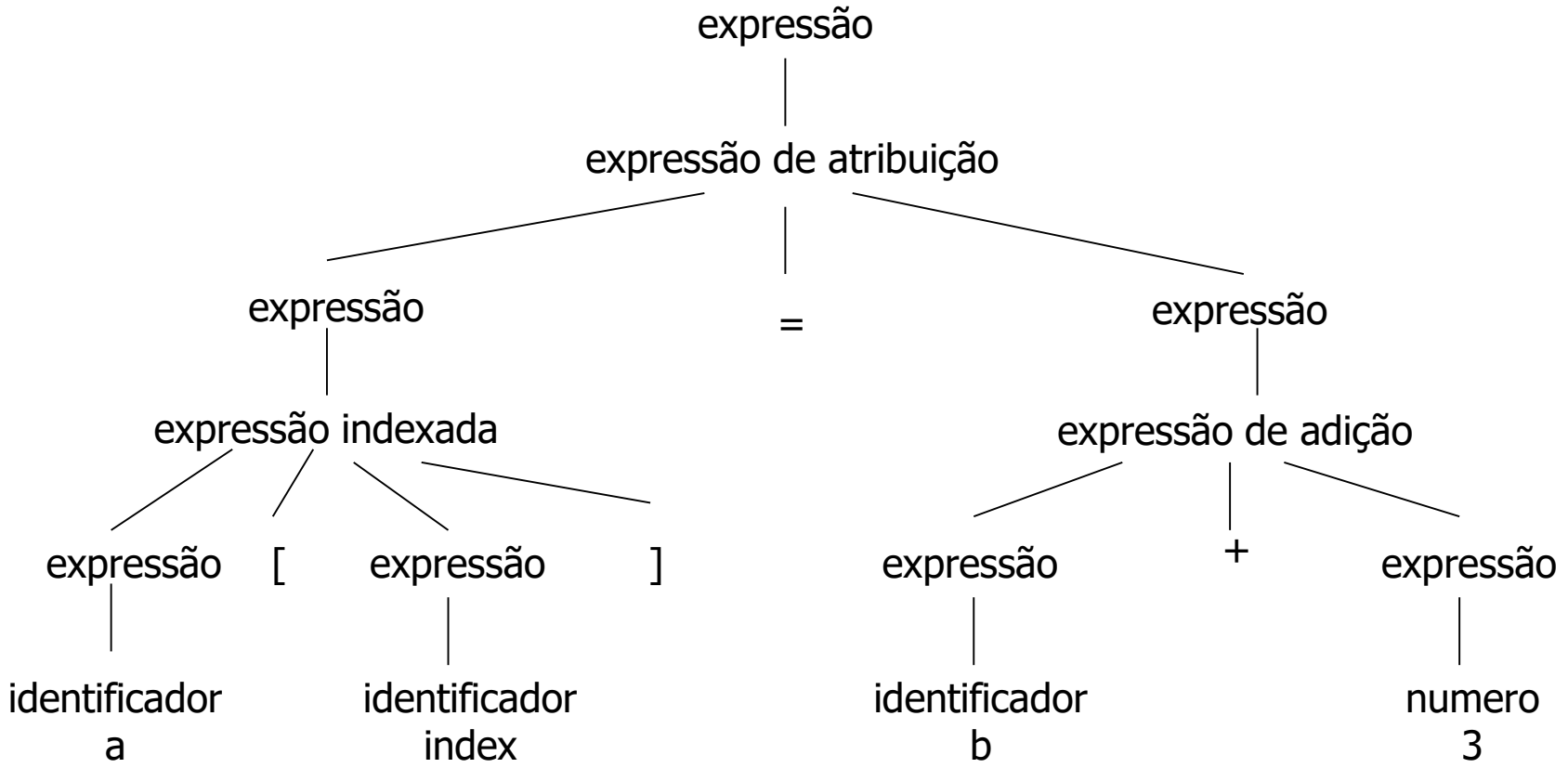


FIGURA 4.1 Posição do analisador sintático no modelo de compilador.

Analisador sintático ou parsing

- Análise do programa fonte considerando-o não mais como uma sequência de caracteres, mas sim como uma sequência de lexemas;
- Verificação da ordem como são colocados os lexemas do programa fonte;

Analisador sintático ou parsing



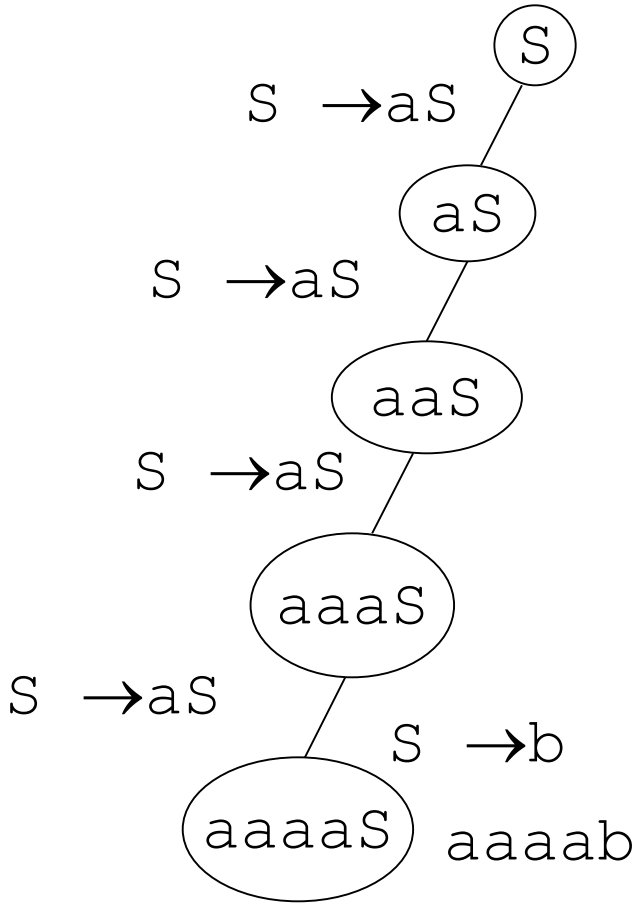
Algoritmo para derivação

- Objetivo – Extrair a estrutura sintática de um programa
  - Primeiro, deveremos especificar a estrutura sintática da linguagem.
  - Uso de gramáticas
- Comece com o símbolo inicial
- Repita até que não exista mais “não terminais”
  - escolha um “não terminal”
  - escolha uma produção para o “não terminal”
  - substitua a ocorrência do “não terminal” na derivação com o lado direito da produção.
- Casos especiais de derivação
  - Leftmost
  - Rightmost

Continuação ...

Seja G = (V,Σ,P,S) onde V = {S}, Σ={a,b} e P é dada pelas regras:

S→aS|b



Derivação mais à Esquerda

Leftmost derivation

(1) $S \rightarrow \varepsilon$

(2) $S \rightarrow SS$

(3) $S \rightarrow (S)$

$w \rightarrow (())\,()$

$S \overset{n}{\underset{L}{\rightarrow}} w$

$S \rightarrow SS$ (2)

$\rightarrow (S)\,S$ (3)

$\rightarrow ((S))\,S$ (3)

$\rightarrow (())\,S$ (1)

$\rightarrow (())\,(S)$ (3)

$\rightarrow (())\,()$ (1)

$S \overset{6}{\underset{L}{\rightarrow}} w$

Derivação mais à direita

rightmost derivation

(1) $S \rightarrow \varepsilon$

(2) $S \rightarrow SS$

(3) $S \rightarrow (S)$

$w \rightarrow (())\,()$

$S \overset{n}{\underset{R}{\rightarrow}} w$

$S \rightarrow SS$ (2)

$\rightarrow S(S)$ (3)

$\rightarrow S((S))$ (3)

$\rightarrow S(())$ (1)

$\rightarrow (S)\,(())$ (3)

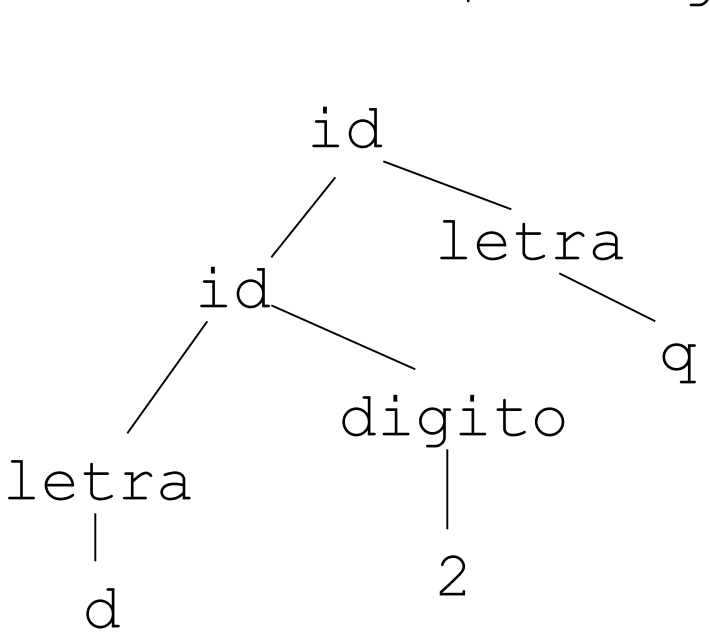
$\rightarrow (())\,()$ (1)

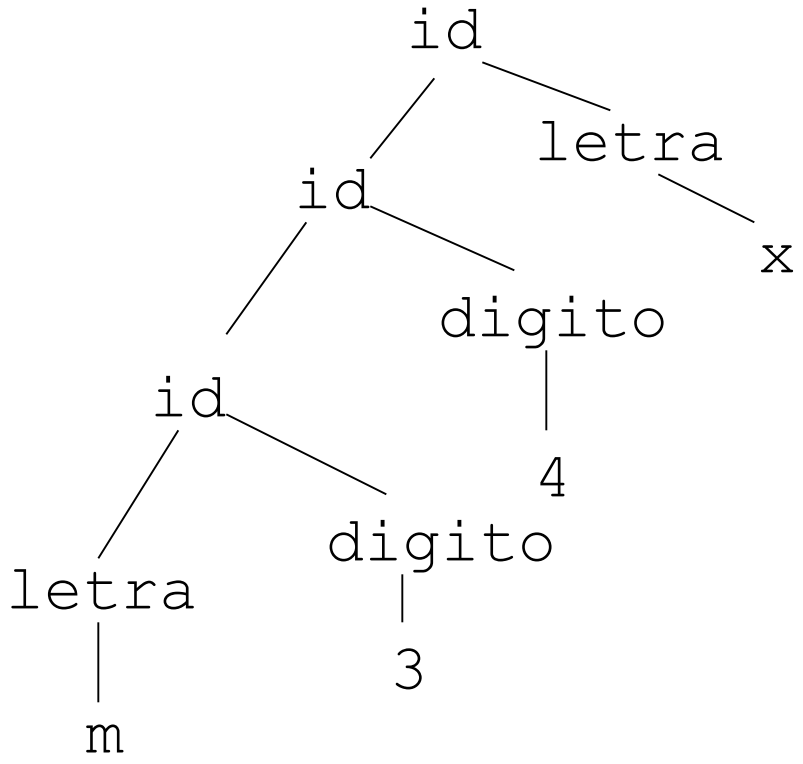
$S \overset{6}{\underset{R}{\rightarrow}} w$

Continuação ...

Seja G = (V,Σ,P,S) onde V = {id,letra,digito}, Σ={a-b,0-9} e P é dada pelas regras:

id→id letra|id digito|letra





Problemas de uma gramática

Ambiguidade: uma gramática é ambígua quando é possível criar árvores de derivações diferentes para gerar a mesma palavra.

Recursão à esquerda: quando o primeiro símbolo do lado direito da regra da produção é o mesmo não terminal do lado esquerdo da produção.

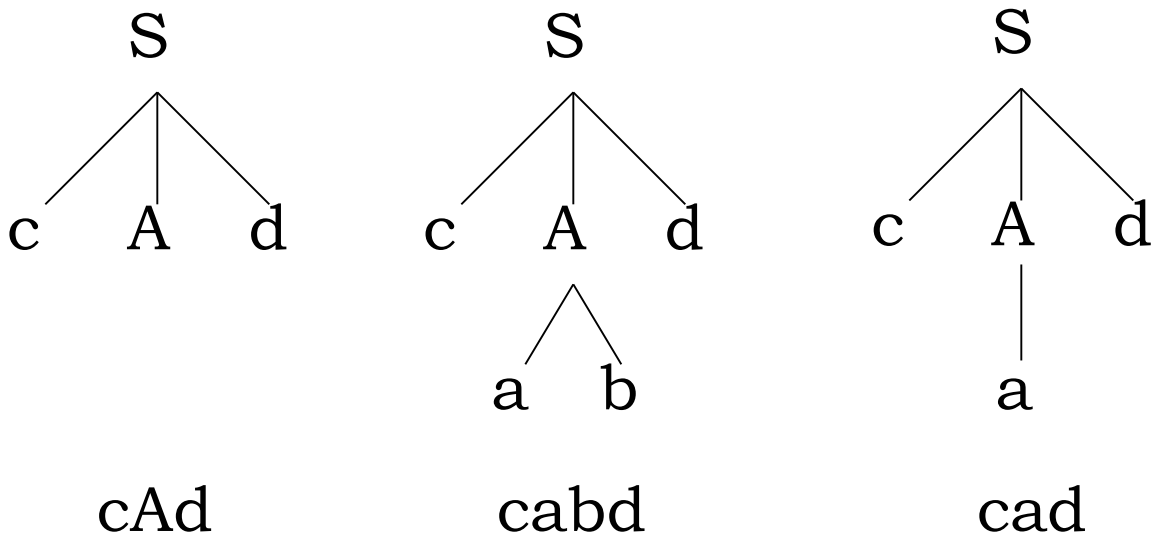
Fatoração: quando existe um não-determinismo nas regras de produções da gramática.

Análise Sintática de
Descendência Recursiva

A análise sintática top-down pode ser vista como uma tentativa de se encontrar uma derivação mais à esquerda para uma cadeia de entrada ou árvore gramatical.

$S \rightarrow cAd$

$A \rightarrow ab \mid a$



Análise Sintática de
Descendência Recursiva

É uma tentativa de encontrar uma derivação mais à esquerda para uma cadeia de entrada.

- Não pode haver recursividade à esquerda;
- A gramática deverá estar fatorada à esquerda.

Gramática G1

E → E+T | E-T | T
T → T*F | T /F| F
F → (E) | id

Resolver a Recursividade à esquerda

E → TE′
E′ → +TE′ | -TE′ | ε
T → FT′
T′ → *FT′ | /FT′ | ε
F → (E) | id

Análise Sintática de
Descendência Recursiva

```
E →TE'
```

```
E( ) {
    T( );
    E'( );
}
```

```
T →FT'
```

```
T( ){
    F( );
    T'( );
}
```

```
E' →+TE'|-TE'|ε
```

```
E'( ) {
    if(token = "+"  ) {
        reconhece ( "+" );
        T( );
        E'( );
    }else if(token = "-"){
            reconhece( "-" );
            T( );
            E'( );}
        else
            break;
     }
}
```

Análise Sintática de
Descendência Recursiva

T' →*FT'|/FT'|ε

```
T'( ) {
  if(token = "*"  ){
    reconhece ( "*" );
    F( );
    T'( );
  }else if(token = "/"){
    reconhece( "/" );
    F( );
    T'( );
  }else
    break;
}
```

F →(E)|id

```
F( ) {
  if(token = "("  ){
    reconhece ( "(" );
    E( );
    reconhece( ")" );
  }
  else
    reconhece( "id");
}
```

Análise Sintática

Gramática G_2

$S \rightarrow aAb \mid b$

$A \rightarrow aA \mid B$

$B \rightarrow bB \mid \varepsilon$

```
S( ){
 if(token = "a"){
    reconhece ("a");
    A( );
    reconhece("b");
 }else if (token = "b"){
    reconhece ("b");
  }
}
```

```
A ( ) {
  if(token = "a"){
       reconhece ("a");
       A( );
    }else B( );
}
```

```
B ( ) {
   if(token = "b"){
       reconhece ("b");
       B( );
   }else
      break
}
```

Análise Sintática

Gramática G_3

$S \rightarrow Sab|c|\ \varepsilon$

$S \rightarrow cS'$

$S' \rightarrow abS'|\varepsilon$

```
S(){
  if(token = "c")
    reconhece("c");
    S'();
}
```

```
S'( ) {
   if(token = "a"  ){
      reconhece ( "a" );
      reconhece("b");
      S'( );
   }else
      break;
}
```

Análise Sintática

Gramática G_4

S →EF
E →1E|0E|ε
F →D|λ
D →1D|0D|A
A →1|0

```
S(){
   E();
   F();
}
```

```
D(){
   if(isdigit(token)=1){
        reconhece (1);
        D();
   } else if(isdigit(token)=0){
        reconhece(0);
        D();
   } else A ();
}
A(){
   if(isdigit(token)=1){
        reconhece (1);
        else
      reconhece (0);
}
```

Análise Sintática

```
E(){
     if(isdigit(token)=1){
          reconhece (1);
          E ( );
       } else if(isdigit(token)=0){
          reconhece ( 0);
          E ( );
        } else break;
}
F(){
    if(“ε”)
        break
      else
        D();
}
```

Análise Sintática

Gramática G_5

```
E →TR
R →+TR|-TR| ε
T →0|1|2| ...|9
```

```
E ( ){
    T( );
    R( );
}
```

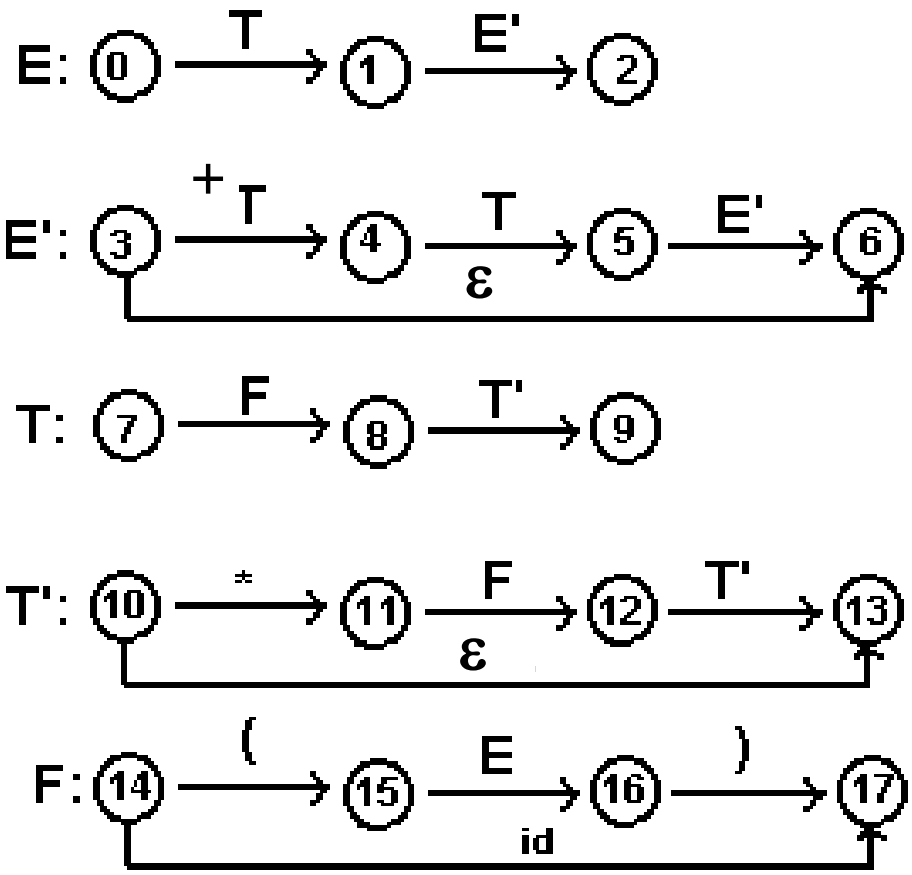
```
R(){
  if(token =“+”){
     reconhece ( “+” );
     T( );
     R( );
  } else if(token =“-”)
     reconhece ( “-”);
     T( );
     R( );
  } else break;
}
```

```
T ( ) {
    if (isdigit(token))
      reconhece (digit);
}
```

Analisadores sintáticos

Diagrama de transição para analisadores sintáticos preditivos

E →TE′
E′ →+TE′| ε
T →FT′
T′ →*FT′| ε
F →(E)|id



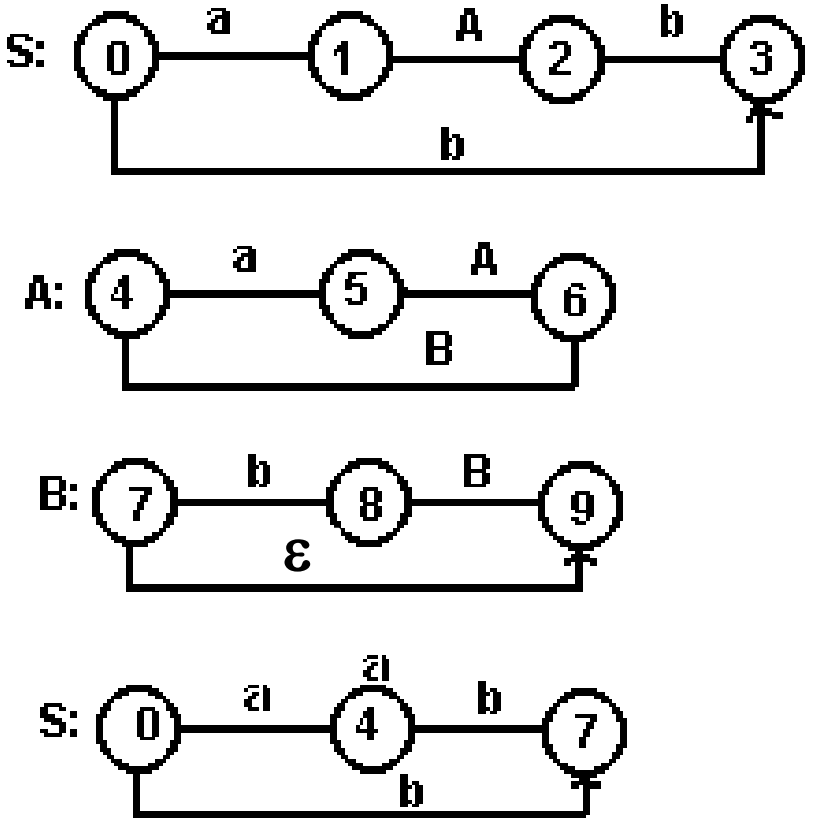
Analisadores sintáticos

Gramática

S →aAb|b

A →Aa|B

B →Bb|ε



Análise Sintática Top-down

A construção de um analisador sintático preditivo é auxiliado por duas funções associadas à gramática G. Essas funções, **PRIMEIRO** e **SEGUINTE** nos permitem preencher as entradas de uma tabela sintática preditiva para G.

Algoritmos para **first - PRIMEIRO**

1. Se A $\rightarrow$ bx; colocar b no first de A
2. Se A $\rightarrow$ ε; colocar ε no first de A
3. Se A $\rightarrow$ BCD....
   - Se $\varepsilon \in$ B; colocar first de C no first de A
   - Se $\varepsilon \notin$ B; colocar first de B no first de A

Análise Sintática Top-down

Algoritmos para **follow - SEGUINTE**

1. Ponha $ (fim de string) no follow de S (Símbolo inicial)
2. Se $A \rightarrow \alpha B \beta$ e a $\in$ ao first de β, colocar a no follow de B. (tudo no primeiro de β exceto ε, é seguinte de B)
3. Se $A \rightarrow \alpha B \beta$ e $\beta =^* \varepsilon$ (β produz ε em um ou mais passos). colocar follow de A no follow de B.

Análise Sintática Top-down

Gramática G_1

- $S \rightarrow aB|d$
- $B \rightarrow cDb|\varepsilon$
- $D \rightarrow a|\varepsilon$

	first	follow
S	a,d	$
B	c, ε	$
D	a, ε	b

Gramática G_2

- $S \rightarrow aSa|A$
- $A \rightarrow bB$
- $B \rightarrow cBc|c$

	first	follow
S	a,b	a,$
A	b	a,$
B	c	a,c,$

Análise Sintática Top-down

Gramática G_3

- Z → XYZ|d
- Y → c| ε
- X → Y|a

	first	follow
Z	d,a,c, ε	$
Y	c, ε	$,a,c,d
X	a,c, ε	$,a,c,d

Gramática G_4

- E → EST|T
- S → +|-
- T → TMF|F
- M → *
- F → (E)|n

Gramática G_4

- E → TX
- X → STX| ε
- S → +|-
- T → FY
- Y → MFY| ε
- M → *
- F → (E)|n

	first	follow
E	n,(	$,)
X	+,-, ε	$,)
S	+,-	n,(
T	n,(	+,-,$,)
Y	*, ε	+,-,$,)
M	*	n,(
F	n,(	*,+,-,$,)

Análise Sintática Top-down

Gramática G_5

- S →(A)|b
- A → B:A|B
- B → a| ε

	first	follow
S	b,(	$
A	a, ε ,:	)
B	a, ε	:,)

Gramática G_6

- S →(A)|b
- A → A;B|B
- B → a| ε
- -- fatoração ---
- S → (A)|b
- A → BX
- X → ;BX| ε
- B → a| ε

	first	follow
S	b,(	$
A	a,;, ε	)
X	;, ε	)
B	a, ε	;,)

Análise Sintática Top-down

Gramática G_7

E → E+T|E-T|T

E → T*F|T/F|F

F → (E)|a

---fatoração ---

E → TE′

E′ →+TE′|-TE′| ε

T →FT′

T′ →*FT′|/FT′| ε

F →(E)|a

	first	follow
E	(,a	$,)
E′	+,-, ε	$,)
T	(,a	+,-,),$
T′	*,/, ε	+,-,),$
F	(,a	*,/,+,-,),$

Análise Sintática Top-down

Gramática G_8

D → var E

E → F;E| ε

F → V:T

V → G;V|G

T → int

G → a

--- fatoração ---

D →var E

E →F;E| ε

F →V:T

V →GX

X →;V| ε

T →int

G →a

	first	follow
D	var	$
E	a, ε	$
F	a	;
V	a	:
X	;, ε	:
T	int	;
G	a	;,:

Análise Sintática Top-down

Gramática G_9

```
S →iEtS|iEtSeS|a
-----------------
S →iEtSX'|a
X' →eS| ε
E →b
```

	first	follow
A	i,a	$,e
X'	e, ε	$,e
E	b	t

Gramática G_{10}

```
S →(L)|a
L →L:S|S
--------------------
S →(L)|a
L →SL'
L' →:SL'|ε
```

	first	follow
S	(,a	$, :,)
L	(,a	)
L'	:, ε	)