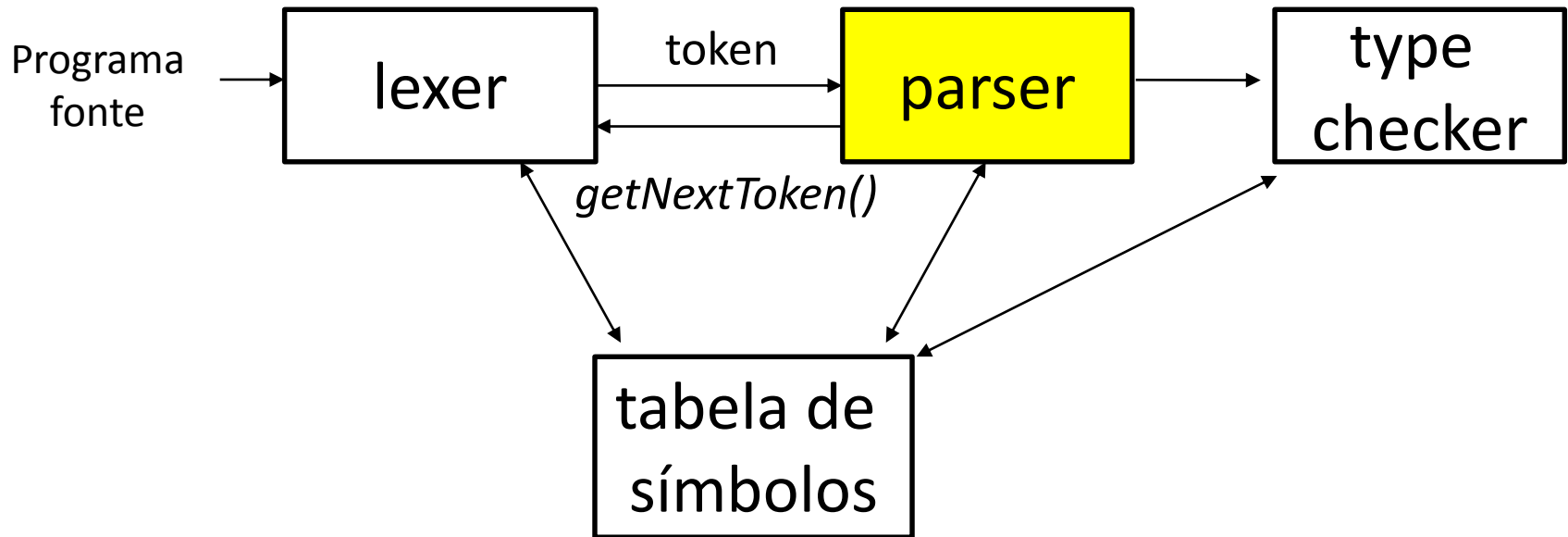


Análise Sintática

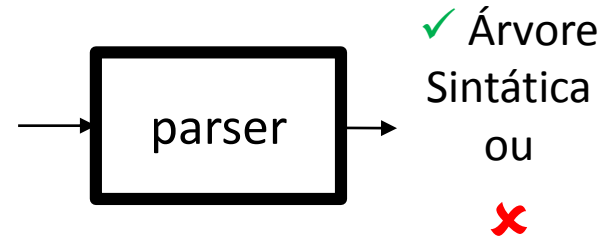
Análise Sintática

- Sintaxe (de uma linguagem)
 - Definição das strings estruturalmente válidas de uma linguagem
- Analisador sintático
 - Checa se a string respeita sintaxe

Analizador Sintático



Parsing



- Caso sucesso:
 - A sintaxe do programa está correta
 - A string de entrada é “bem formada”
- Caso contrário:
 - erro de sintaxe; alguma regra sintática foi violada
- Importante:
 - O programa pode ainda conter erros capturados ou não pelo type checker

Especificação da Sintaxe

- Historicamente, gramáticas livres de contexto são um formalismo adequado de especificação
 - Suficientemente expressivo
 - Fácil de especificar, de manter, e entender

Exercício. Qual das strings a seguir fazem parte de $L(G)$?

G:
$$\begin{array}{l} S \rightarrow aXa \\ X \rightarrow bY \mid \varepsilon \\ Y \rightarrow cXc \mid \varepsilon \end{array}$$

- abcba
- acca
- aba
- abcbcbaba

Derivação

- Derivação: Dada uma gramática G , produz uma string s que faz parte de $L(G)$

Gramática G

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

Derivação de string em G

$S \rightarrow a\underline{A}Be \rightarrow a\underline{A}bcBe \rightarrow$
 $a\underline{A}bcbcBe \rightarrow abbcbc\underline{B}e \rightarrow$
 $abbcbcde$

Parsing

- Parsing: Dada uma string s em $L(G)$, produz uma árvore sintática que demonstra como se obter derivação de s

Gramática G

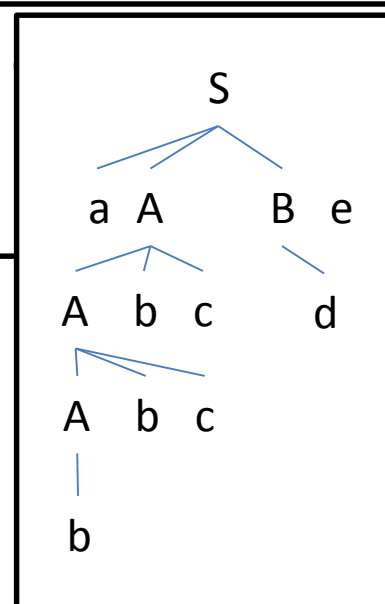
$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

Derivação de string em G

$S \rightarrow a\underline{A}Be \rightarrow$
 $a\underline{A}bcbcbBe \rightarrow$
 $abbcbbcde$



Exercício. Qual das seguintes derivações são válidas em G ?

• S

aXa

aa

• S

aXa

$abYa$

$acXca$

$acca$

• S

aXa

$abYa$

$abcXcda$

$abccda$

• S

aXa

$abYa$

$abcXca$

$abcbYca$

$abcbdca$

$S \rightarrow aXa$

$X \rightarrow bY \mid \varepsilon$

$Y \rightarrow cXc \mid d \mid \varepsilon$

Top-down e Bottom-up parsing

Considerando a ordem de criação da árvore sintática, **top-down** constrói o nó raiz primeiro e depois os internos em direção aos nós folha. **bottom-up** faz o contrário.

TOP-DOWN PARSING

Top-down parser

- Procura sequência de derivações mais a esquerda para se obter uma string de entrada
- O parse da string **abbcbcde** é caracterizado pela sequência de derivações abaixo.

$$S \rightarrow aABe$$
$$A \rightarrow Abc \mid b$$
$$B \rightarrow d$$
$$\begin{aligned} S &\rightarrow a\underline{A}Be \rightarrow a\underline{A}bcBe \rightarrow \\ &a\underline{A}bcbcbBe \rightarrow abbcbbc\underline{B}e \rightarrow \\ &abbcbbcde \end{aligned}$$

Top-down parser

- Procura sequência de derivações mais a esquerda para se obter uma string de entrada
- O parse da string **abbcbcd** é caracterizado pela sequência de derivações abaixo.

$S \rightarrow aABe$
 $A \rightarrow Abc \mid b$
 $B \rightarrow d$

$S \rightarrow a\underline{A}Be \rightarrow a\underline{A}bcBe \rightarrow$
 $a\underline{A}bcbBe \rightarrow abbc\underline{b}cBe \rightarrow$
 $abbc\underline{b}cd$

Note a preferência da
produção mais à esquerda

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

a b b c b c d e S

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow \mathbf{aABe}$

$A \rightarrow Abc \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

a b b c b c d e

aABe

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$
 $A \rightarrow Abc \mid b$
 $B \rightarrow d$

a b b c b c d e

ABe

escolha!

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid \mathbf{b}$

$B \rightarrow d$

a b b c b c d e

bBe

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid \mathbf{b}$

$B \rightarrow d$

a **b** b c b c d e Be



erro. Backtrack!

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$
 $A \rightarrow Abc \mid b$
 $B \rightarrow d$

a b b c b c d e

ABe

Faz outra
escolha!

Restaura estado
anterior a última
escolha

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow \textcolor{red}{Abc} \mid b$

$B \rightarrow d$

a

b

b

c

b

c

d

e

AbcBe

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$
 $A \rightarrow Abc \mid b$
 $B \rightarrow d$

a b b c b c d e

AbcBe

escolha
novamente!

Símbolo A
novamente
na posição
mais à
esquerda

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$
 $A \rightarrow Abc \mid \mathbf{b}$
 $B \rightarrow d$

a b b c b c d e

bbcBe

Outra
escolha!

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid \mathbf{b}$

$B \rightarrow d$

a b b c b c d e

bcBe

O método


- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid \mathbf{b}$

$B \rightarrow d$

a **b** **b** **c** b c d e Be

 erro. Backtrack!

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

a

b

b

c

b

c

d

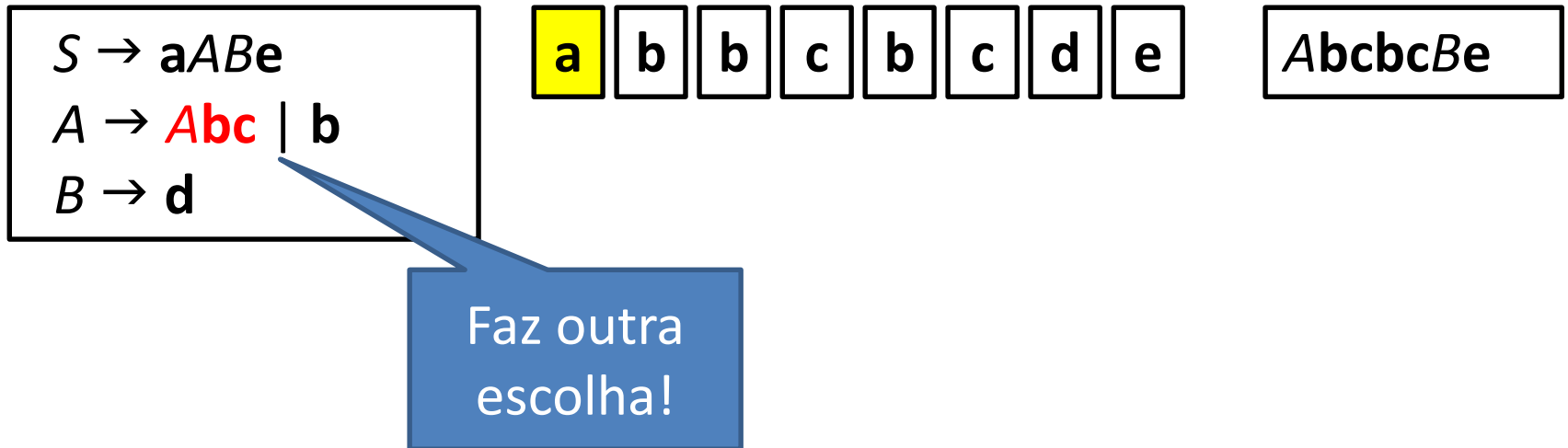
e

AbcBe

Restaura estado
anterior a última
escolha

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar



O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid \mathbf{b}$

$B \rightarrow d$

a b b c b c d e

bbcbcbBe

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

a **b** **b** **c** **b** **c** d e Be

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

a b b c b c d e de

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

a b b c b c d e

O método

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

a b b c b c d e

derivação
correspondente!

$S \rightarrow a\underline{A}Be \rightarrow a\underline{A}bcBe \rightarrow$
 $a\underline{A}bcbcbBe \rightarrow abbcbbc\underline{B}e \rightarrow$
 $abbcbbcde$

Exercício. Construa a árvore sintática associada a derivação abaixo

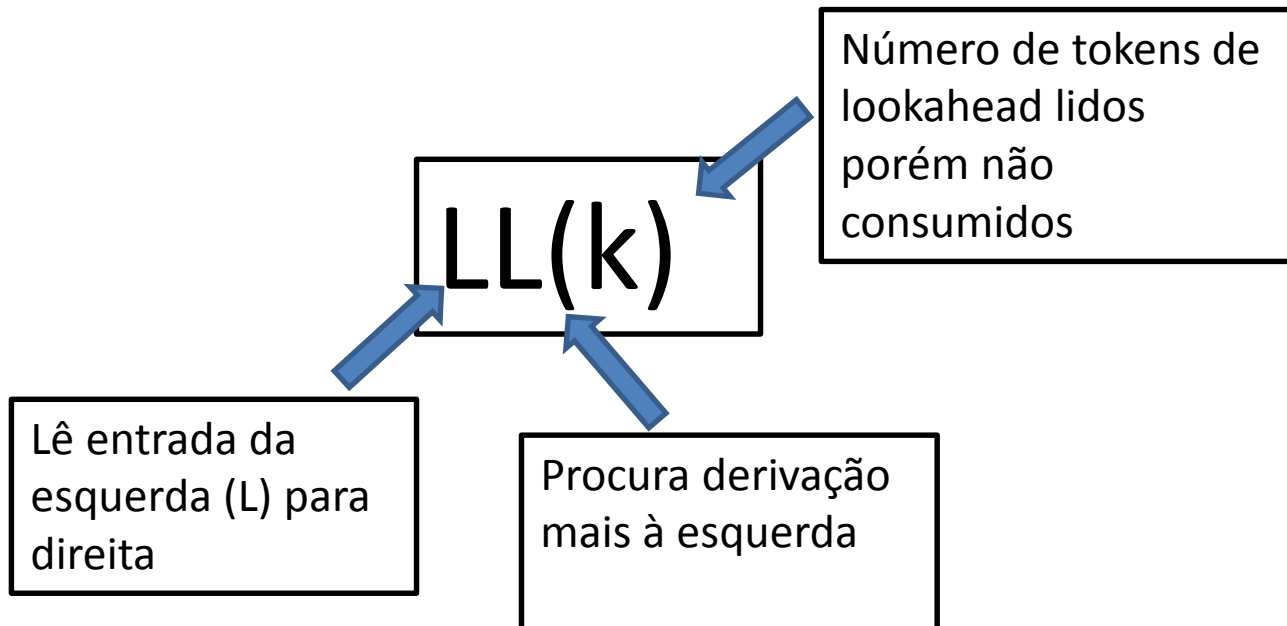
$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

$S \rightarrow a\underline{A}Be \rightarrow a\underline{A}bcBe \rightarrow$
 $a\underline{A}bcbcbBe \rightarrow abbcbbc\underline{B}e \rightarrow$
 $abbcbbcde$

Classificação de parsers



Ambiguidade e Recursão à esquerda

- Ambiguidade: Existe mais de uma forma de se derivar a string em uma gramática ambígua
 - Mais de uma parse tree
- Recursão à esquerda: dificuldade em saber quando parar de aplicar uma produção

$S \rightarrow aABe$

$A \rightarrow A**b**c \mid b$

$B \rightarrow d$

$a\underline{A}Be \rightarrow a\underline{A}bcBe \rightarrow a\underline{A}bcbcbBe$

$\rightarrow a\underline{A}bcbcbcbBe \rightarrow$

$a\underline{A}bcbcbcbcbcbBe \rightarrow \dots$

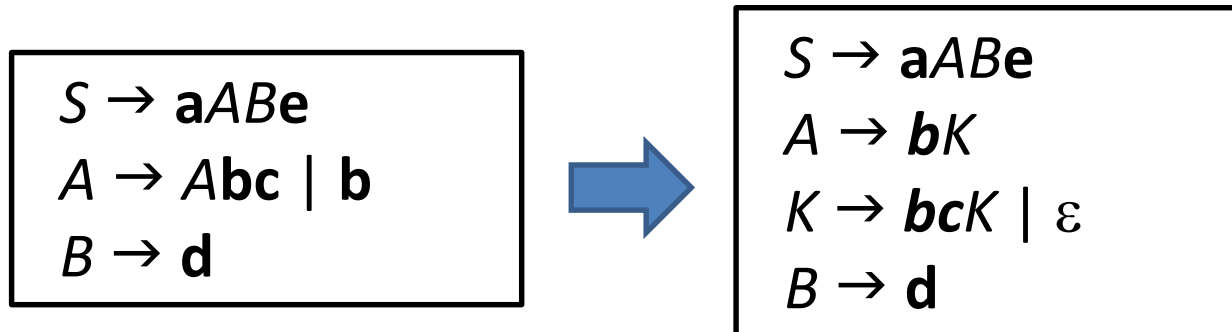
complicador

Exercício: Quais das seguintes gramáticas são ambíguas?

- $S \rightarrow SS \mid a \mid b$
- $E \rightarrow E + E \mid id$
- $S \rightarrow Sa \mid Sb \mid a$
- $E \rightarrow E' \mid E' + E$
 $E' \rightarrow -E' \mid id \mid (E)$

Predictive parsing

- É um parser **top-down**
- É um parser que não requer backtracking
 - Simples de construir manualmente
 - Mas, requer modificação na gramática para tratar recursão à esquerda e ambigüidade
- Eliminação de recursão à esquerda (fatoração):



Exercício: Escolha a gramática que elimina recursão à esquerda corretamente da gramática abaixo

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow id \mid (E) \end{aligned}$$

- $$\begin{aligned} E &\rightarrow id + E \mid E + T \mid T \\ T &\rightarrow id \mid (E) \end{aligned}$$

- $$\begin{aligned} E &\rightarrow E' + T \mid T \\ E' &\rightarrow id \mid (E) \\ T &\rightarrow id \mid (E) \end{aligned}$$

- $$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow id \mid (E) \end{aligned}$$

- $$\begin{aligned} E &\rightarrow E + id \mid E + (E) \\ &\quad \mid id \mid (E) \end{aligned}$$

BOTTOM-UP PARSING

Bottom-up parser

- Procura sequência de derivações **mais a direita** para se obter uma string de entrada
- O parse da string **abbcbcdc** é caracterizado pela sequência de derivações abaixo.

$$\begin{array}{l} S \rightarrow aABe \\ A \rightarrow Abc \mid b \\ B \rightarrow d \end{array}$$
$$S \rightarrow aA\underline{B}e \rightarrow a\underline{A}de \rightarrow a\underline{A}bcde \rightarrow a\underline{A}bcbcdc \rightarrow abbcbbcdc$$

O método

- Encontre padrões que casam com lado direito da produção e substitua pelo lado esquerdo

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

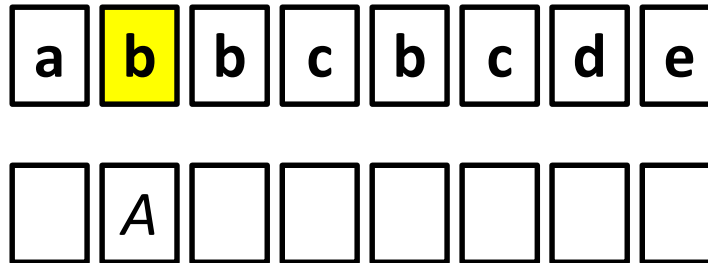
$B \rightarrow d$

a b b c b c d e

O método

- Encontre padrões que casam com lado direito da produção e substitua pelo lado esquerdo

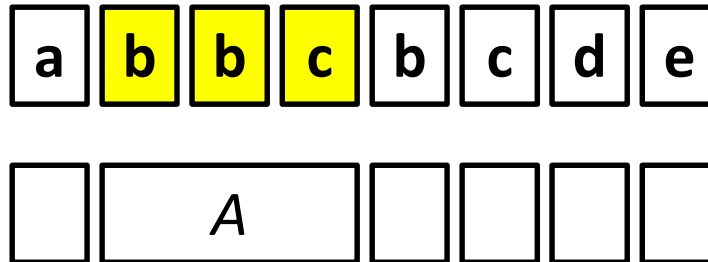
$S \rightarrow aABe$
 $A \rightarrow Abc \mid \mathbf{b}$
 $B \rightarrow d$



O método

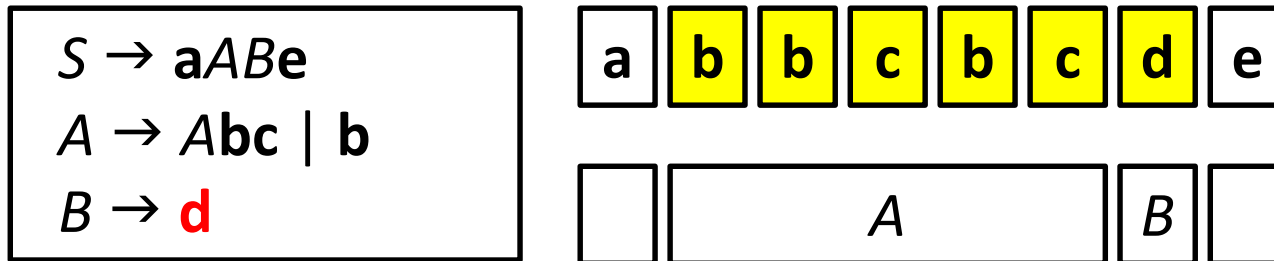
- Encontre padrões que casam com lado direito da produção e substitua pelo lado esquerdo

$S \rightarrow aABe$
 $A \rightarrow \textcolor{red}{Abc} \mid b$
 $B \rightarrow d$



O método

- Encontre padrões que casam com lado direito da produção e substitua pelo lado esquerdo



O método

- Encontre padrões que casam com lado direito da produção e substitua pelo lado esquerdo

$S \rightarrow \mathbf{aABe}$
 $A \rightarrow Abc \mid \mathbf{b}$
 $B \rightarrow \mathbf{d}$

a b b c b c d e

S

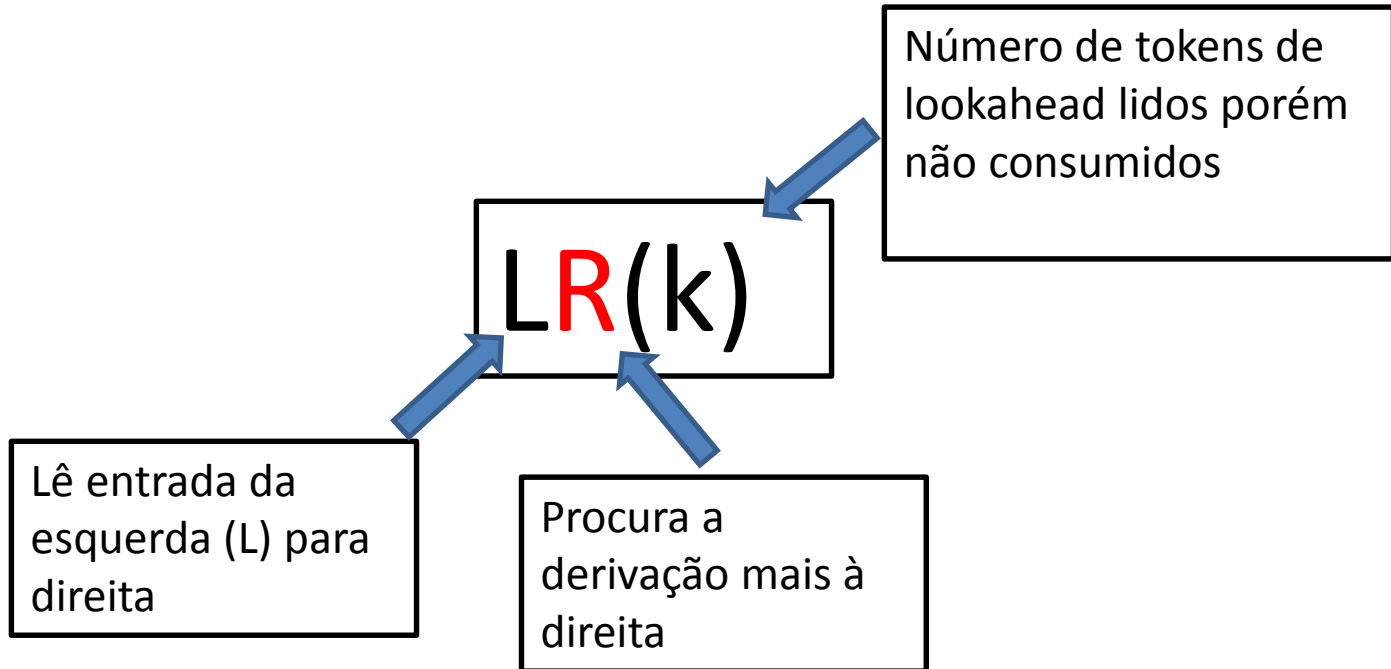
derivação
correspondente!

$S \rightarrow aABe \rightarrow aAde \rightarrow aAbcde \rightarrow$
 $aAbcbcbde \rightarrow abbcbbcbde$

Shift-reduce parsers

- Usa uma pilha e uma tabela
 - Pilha: armazena tokens para salvar contexto
 - Tabela: determina as opções atuais de ação
- Possíveis ações
 - Shift (push): coloca tokens na pilha
 - Reduce: determina uma produção
 - Accept: finaliza. reconhece string.
 - Error: Nenhuma ação é possível

Terminologia: classificação de parsers



Top-down e Bottom-up

- Em geral, bottom-up é mais flexível
 - Coloca menos restrições na gramática
- Bem mais trabalhoso de se escrever e manter manualmente. Porém...
 - Yacc e Bison geram parser de gramática LALR(1), um subconjunto LR(1)

TRADUÇÃO DIRIGIDA POR SINTAXE

Tradução dirigida por sintaxe

- O que faz?
 - Forma simples de definição semântica
- Como funciona?
 - Associa ações a produções de uma gramática
- Para que serve?
 - **Construir árvore sintática**
 - Checagem de tipos, etc.

Gramática de Atributos

- BNF com ações
 - Conceitualmente, ações criam e associam **atributos** aos nós da árvore sintática

O processo de avaliação dos atributos é chamado “anotação” ou “decoreação” da *parse tree*.

Exemplo

Produção

$L \rightarrow E \text{ '\n'}$

$E \rightarrow E_1 + T$

$E \rightarrow T$

$T \rightarrow T_1 * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow \text{digit}$

Regra semântica

`print (E.val)`

`E.val = E1.val + T.val`

`E.val = T.val`

`T.val = T1.val * F.val`

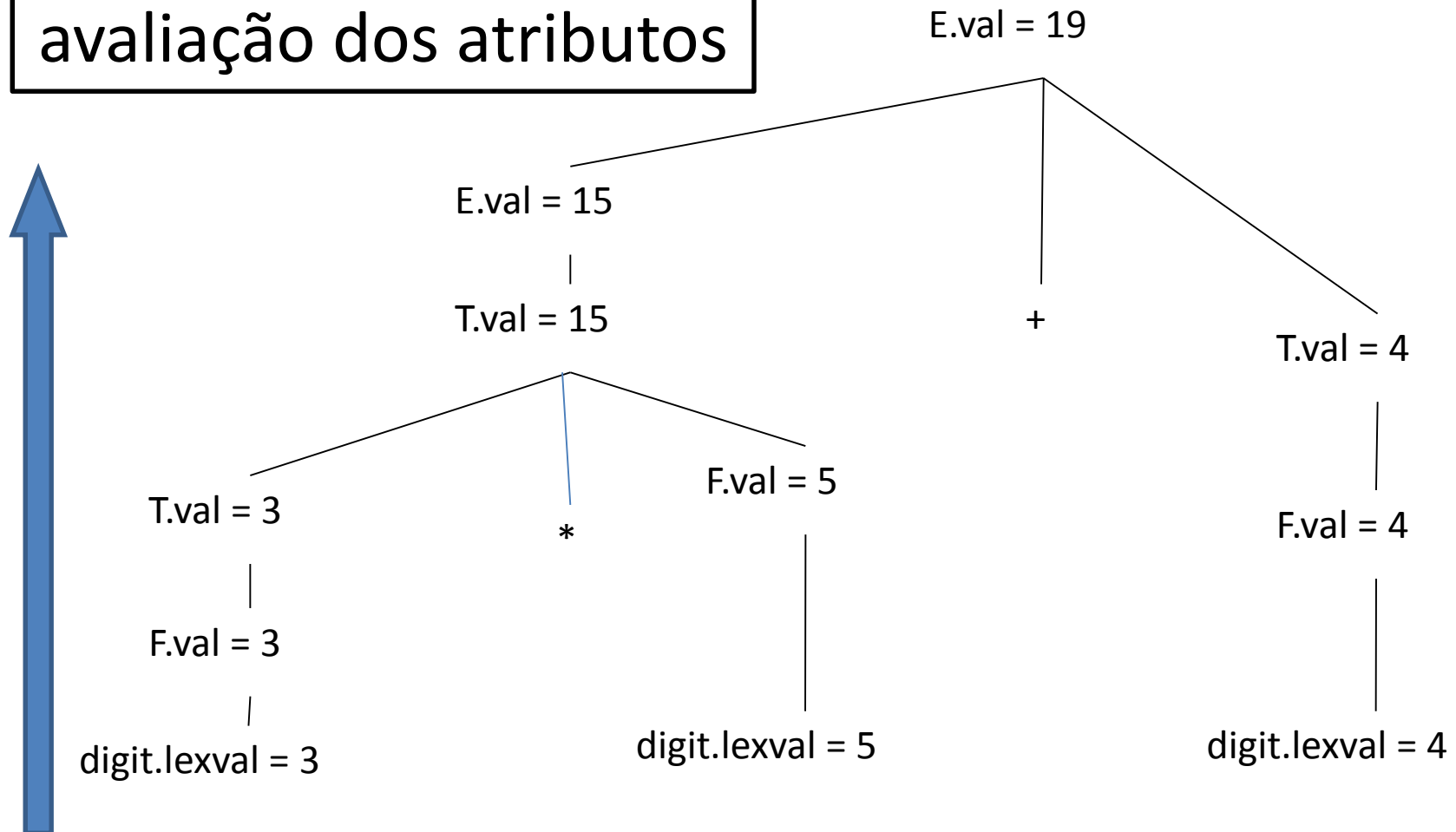
`T.val = F.val`

`F.val = E.val`

`F.val = digit.lexval`

Árvore de “3 * 5 + 4” decorada

Note a direção da
avaliação dos atributos



Exemplo em yacc

...

```
line    : expr '\n' { printf("%d\n", $1); }  
        ;
```

```
expr    : expr '+' term    { $$ = $1 + $3; }  
        | term  
        ;
```

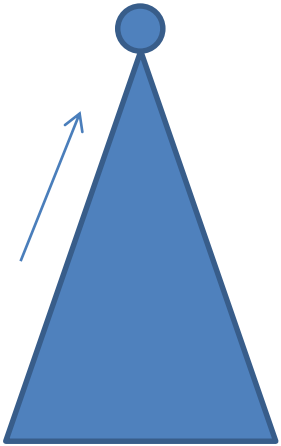
```
term    : term '*' factor { $$ = $1 * $3; }  
        | factor  
        ;
```

```
factor  : '(' expr ')'      { $$ = $2; }  
        | DIGIT  
        ;
```

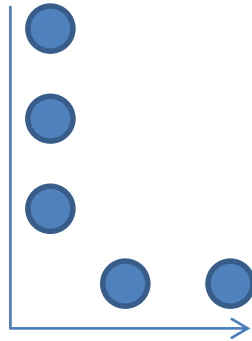
...

Dois tipos de atributo

- Sintetizados (dependem dos nós filho)
- Herdados (dependem de nós pai e irmão)



sintetizados



herdados

Direção da seta
indica direção em
que o valor do
atributo é calculado

Atributos sintetizados

- Implementação simples: anota-se parse tree com busca bottom-up (pós-ordem)

Muito usada na prática!

Uma “S-attributed grammar” usa **apenas** atributos sintetizados

Atributos herdados

- Úteis para especificar contexto
 - Por exemplo se um identificador usado em uma expressão é definido no contexto de uso
- É sempre possível trabalhar apenas com atributos sintetizados
 - Porém, definições tornam-se mais elaboradas
- Visto em análise semântica

Exercício

- Defina regras semânticas para construção das árvores sintáticas da gramática abaixo

$expr \rightarrow expr + factor \mid expr - factor \mid factor$

$factor \rightarrow digit \mid (expr)$

$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Resposta

Considere que há um atributo chamado `res` (para resultado).

```
expr :: expr + factor
        { expr.res = new Add(expr.res, factor.res) }
| expr - factor
        { expr.res = new Sub(expr.res, factor.res) }
| factor
        { expr.res = factor.res }
```

```
factor :: digit      { factor.res = digit.res }
        | ( expr ) { factor.res = expr.res }
```

```
digit :: 0 { digit.res = new Num(0) }
        | 1
        ...
```

Resposta

Defina os tipos de dados

```
public interface Expression {}

public class Add implements Expression {
    Add(Expression e1, Expression e2) { ... } ...
}

public class Sub implements Expression {
    Sub(Expression e1, Expression e2) { ... } ...
}

public class Num implements Expression {
    Num(int n) {...}
}
```

Resposta

Ex: $5 + (3 - 2)$

```
new Add(new Digit(5), new Sub(new Digit(3), new Digit(2)))
```