

Análise sintática

Análise sintática *top-down*

Prof. Edson Alves

Faculdade UnB Gama

Análise sintática recursiva-descendente

- ▶ A análise sintática *top-down* pode ser interpretada como uma tentativa de se encontrar uma derivação mais à esquerda para uma cadeia da entrada
- ▶ Outra forma de interpretar esta análise é como uma tentativa de construção de uma árvore gramatical a partir da raiz, criando os nós da árvore em pré-ordem
- ▶ No caso geral, a análise sintática recursiva-descendente pode envolver algum nível de retrocesso
- ▶ Um caso especial da análise sintática recursiva-descente é o dos analisadores preditivos, que não demandam retrocesso

Retrocesso

- ▶ Certas gramáticas impõem ao analisador recursivo-descendente a necessidade de um retrocesso na análise
- ▶ Por exemplo, considere a entrada $w = cad$ e a gramática

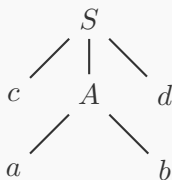
$$\begin{aligned} S &\rightarrow cAd \\ A &\rightarrow ab \mid a \end{aligned}$$

- ▶ A análise inicia construindo a raiz da árvore sintática, rotulada como S
- ▶ Ao ler o primeiro símbolo da entrada, o caractere c , o analisador expande a produção de S para gerar os filhos e obter



Retrocesso

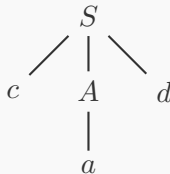
- ▶ O filho mais à esquerda reconhece o caractere c , de modo que o apontado avança para o segundo símbolo da entrada, o caractere a
- ▶ A análise usa a primeira alternativa das produções- A para obter



- ▶ A folha à esquerda reconhece o caractere a , mas a folha à direita, rotulada b , não reconhece o caractere d
- ▶ Aqui acontece o retrocesso: os filhos são descartados e, ao retornar ao nó A , é usada a segunda alternativa das produções- A

Retrocesso

- ▶ Esta nova substituição resulta em



- ▶ Nesta nova árvore o caractere a é reconhecido, e a folha mais à esquerda reconhece o caractere d , de modo que esta árvore produz a cadeia w , finalizando a análise sintática
- ▶ Gramáticas recursivas à esquerda podem levar a laços infinitos, mesmo com o retrocesso
- ▶ Isto pode acontecer com múltiplas expansões de um não-terminal que não consome a entrada

Analísadores sintáticos preditivos

- ▶ Uma escrita cuidadosa da gramática, por meio de eliminação de recursão à esquerda e do uso de fatorações à esquerda, pode dispensar complementamente o retrocesso
- ▶ Neste caso, a análise sintática recursiva-descendente se torna uma análise gramatical preditiva
- ▶ Na análise gramatical preditiva a alternativa a ser escolhida deve ser detectável examinando-se apenas o primeiro símbolo da cadeia que a mesma deriva
- ▶ Por exemplo, as produções abaixo permitem uma análise preditiva:

```
cmd  →  if expr then cmd else cmd  
      |  while expr do cmd  
      |  begin lista_de_commandos end
```

Diagramas de transição para analisadores sintáticos preditivos

- ▶ Assim como foi feito na análise léxica, é possível criar diagramas de transição para analisadores sintáticos
- ▶ Cada não-terminal da gramática deve ter um diagrama próprio
- ▶ Os rótulos das arestas são tokens e não-terminais
- ▶ Uma transição rotulada por um token deve ser seguida se o token do rótulo for o próximo token da entrada
- ▶ Uma transição rotulada por um não-terminal A implica em uma passagem pelo diagrama de A

Geração de diagramas de transição para analisadores sintáticos preditivos

Input: uma gramática G

Output: os diagramas de transição para todos os não-terminais de G

- 1: Elimine qualquer recursão à esquerda de G , se necessário
- 2: Fatore G à esquerda, se necessário
- 3: **for** cada não-terminal A de G **do**
- 4: crie um estado inicial e um estado final (de retorno)
- 5: **for** cada produção $A \rightarrow X_1 X_2 \dots X_n$ **do**
- 6: crie um percurso que parte do estado inicial até o estado final com arestas rotuladas por X_1, X_2, \dots, X_n

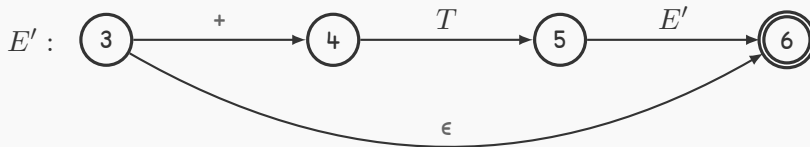
Comportamento do analisador sintático preditivo

- ▶ O analisador parte do estado inicial do símbolo de partida
- ▶ Estando o analisador no estado s , e se este possui uma aresta rotulada com o token a apontando para o estado t , e o próximo token da entrada é a , o analisador reconhece a e se move para o estado t
- ▶ Se a aresta é rotulada pelo não-terminal A , o analisador segue para o estado inicial de A , sem reconhecer o próximo token da entrada
- ▶ Se, em algum momento, o analisador atinge o estado final de A , ele deve seguir para t , tendo “lido A ” (isto é, reconhecendo quaisquer tokens que surgiram no caminho do estado inicial ao final de A)
- ▶ Se a aresta é rotulada por ϵ , o analisador segue para t sem reconhecer a entrada

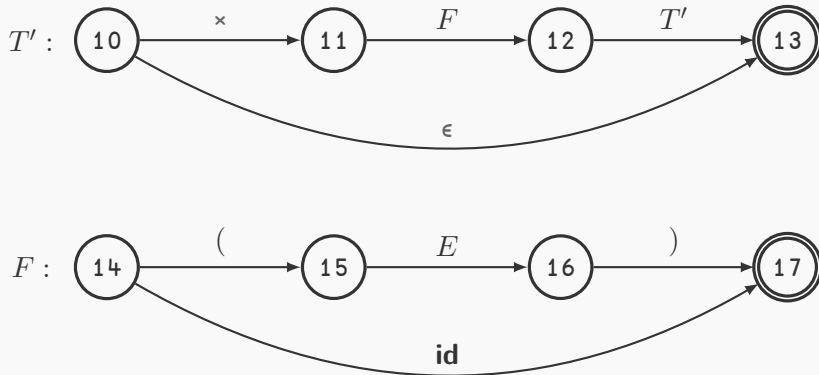
Exemplo de gramática G sem recursão à esquerda e fatorada à esquerda

$$\begin{aligned}E &\rightarrow TE' \\E' &\rightarrow +TE' \mid \epsilon \\T &\rightarrow FT' \\T' &\rightarrow \times FT' \mid \epsilon \\F &\rightarrow (E) \mid \mathbf{id}\end{aligned}$$

Diagramas de transição para a análise sintática preditiva da gramática G



Diagramas de transição para a análise sintática preditiva da gramática G

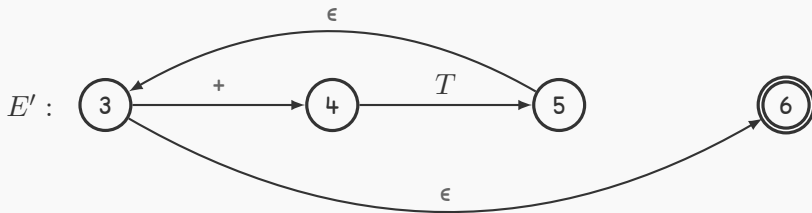


Limitações da análise gramatical preditiva

- ▶ A análise gramatical preditiva só pode ser realizada se os diagramas dos não-terminais forem determinísticos, isto é, caso não exista mais de uma transição de um mesmo estado para outros com o mesmo rótulo
- ▶ Caso exista ambiguidades, estas devem ser resolvidas de forma *ad hoc*
- ▶ Se não for possível eliminar as ambiguidades, não será possível realizar uma análise sintática preditiva
- ▶ Neste caso, será preciso conduzir uma análise sintática recursiva-descendente com retrocesso, avaliando cada um dos caminhos possíveis

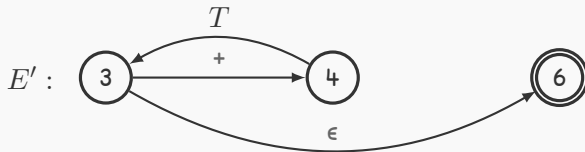
Simplificação dos diagramas de transição

- ▶ Os diagramas de transição podem ser simplificados por meio da aplicação da substituição de uns nos outros e da observação das produções
- ▶ Por exemplo, o diagrama de E' pode ser modificado por meio de uma aresta retornando diretamente para E' , sem recursão:

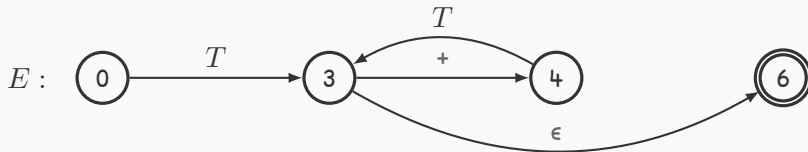


Simplificação dos diagramas de transição

- ▶ A transição- ϵ de 5 para 3 pode ser eliminada, de modo que o estado 5 pode ser removido:

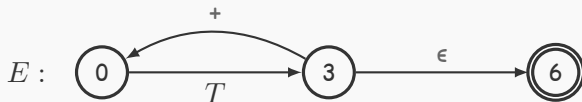


- ▶ Estas modificações podem ser inseridas no diagrama de E , resultando em



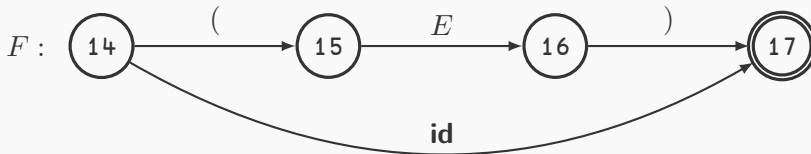
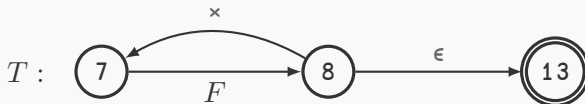
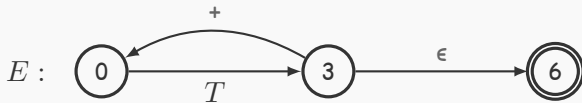
Simplificação dos diagramas de transição

- ▶ Por fim, o estado 4 também pode ser eliminado:



- ▶ Aplicando técnicas semelhantes os diagramas apresentados anteriormente podem ser simplificados, reduzindo substancialmente o número de estados e consequentemente a memória usada pelo analisador sintático
- ▶ A simplificação também pode eliminar recursões, reduzindo o tempo de execução do analisador sintático

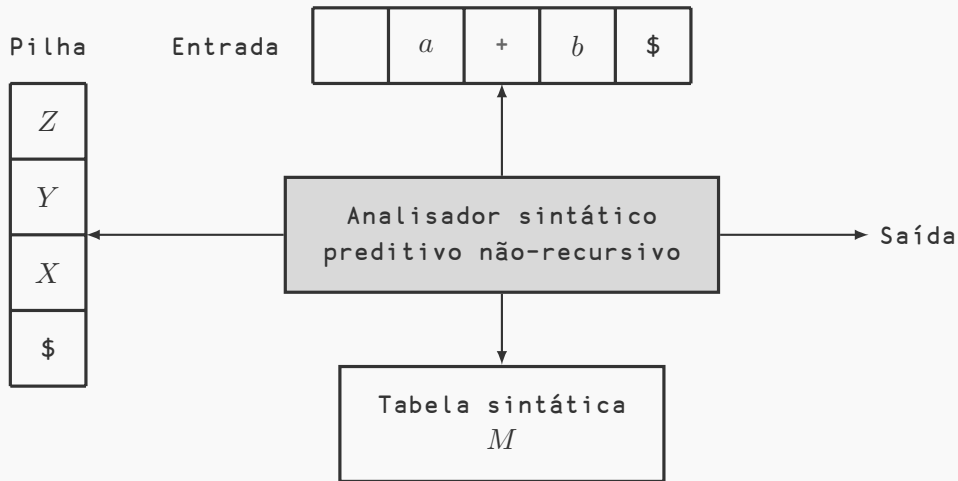
Diagramas simplificados para expressões aritméticas



Analizador preditivo não-recursivo

- ▶ É possível construir um analisador sintático preditivo não-recursivo, no qual as chamadas recursivas são eliminadas por meio do uso de uma pilha explícita
- ▶ Seja recursivo ou não, o principal problema a ser resolvido por um analisador sintático é o de identificar a produção que deve ser aplicada a um não-terminal
- ▶ Um analisador sintático não-recursivo busca em uma tabela sintática pela produção a ser aplicada
- ▶ Tal tabela pode ser construída diretamente a partir de certas gramáticas

Modelo de um analisador sintático preditivo não-recursivo



Estrutura de um analisador sintático preditivo não-recursivo

- ▶ Um analisador sintático preditivo não-recursivo é composto por um *buffer* de entrada, uma pilha, uma tabela sintática e um fluxo de saída
- ▶ O *buffer* de entrada contém a cadeia a ser analisada, seguida de um sentinela que indique o fim da cadeia (assuma que o sentinela é o caractere \$)
- ▶ A pilha contém símbolos gramaticais, onde o sentinela indica o fundo da pilha
- ▶ Inicialmente a pilha deve conter o símbolo de partida da gramática logo acima do sentinela
- ▶ A tabela sintática é uma matriz $M[A, a]$ cuja primeira dimensão contém não-terminais A e a segunda contém terminais a ou o sentinela \$

Algoritmo para o analisar sintático preditivo não-reativo

Input: Uma cadeia w e uma tabela sintática M para uma gramática G

Output: Se $w \in L(G)$, uma derivação mais à esquerda de w , caso contrário sinaliza um erro

```
1:  $a \leftarrow$  primeiro símbolo de  $w$ 
2: repeat
3:    $X \leftarrow$  topo da pilha
4:   if  $X$  é um terminal then
5:     if  $X = a$  then
6:       remova  $X$  da pilha
7:        $a \leftarrow$  próximo símbolo de  $w$ 
8:     else
9:       sinalize um erro
```

Algoritmo para o analisar sintático preditivo não-recursivo

```
10:   else if  $X$  é um não-terminal then
11:       if  $M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$  then
12:           remova  $X$  da pilha
13:           empilhe  $Y_k, Y_{k-1}, \dots, Y_1$ , com  $Y_1$  no topo da pilha
14:           escreva a produção  $X \rightarrow Y_1 Y_2 \dots Y_k$  na saída
15:       else
16:           sinalize um erro
17: until  $X = \$$ 
```

Tabela sintática para a gramática de expressões aritméticas

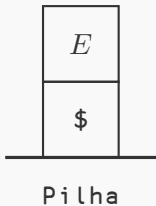
Não-terminal	Símbolo da entrada					
	id	+	×	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow \times FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow \text{id}$			$F \rightarrow (E)$		

Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”

Entrada

Saída

id + id × id \$



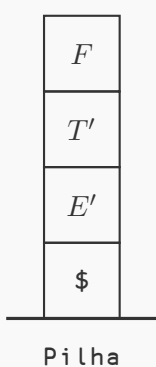
Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Entrada

id + id × id \$
↑

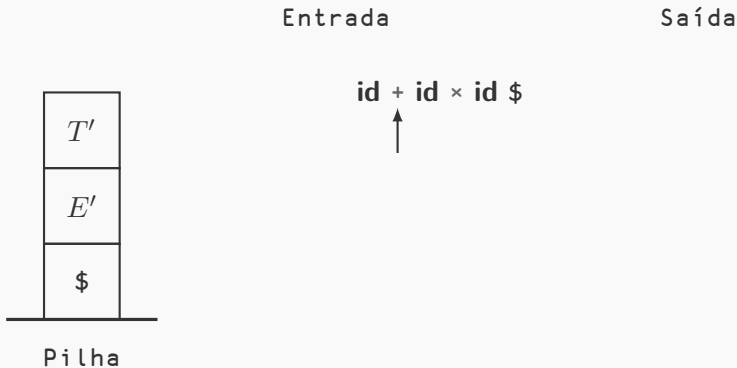
Saída

$T \rightarrow FT'$

Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



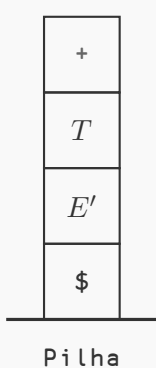
Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



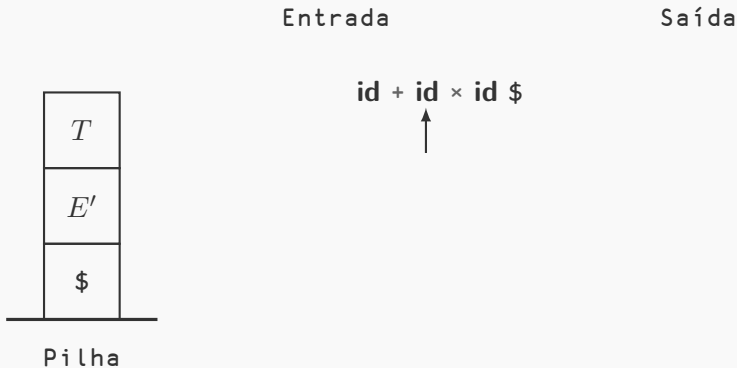
Entrada

id + id × id \$
↑

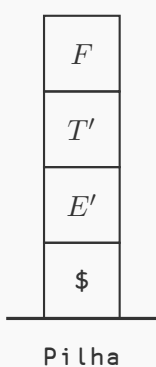
Saída

$E' \rightarrow +TE'$

Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Entrada

id + id × id \$
↑

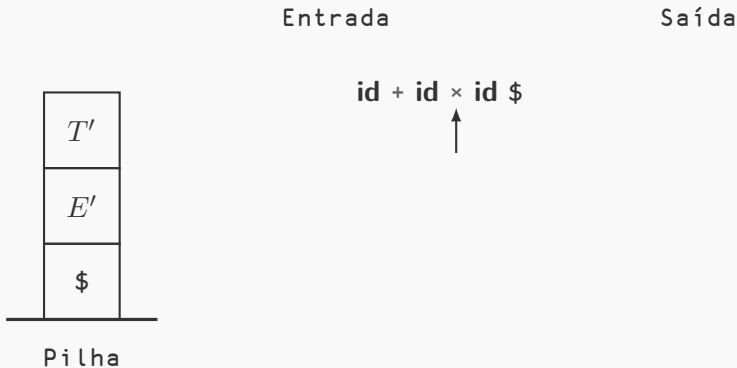
Saída

$T \rightarrow FT'$

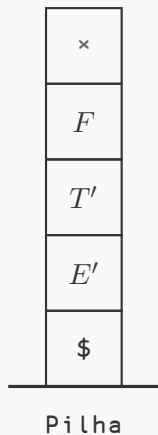
Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



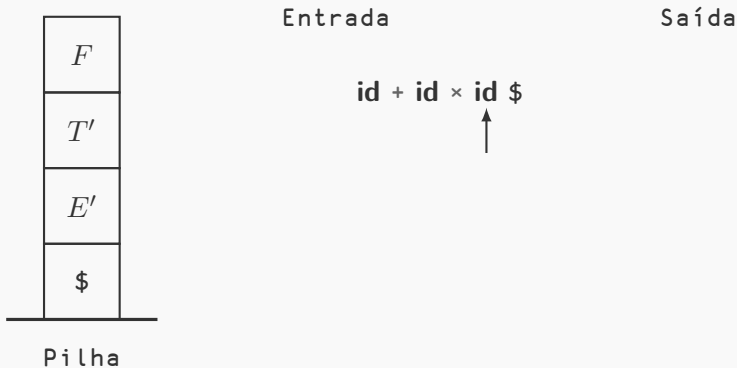
Entrada

id + id × id \$
 ↑

Saída

$T' \rightarrow \times FT'$

Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”



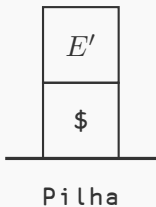
Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”

Entrada

Saída

id + id × id \$
↑

$T' \rightarrow \epsilon$



Comportamento do analisador preditivo não-recursivo para a cadeia “id+id×id”

Entrada

Saída

id + id × id \$
↑

$E' \rightarrow \epsilon$



Pilha

Funções auxiliares

- ▶ Duas funções associadas à gramática G auxiliam a construção de um analisador sintático preditivo não-recursivo
- ▶ Estas funções, $\text{PRIMEIRO}()$ e $\text{SEGUINTE}()$, apoiam o preenchimento da tabela sintática preditiva para G
- ▶ Os tokens produzidos pela função $\text{SEGUINTE}()$ também podem ser usados como tokens de sincronização na recuperação de erros na modalidade de desespero

Definição

Seja α uma cadeia de símbolos gramaticais. Então $\text{PRIMEIRO}(\alpha)$ é o conjunto de terminais que começam as cadeias derivadas a partir de α . Se $\alpha \xRightarrow{*} \epsilon$ então $\epsilon \in \text{PRIMEIRO}(\alpha)$.

Funções auxiliares

Definição

Seja A um não-terminal. Então $\text{SEGUINTE}(A)$ é o conjunto de terminais a que podem aparecer imediatamente à direita de A em alguma forma sentencial, isto é, o conjunto de terminais a tais que existe uma derivação $S \xRightarrow{*} \alpha A a \beta$ para algum α e β .

Se A puder ser o símbolo mais à direita em alguma forma sentencial, então $\$ \in \text{SEGUINTE}(A)$.

Algoritmo para o cálculo de PRIMEIRO()

Input: um símbolo gramatical X

Output: o conjunto $\text{PRIMEIRO}(X)$

- 1: **if** X é um terminal **then**
- 2: $\text{PRIMEIRO}(X) = \{ X \}$
- 3: **if** $X \rightarrow \epsilon$ **then**
- 4: adicione ϵ ao conjunto $\text{PRIMEIRO}(X)$
- 5: **if** X é um não terminal e $X \rightarrow Y_1 Y_2 \dots Y_k$ **then**
- 6: coloque a em $\text{PRIMEIRO}(X)$ se $a \in \text{PRIMEIRO}(Y_i)$ e $Y_1, Y_2, \dots, Y_{i-1} \xRightarrow{*} \epsilon$
- 7: se $\epsilon \in \text{PRIMEIRO}(Y_j)$ para todo $j = 1, 2, \dots, k$, coloque ϵ em $\text{PRIMEIRO}(X)$

Algoritmo para o cálculo de PRIMEIRO()

Input: uma cadeia $Y = X_1X_2 \dots X_n$

Output: o conjunto PRIMEIRO(Y)

- 1: **for** $i = 1, n$ **do**
- 2: adicione todos os símbolos diferente de ϵ de PRIMEIRO(X_i) em PRIMEIRO(Y)
- 3: **if** $\epsilon \notin \text{PRIMEIRO}(X_i)$ **then**
- 4: interrompa o laço
- 5: **if** $\epsilon \in \text{PRIMEIRO}(X_j)$ para todo $j = 1, 2, \dots, n$ **then**
- 6: adicione ϵ ao conjunto PRIMEIRO(Y)

Algoritmo para o cálculo de $\text{SEGUINTE}()$

Input: uma gramática G

Output: os conjuntos $\text{SEGUINTE}(A)$ para todos não-terminais A em G

- 1: Coloque $\$$ no conjunto $\text{SEGUINTE}(S)$, onde S é o símbolo de partida
- 2: **while** há algo a ser adicionado a qualquer conjunto $\text{SEGUINTE}()$ **do**
- 3: **if** existe uma produção $A \rightarrow \alpha B \beta$ **then**
- 4: adicione todos os elementos diferentes de ϵ do $\text{PRIMEIRO}(\beta)$ em $\text{SEGUINTE}(B)$
- 5: **if** existe uma produção $A \rightarrow \alpha B$ ou uma produção $A \rightarrow \alpha B \beta$ onde $\epsilon \in \text{PRIMEIRO}(\beta)$ **then**
- 6: adicione todos os elementos de $\text{SEGUINTE}(A)$ em $\text{SEGUINTE}(B)$

Gramática para expressões aritméticas e conjuntos auxiliares

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$E' \rightarrow \times FT' \mid \epsilon$$

$$F \rightarrow (E) \mid \text{id}$$

$$\text{PRIMEIRO}(E) = \text{PRIMEIRO}(T) = \text{PRIMEIRO}(F) = \{ (, \text{id} \}$$

$$\text{PRIMEIRO}(E') = \{ +, \epsilon \}$$

$$\text{PRIMEIRO}(T') = \{ \times, \epsilon \}$$

$$\text{SEGUINTE}(E) = \text{SEGUINTE}(E') = \{), \$ \}$$

$$\text{SEGUINTE}(T) = \text{SEGUINTE}(T') = \{ +,), \$ \}$$

$$\text{SEGUINTE}(F) = \{ +, \times,), \$ \}$$

Construção de tabelas sintáticas preditivas

Input: uma gramática G

Output: uma tabela sintática preditiva M

- 1: **for** cada produção $A \rightarrow \alpha$ de G **do**
- 2: **for** cada terminal $a \in \text{PRIMEIRO}(\alpha)$ **do**
- 3: adicione $A \rightarrow \alpha$ na posição $M[A, a]$
- 4: **if** $\epsilon \in \text{PRIMEIRO}(\alpha)$ **then**
- 5: **for** cada terminal $b \in \text{SEGUINTE}(A)$ **do**
- 6: adicione $A \rightarrow \alpha$ na posição $M[A, b]$
- 7: **if** $\$ \in \text{SEGUINTE}(A)$ **then**
- 8: adicione $A \rightarrow \alpha$ na posição $M[A, \$]$

Ambiguidades na tabela sintática preditiva

- ▶ Embora o algoritmo de produção de tabelas sintáticas preditivas possa ser aplicado em qualquer gramática G , algumas gramáticas produzirão múltiplas entradas para determinados pares (A, a)
- ▶ Considere a gramática que abstrai o comando **if-then-else**:

$$S \rightarrow iEtSS' \mid a$$

$$S' \rightarrow eS \mid \epsilon$$

$$E \rightarrow b$$

- ▶ Esta gramática possui duas entradas para $M[S', e]$: a saber $S' \rightarrow eS$ e $S' \rightarrow \epsilon$, uma vez que $\text{PRIMEIRO}(S') = \{e, \$\}$
- ▶ Neste caso, a ambiguidade pode ser resolvida optando pela primeira produção e descartando a segunda ($S' \rightarrow \epsilon$)

Tabela sintática para a gramática que abstrai o comando if-then-else

Não-terminal	Símbolo da entrada					
	a	b	e	i	t	$\$$
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$S' \rightarrow \epsilon$ $S' \rightarrow eS$			$S' \rightarrow \epsilon$
E		$E \rightarrow b$				

Gramáticas $LL(1)$

Definição

Uma gramática cuja tabela sintática não possui entradas multiplamente definidas são denominadas gramáticas $LL(1)$. O primeiro L da sigla indica a varredura da esquerda para a direita (*left to right*); o segundo L diz respeito à produção de uma derivação mais à esquerda da cadeia (*left linear*); o número 1 sinaliza que é necessário um único token de *lookahead* a cada passo para a tomada de decisões sintáticas.

Propriedades das gramáticas $LL(1)$

- ▶ Nenhuma gramática recursiva à esquerda ou ambígua pode ser $LL(1)$
- ▶ Uma gramática G é $LL(1)$ se, e somente se, sempre que $A \rightarrow \alpha \mid \beta$ forem duas produções distintas de G , valem as seguintes condições:
 1. α e β não derivam, ao mesmo tempo, cadeias começando pelo mesmo terminal a , qualquer que seja a
 2. no máximo um dos dois (α ou β) deriva a cadeia vazia
 3. se $\beta \xRightarrow{*} \epsilon$, então α não deriva qualquer cadeia começando por um terminal em $\text{SEGUINTE}(A)$
- ▶ A gramática para expressões aritméticas é $LL(1)$
- ▶ A gramática que generaliza o comando **if-then-else** não é $LL(1)$

Recuperação de erros na análise preditiva

- ▶ A pilha de um analisador preditivo não-recursivo torna explícitos os terminais e os não-terminais que o mesmo espera reconhecer com o restante da entrada
- ▶ Um erro é detectado na análise preditiva em dois casos
- ▶ O primeiro deles acontece quando o terminal do topo da pilha não reconhece o próximo símbolo da entrada
- ▶ O segundo caso ocorre quando o não-terminal A está no topo da pilha, a é o próximo símbolo da entrada e a entrada $M[A, a]$ da tabela sintática está vazia
- ▶ A recuperação de erros na modalidade do desespero pula os símbolos da entrada até encontrar um token pertencente ao conjunto pré-definido dos tokens de sincronização
- ▶ A efetividade desta abordagem depende da escolha dos tokens de sincronização

Referências

1. **AHO**, Alfred V, **SETHI**, Ravi, **ULLMAN**, Jeffrey D. *Compiladores: Princípios, Técnicas e Ferramentas*, LTC Editora, 1995.