

ANALISE SINTÁTICA

BOTTOM-UP

ANÁLISE BOTTOM-UP

- ▶ Análise empilhar e reduzir
- ▶ Análise de precedência de operadores
- ▶ Análise LR



ANÁLISE GRAMATICAL EMPILHAR E REDUZIR

- ▶ Tenta construir a árvore gramatical começando pelas folhas e para o topo.
- ▶ Consiste em “reduzir” uma cadeia ao símbolo de partida de uma gramática.



Análise Gramatical

Empilhar e Reduzir

▶ A cada passo de redução:

- ▶ Uma subcadeia particular, que reconheça o lado direito de uma produção, é substituída por um símbolo a esquerda daquela produção.
- ▶ Se a subcadeia tiver sido escolhida corretamente a cada passo, uma derivação mais a direita terá sido rastreada na ordem inversa.



Empilhar e Reduzir

Considere a gramática:

► Cadeia: abbcde

$S \rightarrow aABe$

a**b**bcde

$A \rightarrow Abc \mid b$

a**A**bcde

$B \rightarrow d$

aA**d**e

a**A**Be

S

Derivação mais a direita

$S \rightarrow aABe \rightarrow aAde \rightarrow aAbcde \rightarrow abbcde$



Handles

- ▶ É uma subcadeia que reconhece o lado direito de uma produção;
- ▶ A redução ao não-terminal do lado esquerdo da produção representa um passo ao longo do percurso de uma derivação mais a direita;
- ▶ Em muitos casos a cadeia mais e esquerda que reconhece o lado direito de uma produção, não é um handle.

Ex: Gramática Anterior.

a**b**bcde

a**A**bcde

a**AA**cde // b não é um handle.

- ▶ Handle, e cada subcadeia de uma redução, que consiga chegar ao símbolo inicial da gramática.
-



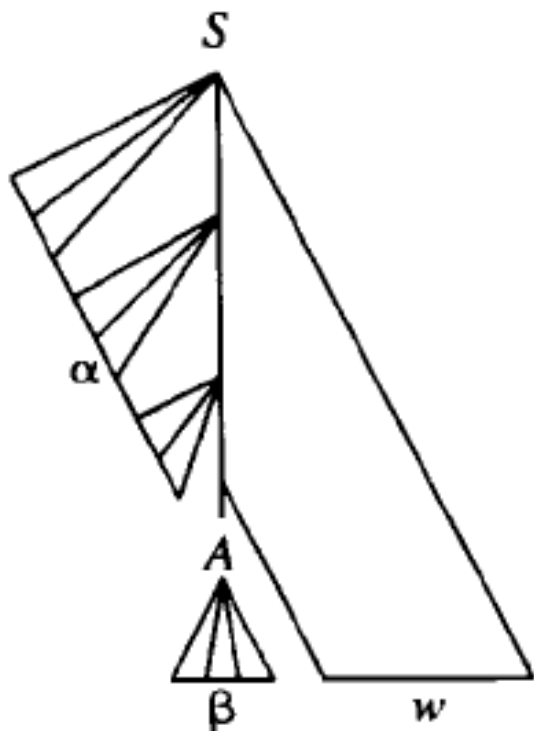
Handles

- ▶ Observações:
- ▶ A cadeia w à direita do handle contém apenas símbolos terminais;
- ▶ Se a gramática for inambígua, cada forma sentencial a direita possui apenas um handle.



Handles

- ▶ Podemos dizer que “ a subcadeia β é um handle de $\alpha\beta w$ ” se a posição de β e a produção $A \rightarrow \beta$ forem claras.



- ▶ Handle $A \rightarrow \beta$ na produção $\alpha\beta w$.
- ▶ A redução de β para A pode ser chamada de “poda do handle”

Handles

Considere a seguinte gramática

- (1) $E \rightarrow E + E$
- (2) $E \rightarrow E * E$
- (3) $E \rightarrow (E)$
- (4) $E \rightarrow \mathbf{id}$

$$\begin{aligned} E &\Rightarrow_{\text{mad}} \underline{E + E} && \text{Precedência } * \\ &\Rightarrow_{\text{mad}} E + \underline{E * E} \\ &\Rightarrow_{\text{mad}} E + E * \underline{\mathbf{id}_3} \\ &\Rightarrow_{\text{mad}} E + \underline{\mathbf{id}_2} * \mathbf{id}_3 \\ &\Rightarrow_{\text{mad}} \underline{\mathbf{id}_1} + \mathbf{id}_2 * \mathbf{id}_3 \end{aligned}$$

$$\begin{aligned} E &\Rightarrow_{\text{mad}} \underline{E * E} && \text{Precedência } + \\ &\Rightarrow_{\text{mad}} E * \underline{\mathbf{id}_3} \\ &\Rightarrow_{\text{mad}} \underline{E + E} * \mathbf{id}_3 \\ &\Rightarrow_{\text{mad}} E + \underline{\mathbf{id}_2} * \mathbf{id}_3 \\ &\Rightarrow_{\text{mad}} \underline{\mathbf{id}_1} + \mathbf{id}_2 * \mathbf{id}_3 \end{aligned}$$



A Poda do Handle

FORMA SENTENCIAL À DIREITA	HANDLE	PRODUÇÃO REDUTORA
$\text{id}_1 + \text{id}_2 * \text{id}_3$	id_1	$E \rightarrow \text{id}$
$E + \text{id}_2 * \text{id}_3$	id_2	$E \rightarrow \text{id}$
$E + E * \text{id}_3$	id_3	$E \rightarrow \text{id}$
$E + E * E$	$E * E$	$E \rightarrow E * E$
$E + E$	$E + E$	$E \rightarrow E + E$
E		

- Note que a sequência de formas sentenciais nesse exemplo é somente o inverso da sequência da primeira derivação do exemplo anterior.

A Poda do Handle

- ▶ Dois problemas devem ser considerados:
 1. Localizar a subcadeia a ser reduzida numa forma sentencial à direita
 2. Determinar que produção escolher, no caso de existir mais de uma.

Trataremos desses dois problemas mais tarde.



Implementando a pilha da Análise Sintática de Empilhar e Reduzir

► Implementando:

1. Pilha para guardar os símbolos gramaticais;
2. Buffer de entrada para a cadeia w ;
3. \$ no fundo da pilha e a direita da entrada;



Implementando a pilha da Análise Sintática de Empilhar e Reduzir

Iniciamos com a pilha está vazia e a cadeia w na entrada

PILHA	ENTRADA
\$	w \$



Implementando a pilha da Análise Sintática de Empilhar e Reduzir

- ▶ O analisador opera empilhando zero ou mais símbolos na pilha até que o handle β surja no topo da pilha.
- ▶ Reduz β para o lado esquerdo da produção apropriada.
- ▶ Repete o ciclo até que tenha detectado um erro ou que a pilha contenha apenas o símbolo de partida e a entrada esteja vazia.

▶ PILHA	ENTRADA
▶ \$\$	\$



Implementando a pilha da Análise Sintática de Empilhar e Reduzir

► Exemplo:

PILHA	ENTRADA	AÇÃO
(1) \$	id₁ + id₂ * id₃ \$	empilhar
(2) \$ id₁	+ id₂ * id₃ \$	reduzir por $E \rightarrow \text{id}$
(3) \$ E	+ id₂ * id₃ \$	empilhar
(4) \$ E +	id₂ * id₃ \$	empilhar
(5) \$ E + id₂	* id₃ \$	reduzir por $E \rightarrow \text{id}$
(6) \$ E + E	* id₃ \$	empilhar
(7) \$ E + E *	id₃ \$	empilhar
(8) \$ E + E * id₃	\$	reduzir por $E \rightarrow \text{id}$
(9) \$ E + E * E	\$	reduzir por $E \rightarrow E * E$
(10) \$ E + E	\$	reduzir por $E \rightarrow E + E$
(11) \$ E	\$	aceitar

Implementando a pilha da Análise Sintática de Empilhar e Reduzir

► Existem quatro operação possíveis:

1. Empilhar
2. Reduzir
3. Aceitar
4. Erro



Prefixos Variáveis

- ▶ **Def(Livro):** Um prefixo de uma forma sentencial a direita, o qual não se estende para além do limite a direita do *handle* mais a direita, daquela forma sentencial.
- ▶ **Professor:** É sempre possível adicionar símbolos terminais ao final do prefixo variável de modo a obter uma forma sentencial a direita;



Conflitos durante a análise sintática de empilhar e reduzir

- ▶ Existem Gramáticas Livres de Contexto (**Gramáticas não LR (k)**) para as quais o analisador empilhar e reduzir não pode ser usado;

Podem existir duas situações:

1. Mesmo conhecendo toda a pilha e o próximo símbolo de entrada, não pode decidir entre empilhar e reduzir;
2. Não pode decidir qual das diversas reduções alternativas realizar;



Exemplo: Gramáticas Ambíguas

- ▶ Uma Gramática Ambígua jamais poderá ser LR;

cmd → **if** *exp* **then** *cmd*
 | **if** *exp* **then** *cmd* **else** *cmd*
 | **outro**

Se tivermos um analisador sintático com uma única configuração

PILHA

... **if** *exp* **then** *cmd*

ENTRA

else ...\$

Não podemos dizer se **if** *exp* **then** *cmd* é o *cmd*, não importa o que apareça abaixo do mesmo na pilha.

FALAREMOS MAIS
SOBRE ISSO DEPOIS.



ANÁLISE SINTÁTICA DE PRECEDÊNCIA DE OPERADORES

Gramática de Precedência de Operadores

► Propriedades:

1. Nenhum lado direito de produção seja ε
2. Não tenha dois não-terminais subjacentes
3. Os operadores tem precedência uns sobre os outros

► Três relações de precedência:

1. $a <\bullet b \rightarrow$ a “confere precedência a” b;
2. $a = b \rightarrow$ a “tem a mesma precedência que” b;
3. $a \bullet > b \rightarrow$ a “tem precedência sobre ” b.

Essas relações de precedência guiam a seleção de *handles*.



Usando Relações de Precedência de Operadores

- ▶ O objetivo das relações de precedência é delimitar o *handle* de uma forma sentencial à direita.
- ▶ Com:
 - $\leftarrow \bullet \rightarrow$ assinalando o limite à esquerda;
 - $= \rightarrow$ marcando o interior do *handle*;
 - $\bullet \rightarrow$ marcando o limite à direita.



Precedência de Operadores

► Por exemplo:

Forma sentencial à direita: **id + id * id;**

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \uparrow E \mid (E) \mid -E \mid \text{id}$$

	id	+	*	\$
id		.>	.>	.>
+	<.	.>	<.	.>
*	<.	.>	.>	.>
\$	<.	<.	<.	

Relações de precedência de operadores.

$$\$ < \cdot \text{id} \cdot > + < \cdot \text{id} \cdot > * < \cdot \text{id} \cdot > \$$$



Precedência de Operadores

► O *handle* pode ser encontrado:

1. Esquadrilhar a cadeia a partir da esquerda até encontrar o primeiro $\bullet>$;
2. Esquadrilhar de volta por sobre as relações $=$ até encontrar $<\bullet$;
3. O *handle* é tudo a esquerda do primeiro $\bullet>$ e á direita de $<\bullet$.

$\$ < \text{id} \bullet > + < \bullet \text{id} \bullet > * < \bullet \text{id} \bullet > \$$



Precedência de Operadores

GRAMÁTICA:

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \uparrow E \mid (E) \mid -E \mid \text{id}$

Cadeia: **id + id * id**

Reduzir **id** para **E**.

E + id * id

Reduzir os dois **id**'s restantes

E + E * E

E + E * E

Considerando a Cadeia:

\$ + * \$ (remoção do ñ-term.)

Temos:

\$ <• + <• * •> \$

Da cadeia:

E + E * E

Handle: **E * E**



Precedência de Operadores

► Observações Importantes:

1. Pode parecer que toda sentença precise ser esquadrilhada a cada passo para encontrarmos o handle. Isso pode ser resolvido usando uma pilha para armazenar os símbolos já esquadrilhados.
2. Se nenhuma relação de precedência vigorar entre um par de terminais, então um erro sintático foi detectado. Invoca-se a rotina de recuperação de erros.



Relação de precedência de operadores a partir da associatividade

► **IMPORTANTE:**

- Existe uma liberdade para criar relações de precedência de operadores a qualquer ponto em que as vejamos adequadas e esperamos que o algoritmo de análise sintática de precedência de operadores irá funcionar corretamente quando guiados por elas.



Relação de precedência de operadores a partir da associatividade

► REGRAS:

1. Se θ_1 possuir maior precedência que θ_2 , fazer $\theta_1 > \theta_2$ e $\theta_2 < \theta_1$.
2. Se θ_1 e θ_2 são de igual precedência (podem ser o mesmo operador), fazer $\theta_1 > \theta_2$ e $\theta_2 > \theta_1$ se forem associativos a esquerda e $\theta_1 < \theta_2$ e $\theta_2 < \theta_1$ se forem associativos a direita.



Relação de precedência de operadores a partir da associatividade

3. Fazer:

$\theta < \mathbf{id},$	$\mathbf{id} > \theta,$	$\theta < (,$	$(< \theta,$
$) > \theta,$	$\theta >)$	$\theta > \$$	$\$ < \theta,$

Para todos os operadores θ .

Fazer também:

$(=)$	$\$ < ($	$\$ < \mathbf{id}$
$(< ($	$\mathbf{id} > \$$	$) > \$$
$(< \mathbf{id}$	$\mathbf{id} >)$	$) >)$

Esses regras asseguram o uso dos parênteses e o \$ como marcados de extremidades



Precedência de Operadores

Exemplo:

Gramática:

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \uparrow E \mid (E) \mid -E \mid \text{id}$

	+	-	*	/	\uparrow	id	()	\$
+	.>	.>	<.	<.	<.	<.	<.	.>	.>
-	.>	.>	<.	<.	<.	<.	<.	.>	.>
*	.>	.>	.>	.>	<.	<.	<.	.>	.>
/	.>	.>	.>	.>	<.	<.	<.	.>	.>
\uparrow	.>	.>	.>	.>	<.	<.	<.	.>	.>
id	.>	.>	.>	.>	.>			.>	.>
(<.	<.	<.	<.	<.	<.	<.	=.	
)	.>	.>	.>	.>	.>			.>	.>
\$	<.	<.	<.	<.	<.	<.	<.		

. Relações de precedência de operadores.

Precedência de Operadores

1. \uparrow possui maior precedência e é associativo a direita;
2. $*$ e $/$ possui a segunda maior precedência e é associativo a esquerda;
3. $+$ e $-$ possui a menor precedência e é associativo a esquerda;



FUNÇÕES DE PRECEDÊNCIA

- ▶ Os compiladores que usam análise de precedência de operadores não precisam usar a tabela de relações;
- ▶ A tabela pode ser codificada por duas funções, f e g , que manipulam símbolos terminais em inteiros;
- ▶ Para quaisquer símbolos a e b ;
 1. $f(a) < g(b)$, sempre que $a < \bullet b$;
 2. $f(a) = g(b)$, sempre que $a = b$;
 3. $f(a) > g(b)$, sempre que $a \bullet > b$;

A relação de precedência entre a e b pode ser determinada entre a comparação numérica entre $f(a)$ e $g(b)$.



Exemplo

	+	-	*	/	↑	id	()	\$
+	·>	·>	<.	<.	<.	<.	<.	·>	·>
-	·>	·>	<.	<.	<.	<.	<.	·>	·>
*	·>	·>	·>	·>	<.	<.	<.	·>	·>
/	·>	·>	·>	·>	<.	<.	<.	·>	·>
↑	·>	·>	·>	·>	<.	<.	<.	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(<.	<.	<.	<.	<.	<.	<.	=.	
)	·>	·>	·>	·>	·>			·>	·>
\$	<.	<.	<.	<.	<.	<.	<.		

. Relações de precedência de operadores.

	+	-	*	/	↑	()	id	\$
<i>f</i>	2	2	4	4	4	0	6	6	0
<i>g</i>	1	1	3	3	5	5	0	5	0



ANALISADORES SINTÁTICOS LR

ANALISADORES SINTÁTICOS LR

- ▶ A técnica é chamada de LR (k);

VANTAGENS:

1. Pode reconhecer todas as construções de uma linguagem de programação estrita por uma glc;
2. É o método mais geral dentro os métodos sem retrocesso de empilhar e reduzir;
3. Implementação eficiente;
4. Detecção rápida dos erros



ANALISADORES SINTÁTICOS LR

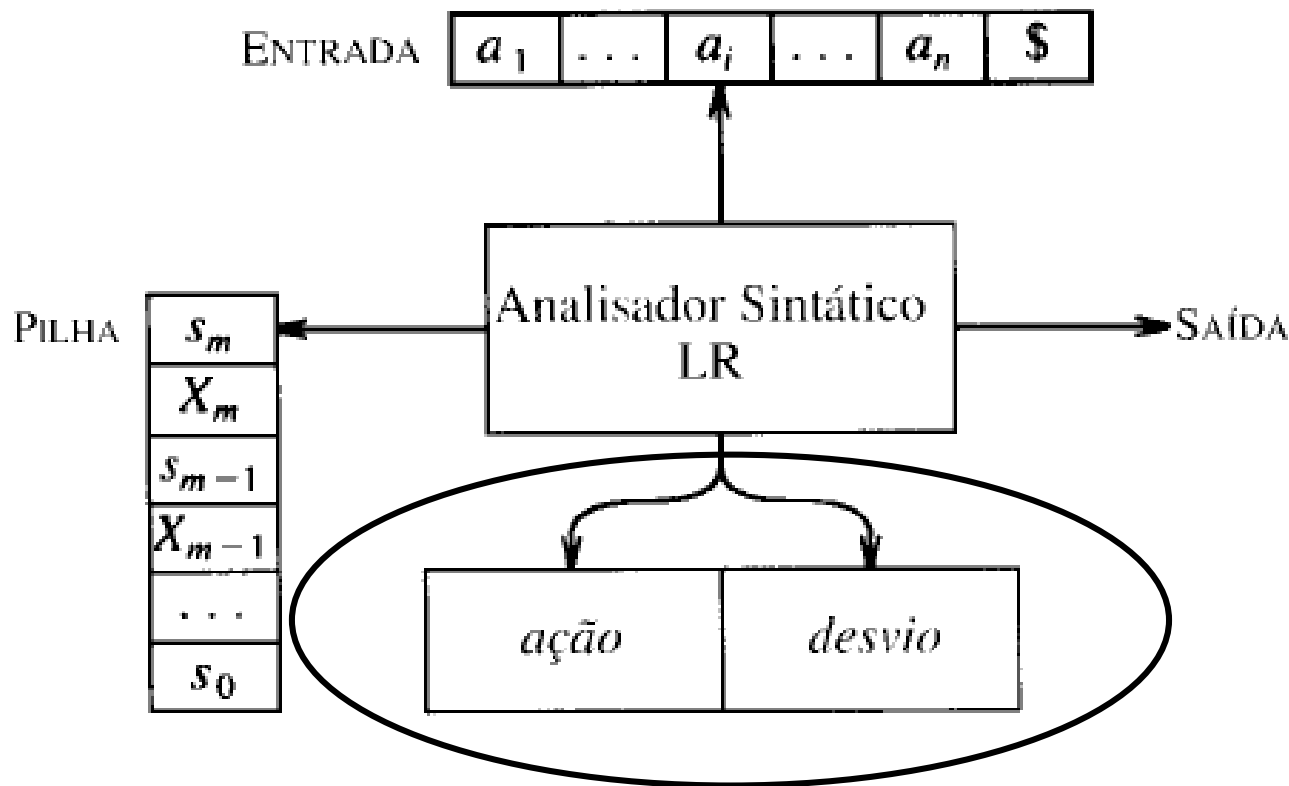
DESVANTAGEM:

- ▶ É muito trabalhoso construir um analisador sintático LR manualmente;
- ▶ Em geral usa-se uma ferramenta especializada - Yacc;
- ▶ Três técnicas:
 1. LR simples (SLR);
 2. LR canônico;
 3. LR lookahead (LALR).



Algoritmo de Análise Sintática LR

► Esquema:

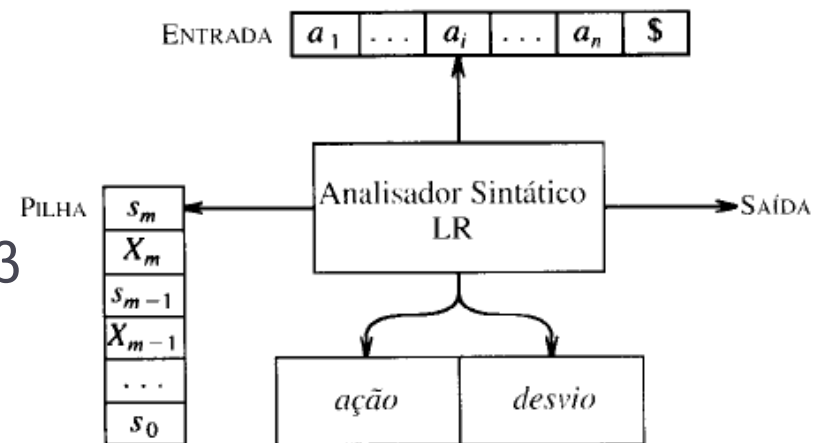


OBS:
Numa implementação
os símbolos
gramaticais não
precisam figurar na
pilha
durante a análise
 $s_i \rightarrow$ símbolo estado;

Algoritmo de análise sintática LR

► Programa Diretor:

1. Determina s_m e a_i ;
2. Consulta em ação $[s_m, a_i]$:
 1. Empilhar s , onde s é um estado
 2. Reduzir através da produção $A \rightarrow \beta$
 3. Aceitar
 4. Erro



A função desvio toma um estado e um símbolo como argumentos e produz um estado de saída. Ela é uma função de transição de um AFD.



Algoritmo de Análise Sintática LR

► Configuração (par):

1. Primeiro componente: conteúdo da pilha;
2. Segundo componente: entrada ainda não consumida;

$(s_0 X_1 s_1 X_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \$)$

Igual ao empilhar e reduzir;

A novidade é a presença dos **estados** na pilha;



Algoritmo de Análise Sintática LR

- ▶ O movimento é determinado por $[s_m, a_i]$
- 1. Se ação $[s_m, a_i] = \text{empilhar } s$. **Empilhar é executado.**
O analisador empilha tanto o símbolo corrente de entrada quanto o próximo estado s ;
- 2. Se ação $[s_m, a_i] = \text{reduzir } A \rightarrow \beta$. **Reduzir é executado.**
Onde $s = \text{desvio } [s_{m-r}, A]$, $r = \text{comprimento de } \beta$

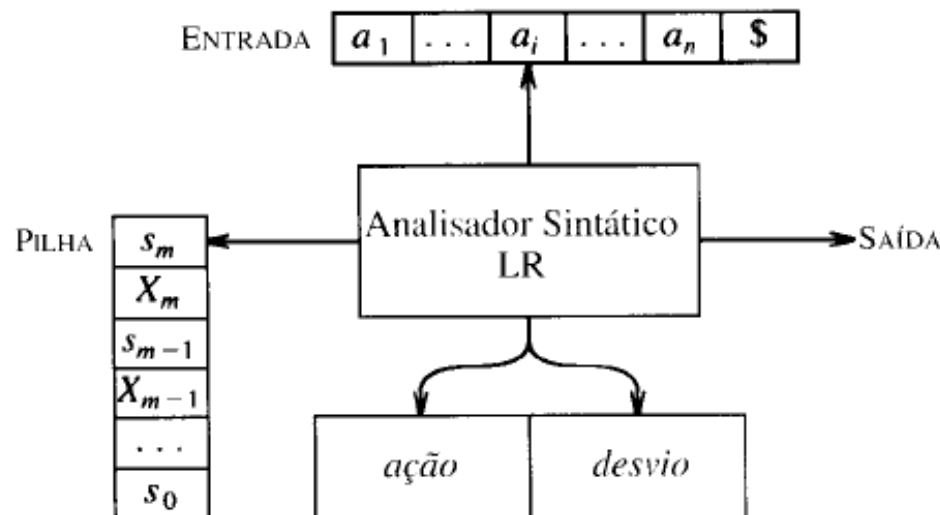
O analisador remove da pilha $2r$ símbolos (r símbolos de estado, e r símbolos gramaticais). Expondo o estado s_{m-r}

Em seguida empilha A e s



Algoritmo de Análise Sintática LR

3. Se ação[s_m, a_i] = aceitar. **Análise sintática completa.**
4. Se ação[s_m, a_i] = erro. **Chamar procedimento de recuperação de erros.**



Exemplo

$$(1) \quad E \rightarrow E + T$$

$$(2) \quad E \rightarrow T$$

$$(3) \quad T \rightarrow T * F$$

$$(4) \quad T \rightarrow F$$

$$(5) \quad F \rightarrow (E)$$

$$(6) \quad F \rightarrow \text{id}$$

O código para cada ação é:

1. *si* significa empilhar o símbolo de entrada mais o estado *i*,

2. *rj* significa reduzir através da produção de número *j*,

3. *acc* significa aceitar,

4. uma entrada em branco significa um erro.

ESTADO	ação						desvio		
	id	+	*	()	\$	<i>E</i>	<i>T</i>	<i>F</i>
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

PILHA	ENTRADA	AÇÃO
(1) 0	id * id + id \$	empilhar
(2) 0 id 5	* id + id \$	reduzir por $F \rightarrow \mathbf{id}$
(3) 0 F 3	* id + id \$	reduzir por $T \rightarrow F$
(4) 0 T 2	* id + id \$	empilhar
(5) 0 T 2 * 7	id + id \$	empilhar
(6) 0 T 2 * 7 id 5	+ id \$	reduzir por $F \rightarrow \mathbf{id}$
(7) 0 T 2 * 7 F 10	+ id \$	reduzir por $T \rightarrow T * F$
(8) 0 T 2	+ id \$	reduzir por $E \rightarrow T$
(9) 0 E 1	+ id \$	empilhar
(10) 0 E 1 + 6	id \$	empilhar
(11) 0 E 1 + 6 id 5	\$	reduzir por $F \rightarrow \mathbf{id}$
(12) 0 E 1 + 6 F 3	\$	reduzir por $T \rightarrow F$
(13) 0 E 1 + 6 T 9	\$	$E \rightarrow E + T$
(14) 0 E 1	\$	aceitar

- (1) $E \rightarrow E + T$
(2) $E \rightarrow T$
(3) $T \rightarrow T * F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow \mathbf{id}$

ESTADO	ação						desvio		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Construção da Tabela SLR

INTRODUÇÃO

- ▶ Gramática LR \rightarrow uma gramática livre de contexto para a qual podemos construir uma tabela sintática;
- ▶ L \rightarrow *left-to-right*
- ▶ R \rightarrow *right most derivation*
- ▶ Um analisador LR não precisa varrer toda a pilha para saber quando o *handle* surge no topo;



Gramática LR (pontos importantes)

- ▶ Uma gramática que podemos construir uma tabela sintática é denominada de LR;
- ▶ Existem GLC que não são LR;
- ▶ Para uma gramática ser LR deve ser possível construir um analisador de empilhar e reduzir, que seja capaz de reconhecer os handles;
- ▶ Uma gramática que pode ser decomposta por um analisador LR examinando até k símbolos de entrada a cada movimento é chamada de gramática LR (k);
- ▶ Gramáticas LR podem descrever mais linguagens que as gramática LL;



Função Desvio

- ▶ Definição: Autômato finito que pode, através da leitura dos símbolos gramaticais da pilha, determinar qual o *handle*.
- ▶ O autômato não precisa ler toda a pilha a cada movimento;
- ▶ O símbolo do estado no topo da pilha é o estado que o AF estaria se tivesse lido os símbolos gramaticais.



LR(k)

- ▶ Uma gramática LR(k) examina até k símbolos de entrada a cada movimento;
- ▶ Por serem menos restritivos as gramáticas LR podem descrever mais LP que as LL.



ITEM

- ▶ Para uma gramática G um item é uma produção de G com um ponto em algum lugar de suas posições no lado direito.

EX:

G:

$A \rightarrow XYZ$

4 ITENS

$A \rightarrow \cdot XYZ$

$A \rightarrow X \cdot YZ$

$A \rightarrow XY \cdot Z$

$A \rightarrow XYZ \cdot$

$A \rightarrow \varepsilon$

1 ITEM

$A \rightarrow \cdot$

ITEM

- ▶ Um item pode ser representado por um par de inteiros
- ▶ (x,y)
 - ▶ $x \rightarrow$ representa o número da produção;
 - ▶ $y \rightarrow$ representa a posição do ponto.
- ▶ Um item indica quanto de uma produção já examinamos a uma dada altura de uma análise sintática.
 - ▶ $A \rightarrow X \cdot YZ$
 - ▶ Acabamos de ler X e esperamos ver a cadeia YZ



SLR – Ideia Central

- ▶ Construir, a partir da gramática, um autômato finito determinístico que reconheça prefixos variáveis.
- ▶ Agruparemos esse itens com conjuntos, os quais dão origem aos estados do analisador sintático.
- ▶ O agrupamento dos itens é, de fato, um processo de construção de subconjuntos.



Coleção LR(0) Canônica

- ▶ Coleção do conjunto de itens LR(0)
- ▶ Para a construção da coleção LR(0) Canônica definimos:
 1. Uma gramática aumentada;
 2. Duas funções:
 - ▶ Fechamento
 - ▶ Desvio



GRAMÁTICA AUMENTADA

- ▶ Sendo G uma gramática com um símbolo de partida S ;
- ▶ G' é a gramática aumentada para G .
 - ▶ Novo símbolo de partida S' [Produção $S' \rightarrow S$]
- ▶ Objetivo da Produção $S' \rightarrow S$
 - ▶ Indicar ao analisador sintático quando o mesmo deve parar de analisar e anunciar a aceitação da entrada.



Operação Fechamento

- ▶ Se I for um conjunto de itens para a gramática G , então o *fechamento*(I) é o conjunto de itens construídos a partir de I por duas regras:
 1. Cada item de I é adicionado ao fechamento de I
 2. Se $A \rightarrow \alpha \cdot B\beta$ estiver em fechamento de I e $B \rightarrow \gamma$ for produção, adicionar o item $B \rightarrow \gamma$, caso não esteja lá. Aplicamos essa regra até que não possam mais ser adicionados itens ao *fechamento*(I).
- ▶ Se $A \rightarrow \alpha \cdot B\beta$ está em *fechamento*(I) esperamos $B\beta$ e se $B \rightarrow \gamma$ também esperamos ver γ .



Função Fechamento (Algoritmo)

função *fechamento*(I)

início

J:=I

repetir

para cada item $A \rightarrow \alpha \cdot B\beta$ em J e cada
produção $B \rightarrow \gamma$ de G tal que $B \rightarrow \cdot \gamma$ não
esteja em J faça

incluir $B \rightarrow \cdot \gamma$ a J

até que não possam mais ser adicionados itens

fim



Operação Fechamento (Exemplo)

- ▶ Gramática de Expressões (aumentada)
- ▶ Se I for o conjunto $\{[E' \rightarrow \cdot E]\}$, então o fechamento(I) contém os itens.

$$E' \rightarrow E$$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

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

$$E' \rightarrow \cdot E$$

$$E \rightarrow \cdot E + T$$

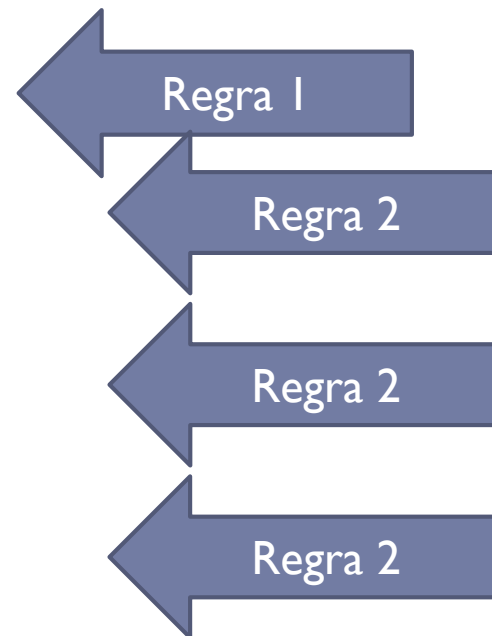
$$E \rightarrow \cdot T$$

$$T \rightarrow \cdot T * F$$

$$T \rightarrow \cdot F$$

$$F \rightarrow \cdot (E)$$

$$F \rightarrow \cdot \mathbf{id}$$



Criar o conjunto de todos os itens para a gramática:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

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

Gramática Aumentada

$$E' \rightarrow E$$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

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



Conjunto de Itens

$$E' \rightarrow \cdot E$$

$$E' \rightarrow E \cdot$$

$$E \rightarrow \cdot E + T$$

$$E \rightarrow E \cdot + T$$

$$E \rightarrow E + \cdot T$$

$$E \rightarrow E + T \cdot$$

$$E \rightarrow \cdot T$$

$$E \rightarrow T \cdot$$

$$T \rightarrow \cdot T * F$$

$$T \rightarrow T \cdot * F$$

$$T \rightarrow T * \cdot F$$

$$T \rightarrow T * F \cdot$$

$$T \rightarrow \cdot F$$

$$T \rightarrow F \cdot$$

$$F \rightarrow \cdot (E)$$

$$F \rightarrow (\cdot E)$$

$$F \rightarrow (E \cdot)$$

$$F \rightarrow (E) \cdot$$

$$F \rightarrow \cdot id$$

$$F \rightarrow id \cdot$$



Fechamento

I₀

$E' \rightarrow \cdot E$
 $E \rightarrow \cdot E + T$
 $E \rightarrow \cdot T$
 $T \rightarrow \cdot T * F$
 $T \rightarrow \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot id$

I₁

$E' \rightarrow E \cdot$
 $E \rightarrow E \cdot + T$

I₂

$E \rightarrow E + \cdot T$
 $T \rightarrow \cdot T * F$
 $T \rightarrow \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot id$

I₃

$E \rightarrow E + T \cdot$
 $T \rightarrow T \cdot * F$

I₄

$E \rightarrow T \cdot$
 $T \rightarrow T \cdot * F$

I₅

$T \rightarrow T * \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot id$

I₆

$T \rightarrow T * F \cdot$

I₇

$T \rightarrow F \cdot$

I₈

$F \rightarrow (\cdot E)$
 $E \rightarrow \cdot E + T$
 $E \rightarrow \cdot T$
 $T \rightarrow \cdot T * F$
 $T \rightarrow \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot id$

I₉

$F \rightarrow (E \cdot)$
 $E \rightarrow E \cdot + T$

I₁₀

$F \rightarrow (E) \cdot$

I₁₁

$F \rightarrow id \cdot$



Operação Desvio

Desvio(I,X)

- ▶ I conjunto de itens
- ▶ X símbolo gramatical
- ▶ Definida como fechamento do conjunto de todos os itens $[A \rightarrow \alpha X \cdot \beta]$ tais que $[A \rightarrow \alpha \cdot X \beta]$ esteja em I.
- ▶ Se I for o conjunto de Itens válidos para algum prefixo variável γ , então *desvio(I,X)* será o conjunto de itens válidos para o prefixo variável γX .



Operação Desvio (Exemplo)

- ▶ Se I for o conjunto de itens

$$E' \rightarrow E \cdot$$

$$E \rightarrow E \cdot + T$$

desvio($I, +$)

$$E \rightarrow E + \cdot T$$

$$T \rightarrow \cdot T * F$$

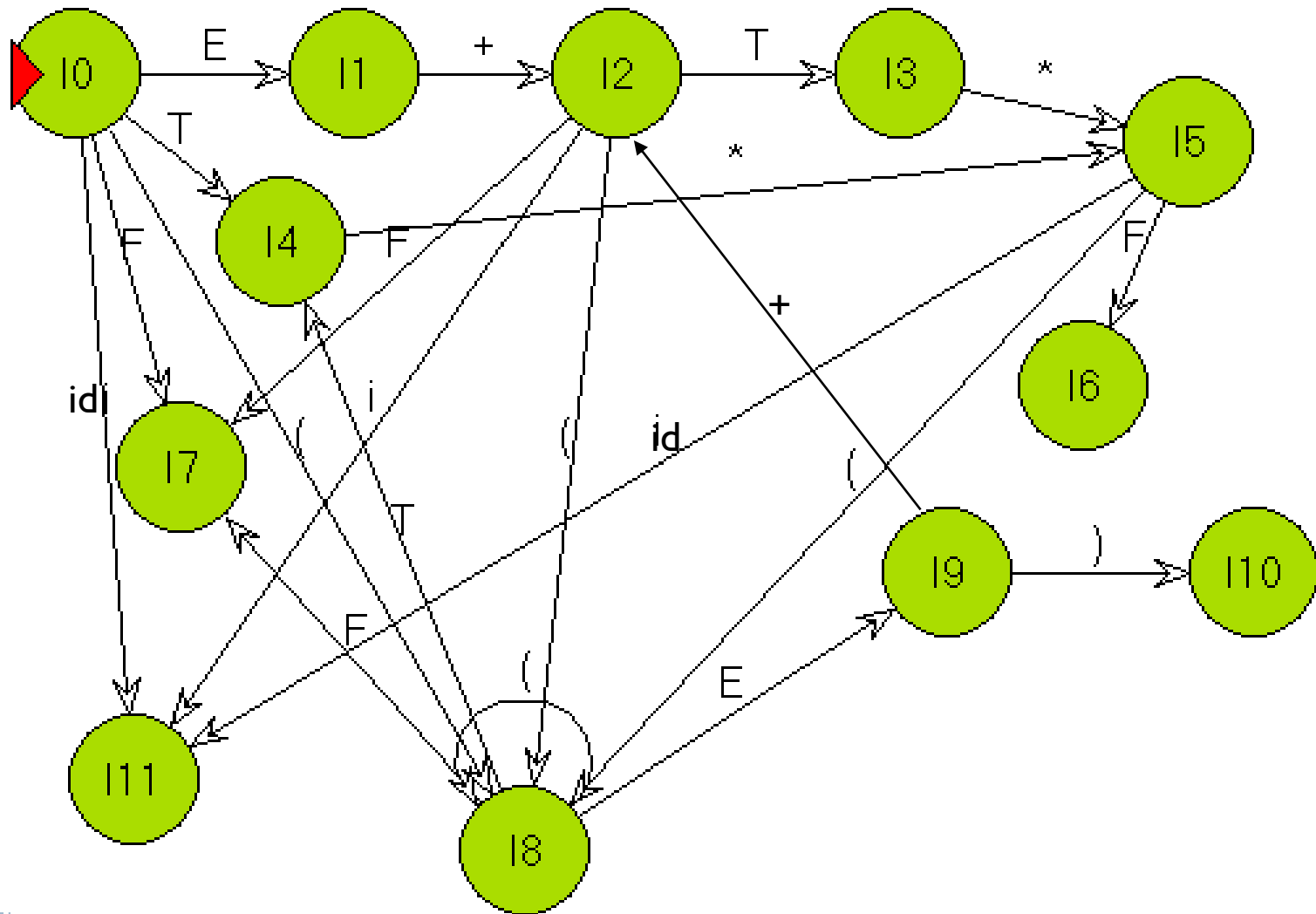
$$T \rightarrow \cdot F$$

$$F \rightarrow \cdot (E)$$

$$F \rightarrow \cdot id$$



Diagrama de Transições para AF para prefixos variáveis



Exemplo:

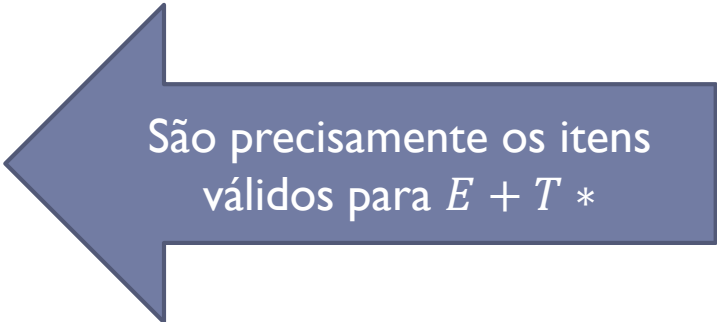
- ▶ A cadeia $E + T *$ é um prefixo variável para a gramática de expressões;
- ▶ Após ler $E + T *$ estamos no estado I_5 ;

I_5

$T \rightarrow T * \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot id$



São precisamente os itens
válidos para $E + T *$

$E' \rightarrow E$

$\rightarrow E + T$

$\rightarrow E + T * F$

$E' \rightarrow E$

$\rightarrow E + T$

$\rightarrow E + T * F$

$\rightarrow E + T * (E)$

$E' \rightarrow E$

$\rightarrow E + T$

$\rightarrow E + T * F$

$\rightarrow E + T * id$

Construção da Tabela Sintática (SLR)

- ▶ **Entrada** → Gramática G'
- ▶ **Saída** → As funções sintáticas SLR ação e desvio para G'



Método

1. Construir $C = \{I_0, I_1, \dots, I_n\}$, a coleção do conjunto de itens LR(0) para G' .
2. O estado i é construído a partir de I_i . Ações sintáticas para o estado i .
 - ▶ Se $[A \rightarrow \alpha \cdot a\beta]$ estiver em I_i e $\text{desvio}(I_i, a) = I_j$.
Ação $[i, a]$ = “empilhar j ”. ‘ a ’ precisa ser terminal;
 - ▶ Se $[A \rightarrow \alpha \cdot]$ estiver em I_i .
Ação $[i, a]$ = “redizer através de $A \rightarrow \alpha$ ”, para todo a em $\text{seguinte}(A)$. ‘ A ’ não pode ser S' .
 - ▶ Se $[S' \rightarrow S \cdot]$ estiver em I_i .
Ação $[i, \$]$ = “aceitar”



Método

3. As transições de desvio para o estado i são construídas para todos os não terminais A , usando-se a seguinte regra.

Se $\text{desvio}(l_i, A) = l_j$, então $\text{desvio}[i, A] = j$.

4. O que não foi definido em 2 e 3 é erro.
5. Estado inicial $[S' \rightarrow S]$



Exemplo

AÇÃO							DESVIO		
	id	+	*	()	\$	E	T	F
0	E11			E8			1	4	7
1		E2				acc			
2	E11			E8				3	7
3		R1	E5		R1	R1			
4		R2	E5		R2	R2			

I₀: $F \rightarrow \cdot (E)$ ação[0,(] empilhar 8
 $F \rightarrow \cdot id$ ação[0,id] empilhar 11

I₁: $E' \rightarrow E \cdot$ ação[1,\$] acc
 $E \rightarrow T \cdot +F$ ação[1,+] empilhar 2

I₂: $F \rightarrow \cdot (E)$ ação[1,+] empilhar 8
 $F \rightarrow \cdot id$ ação[0,id] empilhar 11

I₄: $E \rightarrow T \cdot$ Seg(E) = { \$, +,) } ação[4,\$]=[4,+]=[4,)] → Reduzir $E \rightarrow T$
 $E \rightarrow T \cdot * F$ ação[2,*] empilhar 5

- (1) $E \rightarrow E + T$
 - (2) $E \rightarrow T$
 - (3) $T \rightarrow T * F$
 - (4) $T \rightarrow F$
 - (5) $F \rightarrow (E)$
 - (6) $F \rightarrow id$