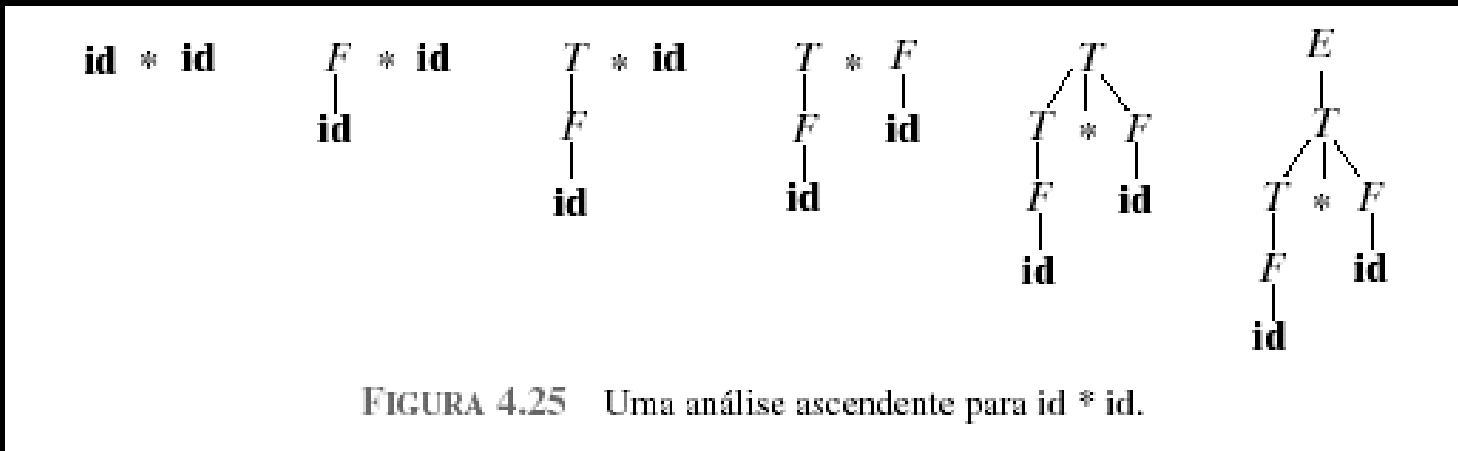


Análise Sintática Ascendente ("Bottom-Up")

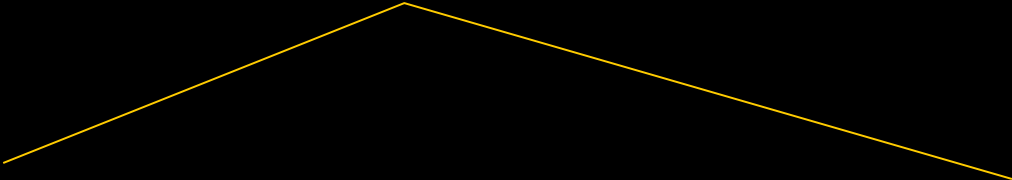
Construção da árvore a partir das folhas



Objetivo - redução da cadeia ao símbolo inicial da gramática

Análise Sintática Ascendente ("Bottom-Up")

(Empilhar-Reduzir) Shift-Reduce Parsing



Precedência Simples
e Precedência de
Operadores

Métodos LR

Análise Sintática Ascendente ("Bottom-Up")

Passo: reduzir uma sub-cadeia a um não-terminal

Ex:

$S \rightarrow aAcBe$

$A \rightarrow Ab/b$

$B \rightarrow d$

sent: abbcde

1. b por A ou d por B?
2. Ab por A ou b por A?
3. d por B $\rightarrow aAcBe$
4. aAcBe por S $\rightarrow S$

Análise Sintática Ascendente ("Bottom-Up")

Ex:

$$S \rightarrow aAcBe$$
$$A \rightarrow Ab/b$$
$$B \rightarrow d$$

sent: abbcde

Derivação mais à direita:

$S \rightarrow aAcBe \rightarrow aAcde \rightarrow aAbcde \rightarrow$
 $abbcde$

Análise Sintática Ascendente (“Bottom-Up”)

Handle:

- lado direito de uma regra
- cuja redução representa um passo em direção à construção de uma derivação mais à direita.

Análise Sintática Ascendente ("Bottom-Up")

Handle: definição - um handle de uma forma sentencial à direita, γ , consiste de:

- uma produção $A \rightarrow \beta$
- uma posição de γ onde β pode ser encontrada e substituída por A , para produzir a forma sentencial anterior, numa derivação mais à direita de γ

Análise Sintática Ascendente ("Bottom-Up")

Isto é:

se $S \rightarrow \alpha A w \rightarrow \alpha \beta w$, então

$A \rightarrow \beta$, na posição que segue α , é um
handle de $\alpha \beta w$ [$w \in V_T^*$]

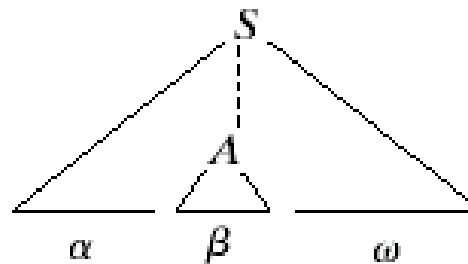


FIGURA 4.27 Um handle $A \rightarrow \beta$ na árvore de derivação para $\alpha \beta w$.

Análise Sintática Ascendente ("Bottom-Up")

no Ex1: handle

$abbcde = A \rightarrow b$ (posição 2)

$aAbcde = A \rightarrow Ab$ (posição 2)

$aAcde = B \rightarrow d$ (posição 4)

$aAcBe = S \rightarrow aAcBe$ (posição 1)

Análise Sintática Ascendente (“Bottom-Up”)

FORMA SENTENCIAL À DIREITA	HANDLE	PRODUÇÃO DE REDUÇÃO
$id_1 * id_2$	id_1	$F \rightarrow id$
$F * id_2$	F	$T \rightarrow F$
$T * id_2$	id_2	$F \rightarrow id$
$T * F$	$T * F$	$E \rightarrow T * F$

FIGURA 4.26 Handles durante a análise de $id_1 * id_2$.

Análise Sintática Ascendente (“Bottom-Up”) – Poda do Handle

Uma derivação mais à direita na ordem inversa pode ser obtida através da “poda dos handles”.

Para isso, localizamos o handle β_n em γ_n e substituímos β_n pelo lado direito de alguma produção $A_n \rightarrow \beta_n$, de modo a obtermos a enésima menos uma forma sentencial à direita γ_{n-1} .

Análise Sintática Ascendente ("Bottom-Up") – Poda do Handle

O processo continua até que seja encontrado o símbolo inicial da gramática.

Ex: considere a gramática $E \rightarrow E + E \mid E * E \mid (E) \mid id$ e a cadeia de entrada **id1 + id2 * id3**. A sequência de reduções é apresentada na tabela abaixo:

Análise Sintática Ascendente (“Bottom-Up”) – Poda do Handle

Forma sentencial à direita	Handle	Produção Redutora
id1 + id2 * id3	id1_	$E \rightarrow id$
E + id2 * id3	id2	$E \rightarrow id$
E + E * id3	id3	$E \rightarrow id$
E + E * E	$E * E$	$E \rightarrow E * E$
E + E	$E + E$	$E \rightarrow E + E$
E		

Análise Sintática Ascendente (“Bottom-Up”) – Poda do Handle

Obs: as formas $id1$, $id2$, e $id3$ servem meramente como ilustração para indicar qual elemento está sendo substituído. Todos obedecem à regra $E \rightarrow id$.

Implementação da Pilha da Análise Sintática Empilhar e Reduzir (Shift-Reduce)

Dois problemas precisam ser resolvidos se estivermos dispostos a analisar sintaticamente através da poda de handles.

Implementação da Pilha da Análise Sintática Empilhar e Reduzir (Shift-Reduce)

1. localizar a subcadeia a ser reduzida numa forma sentencial à direita.
2. determinar que produção escolher no caso de existir mais de uma produção com aquela subcadeia no lado direito

Implementação da Pilha da Análise Sintática Empilhar e Reduzir (Shift-Reduce)

Antes disso,

Vamos verificar os tipos de estruturas de dados usadas num analisador sintático de empilhar/reduzir

Implementação da Pilha da Análise Sintática Empilhar e Reduzir (Shift-Reduce)

Uma forma conveniente é usar um pilha para guardar os símbolos gramaticais e um *buffer* de entrada para a cadeia w a ser decomposta.

Implementação da Pilha da Análise Sintática Empilhar e Reduzir (Shift-Reduce)

Usamos \$ para marcar o fundo da pilha e também o final à direita da entrada.

Inicialmente, a pilha está vazia e a cadeia w está como segue.

PILHA

\$

ENTRADA

w \$

Implementação da Pilha da Análise Sintática Empilhar e Reduzir (Shift-Reduce)

O analisador sintático opera empilhando zero ou mais símbolos até que um handle β surja no topo da pilha. Reduz então β para o lado esquerdo da produção apropriada.

Implementação da Pilha da Análise Sintática Empilhar e Reduzir (Shift-Reduce)

Repete-se o ciclo até que tenha detectado um erro ou que a pilha contenha o símbolo de partida e a entrada esteja vazia

PILHA

\$S

ENTRADA

\$

Implementação da Pilha da Análise Sintática Empilhar e Reduzir (Shift-Reduce)

Ex: Vamos rastrear as ações que um analisador sintático realizaria para decompor a cadeia **id1 + id2 * id3**, de acordo com a gramática $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$, em sua primeira derivação

Análise Sintática Ascendente ("Bottom-Up")

	Pilha	INPUT	AÇÃO
1	\$	$\text{id}_1 + \text{id}_2 * \text{id}_3$ \$	shift
2	\$ id_1	$+ \text{id}_2 * \text{id}_3$ \$	reduce $E \rightarrow \text{id}$
3	\$ E	$+ \text{id}_2 * \text{id}_3$ \$	shift
4	\$ E +	$\text{id}_2 * \text{id}_3$ \$	shift
5	\$ E + id_2	$* \text{id}_3$ \$	reduce $E \rightarrow \text{id}$
6	\$ E + E	$* \text{id}_3$ \$	shift
7	\$ E + E *	id_3 \$	shift
8	\$ E + E * id_3	\$	reduce $E \rightarrow \text{id}$
9	\$ E + E * E	\$	reduce $E \rightarrow E * E$
10	\$ E + E	\$	reduce $E \rightarrow E + E$
11	\$ E	\$	aceita

4 operações:

SHIFT (empilhar) = próximo símbolo entrada é empilhado

REDUCE (reduzir) = símbolo mais à direita de um handle no topo; deve localizar extremidade esquerda do handle e decidir sobre regra.

ACEITAÇÃO = término da cadeia e pilha com S (símbolo inicial)

ERRO = descobre erro sintático e chama rotina de recuperação

Análise Sintática Ascendente ("Bottom-Up")

Uso de uma pilha:

- Empilha (SHIFT) símbolos da cadeia até detectar o símbolo mais à direita de um handle;
- Desempilha o handle (REDUCE) e empilha o lado esquerdo da respectiva regra.

Repetir o ciclo até situação de erro ou
terminar entrada e pilha = S

Análise Sintática Ascendente (“Bottom-Up”)

Um fato importante que justifica o uso de pilha numa análise sintática de empilhar e reduzir é que o handle irá sempre aparecer no topo da pilha, nunca dentro da mesma.

Análise Sintática Ascendente (“Bottom-Up”)

Como decidir sobre Shift ou Reduce?

Surgem dois tipos de conflitos:

- conflito shift/reduce: o que fazer com o símbolo do topo da pilha?
- conflito reduce/shift: qual regra aplicar?

Análise Sintática Ascendente ("Bottom-Up")

Exemplo de conflitos shift/reduce:

Cmd \rightarrow if Expr then Cmd |
 if Expr then Cmd else Cmd |

...

Análise Sintática Ascendente ("Bottom-Up")

Pilha

....if Expr then Cmd

Input

else.....\$

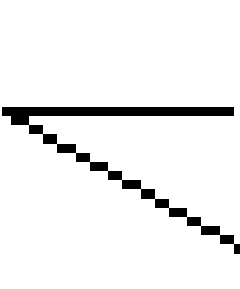
shift ou reduce?

Obs: não podemos dizer se *if Expr then Cmd* é o handle.

Gramáticas ambíguas

Análise Sintática Ascendente ("Bottom-Up")

Exemplo de conflito reduce/reduce

L.P. onde $A(i,j)$  array
chamada de subprograma

Análise Sintática Ascendente ("Bottom-Up")

Como resolver conflito reduce/reduce:
considere hipótese: Análise Léxica
retorna "id" para todo identificador

1. $\langle \text{cmd} \rangle ::= \text{id} (\langle \text{param_list} \rangle)$
2. $\langle \text{cmd} \rangle ::= \langle \text{expr} \rangle := \langle \text{expr} \rangle$
3. $\langle \text{param_list} \rangle ::= \langle \text{param_list} \rangle , \langle \text{param} \rangle$
4. $\langle \text{param_list} \rangle ::= \langle \text{param} \rangle$
5. $\langle \text{param} \rangle ::= \text{id}$

Análise Sintática Ascendente ("Bottom-Up")

cont..

6. $\langle \text{expr} \rangle ::= \text{id} (\langle \text{expr_list} \rangle)$

7. $\langle \text{expr} \rangle ::= \text{id}$

8. $\langle \text{expr_list} \rangle ::= \langle \text{expr_list} \rangle , \langle \text{expr} \rangle$

9. $\langle \text{expr_list} \rangle ::= \langle \text{expr} \rangle$

Análise Sintática Ascendente ("Bottom-Up")

Pilha

..... $id_1(id_2$

Input

, id).....

Problema: Reduzir id_2 via 5
(procedimento) ou 7 (array)?

- **uma solução:** mudar token id em 1 para
procid – (como? mudando o anal. Léxico)

Exercício

1) Para a gramática $S \rightarrow 0S1 \mid 01$, indique qual o handle em cada uma das formas sentenciais a direita

a) 000111

b) 00S11

Análise Sintática LR(k)

- O termo LR(k) é usado para representar uma análise cuja varredura de entrada acontece da esquerda para a direita (L = left to right), com a construção de uma derivação mais à direita ao contrário (R = rightmost derivation), usando o número k de símbolos de entrada de *lookahead* que serão usados ao se tomar decisões na análise sintática.

Análise Sintática LR(k) – Pontos que atraem esse tipo de análise

- os analisadores LR podem ser elaborados para reconhecer virtualmente todas as construções de linguagens de programação, que forem escritas em linguagens livres de contexto;
- método de decomposição LR é o mais geral de empilhar/reduzir.
- o analisador LR pode detectar um erro sintático tão cedo quanto possível numa varredura da esquerda para direita.

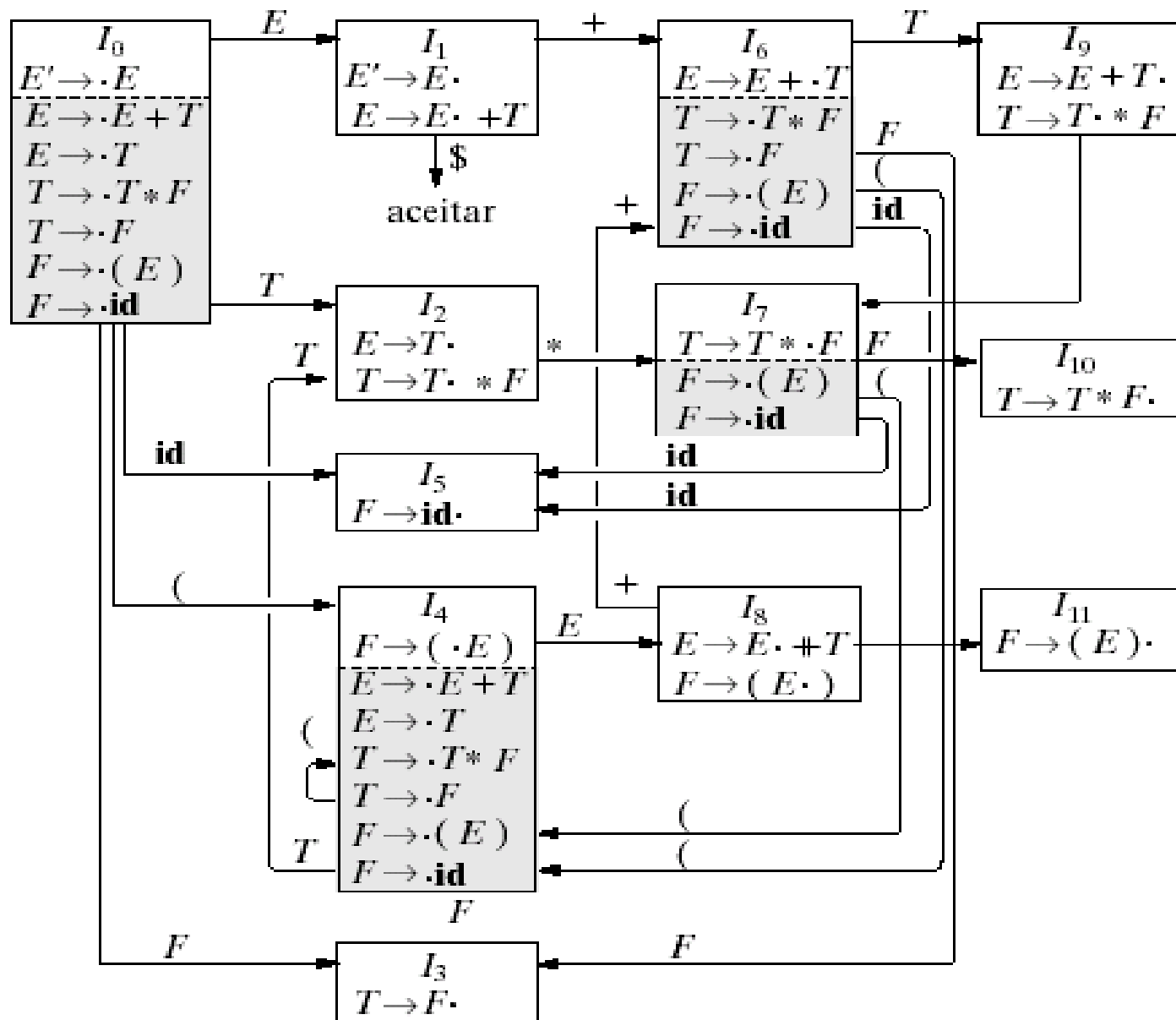


FIGURA 4.31 Autômato LR(0) para a gramática da expressão (4.1).

Desvantagem Principal

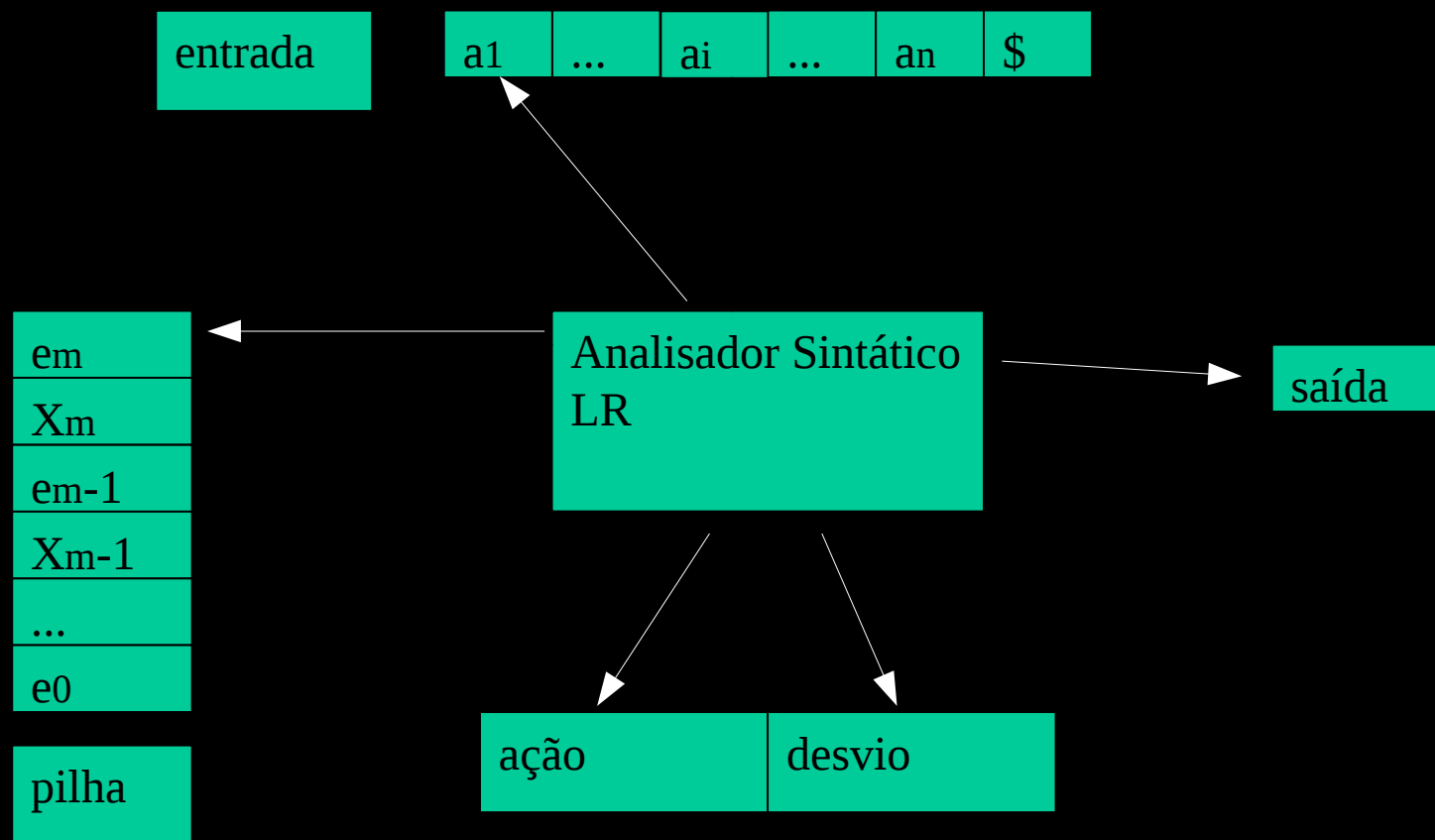
- é um processo muito trabalhoso construir um analisador sintático LR manualmente para uma linguagem de programação
- ESSE PROBLEMA é contornado com o uso de geradores de analisadores LR. ex: o Yacc, que veremos mais tarde!

Tipos de analisadores LR

- SRL (Simple LR) – fáceis de implementar, porém classe restrita de gramáticas
- LR canônicos – mais poderosos, podendo ser aplicado a um grande número de linguagens livres de contexto
- LALR (look ahead LR) – de nível intermediário e implementação eficiente. Funcionam com a maioria das linguagens de programação. O YACC gera esse tipo de analisador

Agora veremos o analisador SLR(1)

Modelo de um analisador sintático LR



- Na pilha, as cadeias são armazenadas na forma $e_0X_1e_1X_2e_2...Xmem$, onde X_i é um símbolo gramatical e e_i é um estado. Cada estado sumariza a informação contida na pilha abaixo do mesmo e a combinação do símbolo do estado no topo da pilha e o símbolo corrente de entrada é usada para indexar a tabela sintática e determinar ação empilhar.

Tabela de Análise

- Tabela de transição de estados formada por duas partes:

Ação: **ei** (empilhar estado e_i); **ri** (reduzir usando a i -ésima produção) e **a** (aceitar cadeia).

Transição: transição de estados com relação aos não terminais.

Como construir as tabelas SLR(1)

$E ::= +EE$

$E ::= *EE$

$E ::= a$

$E ::= b$

Definições iniciais:

- **item**: produção na qual foi marcada uma posição do lado direito. A posição é marcada por •

Exemplo:

$E ::= \bullet + EE \mid + \bullet EE \mid + E \bullet E \mid + EE \bullet$

$E ::= \bullet * EE \mid * \bullet EE \mid * E \bullet E \mid * EE \bullet$

$E ::= \bullet a \mid a \bullet \mid \bullet b \mid b \bullet$

Como construir as tabelas SLR(1)

Os itens são usados para construir os estados. A presença, no topo da pilha, de um estado contendo um item da forma $A ::= \alpha \bullet \beta$ indica que a parte inicial do handle (reduzindo) $\alpha\beta$, que é α , já foi processado e deslocado para a pilha. Num outro exemplo, se tivermos $A ::= \alpha \bullet B\beta$, onde A e B são não terminais e α, β , qualquer símbolo do alfabeto (Σ^*), sua presença, no topo da pilha indicará que já foi empilhada a cadeia α , sendo esperado a seguir, a cadeia $B\beta$. Isso significa que a entrada corrente poderá ser da forma γw , com $\gamma \in \Sigma^*$ (γ , qualquer símbolo do alfabeto), $w \in T^*$ (w qualquer terminal) e $B\beta \Rightarrow^* \gamma$ (γ pode ser obtido a partir de zero ou mais derivações a partir de $B\beta$).

Obs: intuitivamente, um item indica quanto de uma produção já foi examinado num determinado ponto da análise.

Como construir a tabela de estados

1. Definir uma gramática aumentada
2. Definir duas funções: fechamento e desvio

Se G for uma gramática com símbolo de partida S , então G' , a gramática aumentada para G . Ou seja, é G com um novo símbolo de partida S' .

A aceitação ocorre quando e somente quando o analisador sintático estiver para reduzir através de $S' \rightarrow S$.

Duas funções são necessárias para o cálculo dos estados: fechamento e desvio

Operação Fechamento

- Se I = conjunto de itens para G , então $\text{fechamento}(I)$ é o conjunto de itens construídos a partir de I por 2 regras:
- cada item de I é adicionado ao fechamento
- se $A \rightarrow \alpha \bullet B \beta$ estiver em $\text{fechamento}(I)$ e $B \rightarrow \gamma$ for uma produção, adicionar o item $B \rightarrow \bullet \gamma$ a I , se já não estiver lá.
- Aplica-se essa regra até que não possam ser adicionados novos itens ao $\text{fechamento}(I)$.

Duas funções são necessárias para o cálculo dos estados: fechamento e desvio

Operação Fechamento

- Intuitivamente, $A \rightarrow \alpha \bullet B \beta$ em fechamento(I) indica que, em algum ponto do processo de análise gramatical, esperamos poder ver em seguida na entrada, uma subcadeia derivável a partir de $B\beta$. Se $B \rightarrow \gamma$ for uma produção, também esperamos ver uma subcadeia derivável de γ naquele ponto. Por essa razão também incluímos $B \rightarrow \bullet \gamma$ no fechamento(I).

Duas funções são necessárias para o cálculo dos estados: fechamento e desvio

Exemplo: Dada a gramática de expressões aumentadas:

- $E' ::= E$
- $E ::= E + T \mid T$
- $T ::= T * F \mid F$
- $F ::= (E) \mid \text{id}$

Duas funções são necessárias para o cálculo dos estados: fechamento e desvio

Se I for um conjunto de um item $\{ [E' ::= \bullet E] \}$, então o fechamento(I) contém os itens:

- $E' ::= \bullet E$ (da primeira regra)
- $E ::= \bullet E + T$
- $E ::= \bullet T$
- $T ::= \bullet T * F$
- $T ::= \bullet F$
- $F ::= \bullet (E)$
- $F ::= \bullet id$

(da segunda regra)

$E' ::= E$

$E ::= E + T \mid T$

$T ::= T * F \mid F$

$F ::= (E) \mid id$

Duas funções são necessárias para o cálculo dos estados: fechamento e desvio

Pela regra 1 $E' ::= \bullet E$ é colocado em fechamento(I). Como existe um E imediatamente à direita do ponto, adicionamos, pela regra 2, as produções $E ::= \bullet E + T$ e $E ::= \bullet T$.

- Existe agora um T imediatamente à direita do ponto e, conseqüentemente, adicionamos $F ::= \bullet (E)$ e $F ::= \bullet id$. A partir daí nenhum outro item é colocado no fechamento(I).

Duas funções são necessárias para o cálculo dos estados: fechamento e desvio

Operação Desvio

- $\text{Desvio}(I, X)$, onde I é o conjunto de itens e X é um símbolo gramatical é definida como o fechamento do conjunto de todos os itens $[A \rightarrow \alpha X \bullet \beta]$ tais que $[A \rightarrow \alpha \bullet X \beta]$ esteja em I .

Duas funções são necessárias para o cálculo dos estados: fechamento e desvio

- Intuitivamente: se I for o conjunto de itens válidos para algum prefixo viável γ , então $\text{desvio}(I, X)$ será o conjunto de itens válidos para o prefixo viável γX .

Duas funções são necessárias para o cálculo dos estados: fechamento e desvio

exemplo: se I for o conjunto de dois itens $\{[E' ::= E \bullet], [E ::= E \bullet + T]\}$, então o desvio $(I, +)$ consiste em:

- $E ::= E + \bullet T$
- $T ::= \bullet T * F$
- $T ::= \bullet F$
- $F ::= \bullet (E)$
- $F ::= \bullet id$
- Computamos o desvio $(I, +)$ através do exame dos itens com $+$ imediatamente à direita do ponto. $E' ::= E \bullet$ não é um desses itens, mas $E ::= E \bullet + T$ o é. Movemos, então, o ponto por sobre o $+$ para obtermos $\{E ::= E + \bullet T\}$ e em seguida, realizamos o fechamento do conjunto.

$$E' ::= E$$
$$E ::= E + T \mid T$$
$$T ::= T * F \mid F$$
$$F ::= (E) \mid id$$

A construção de um conjunto de itens

- A construção partirá de um estado inicial, e, aplicando as funções **fechamento** e **desvio**, calculará novos estados, até que não seja possível prosseguir na construção. Uma vez que o número de itens é finito, o número de estados distintos também o é.

A construção de um conjunto de itens

- **Convenção 1:** para a raiz S , acrescenta-se a produção $S' ::= S\#$. O símbolo S' passa a ser a nova raiz.
- **Convenção 2:** toda cadeia de entrada será seguida do símbolo $\#$.

Assim, o primeiro estado **0** deve conter o item $S' ::= \bullet S\#$, e se $S' ::= S\bullet\#$ estiver no topo da pilha quando o símbolo seguinte a ser consultado é $\#$, então a cadeia deverá ser aceita.

Algoritmo para determinar os estados de uma gramática, onde C é uma coleção de estados, ou seja, um conjunto de conjuntos de itens:

- Adota-se o estado $0 = \{\text{fechamento } \{S' ::= \bullet S\# \})\}$ como o valor inicial de C
- Se existe um estado e de C , e um símbolo $X \in \Sigma$ (qualquer elemento da gramática), tais que $e' = \text{desvio}(e, X)$ não é vazio, e e' não está em C , então acrescenta-se e' à coleção C
- O passo2 é repetido até que não possam mais acrescentar estados à coleção C .
- C é a coleção de estados tipo LR (0) da gramática.

- Exemplo: Considere a gramática aumentada

$$E' ::= E\#$$

$$E ::= +EE$$

$$E ::= *EE$$

$$E ::= a$$

$$E ::= b$$

Aplicando-se as construções acima, obtém-se os seguintes estados

$I_0: \quad E' ::= \bullet E \#$

$E ::= \bullet + EE \mid \bullet * EE \mid \bullet a \mid \bullet b$

$\text{goto}(I_0, E) = I_1 = E' ::= E \bullet \#$

$\text{goto}(I_0, +) = I_2 = E ::= + \bullet EE \mid \bullet + EE \mid \bullet * EE \mid \bullet a \mid \bullet b$

$\text{goto}(I_0, *) = I_3 = E ::= * \bullet EE \mid \bullet + EE \mid \bullet * EE \mid \bullet a \mid \bullet b$

$\text{goto}(I_0, a) = I_4 = E ::= a \bullet$

$\text{goto}(I_0, b) = I_5 = E ::= b \bullet$

$\text{goto}(I_2, E) = I_6 = E ::= + E \bullet E \mid \bullet + EE \mid \bullet * EE \mid \bullet a \mid \bullet b$

$\text{goto}(I_3, E) = I_7 = E ::= * E \bullet E \mid \bullet + EE \mid \bullet * EE \mid \bullet a \mid \bullet b$

$\text{goto}(I_6, E) = I_8 = E ::= + EE \bullet$

$\text{goto}(I_7, E) = I_9 = E ::= * EE \bullet$

- Os valores da função desvio fornecem as entradas e_j da tabela. Conforme verificado acima existem estados com itens completos apenas (4,5,8 e 9) ou estados com itens incompletos apenas (0, 1, 2, 3, 6 e 7).
- Esses últimos (os incompletos) indicam que um estado correspondente ao próximo símbolo consultado deve ser deslocado na pilha.

- Os estados constituídos de itens completos indicam que os últimos estados empilhados correspondem a um handle (redutendo) e que, portanto, deve haver redução, independentemente do próximo símbolo.
- Se um estado que está no topo da pilha contém um único item da forma $A ::= \alpha \bullet$, então deve-se aplicar uma redução utilizando a produção $A ::= \alpha$. Assim todas as entradas r_j (j é o número da regra) são obtidas, e colocadas nas entradas dos $\text{Follow}(A)$.

Veja na tabela:

r0 $E' ::= E\#$

r1 $E ::= +EE$

r2 $E ::= *EE$

r3 $E ::= a$

r4 $E ::= b$

I0: $E' ::= \bullet E\#$

$E ::= \bullet +EE \mid \bullet *EE \mid \bullet a \mid \bullet b$

goto(I0,E) = I1 = $E' ::= E\bullet\#$

goto(I0,+) = I2 = $E ::= +\bullet EE \mid \bullet +EE \mid \bullet *EE \mid \bullet a \mid \bullet b$

goto(I0,*) = I3 = $E ::= *\bullet EE \mid \bullet +EE \mid \bullet *EE \mid \bullet a \mid \bullet b$

goto(I0,a) = I4 = $E ::= a\bullet$

goto(I0,b) = I5 = $E ::= b\bullet$

goto(I2,E) = I6 = $E ::= +E\bullet E \mid \bullet +EE \mid \bullet *EE \mid \bullet a \mid \bullet b$

goto(I3,E) = I7 = $E ::= *E\bullet E \mid \bullet +EE \mid \bullet *EE \mid \bullet a \mid \bullet b$

goto(I6,E) = I8 = $E ::= +EE\bullet$

goto(I7,E) = I9 = $E ::= *EE\bullet$

As reduções r = vão nas entradas Follow(X)- Follow(E) = {#,+,* ,a,b}

ação

transição

	+	*	a	b	#	E
0	e2	e3	e4	e5		1
1					a	
2	e2	e3	e4	e5		6
3	e2	e3	e4	e5		7
4	r3	r3	r3	r3	r3	
5	r4	r4	r4	r4	r4	
6	e2	e3	e4	e5		8
7	e2	e3	e4	e5		9
8	r1	r1	r1	r1	r1	
9	r2	r2	r2	r2	r2	

Passo	Pilha	Símbolo Reduzido	cadeia de entrada	ação
0	0		+ * a + b a a #	e2
1	0 + 2		* a + b a a #	e3
2	0 + 2 * 3		a + b a a #	e4
3	0 + 2 * 3 a 4		+ b a a #	r3
4	0 + 2 * 3	E	+ b a a #	e7
5	0 + 2 * 3 E 7		+ b a a #	e2
6	0 + 2 * 3 E 7 + 2		b a a #	e5
7	0 + 2 * 3 E 7 + 2 b 5		a a #	r4
8	0 + 2 * 3 E 7 + 2	E	a a #	e6
9	0 + 2 * 3 E 7 + 2 E 6		a a #	e4
10	0 + 2 * 3 E 7 + 2 E 6 a 4		a #	r3
11	0 + 2 * 3 E 7 + 2 E 6	E	a #	e8
12	0 + 2 * 3 E 7 + 2 E 6 E 8		a #	r1
13	0 + 2 * 3 E 7	E	a #	e9
14	0 + 2 * 3 E 7 E 9		a #	r2
15	0 + 2	E	a #	e6
16	0 + 2 E 6		a #	e4
17	0 + 2 E 6 a 4		#	r3
18	0 + 2 E 6	E	#	e8
19	0 + 2 E 6 E 8		#	r1
20	0	E	#	e1
21	0 E 1		#	

Exercício

- 1) Para a gramática abaixo, crie o conjunto de itens e a tabela de análise SRL e os passos do analisador para reconhecer a sentença

$\text{id} \ \& \ \text{id} \ \vee \ \text{id}$

$E \rightarrow E \vee T$

$E \rightarrow T$

$T \rightarrow T \& F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow \text{id}$