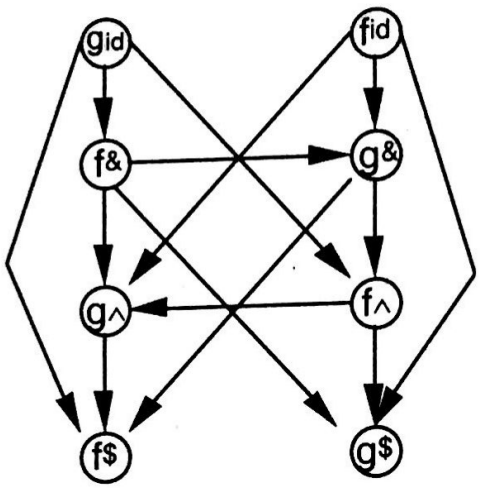


Figura 3.12. Grafo para obter as funções de precedência.

SEMANA 5

3.3.3 Analisadores LR(k)

Os analisadores LR (Left to right with Rightmost derivation) são analisadores redutores eficientes que lêem a sentença em análise da esquerda para a direita e produzem uma derivação mais à direita ao reverso, considerando **k** símbolos sob o cabeçote de leitura. Dentre as vantagens identificadas nesses analisadores, destacam-se:

1) são capazes de reconhecer, praticamente, todas as estruturas sintáticas definidas por gramáticas livres do contexto;

2) o método de reconhecimento LR é mais geral que o de precedência de operadores e que qualquer outro do tipo empilha–reduz e pode ser implementado com o mesmo grau de eficiência;

3) analisadores LR são capazes de descobrir erros sintáticos no momento mais cedo, isto é, já na leitura da sentença em análise.

A principal desvantagem desses analisadores é a dificuldade de implementação dos mesmos, sendo necessário utilizar ferramentas automatizadas na construção da tabela de análise. Há, basicamente, três tipos de analisadores LR:

1) SLR (Simple LR), fáceis de implementar, porém aplicáveis a uma classe restrita de gramáticas;

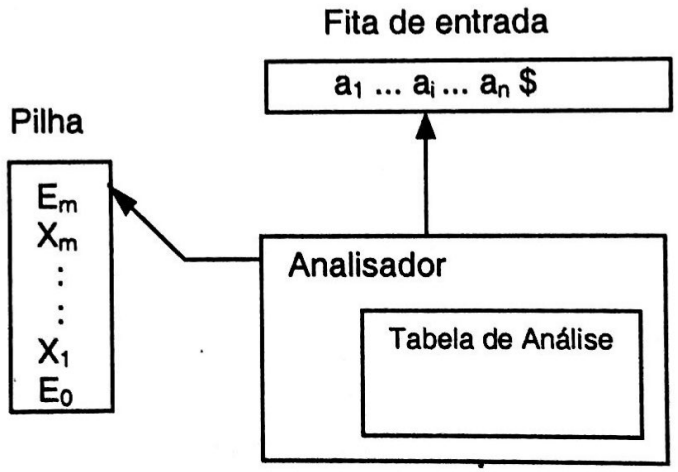
2) LR Canônicos, mais poderosos, podendo ser aplicados a um grande número de linguagens livres do contexto; e

3) LALR (Look Ahead LR), de nível intermediário e implementação eficiente, que funciona para a maioria das linguagens de programação. O YACC gera esse tipo de analisador.

Nesta seção, será apresentada em detalhe a construção de um analisador SLR(1).

Funcionamento dos Analisadores LR

A estrutura genérica de um analisador LR(1) é mostrada na Figura 3.13. A fita de entrada mostra a sentença (a_1 ... a_i ... a_n \$) a ser analisada, e a pilha armazena símbolos da gramática (X_j) intercalados com estados (E_j) do analisador. O símbolo da base da pilha é E_0, estado inicial do analisador. O analisador é dirigido pela Tabela de Análise, cuja estrutura é mostrada na Figura 3.14.



X_j - símbolo da gramática

E_j - estado

Figura 3.13 Estrutura dos analisadores LR

A Tabela de Análise é uma tabela de transição de estados formada por duas partes: a parte AÇÃO contém ações (empilhar, reduzir, aceitar, ou condição de erro) associadas às transições de estados; e a parte TRANSIÇÃO contém transições de estados com relação aos símbolos não-terminais.

	AÇÃO	TRANSIÇÃO
	TERMINAIS	NÃO-TERMINAIS
ESTADOS	empilha reduz aceita erro	estados

Figura 3.14 Estrutura da Tabela de Análise

O analisador funciona basicamente como segue. Seja $\mathbf{E_m}$ o estado do topo da pilha e $\mathbf{a_i}$ o *token* sob o cabeçote de leitura. O analisador consulta a tabela AÇÃO[E_m, a_i], que pode assumir um dos valores:

a) empilha E_x: causa o empilhamento de "a_i E_x";

b) reduz n (onde n é o número da produção $A \rightarrow \beta$): causa o desempilhamento de 2r símbolos, onde $r = |\beta|$, e o empilhamento de "AE_y" onde E_y resulta da consulta à tabela de TRANSIÇÃO[E_{m-r}, A];[6]

c) aceita: o analisador reconhece a sentença como válida;

d) erro: o analisador pára a execução, identificando um erro sintático.

O funcionamento do analisador pode ser entendido, considerando as transformações que ocorrem na pilha e na fita de entrada para cada ação. No texto que segue, consideram-se as configurações da pilha e da fita representadas por pares da forma (<...pilha...>, <...fita...>). A configuração inicial de um analisador LR é:

(<E_0>, <a_1 a_2 ... a_n \$>)

Considerando que a configuração atual é como segue:

(<E_0 X_1 E_1 X_2 E_2 ... X_m E_m>, <a_i a_{i+1} ... a_n \$>)

tem-se que a configuração resultante após cada ação é:

AÇÃO[E_m, a_i] = empilha X :

(<E_0 X_1 E_1 X_2 E_2 ... X_m E_m a_i X>, <a_{i+1} ... a_n \$>)

AÇÃO[E_m, a_i] = reduz $A \rightarrow \beta$:

(<E_0 X_1 E_1 X_2 E_2 ... X_{m-r} E_{m-r} A E_y >, <a_i a_{i+1} ... a_n \$>)

[6] E_{m-r} é o estado que está no topo da pilha logo após a operação de redução; após a transição, as 3 posições mais ao topo irão conter E_{m-r} A E_y.

sendo $|\beta| = r$ e TRANSIÇÃO$[E_{m-r}, A] = E_y$. Nesse caso, são desempilhados 2r símbolos, para depois ser empilhado "A E_y".

EXEMPLO 3.20 Movimentos de um analisador SLR(1).

Considerando a gramática abaixo, que gera expressões lógicas, a Figura 3.15 mostra a tabela de análise SLR(1) e os passos do analisador para reconhecer a sentença **id&id∨id**.

1) E → E ∨ T
2) E → T
3) T → T & F
4) T → F
5) F → (E)
6) F → id

	AÇÃO						TRANSIÇÃO		
	id	∨	&	(	)	$	E	T	F
0	e5			e4			1	2	3
1		e6				AC			
2		r2	e7		r2	r2			
3		r4	r4		r4	r4			
4	e5			e4			8	2	3
5		r6	r6		r6	r6			
6	e5			e4				9	3
7	e5			e4					10
8		e6			e11				
9		r1	e7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Pilha	Entrada	Ação/Transição
0	id & id ∨ id $	e5: empilha id 5
0 id 5	& id ∨ id $	r6: reduz F→id TRANSIÇÃO[0 F]
0 F 3	& id ∨ id $	r4: reduz T→F TRANSIÇÃO[0 T]
0 T 2	& id ∨ id $	e7: empilha & 7
0 T 2 & 7	id ∨ id $	e5: empilha id 5
0 T 2 & 7 id 5	∨ id $	r6: reduz F→id TRANSIÇÃO[7 F]
0 T 2 & 7 F 10	∨id $	r3: reduz T→T&F TRANSIÇÃO[0 T]
0 T 2	∨id $	r2: reduz E→T TRANSIÇÃO[0 E]
0 E 1	∨id $	e6: empilha ∨ 6
0 E 1 ∨ 6	id $	e5: empilha id 5

(continuação)

Pilha	Entrada	Ação/Transição
0 E 1 ∨ 6 id 5	$	r6: reduz F→id TRANSIÇÃO[6 F]
0 E 1 ∨ 6 F 3	$	r4: reduz T→F TRANSIÇÃO[6 T]
0 E 1 ∨ 6 T 9	$	r2: reduz E→E∨T TRANSIÇÃO[0 E]
0 E 1	$	ACEITA!

Figura 3.15 Tabela de análise e passos do analisador

exercício
analisar a cadeia: id & (id ∨ id)$

Construção de Analisadores SLR

A construção da tabela de controle para analisadores SLR baseia-se no que se denomina *Conjunto Canônico de Itens LR(0).*

Um *item LR(0),* para uma gramática G, é uma produção com um ponto em alguma posição do lado direito. O ponto é uma indicação de até onde uma produção já foi analisada no processo de reconhecimento. Por exemplo, a produção $A \rightarrow X\ Y\ Z$ origina quatro itens:

$A \rightarrow .XYZ$

$A \rightarrow X.YZ$

$A \rightarrow XY.Z$

$A \rightarrow XYZ.$

enquanto a produção $A \rightarrow \varepsilon$ gera apenas um item:

$A \rightarrow .$

A construção do Conjunto Canônico de Itens LR(0) requer duas operações:

1) acrescentar à gramática a produção $S' \rightarrow S$ (onde S é o símbolo inicial da gramática),
2) computar as funções *closure* e *goto* para a nova gramática.

Cálculo da função closure(I)

Itens validos

Se I é um conjunto de itens LR(0) para G, então o conjunto de itens *closure(I)* é construído a partir de I pelas regras:

1) Todo item em I pertence a *closure*(I);
2) Se $A \rightarrow \alpha.X\beta$ está no conjunto *closure*(I) e $X \rightarrow \gamma$ é uma produção, então adicione $X \rightarrow .\gamma$ ao conjunto.

A regra 2 aumenta o conjunto com as produções dos não–terminais que aparecem com um ponto no lado esquerdo.

EXEMPLO 3.21 Cálculo da função closure(I).

Para as produções:

$S \rightarrow a \mid [\ L\ ]$

$L \rightarrow L\ ;\ S \mid S$

e para I = { $S \rightarrow [\ .\ L\ ]$ }, o conjunto *closure*(I) é o seguinte:

closure({$S \rightarrow [\ .\ L\ ]$}) = {$S \rightarrow [\ .\ L\ ]$, $L \rightarrow .\ L\ ;\ S$, $L \rightarrow .\ S$, $S \rightarrow .\ a$, $S \rightarrow .\ [\ L\ ]$}

Cálculo da função goto(I,X) Transições

Informalmente, *goto*(I,X), "avanço do ponto sobre X em I", consiste em coletar as produções com ponto no lado esquerdo de X, passar o ponto para a direita de X, e obter a função *closure* desse conjunto.

Formalmente, para X símbolo terminal ou não–terminal da gramática, *goto*(I,X) é a função *closure* do conjunto dos itens $A \rightarrow \alpha X.\beta$, tais que $A \rightarrow \alpha.X\beta$ pertence a I.

EXEMPLO 3.22 Cálculo da função goto(I,X).

Considerando o conjunto I = {$S \rightarrow [\ L\ .\ ]$, $L \rightarrow L\ .\ ;\ S$}, o cálculo de *goto*(I, ;) resulta em:

goto(I, ;) = { $L \rightarrow L\ ;\ .\ S$, $S \rightarrow .\ a$, $S \rightarrow .\ [\ L\ ]$}

❑

Algoritmo para obter o Conjunto Canônico de Itens LR(0)

Para uma gramática G, o Conjunto Canônico de Itens LR(0), referido por C, é obtido como segue.

Algoritmo:

Inicialização :

C = { I_0 = *closure* ({ $S' \rightarrow .\ S$ }) } /* os elementos de C serão conjuntos */

Repita :

Para cada conjunto I em C e X símbolo de G, tal que *goto*(I,X) ≠ ϕ

adicione *goto*(I,X) a C

até que todos os conjuntos tenham sido adicionados a C.

Continua no arquivo 'notas de aula' com nova GLC: construção itens válidos e transições

Construção da Tabela de Análise SLR

Dada uma gramática G, obtém-se G', aumentando G com a produção S'→S, onde S é o símbolo inicial de G. A partir de G', determina-se o conjunto canônico C. Finalmente, constróem-se as tabelas AÇÃO e TRANSIÇÃO conforme o algoritmo abaixo.

Algoritmo para construir a tabela SLR para G

Entrada: O conjunto C para G'.

Resultado: A tabela de análise SLR para G' (se ela existir).

Método: Seja C = { I_0, I_1, ..., I_n }. Os estados do analisador são 0, 1, ..., n (0 é o estado inicial). A linha i da tabela é construída a partir do conjunto I_i, como segue.

As ***ações*** do analisador para o estado i são determinadas usando as regras:

1) se *goto*(I_i,a) = I_j, então faça AÇÃO[i, a] = empilha j;

2) se $A \rightarrow \alpha.$ está em I_i, então para todo a em FOLLOW(A), faça AÇÃO[i, a] = reduz n, sendo n o número da produção $A \rightarrow \alpha$.

3) se $S' \rightarrow S.$ está em I_i, então faça AÇÃO[i, $] = aceita.

Se ações conflitantes são geradas pelas regras acima, então a gramática não é SLR(1).

As ***transições*** para o estado i são construídas, usando a regra:

4) se *goto*(I_i, A) = I_j, então TRANSIÇÃO(i, A) = j;

As entradas não definidas pelas regras acima correspondem a situações de erro.

❑

Definição 3.14 Gramática SLR(1).

Uma gramática é dita SLR(1) se, a partir dela, pode-se construir uma tabela de análise SLR(1). Toda gramática SLR(1) é não ambígua, porém existem gramáticas não–ambíguas que não são SLR(1).

EXEMPLO 3.23 Cálculo da tabela SLR.

A seguir, é calculado o conjunto canônico de itens para a gramática que gera listas:

S' → S (produção adicionada)

1) S → a

2) S → [L]

3) L → L ; S

4) L → S

	F	Follow
S	a, [	$,], ;
L	a, [	], ;

Cálculo dos Conjuntos LR(0):

$I_0 = \{ S' \rightarrow .S \ , \ S \rightarrow .a \ , \ S \rightarrow .[\ L\ ] \}$

$goto(I_0, S) = I_1 = \{ S' \rightarrow S. \}$

$goto(I_0, a) = I_2 = \{ S \rightarrow a. \}$

$goto(I_0, [\ ) = I_3 = \{ S \rightarrow [.L\ ], \ L \rightarrow .L\ ;S, \ L \rightarrow .S, \ S \rightarrow .a, \ S \rightarrow .[\ L\ ] \}$

$goto(I_3, L) = I_4 = \{ S \rightarrow [L.\ ] \ , \ L \rightarrow L.\ ;S \}$

$goto(I_3, S) = I_5 = \{ L \rightarrow S. \}$

$goto(I_3, a) = I_2$

$goto(I_3, [\ ) = I_3$

$goto(I_4,]\ ) = I_6 = \{ S \rightarrow [L\ ]. \}$

$goto(I_4, ;) = I_7 = \{ L \rightarrow L\ ; .S \ , \ S \rightarrow .a \ , \ S \rightarrow .[\ L\ ] \}$

$goto(I_7, S) = I_8 = \{ L \rightarrow L\ ; S. \}$

$C = \{I_0, I_1, \ldots, I_{11}\}$, e a tabela SLR resultante é a apresentada na Figura 3.16.

	AÇÃO					TRANSIÇÃO	
	a	[	]	;	$	S	L
0	e2	e3				1	
1					AC		
2			r1	r1	r1		
3	e2	e3				5	4
4			e6	e7			
5			r4	r4			
6			r2	r2	r2		
7	e2	e3				8	
8			r3	r3			

Figura 3.16 Tabela SLR

❑

Convém observar que a tabela SLR é uma representação eficiente do autômato de pilha que reconhece a linguagem. O topo da pilha contém sempre o estado atual do autômato. Dado o estado atual e o *token* de entrada, a tabela indica a ação a ser executada. No caso da ação ser uma redução, a tabela também indica o próximo estado a ser assumido pelo autômato. As entradas em branco correspondem a situações de erro.