

Exame - Lista 2 - CES41 Compiladores

14 de Julho de 2019

Prof. Dr. Fábio Carneiro Mokarzel

Igor Bragaia¹

¹Aluno de Graduação em Engenharia do Instituto Tecnológico de Aeronáutica (ITA).



E-mail: igor.bragaia@gmail.com

Questão 1

Seja a gramática do exemplo da Seção 5.5.3 do Capítulo V dos Slides Teóricos de CES-41 e as tabelas de ações e de transições do analisador LR, no mesmo exemplo. Fazer uma tabela de execução, no mesmo analisador, simulando a análise da seguinte sentença: $id * ((id + id) * ((id + id) * id)) \$$

Resolução

Implementou-se um código em Python que constrói a tabela de execução de acordo com o algoritmo descrito nos slides. O código pode ser consultado nos arquivos anexos ou no link

<https://github.com/igorbragaia/CES-41/blob/master/list2/ex1/ex1.py>

O resultado obtido foi

\$ 0	$id * ((id + id) * ((id + id) * id)) \$$	d 5	
\$ 0 (5, 'id')	$* ((id + id) * ((id + id) * id)) \$$	r 6 (F -> id)	Goto (0, F) = 3
\$ 0 (3, 'F')	$* ((id + id) * ((id + id) * id)) \$$	r 4 (T -> F)	Goto (0, T) = 2
\$ 0 (2, 'T')	$* ((id + id) * ((id + id) * id)) \$$	d 7	
\$ 0 (2, 'T') (7, '*')	$((id + id) * ((id + id) * id)) \$$	d 4	
\$ 0 (2, 'T') (7, '*') (4, '(')	$(id + id) * ((id + id) * id) \$$	d 4	
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(')	$id + id) * ((id + id) * id) \$$	d 5	
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (5, 'id')	$+ id) * ((id + id) * id) \$$	r 6 (F -> id)	Goto (4, F) = 3
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (3, 'F')	$+ id) * ((id + id) * id) \$$	r 4 (T -> F)	Goto (4, T) = 2
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (2, 'T')	$+ id) * ((id + id) * id) \$$	r 2 (E -> T)	Goto (4, E) = 8
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (8, 'E')	$+ id) * ((id + id) * id) \$$	d 6	
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (8, 'E') (6, '+')	$id) * ((id + id) * id) \$$	d 5	
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (8, 'E') (6, '+') (5, 'id')	$) * ((id + id) * id) \$$	r 6 (F -> id)	Goto (6, F) = 3
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (8, 'E') (6, '+') (3, 'F')	$) * ((id + id) * id) \$$	r 4 (T -> F)	Goto (6, T) = 9
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (8, 'E') (6, '+') (9, 'T')	$) * ((id + id) * id) \$$	r 1 (E -> E+T)	Goto (4, E) = 8
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (8, 'E')	$) * ((id + id) * id) \$$	d 11	
\$ 0 (2, 'T') (7, '*') (4, '(') (4, '(') (8, 'E') (11, ')')	$* ((id + id) * id) \$$	r 5 (F -> (E))	Goto (4, F) = 3

\$ 0 (2, 'T') (7, '**') (4, 'I') (3, 'F')	* ((id + id) * id)) \$	r 4 (T -> F)	Goto (4, T) = 2
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T')	* ((id + id) * id)) \$	d 7	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**')	((id + id) * id)) \$	d 4	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I')	(id + id) * id)) \$	d 4	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I')	id + id) * id)) \$	d 5	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (5, 'id')	+ id) * id)) \$	r 6 (F -> id)	Goto (4, F) = 3
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (3, 'F')	+ id) * id)) \$	r 4 (T -> F)	Goto (4, T) = 2
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (2, 'T')	+ id) * id)) \$	r 2 (E -> T)	Goto (4, E) = 8
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (8, 'E')	+ id) * id)) \$	d 6	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (8, 'E') (6, '+')	id) * id)) \$	d 5	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (8, 'E') (6, '+') (5, 'id')) * id)) \$	r 6 (F -> id)	Goto (6, F) = 3
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (8, 'E') (6, '+') (3, 'F')) * id)) \$	r 4 (T -> F)	Goto (6, T) = 9
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (8, 'E') (6, '+') (9, 'T')) * id)) \$	r 1 (E -> E+T)	Goto (4, E) = 8
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (8, 'E')) * id)) \$	d 11	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (4, 'I') (8, 'E') (11, 'I')	* id)) \$	r 5 (F -> (E))	Goto (4, F) = 3
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (3, 'F')	* id)) \$	r 4 (T -> F)	Goto (4, T) = 2
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (2, 'T')	* id)) \$	d 7	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**')	id)) \$	d 5	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (5, 'id'))) \$	r 6 (F -> id)	Goto (7, F) = 10
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (10, 'F'))) \$	r 3 (T -> T*F)	Goto (4, T) = 2
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (2, 'T'))) \$	r 2 (E -> T)	Goto (4, E) = 8
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (8, 'E'))) \$	d 11	
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (4, 'I') (8, 'E') (11, 'I')) \$	r 5 (F -> (E))	Goto (7, F) = 10
\$ 0 (2, 'T') (7, '**') (4, 'I') (2, 'T') (7, '**') (10, 'F')) \$	r 3 (T -> T*F)	Goto (4, T) = 2

\$ 0 (2, 'T') (7, '**') (4, '(') (2, 'T')) \$	r 2 (E -> T)	Goto (4, E) = 8
\$ 0 (2, 'T') (7, '**') (4, '(') (8, 'E')) \$	d 11	
\$ 0 (2, 'T') (7, '**') (4, '(') (8, 'E') (11, ')')	\$	r 5 (F -> (E))	Goto (7, F) = 10
\$ 0 (2, 'T') (7, '**') (10, 'F')	\$	r 3 (T -> T*F)	Goto (0, T) = 2
\$ 0 (2, 'T')	\$	r 2 (E -> T)	Goto (0, E) = 1
\$ 0 (1, 'E')	\$	aceitar	

Questão 2

Construir para a gramática a seguir as tabelas de ações e de transições de um analisador sintático SLR. Dizer se a referida gramática é SLR ou não e justificar a resposta. Mostrar todo o desenvolvimento usado, ou seja, os autômatos não-determinístico e determinístico obtidos, a tabela dos seguintes de todos os não-terminais e a tabela com estados contendo os itens de final de todas as produções da gramática. Não é necessário programar, porém não é proibido.

Produções da gramática:

E	E + T T
T	T * F F
F	P @ F P
P	(E) a a (L)
L	L , E E

Obs.: o não-terminal E é o símbolo inicial da gramática.

Resolução

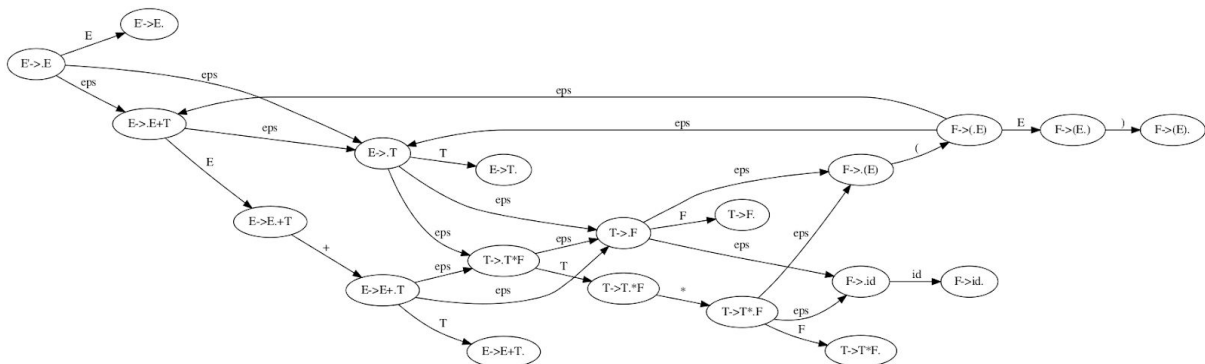
Implementou-se um código em Python que a partir das produções da gramática, cria programaticamente um autômato não-determinístico e então lhe reduz para um autômato determinístico. Em seguida, fazendo uso dos Seguintes (obtidos por meio do código desenvolvido na lista 1), o script em Python segue a implementação dos slides e obtém a tabela de ações e transições pedida. O código pode ser consultado nos arquivos anexos ou no link

<https://github.com/igorbragaia/CES-41/blob/master/list2/ex2/ex2.py>

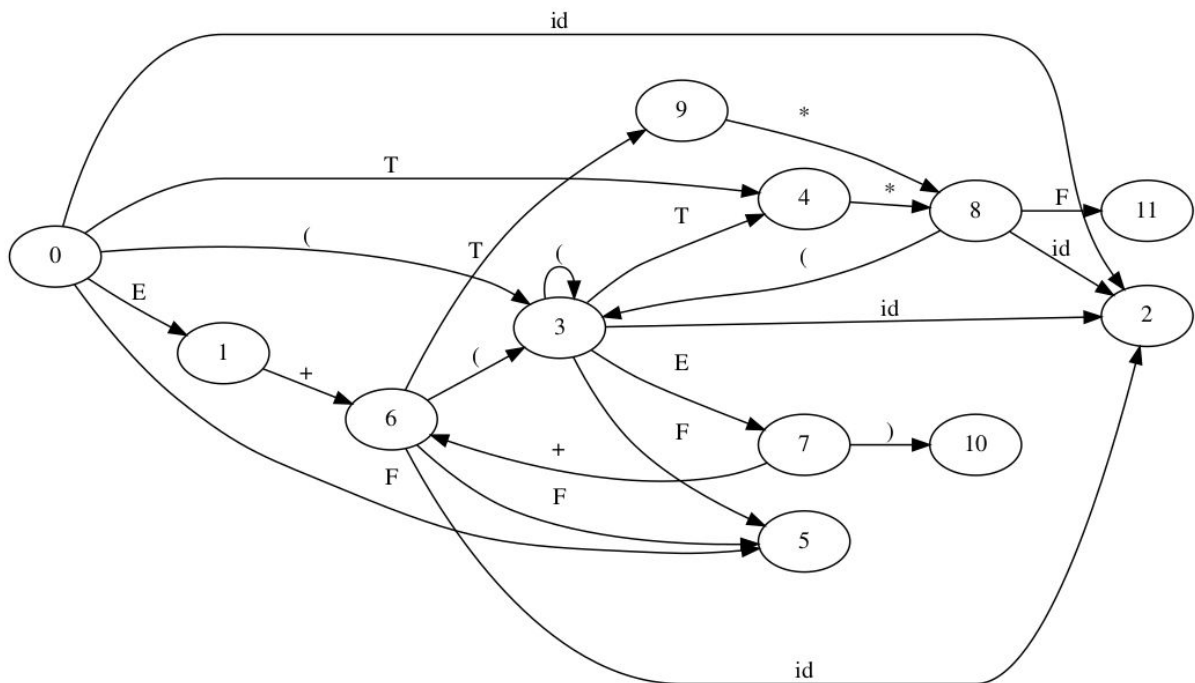
Para o exemplo do slide da aula 5-c, temos um resultado idêntico ao slide, embora com nomenclatura diferente nos nós. Veja a seguir, por exemplo, para a gramática dos slides

E	E + T T
T	T * F F
F	(E) id

O autômato finito não determinístico é



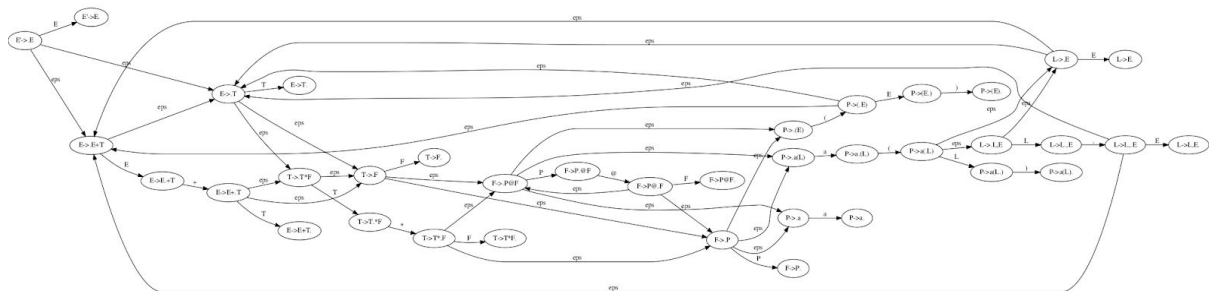
O autômato finito determinístico, por sua vez, é



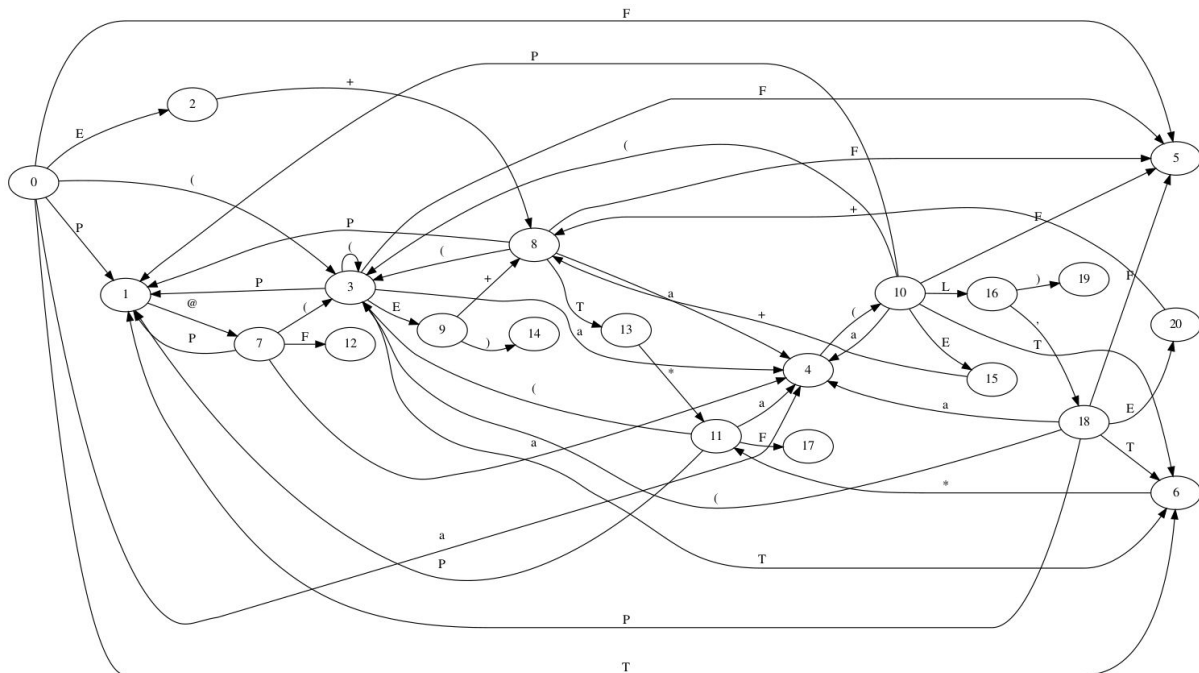
Para a questão da lista, dada a gramática

E **E + T | T**
T **T * F | F**
F **P @ F | P**
P **(E) | a | a (L)**
L **L , E | E**

O autômato finito não determinístico é (veja em melhor resolução no pdf em anexo ou https://github.com/igorbragaia/CES-41/blob/master/list2/ex2/questao_lista/questao_lista_afnd.gv.pdf)



O autômato finito determinístico, por sua vez, é (veja em melhor resolução no pdf em anexo ou https://github.com/igorbragaia/CES-41/blob/master/list2/ex2/questao_lista/questao_lista_afd.gv.pdf)



Usando o código em Python, desenvolvido na questão 3 da lista 1, para gramática em questão, temos que os Seguintes para os não terminais dessa gramática são

Seguintes	
'E'	['\$', '+', ')', ',']
'F'	['*', '\$', '+', ')', ',']
'L'	[')', ',']
'P'	['@', '*', '\$', '+', ')', ',']
'T'	['*', '\$', '+', ')', ',']

Por fim, seguindo a implementação apresentada no slide para obtenção da tabela de ações e transições de um analisador SLR, temos que a tabela de ações e de transições é

	@	+	*	()	a	,	\$	E	T	F	P	L
0				d 3		d 4			2	6	5	1	
1	d 7	r 6	r 6		r 6		r 6	r 6					
2		d 8						act					
3				d 3		d 4			9	6	5	1	
4	r 8	r 8	r 8	d 10	r 8		r 8	r 8					
5		r 4	r 4		r 4		r 4	r 4					
6		r 2	d 11		r 2		r 2	r 2					
7				d 3		d 4					12	1	
8				d 3		d 4				13	5	1	
9		d 8			d 14								
10				d 3		d 4			15	6	5	1	16
11				d 3		d 4					17	1	
12		r 5	r 5		r 5		r 5	r 5					
13		r 1	d 11		r 1		r 1	r 1					
14	r 7	r 7	r 7		r 7		r 7	r 7					
15		d 8			r 11		r 11						
16					d 19		d 18						
17		r 3	r 3		r 3		r 3	r 3					
18				d 3		d 4			20	6	5	1	
19	r 9	r 9	r 9		r 9		r 9	r 9					
20		d 8			r 10		r 10						

Por fim, dado que durante a construção da tabela acima não houve nenhuma entrada com mais de uma ação possível, temos que a gramática é SLR.

Questão 3

Escrever o código intermediário para o programa escrito na linguagem COMP-ITA 2019. Tal código deve ser livre de quádruplas de operadores NOP e de operadores de atribuição desnecessários em determinadas situações. Não usar o programa desenvolvido nos laboratórios.

Resolução

Obs.: A fim de simplificar a construção da tabela, optei por escrever os próprios símbolos ao invés de escrevê-los por extenso. Por exemplo, escrevi “+” ao invés de MAISOP nas quádruplas.

PROGRAMA	rótulo	QUÁDRUPLAS			
program MatrizTransposta {	1	OPENMOD	(MODULO, MatrizTransposta)	(IDLE)	(IDLE)
global: int A[10,10], n;	2	PARAM	(VAR, A)	(IDLE)	(IDLE)
	3	PARAM	(VAR, n)	(IDLE)	(IDLE)
functions:					
void LerMatriz () {	4	OPENMOD	(MODULO, LerMatriz)	(IDLE)	(IDLE)
local: int i, j;	5	PARAM	(VAR, i)	(IDLE)	(IDLE)
	6	PARAM	(VAR, j)	(IDLE)	(IDLE)
statements:					
write ("Dimensao da matriz quadrada: ");	7	PARAM	(CADEIA, "Dimensao da matriz quadrada:")	(IDLE)	(IDLE)
	8	WRITEOP	(INT, 1)	(IDLE)	(IDLE)
do read (n); while (n < 0);	9	PARAM	(VAR, n)	(IDLE)	(IDLE)
	10	READOP	(INT, 1)	(IDLE)	(IDLE)
	11	<	(VAR, n)	(INT, 0)	(VAR, ##1)
	12	JTOP	(VAR, ##1)	(IDLE)	(RÓTULO, 9)
write ("\nElementos da matriz: \n");	13	PARAM	(CADEIA, "Elementos da matriz: \n")	(IDLE)	(IDLE)
	14	WRITEOP	(INT, 1)	(IDLE)	(IDLE)
for (i <- 0; i <= n-1; i <- i+1) for (j <- 0; j <= n-1; j <- j+1) read (A[i,j]);	15	ATRIBOP	(INT, 0)	(IDLE)	(VAR, i)
	16	ATRIBOP	(INT, 0)	(IDLE)	(VAR, j)
	17	IND	(VAR, i)	(IDLE)	(IDLE)
	18	IND	(VAR, j)	(IDLE)	(IDLE)
	19	INDEX	(VAR, A)	(INT, 2)	(VAR, ##2)
	20	PARAM	(VAR, ##3)	(IDLE)	(IDLE)
	21	READOP	(INT, 1)	(IDLE)	(IDLE)
	22	ATRIBPONT	(VAR, ##3)	(IDLE)	(VAR, ##2)
	23	+	(VAR, j)	(INT, 1)	(VAR, ##4)
	24	ATRIBOP	(VAR, ##4)	(IDLE)	(VAR, j)
	25	-	(VAR, n)	(INT, 1)	(VAR, ##5)
	26	<=	(VAR, j)	(VAR, ##5)	(VAR, ##6)
	27	JTOP	(VAR, ##6)	(IDLE)	(RÓTULO, 17)
	28	+	(VAR, i)	(INT, 1)	(VAR, ##7)

	29	ATRIBOP	(VAR, ##7)	(IDLE)	(VAR, i)
	30	<=	(VAR, i)	(VAR, ##5)	(VAR, ##8)
	31	JTOP	(VAR, ##8)	(IDLE)	(RÓTULO, 16)
}	32	RETURNOP	(IDLE)	(IDLE)	(IDLE)
void EscreverMatriz () {	33	OPENMOD	(MODULO, EscreverMatriz)	(IDLE)	(IDLE)
local: int i, j;	34	PARAM	(VAR,i)	(IDLE)	(IDLE)
	35	PARAM	(VAR, j)	(IDLE)	(IDLE)
statements:					
if (n <= 0) write ("Matriz nula");	36	<=	(VAR, n)	(INT,0)	(VAR, ##1)
	37	JFOP	(VAR, ##1)	(IDLE)	(RÓTULO, 41)
	38	PARAM	(CADEIA, "Matriz nula")	(IDLE)	(IDLE)
	39	WRITEOP	(INT,1)	(IDLE)	(IDLE)
	40	JUMPOP	(IDLE)	(IDLE)	(RÓTULO, 60)
else					
for (i <- 0; i <= n-1; i <- i+1) { for (j <- 0; j <= n-1; j <- j+1) write (A[i,j]); write ("\n"); }	41	ATRIBOP	(INT,0)	(IDLE)	(VAR,i)
	42	ATRIBOP	(INT,0)	(IDLE)	(VAR,j)
	43	IND	(VAR,i)	(IDLE)	(IDLE)
	44	IND	(VAR,j)	(IDLE)	(IDLE)
	45	INDEX	(VAR,A)	(INT,2)	(VAR,##2)
	46	CONTAPONTOP	(VAR, ##2)	(IDLE)	(VAR, ##3)
	47	PARAM	(VAR, ##3)	(IDLE)	(IDLE)
	48	WRITEOP	(INT, 1)	(IDLE)	(IDLE)
	49	+	(VAR, j)	(INT, 1)	(VAR, ##4)
	50	ATRIB	(VAR, ##4)	(IDLE)	(VAR, j)
	51	-	(VAR, n)	(INT,1)	(VAR, ##5)
	52	<=	(VAR, j)	(VAR, ##5)	(VAR, ##6)
	53	JTOP	(VAR, ##6)	(IDLE)	(RÓTULO, 43)
	54	PARAM	(CADEIA, "\n")	(IDLE)	(IDLE)
	55	WRITEOP	(INT,1)	(IDLE)	(IDLE)
	56	+	(VAR, i)	(INT, 1)	(VAR, ##7)
	57	ATRIBOP	(VAR, ##7)	(IDLE)	(VAR, i)
	58	<=	(VAR, i)	(VAR, ##5)	(VAR, ##8)
	59	JT	(VAR, ##8)	(IDLE)	(RÓTULO, 42)
}	60	RETURNOP	(IDLE)	(IDLE)	(IDLE)
void Trocar (int i, int j) {	61	OPENMOD	(MODULO, Trocar)	(IDLE)	(IDLE)
local: int aux;	62	PARAM	(VAR, aux)	(IDLE)	(IDLE)
statements:					

aux <- A[i,j];	63	IND	(VAR, i)	(IDLE)	(IDLE)
	64	IND	(VAR, j)	(IDLE)	(IDLE)
	65	INDEX	(VAR, A)	(INT,2)	(VAR, ##1)
	66	CONTAPONTOP	(VAR, ##1)	(IDLE)	(VAR, ##3)
	67	ATRIB	(VAR, ##3)	(IDLE)	(VAR, aux)
A[i,j] <- A[j,i];	68	IND	(VAR, j)	(IDLE)	(IDLE)
	69	IND	(VAR, i)	(IDLE)	(IDLE)
	70	INDEX	(VAR, A)	(INT,2)	(VAR, ##4)
	71	CONTAPONTOP	(VAR, ##4)	(IDLE)	(VAR, ##5)
	72	IND	(VAR, i)	(IDLE)	(IDLE)
	73	IND	(VAR, j)	(IDLE)	(IDLE)
	74	INDEX	(VAR, A)	(INT, 2)	(VAR, ##6)
	75	ATRIBPONTOP	(VAR, ##5)	(IDLE)	(VAR, ##6)
A[j,i] <- aux;	76	IND	(VAR, j)	(IDLE)	(IDLE)
	77	IND	(VAR, i)	(IDLE)	(IDLE)
	78	INDEX	(VAR, A)	(INT,2)	(VAR, ##7)
	79	ATRIBPONTOP	(VAR, aux)	(IDLE)	(VAR, ##7)
}	80	RETURNOP	(IDLE)	(IDLE)	(IDLE)
main {	81	OPENMOD	(MODULO, Main)	(IDLE)	(IDLE)
local: int i, j;	82	PARAM	(VAR, i)	(IDLE)	(IDLE)
	83	PARAM	(VAR, j)	(IDLE)	(IDLE)
statements:					
/*Leitura e escrita da matriz original*/					
call LerMatriz();	84	CALLOP	(FUNCAO, LerMatriz)	(INT, 0)	(IDLE)
write ("\nMatriz original:\n\n");	85	PARAM	(CADEIA," \nMatriz original:\n\n")	(IDLE)	(IDLE)
	86	WRITEOP	(INT, 1)	(IDLE)	(IDLE)
call EscreverMatriz();	87	CALLOP	(FUNCAO, EscreverMatriz)	(INT, 0)	(IDLE)
/*Transformacao da matriz em sua transposta*/					
if (n > 0)	88	>	(VAR, n)	(VAR, 0)	(VAR, ##1)
	89	JFOP	(VAR, ##1)	(IDLE)	(RÓTULO, 106)
for (i <- 0; i <= n-2; i <- i+1) for (j <- i+1; j <= n-1; j <- j+1) call Trocar (i, j);	90	ATRIBOP	(INT, 0)	(IDLE)	(VAR,i)
	91	+	(VAR, i)	(INT,1)	(VAR, ##2)
	92	ATRIBOP	(VAR, ##2)	(IDLE)	(VAR, j)
	93	PARAM	(VAR, i)	(IDLE)	(IDLE)
	94	PARAM	(VAR, j)	(IDLE)	(IDLE)
	95	CALLOP	(FUNCAO, Trocar)	(INT,2)	(IDLE)

	96	+	(VAR, j)	(INT, 1)	(VAR, ##3)
	97	ATRIBOP	(VAR, ##3)	(IDLE)	(VAR, j)
	98	-	(VAR, n)	(INT, 1)	(VAR, ##4)
	99	<=	(VAR, j)	(VAR, ##4)	(VAR, ##5)
	100	JTOP	(VAR, ##5)	(IDLE)	(RÓTULO, 93)
	101	+	(VAR, i)	(INT, 1)	(VAR, ##6)
	102	ATRIBOP	(VAR, ##6)	(IDLE)	(VAR, i)
	103	-	(VAR, n)	(INT, 2)	(VAR, ##7)
	104	<=	(VAR, i)	(VAR, ##7)	(VAR, ##8)
	105	JTOP	(VAR, ##8)	(IDLE)	(RÓTULO, 91)
/*Escrita da matriz transposta */					
write ("\nMatriz transposta:\n\n");	106	PARAM	(CADEIA, "\nMatriz transposta:\n\n")	(IDLE)	(IDLE)
	107	WRITEOP	(INT, 1)	(IDLE)	(IDLE)
call EscreverMatriz();	108	CALLOP	(FUNCAO, EscreverMatriz)	(INT, 0)	(IDLE)
}	109	RETURNOP	(IDLE)	(IDLE)	(IDLE)
}	110	RETURNOP	(IDLE)	(IDLE)	(IDLE)