PCS 3216 – Sistemas de Programação

Aula 21

Roteiro de Construção de Compilador

Exemplo completo – parte 2

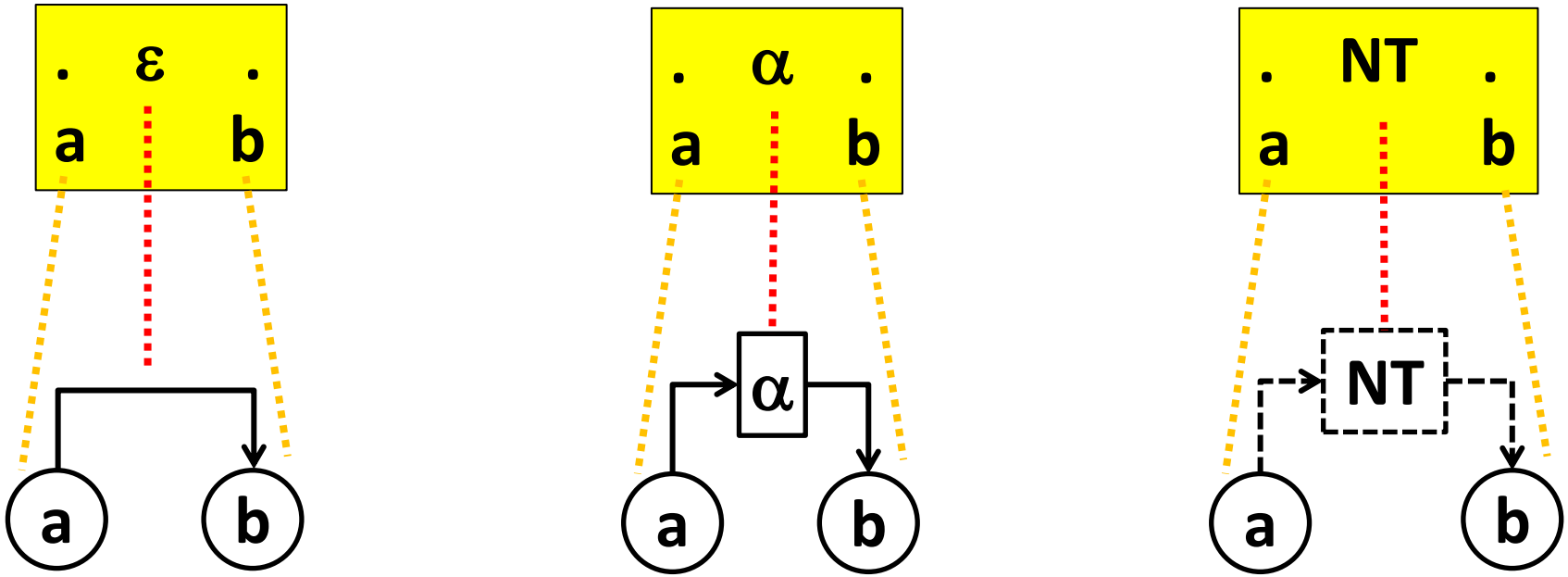
REGRAS PARA A OBTENÇÃO DO AUTÔMATO

- Inicialmente, é importante ter em mente a forma como as regras interativas da Notação de Wirth devem ser interpretadas na gramática da linguagem cujo processador estamos construindo.
- Há os seguintes casos a considerar:
  - Átomos, vazio, não-terminal (transições simples)
  - Concatenação
  - União de conjuntos
    - Agrupamento obrigatório (entre parênteses)
    - Agrupamento optativo (entre colchetes)
  - Agrupamento repetitivo – fecho de Kleene (entre chaves)
- Nos slides seguintes, vamos mostrar, para cada um desses casos, a correspondência com o autômato correspondente:
  - Através da associação de estados às regras da gramática
  - Através da associação de transições entre pares de estados

ELEMENTOS BÁSICOS:
VAZIO, ÁTOMO E NÃO-TERMINAL

Vazio ε, átomo α, não-teminal NT

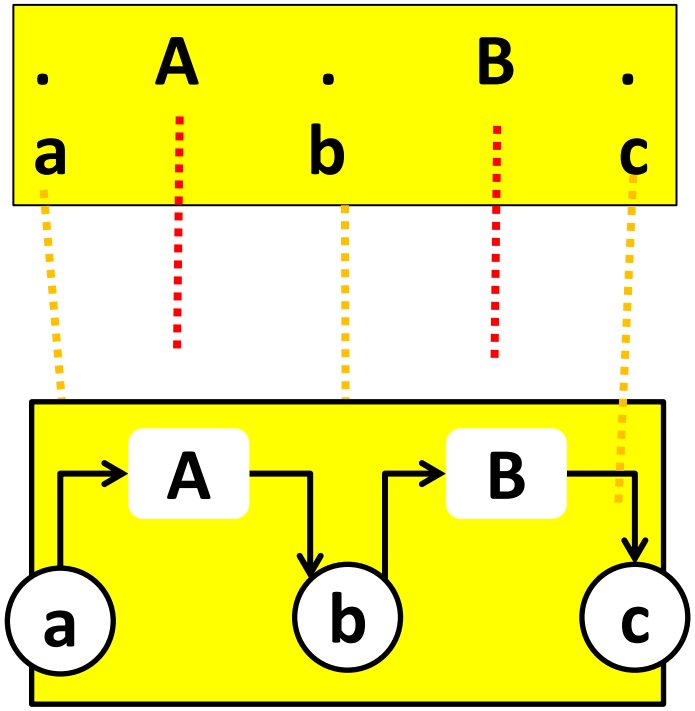
Se dois estados da fórmula sintática em uma regra estiverem justapostos, a transição de um estado para o outro é feita em vazio.
O mesmo ocorre quando na fórmula aparecer explicitamente um ε .
Em qualquer caso, no autômato deverá ser incluída uma transição vazia do primeiro estado para o segundo.
Átomos α e **não-terminais NT** são tratados da mesma forma, usando transições com consumo do **átomo** α ou a chamada da **submáquina NT**, respectivamente.



CONCATENAÇÃO

Concatenação

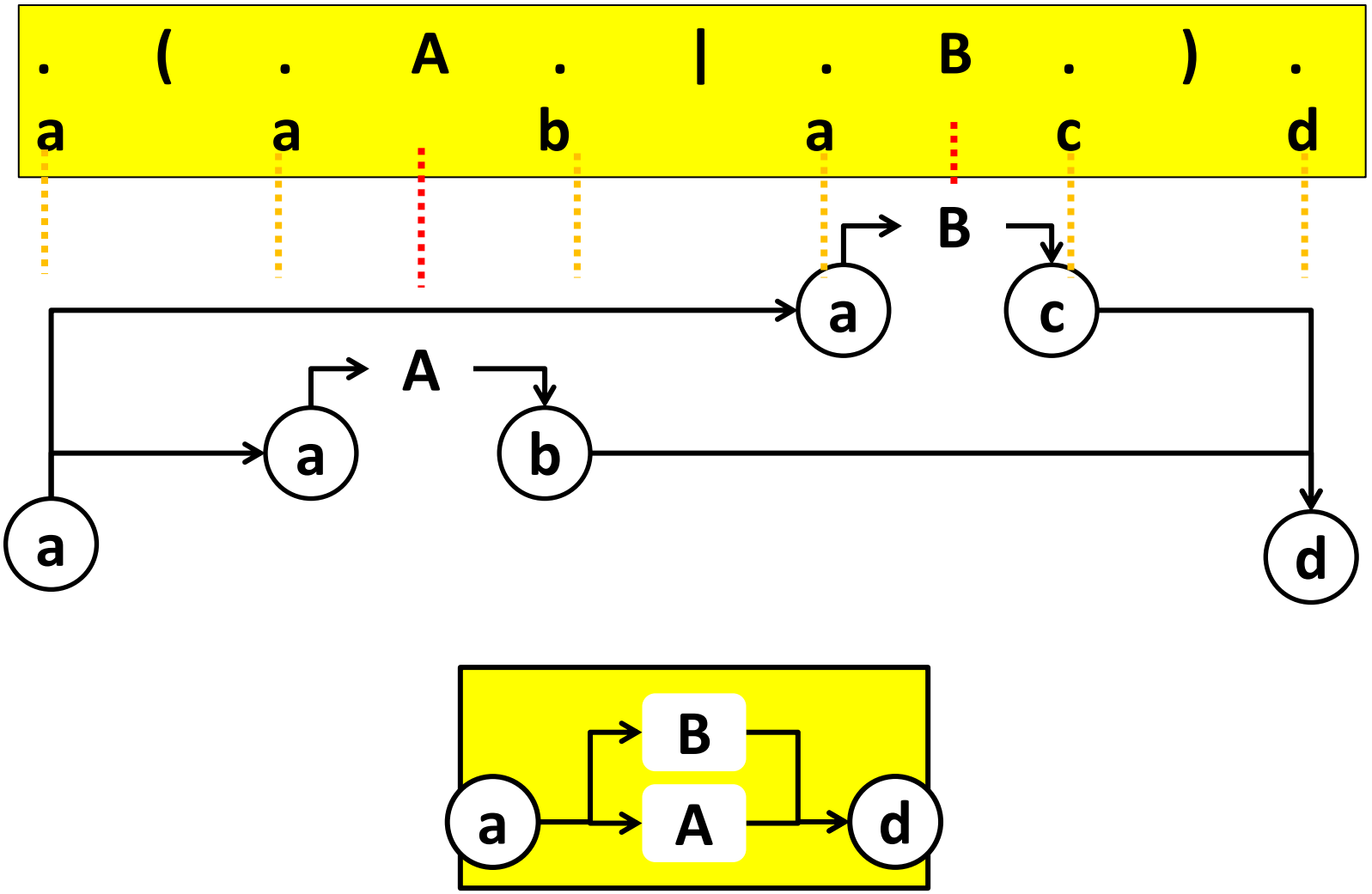
Na Notação de Wirth, denota-se a concatenação de formas sintáticas A,B, ... pela Justaposição das fórmulas que as definem.



UNIÃO DE CONJUNTOS

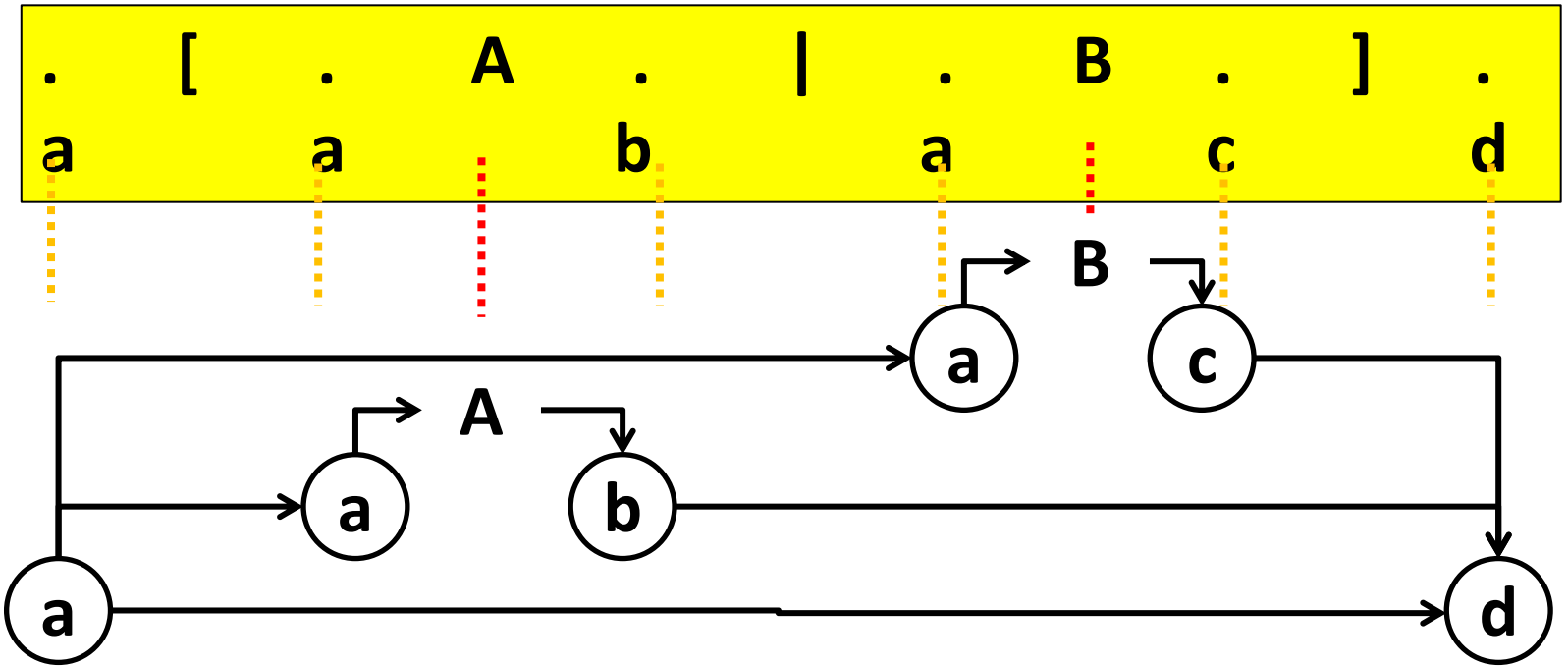
União de conjuntos (obrigatório)

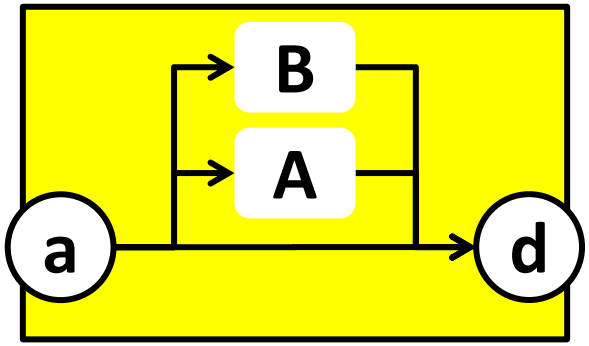
Na Notação de Wirth, denota-se a união de formas sintáticas A,B, ... Separando por meio de barras verticais as fórmulas que as definem. Denotadas entre parênteses.



União de conjuntos (optativo)

Na Notação de Wirth, denota-se a união (optativa)de formas sintáticas A,B, ... Separando por meio de barras verticais as fórmulas que as definem entre colchetes.
A única diferença para o caso obrigatório é a transição em vazio entre os estados a e d.

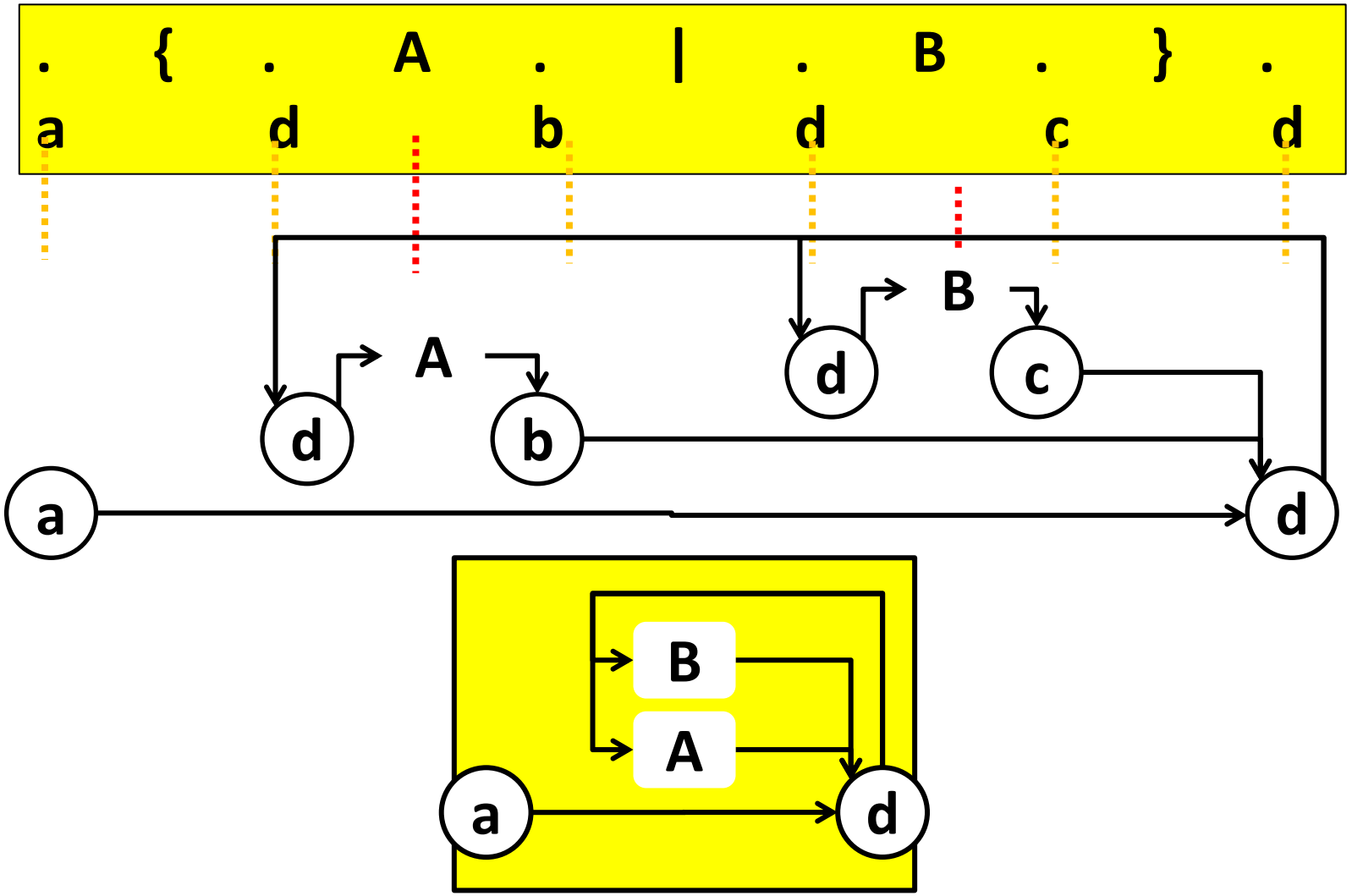




AGRUPAMENTO REPETITIVO (FECHO DE KLEENE)

Fecho de Kleene (repetitivo)

Na Notação de Wirth, denota-se a união (optativa)de formas sintáticas A,B, ... Separando por meio de barras verticais as fórmulas que as definem entre colchetes.
A única diferença para o caso obrigatório é a transição em vazio entre os estados a e d.



MONTAGEM DO RECONHECEDOR
1. ATRIBUIÇÃO DE ESTADOS

- A partir da gramática iterativa, passa-se à construção dos autômatos associados a cada um dos três não-terminais essenciais da mesma:
  - **programa** – raiz da gramática
  - **expressão** – auto-recursivo central
    depende de **expressão**
  - **comando** – auto-recursivo central
    depende de **expressão** e de **comando**
- O passo seguinte é a associação dos diversos pontos das fórmulas das expressões de Wirth aos estados do autômato que estamos construindo.
- No próximo slide, estão demarcadas as regras da gramática associadas aos três não-terminais acima.

Gramática Sintática da Linguagem de Entrada

Em Notação de Wirth iterativa, a sintaxe livre de contexto da linguagem pode ser assim formalizada (em destaque: **programa**, **expressão** e **comando**)

(1) **programa** = seqüência-de-comandos "END".
(2) seqüência-de-comandos = comando { ";" comando }.

(3) **expressão** = termo { ("+" I "-") termo }.
(4) termo = fator { ("*" I "/") fator }.
(5) fator = identificador I número I "(" expressão ")".

(6) **comando** = { rótulo ":" } [atribuição | desvio I leitura I impressão I decisão].
(7) atribuição = "LET" identificador ":=" expressão.
(8) desvio = "GO" "TO" (rótulo I identificador "OF" lista-de-rótulos).
(9) lista-de-rótulos = rótulo { "," rótulo }.
(10) rótulo = identificador.
(11) leitura = "READ" lista-de-Identificadores.
(12) lista-de-Identificadores = [identificador { "," identificador }].
(13) impressão = "PRINT" lista-de-expressões.
(14) lista-de-expressões = [expressão { "," lista-de-expressões }].
(15) decisão = "IF" comparação "THEN" comando "ELSE" comando.
(16) comparação = expressão operador-de-comparação expressão.
(17) operador-de-comparação = ">" I "=" | "<“.

- Podemos agora considerar essa gramática como um sistema de equações cujas variáveis são os não-terminais e cujas constantes são os átomos da linguagem.
- Por meio da eliminação sistemática dos não-terminais considerados não essenciais, podem ser eliminadas as variáveis indesejadas, por suas substituições sucessivas pelas fórmulas de Wirth que as definem em termos de outras variáveis e das constantes da gramática.
- Isso deve ser feito sucessivamente, até que todas as variáveis essenciais sejam expressas apenas em função de constantes e de variáveis essenciais.
- Os slides seguintes mostram a aplicação desse procedimento à gramática que estamos utilizando.

PROGRAMA

Tratamento do não-terminal programa

A partir das regras (1) e (2) que especificam **programa** podemos fazer as manipulações seguintes:

(1) **programa** = seqüência-de-comandos "END".
(2) seqüência-de-comandos = comando { ";" comando }.

Substituindo a expressão (2) de seqüência-de-comandos em (1), obtemos:

programa = comando { ";" comando } "END".

Rebatizando comando como **C** e em seguida numerando a expressão resultante, temos, finalmente:

programa = . **C** . **{** . **;** . **C** . **}** . **END** .
0 1 2 3 4 2 5

EXPRESSÃO

Tratamento do não-terminal expressão

A partir das regras (3) a (5) que especificam **expressão,** podemos fazer as manipulações seguintes:

(3) **expressão** = termo { ("+" | "-") termo }.
(4) termo = fator { ("*" | "/") fator }.
(5) fator = identificador | número | "(" **expressão** ")".

Substituindo (4) em (3), e em seguida (5) na expressão resultante tem-se:

expressão =
(identificador | número | "(" **expressão** ")")
{ ("*" | "/") (identificador | número | "(" **expressão** ")") }
{ ("+" | "-") (identificador | número | "(" **expressão** ")")
{ ("*" | "/") (identificador | número | "(" **expressão** ")" } }.

Rebatizando **expressão** como **E** e em seguida numerando, obtém-se finalmente:

. (. Id . | . N . | . ⟨ . E . ⟩ .) .
0 0 2 0 3 0 4 5 6 1
. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } .
1 7 7 9 7 10 8 8 12 8 13 8 14 15 16 11 7
. { . (. + . | . - .) . (. Id . | . N . | . ⟨ . E . ⟩ .) .
7 17 17 19 17 20 18 18 22 18 23 18 24 25 26 21
. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } . } .
21 27 27 29 27 30 28 28 32 28 33 28 34 35 36 31 27 17

COMANDO

Tratamento do não-terminal comando

Em Notação de Wirth, a sintaxe livre de contexto da linguagem de entrada pode ser assim formalizada (em destaque, **programa**, **expressão** e **comando**)

(6) **comando** = { rótulo ":" } [atribuição | desvio I leitura I impressão I decisão].
(7) atribuição = "LET" identificador ":=" **expressão**.
(8) desvio = "GO" "TO" (rótulo I identificador "OF" lista-de-rótulos).
(9) lista-de-rótulos = rótulo { "," rótulo }.
(10) rótulo = identificador.
(11) leitura = "READ" lista-de-Identificadores.
(12) lista-de-Identificadores = [identificador { "," identificador }].
(13) impressão = "PRINT" lista-de-expressões.
(14) lista-de-expressões = [**expressão** { "," **expressão** }].
(15) decisão = "IF" comparação "THEN" **comando** "ELSE" **comando**.
(16) comparação = **expressão** operador-de-comparação **expressão**.
(17) operador-de-comparação = ">" I "=" | "<".

Efetuando as substituições indicadas pelas cores na gramática acima, e em seguida numerando os estados, tem-se como resultado, para o não-terminal comando, a expressão mostrada no slide seguinte.

comando

```
. { . Id .  : .  } .
0   1    39    40    1

. [ . LET . Id . := . E .
1   1     3    4    5   6

| .GOTO . (  . Id . [  . OF . Id . { . , . id . } . ] . ) .
  1      7    7    9    9    11   12   13   14   15   13   10   8

| . READ .  [  .  Id .  {  .  ,  .  Id .  }  .  ]  .
  1      16    16    18    19    20    21    19    17

| . PRINT  .  [  .  E  .  {  .  ,  .  E .  }  .  ]  .
  1       22    22    24    25    26    27    25    23

| . IF  . E  .  (  .  <  .  |  . = .  |  .  >  .  )  .  E . THEN .  C . ELSE .  C  .  ]  .
  1     28    29    29    31    29  32    29    33    30    34       35    36       37    38    2
```

MONTAGEM DO RECONHECEDOR
2. DIAGRAMAS DE TRANSIÇÕES

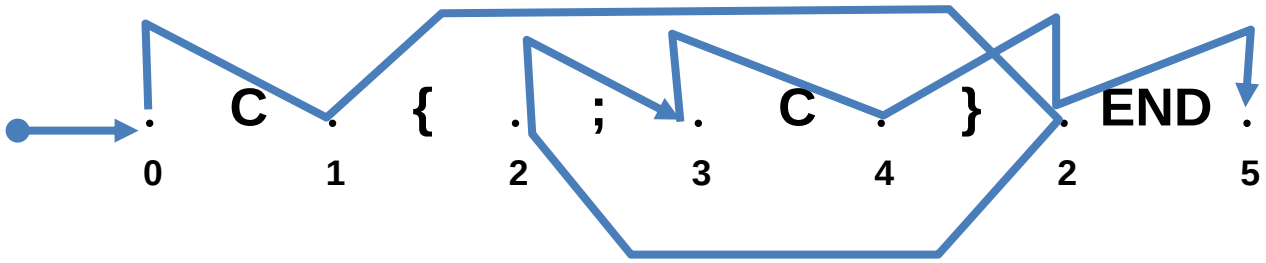
- O passo seguinte é interpretar a numeração dos estados associando cada qual ao respectivo papel no processo de reconhecimento da linguagem:
  - Exaustivamente, identificam-se, na expressão numerada, as sequências de estados percorridos ao ser reconhecido cada tipo de construção sintática da linguagem.
  - Constrói-se um diagrama de estados correspondente, com base nas regras de conversão da gramática em autômato.
  - Eliminam-se estados repetidos, transições em vazio e outras redundâncias do grafo obtido, produzindo-se o diagrama do autômato desejado.
  - Para obter o reconhecedor, pode-se traduzir para a forma de um programa o diagrama de estados obtido neste processo, ou convertê-lo em uma tabela de transições, a ser acoplada a um programa geral que interprete essas tabelas.

PROGRAMA

programa

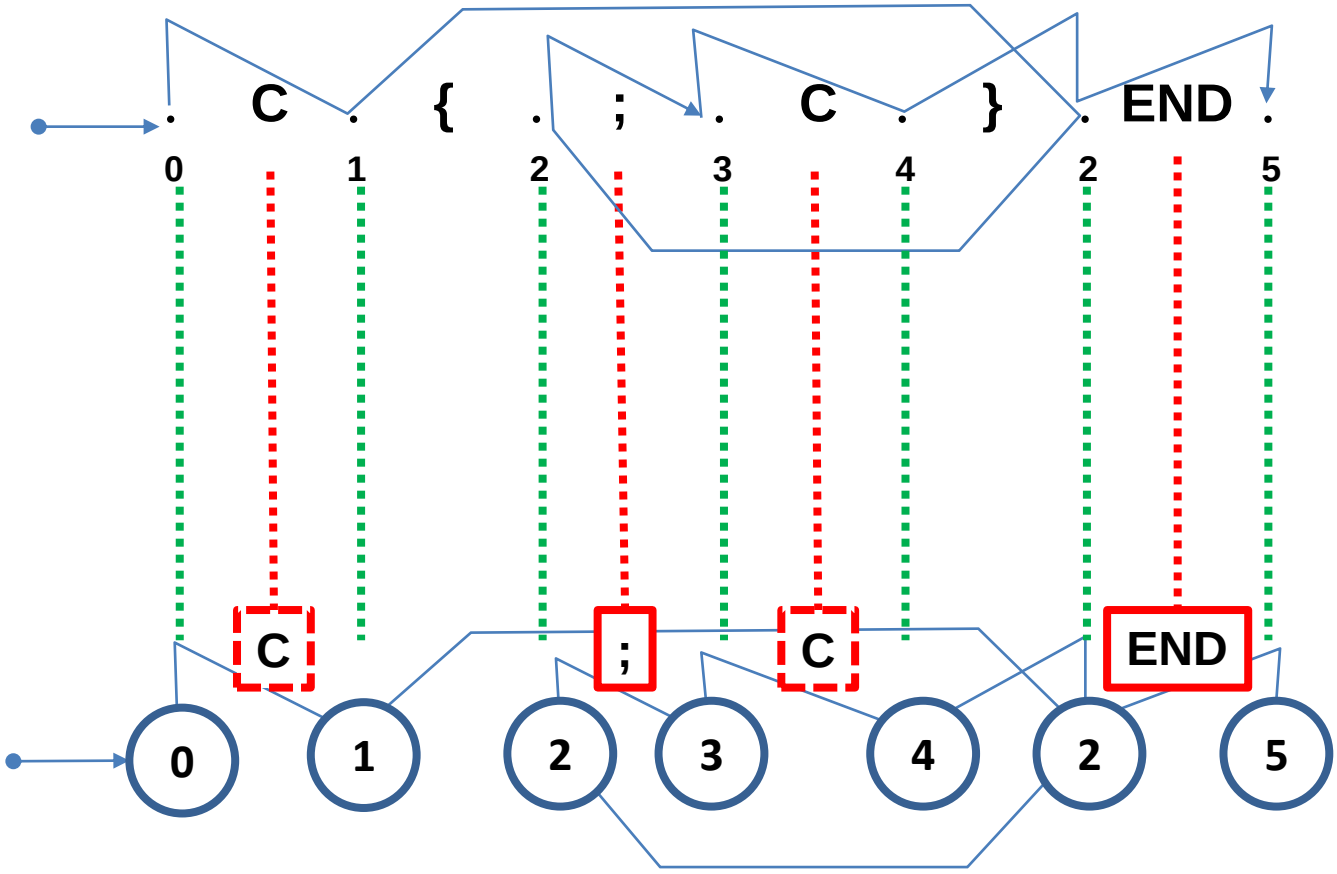
.	C	.	{	.	;	.	C	.	}	.	END	.
0		1		2		3		4		2		5

a) Caminhos possíveis de reconhecimento

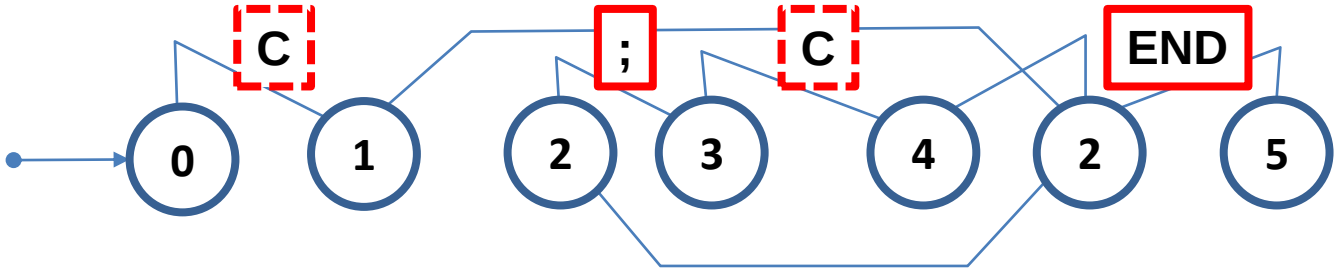


programa

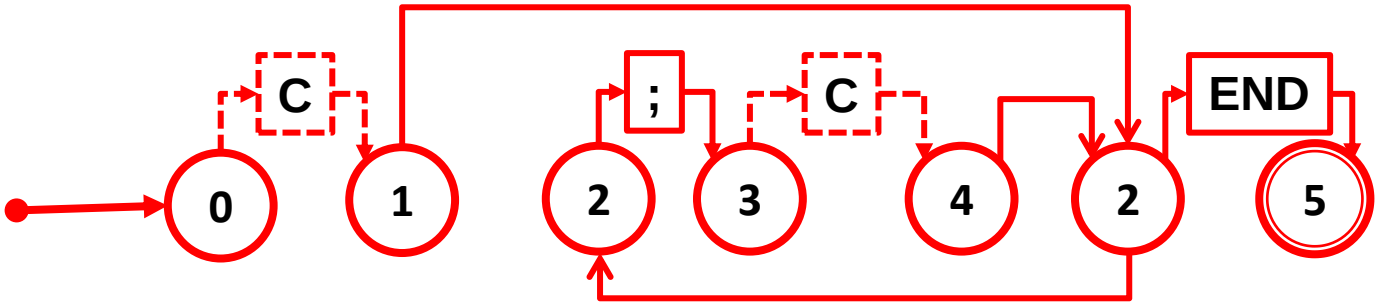
b) Consumo de átomos durante o percurso realizado



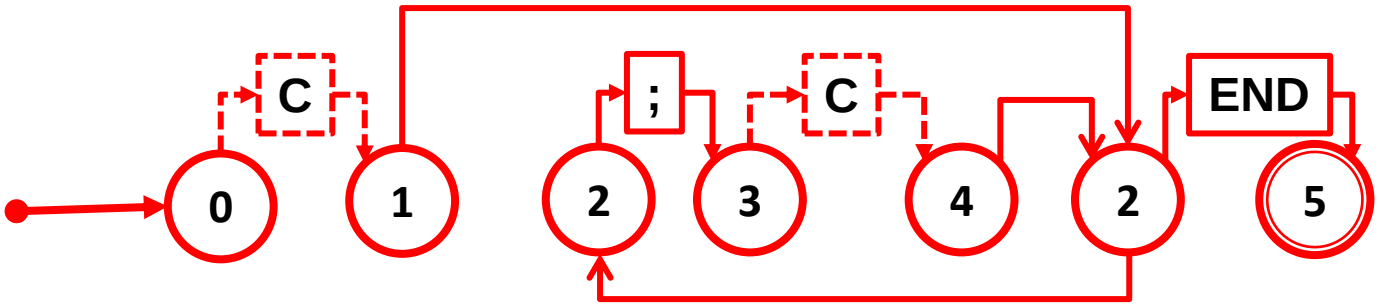
programa



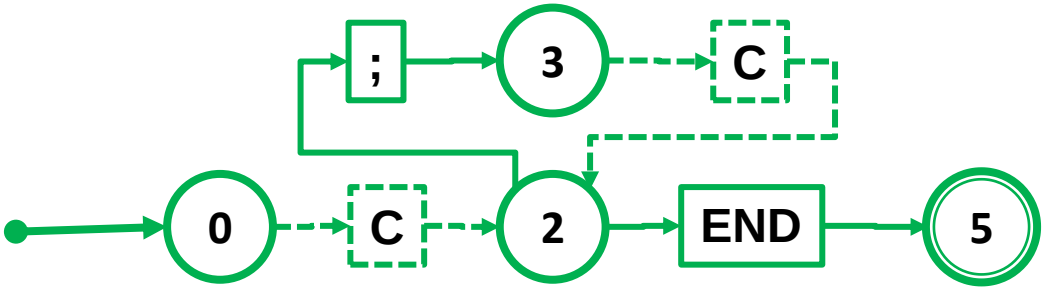
c) Incluindo agora as transições:



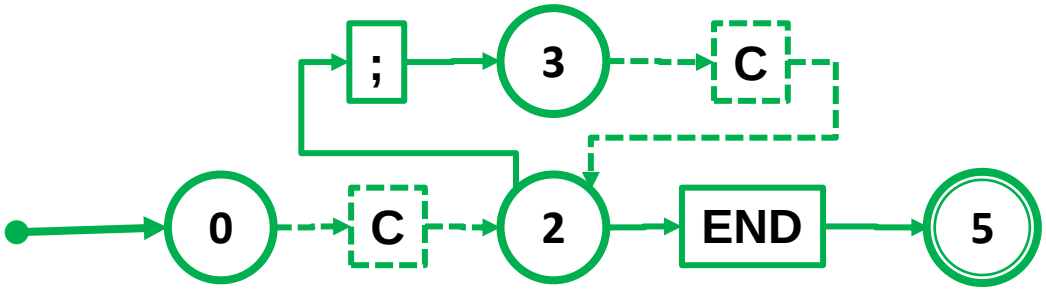
programa



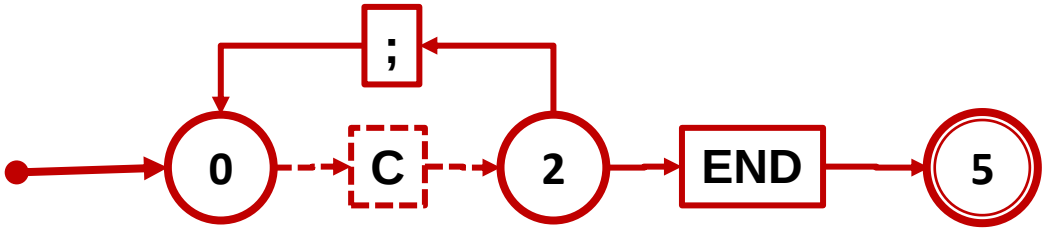
d) Eliminando estados repetidos e transições em vazio:



programa



e) Eliminando redundâncias remanescentes:



Este autômato é mínimo.

programa

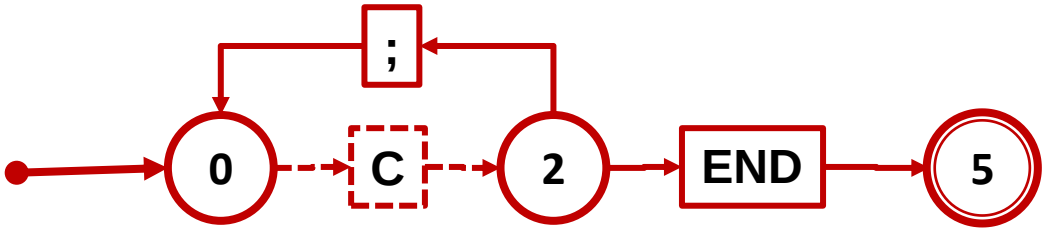
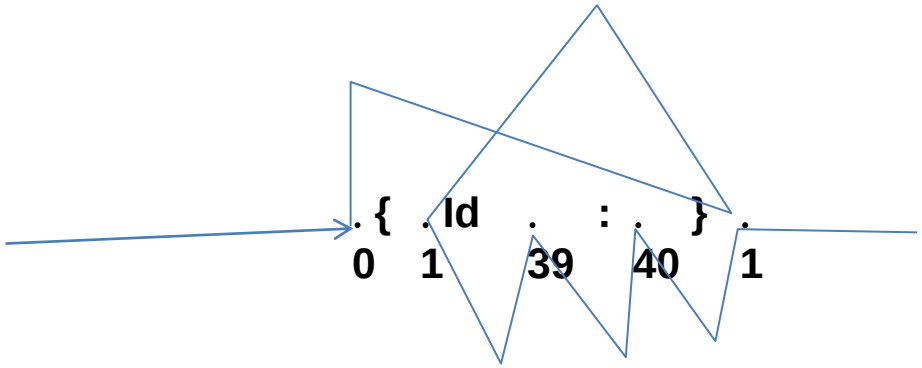


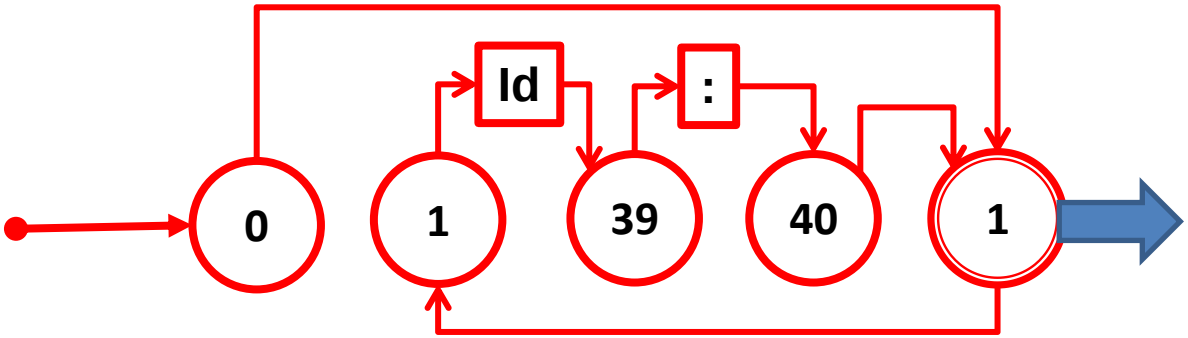
Tabela de transições:

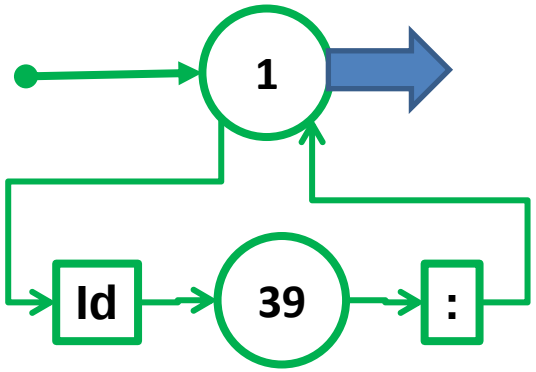
	;	END	C	outros
→ 0			2	
2	0	5		
5 →				

COMANDO

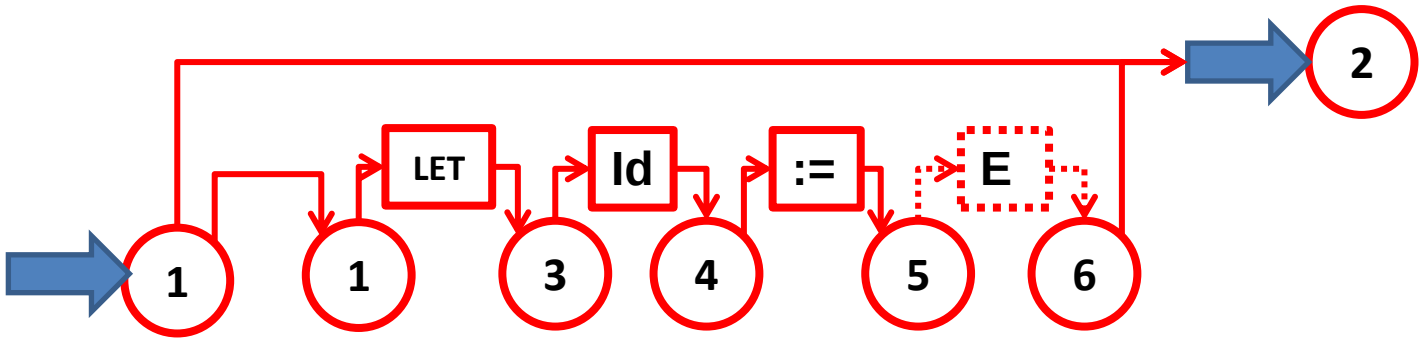
Comando (1)

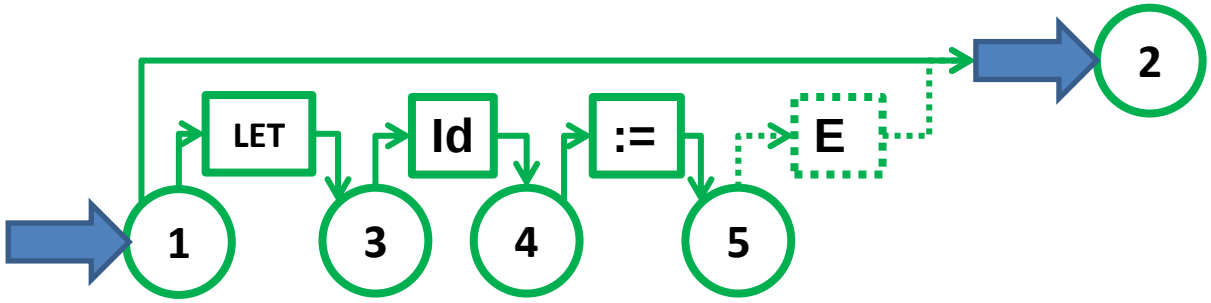






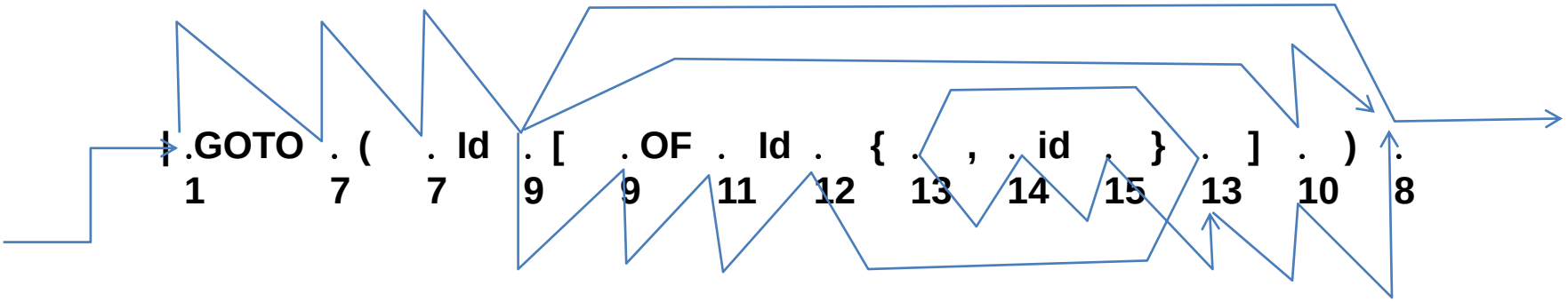
Comando (2)

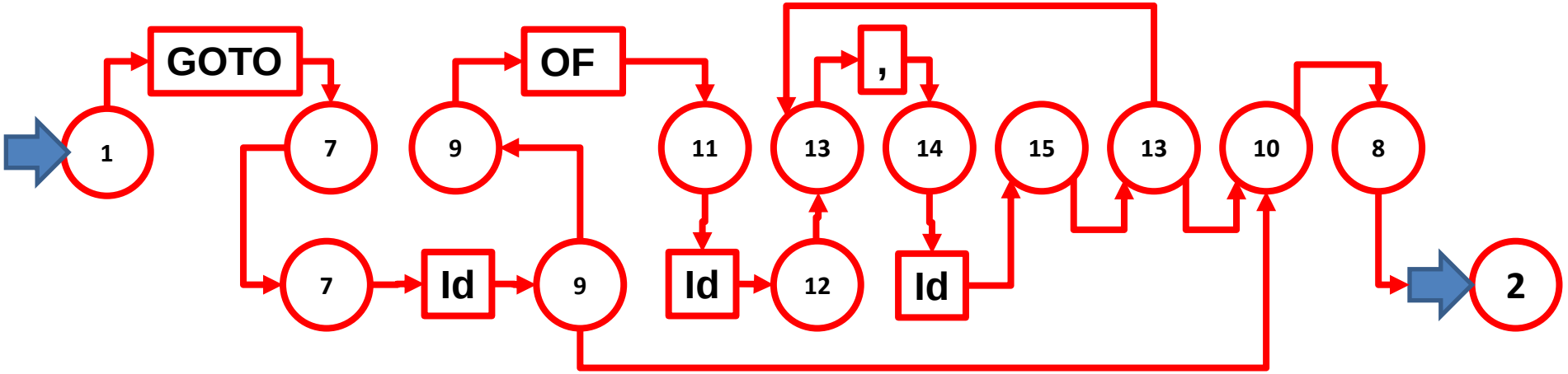


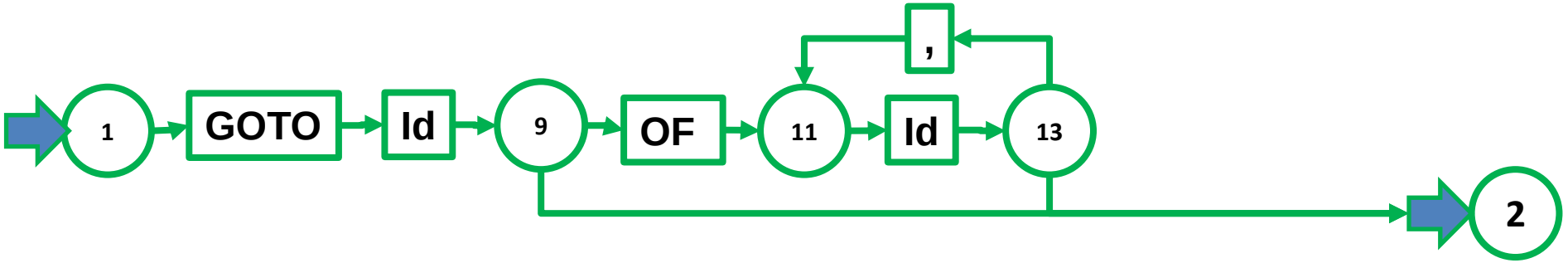


Obs. – a transição em vazio 1-2 ainda não pode ser eliminada pois faltam informações sobre o comportamento do estado 2.

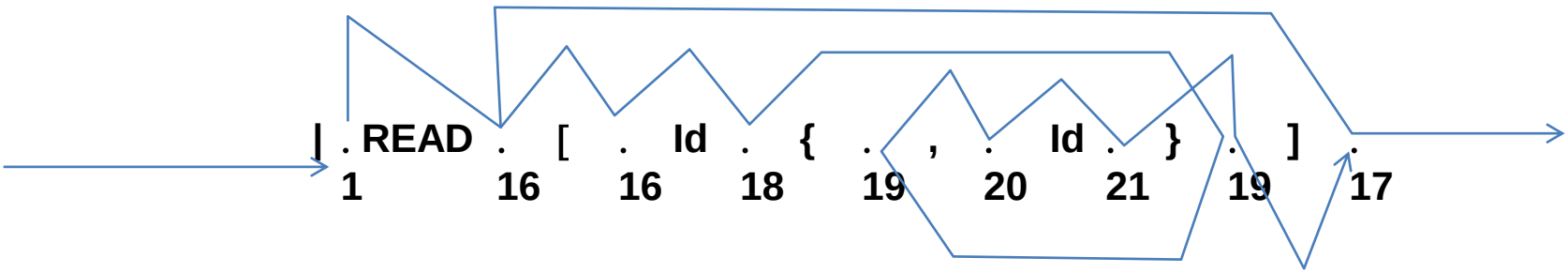
Comando (3)

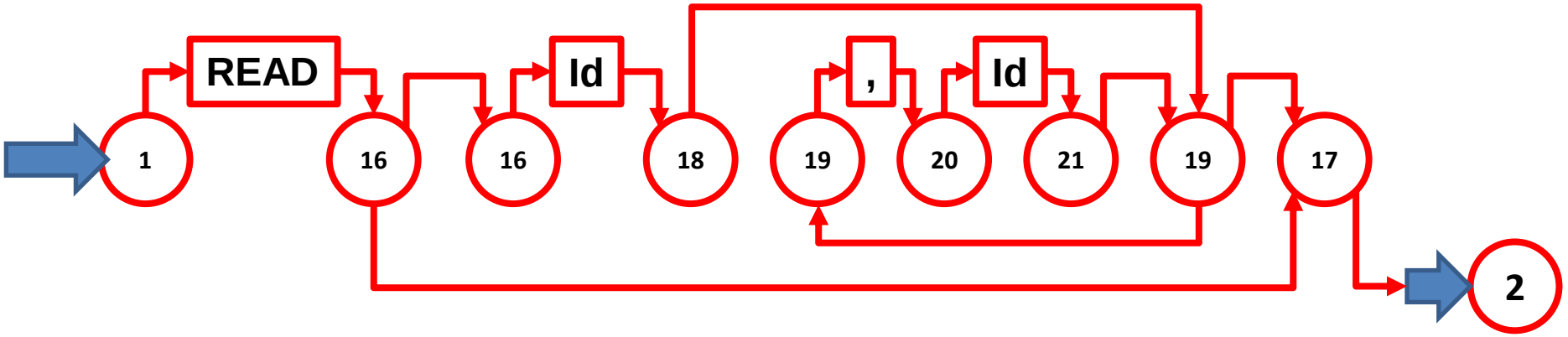

.GOTO . (. Id . [. OF . Id . { . , . id . } .] .) .
1 7 7 9 9 11 12 13 14 15 13 10 8

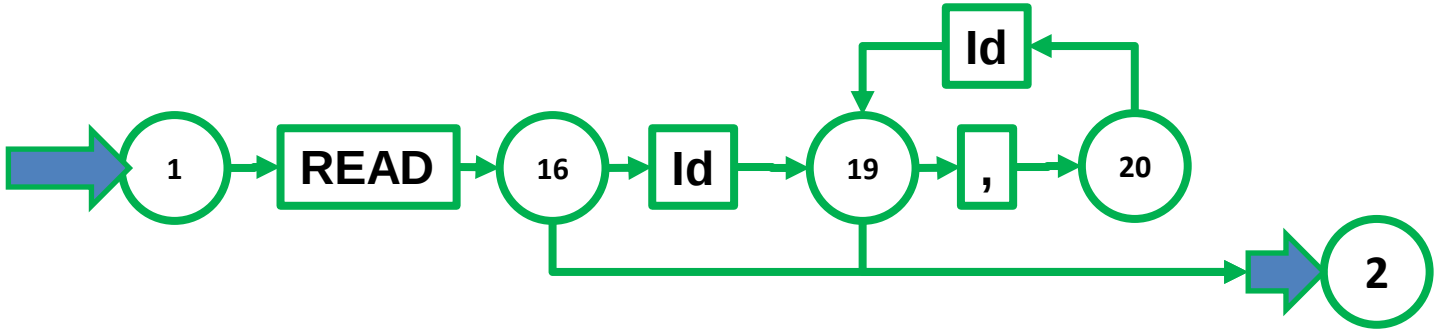

GOTO
OF
,
1
7
9
11
13
14
15
13
10
8
7
Id
9
Id
12
Id
2


,
1
GOTO
Id
9
OF
11
Id
13
2

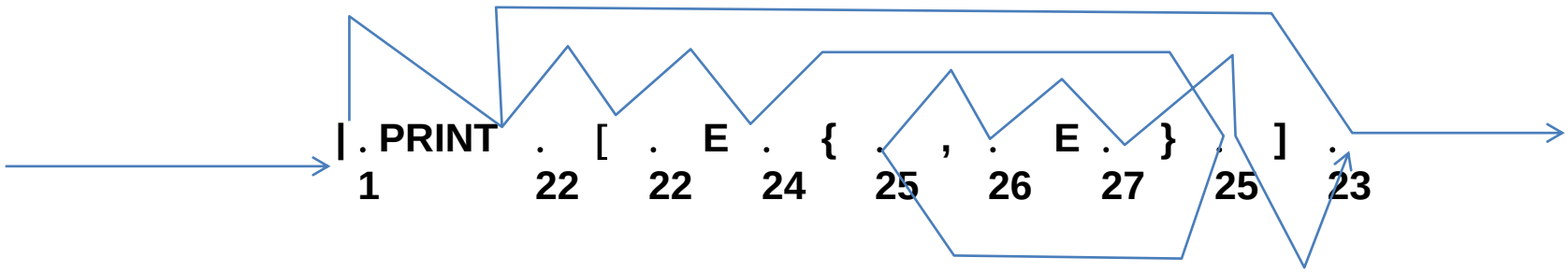
Comando (4)

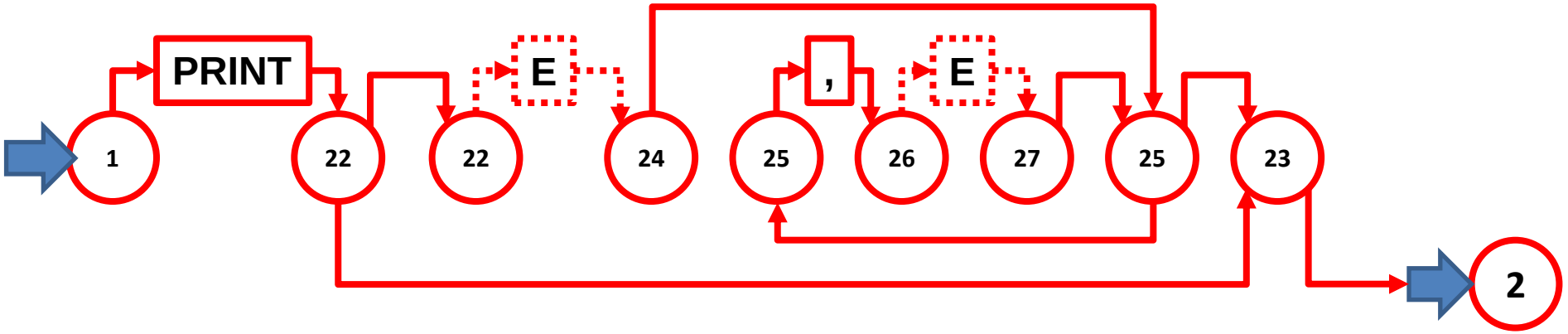

| . READ . [. Id . { . , . Id . } .] .
1 16 16 18 19 20 21 19 17

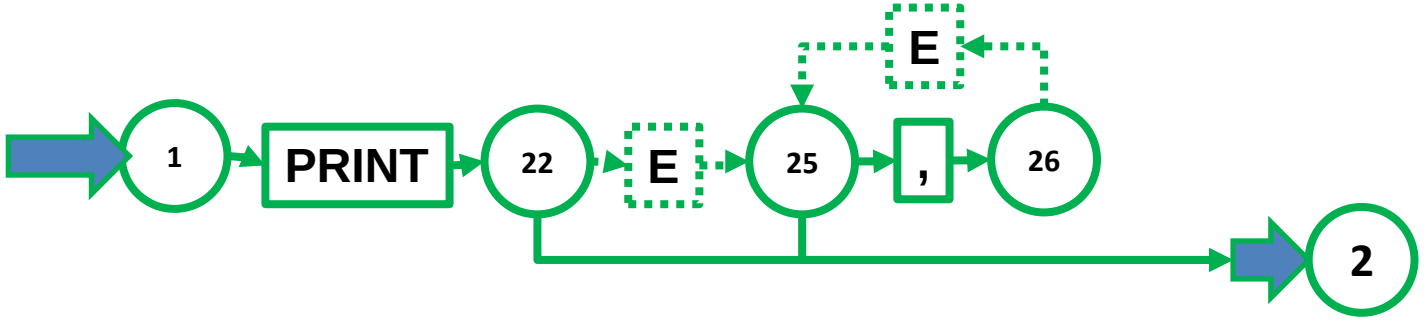

READ
Id
,
Id
1
16
16
18
19
20
21
19
17
2


Id
1
READ
16
Id
19
,
20
2

Comando (5)

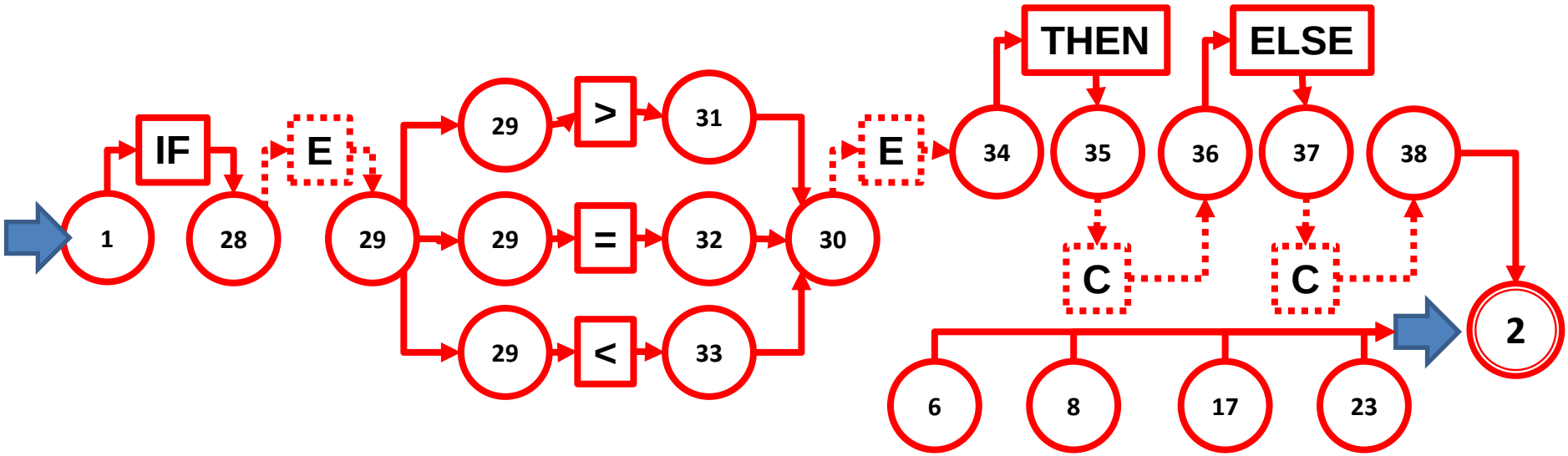

| . PRINT . [. E . { . , . E . } .] .
1 22 22 24 25 26 27 25 23

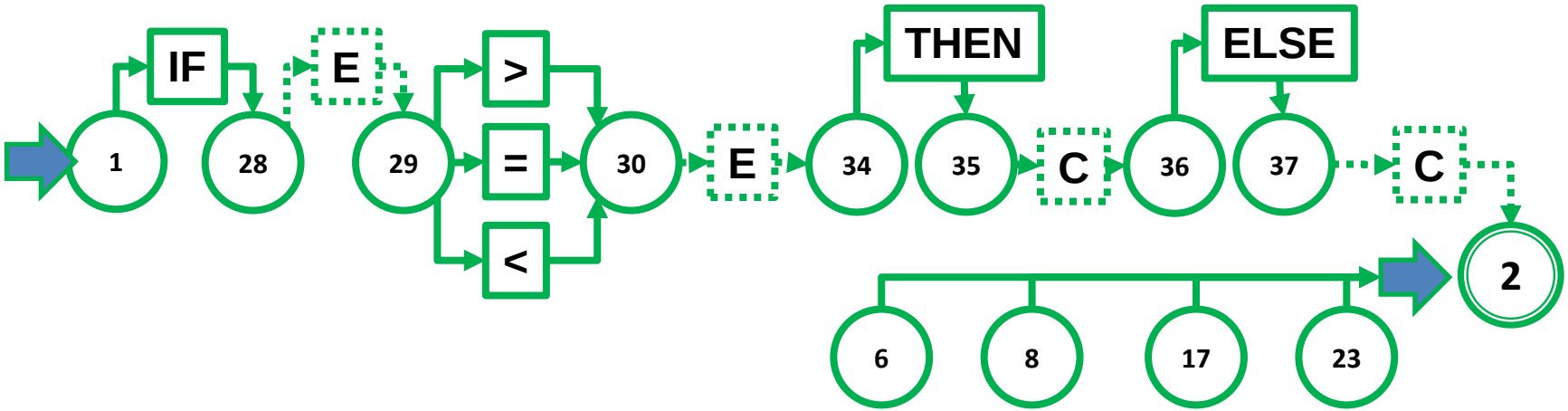

PRINT
E
,
E
1
22
22
24
25
26
27
25
23
2


E
1
PRINT
22
E
25
,
26
2

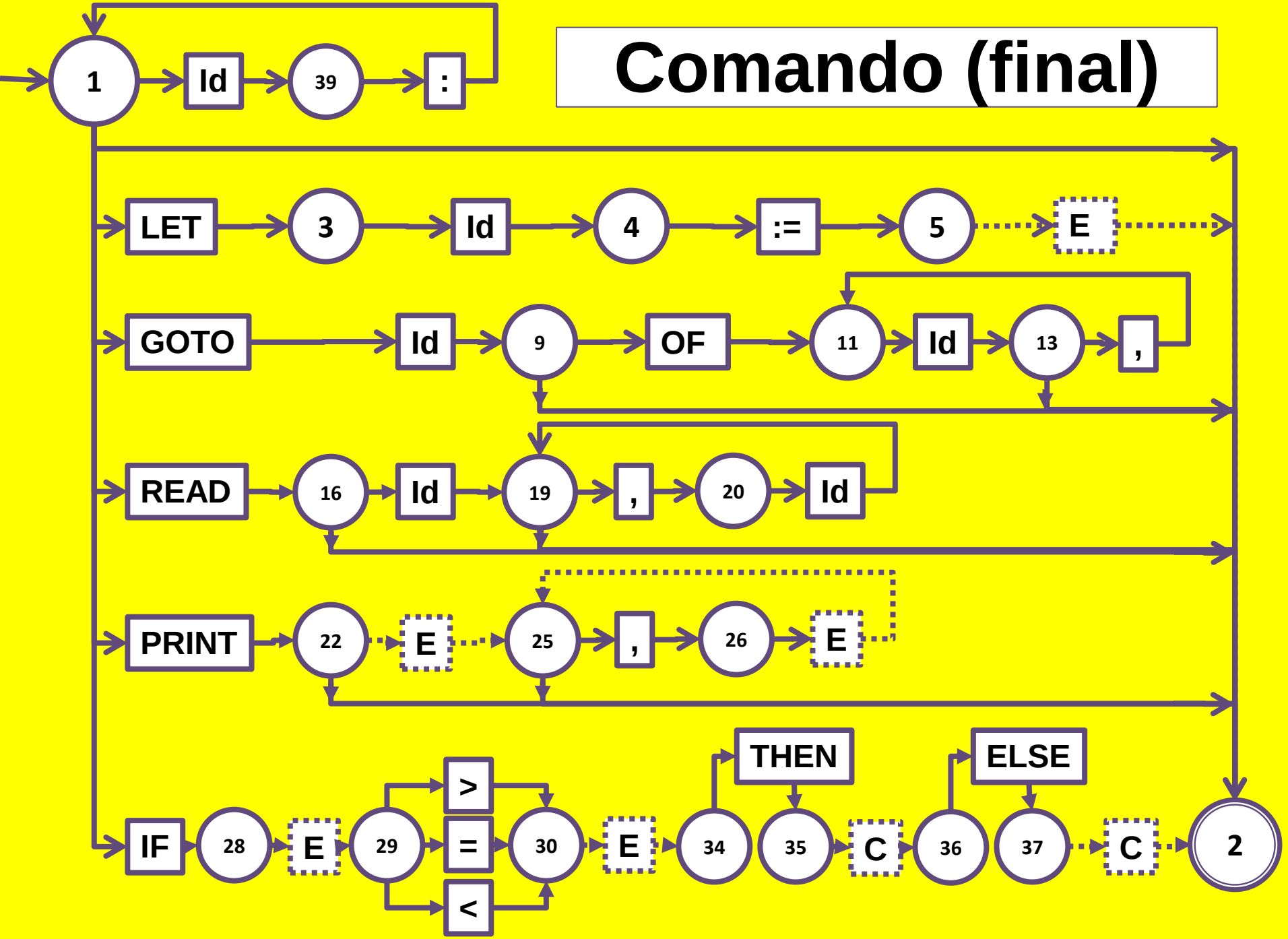
Comando (6)

| . IF . E . (. < . | . = . | . > .) . E . THEN . C . ELSE . C .] .
1 28 29 29 31 29 32 29 33 30 34 35 36 37 38 2





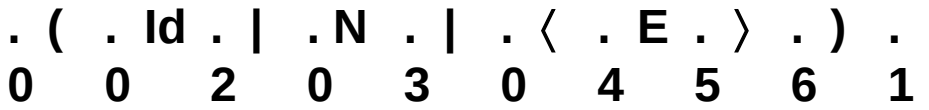
Comando (final)

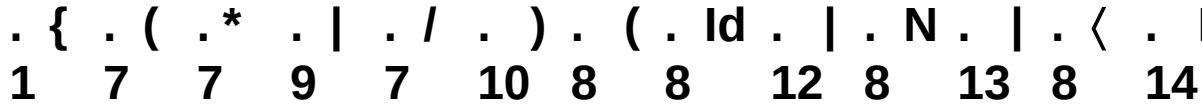


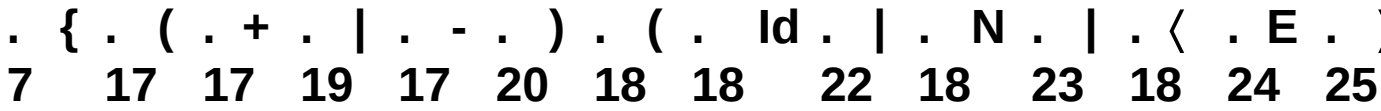
	id	:	LET	:=	GOTO	OF	,	READ	PRINT	IF	>	=	<	THEN	ELSE		C	E	ε
→1	39		3		9			16	22	28									2
3	4																		
4				5															
5																		2	
9						11													2
11	13						11												
13																			2
16	19																		2
19							20												2
20	19																		
22																		25	2
25							26												2
26																		25	
28																		29	
29											30	30	30						
30																		34	
34														35					
35																	36		
36															37				
37																	2		
39		1																	
2→																			

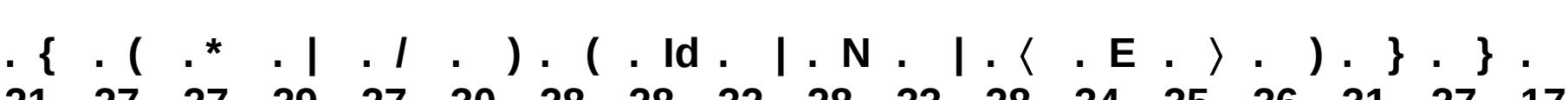
EXPRESSÃO

expressão


. (. Id . | . N . | . ⟨ . E . ⟩ .) .
0 0 2 0 3 0 4 5 6 1

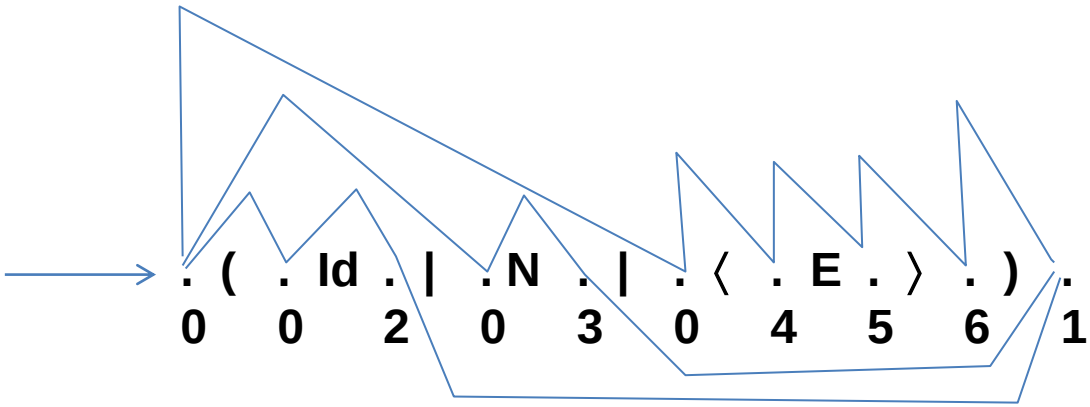

. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } .
1 7 7 9 7 10 8 8 12 8 13 8 14 15 16 11 7


. { . (. + . | . - .) . (. Id . | . N . | . ⟨ . E . ⟩ .) .
7 17 17 19 17 20 18 18 22 18 23 18 24 25 26 21

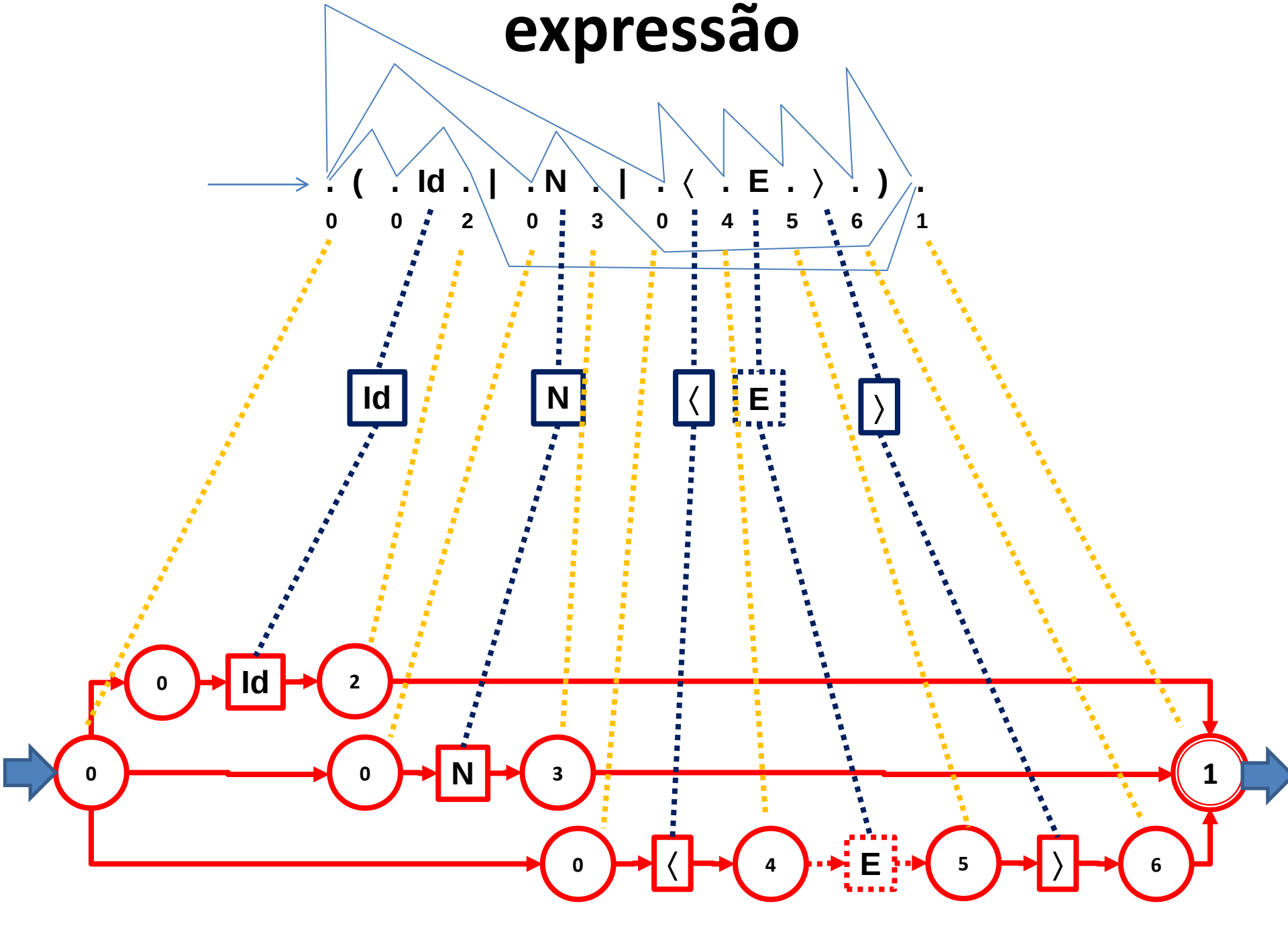

. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } . } .
21 27 27 29 27 30 28 28 32 28 33 28 34 35 36 31 27 17

expressão

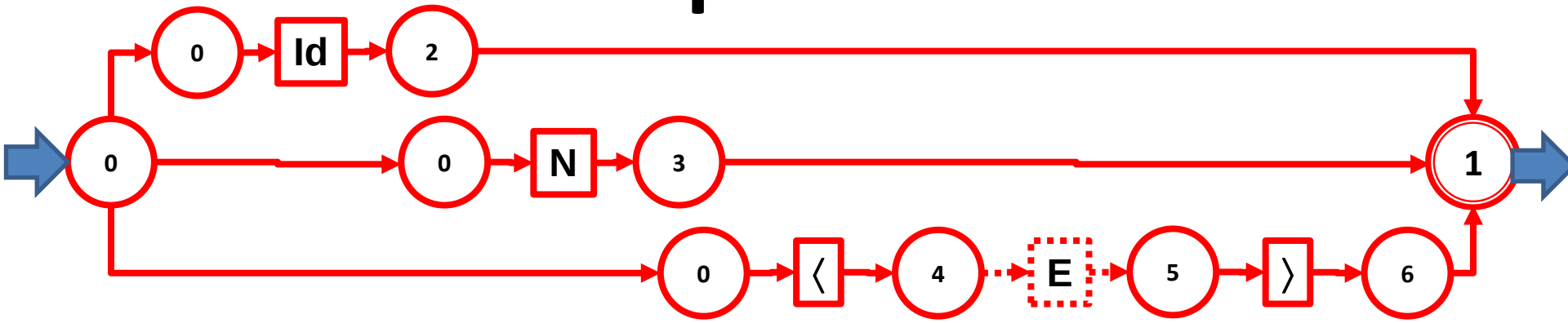
.	(	.	Id	.	|	.	N	.	|	.	⟨	.	E	.	⟩	.	)	.
0		0		2		0		3		0		4		5		6		1

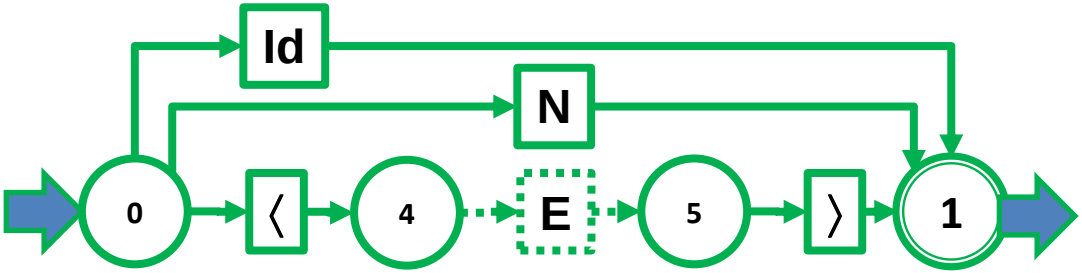


expressão

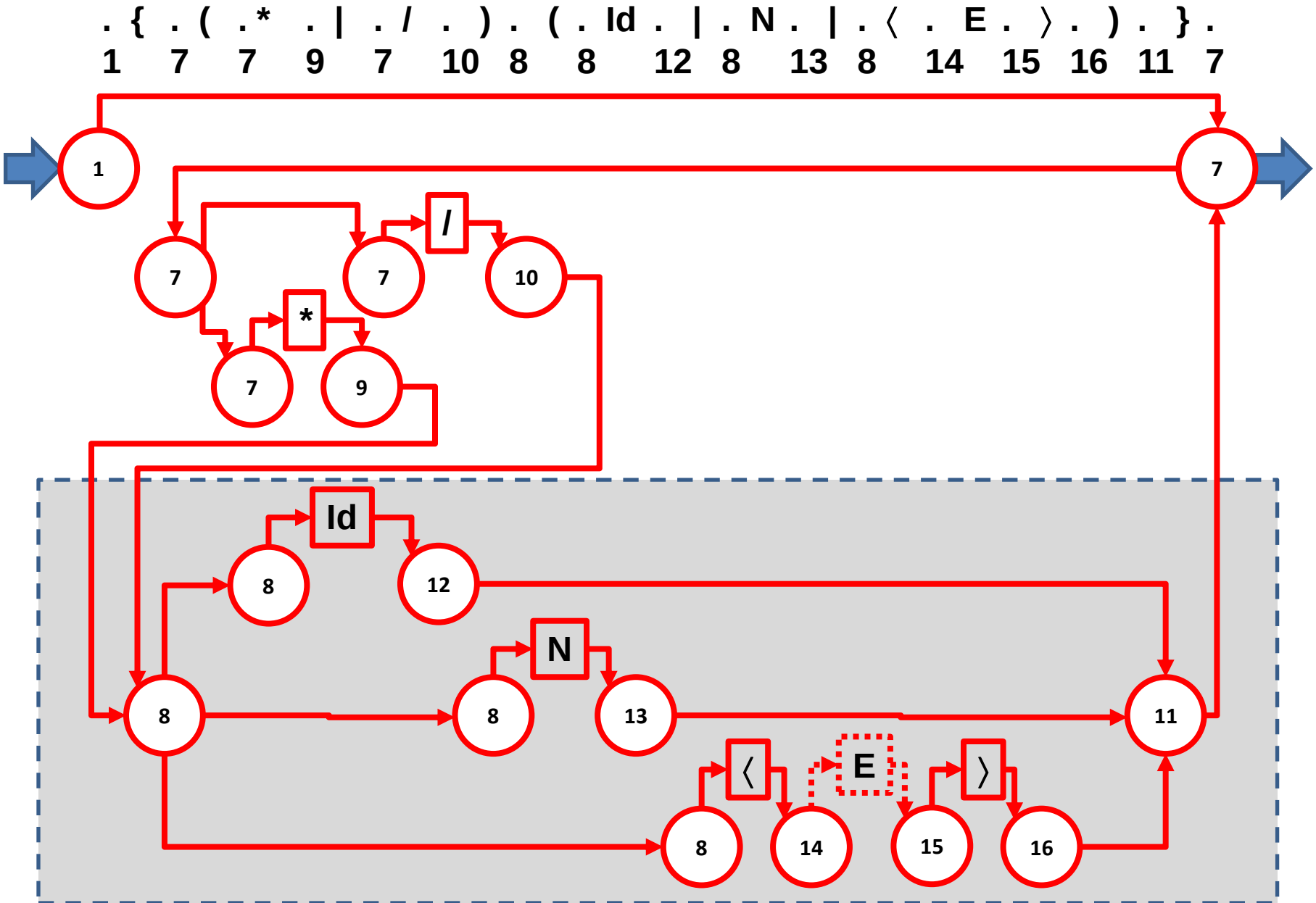


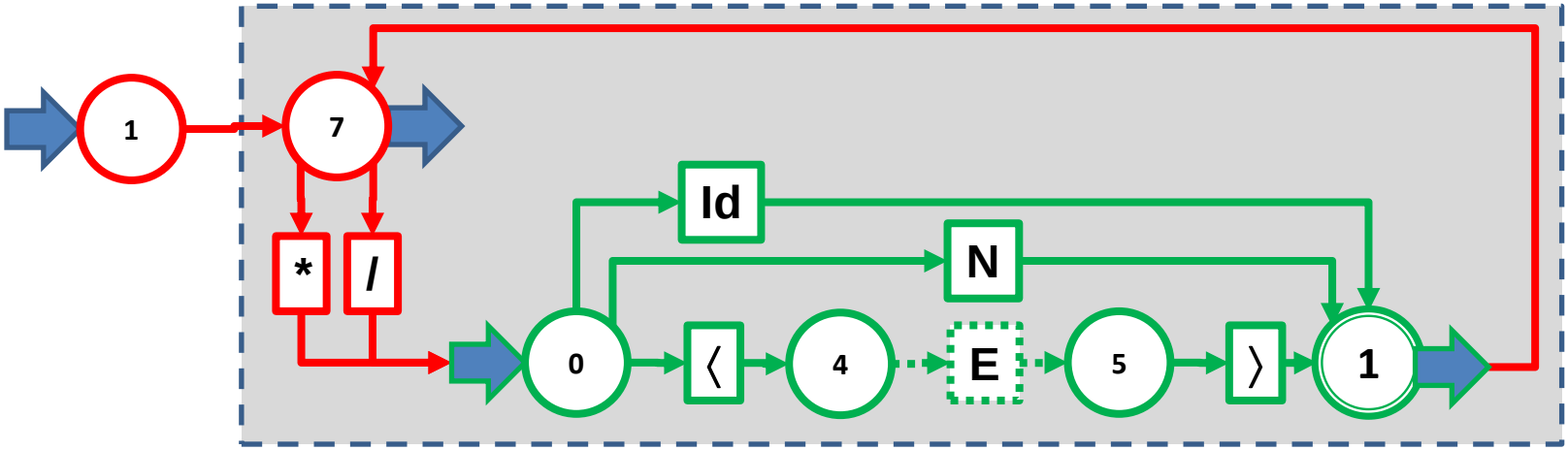
expressão





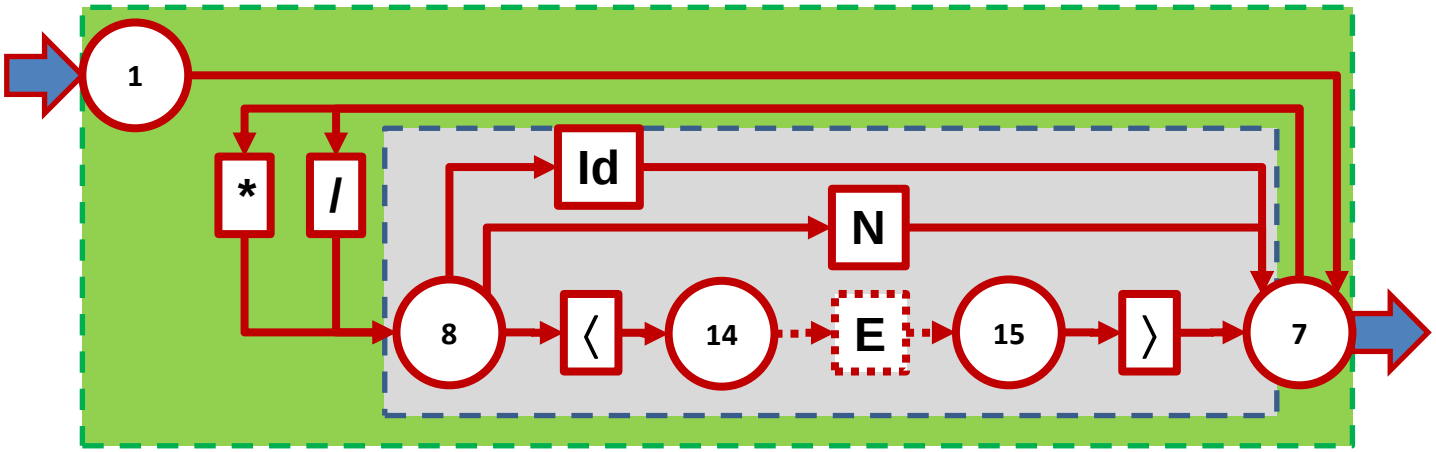
expressão


. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } .
1 7 7 9 7 10 8 8 12 8 13 8 14 15 16 11 7
1
7
7
7
/
10
*
7
9
Id
8
12
N
8
8
13
11
⟨
E
⟩
8
14
15
16


1
7
*
/
0
Id
N
⟨
4
E
5
⟩
1

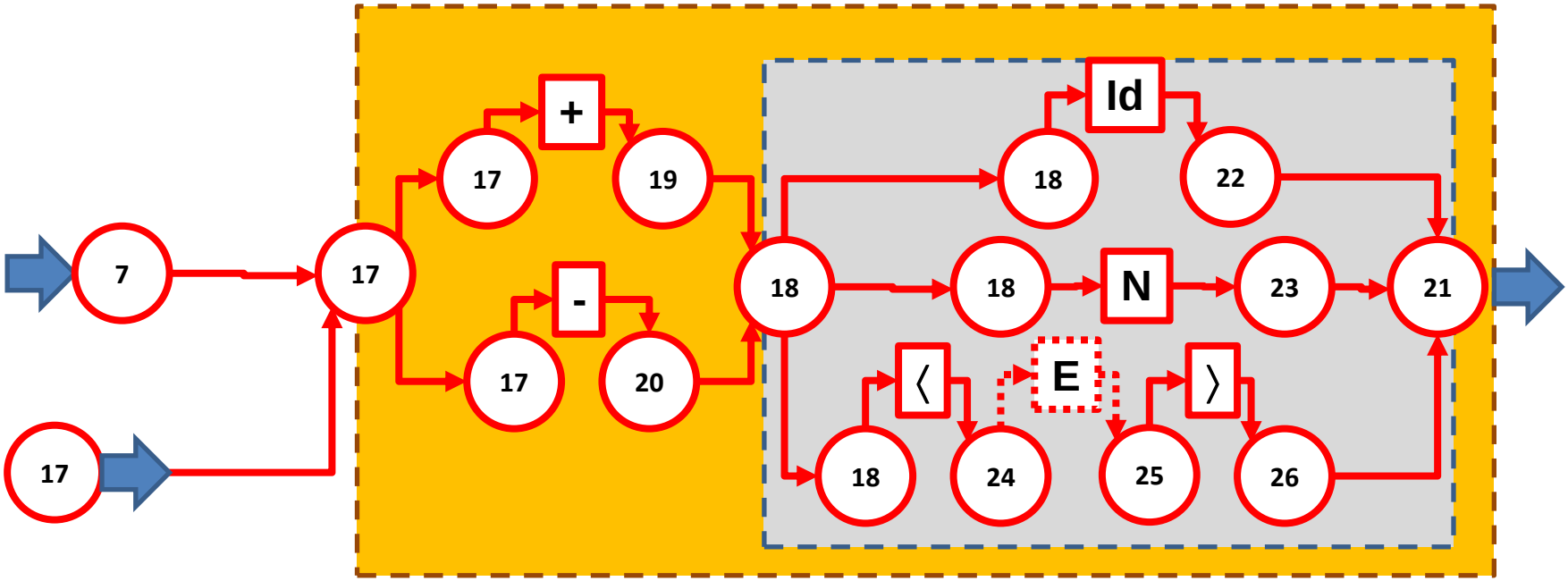
expressão

. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } .
1 7 7 9 7 10 8 8 12 8 13 8 14 15 16 11 7



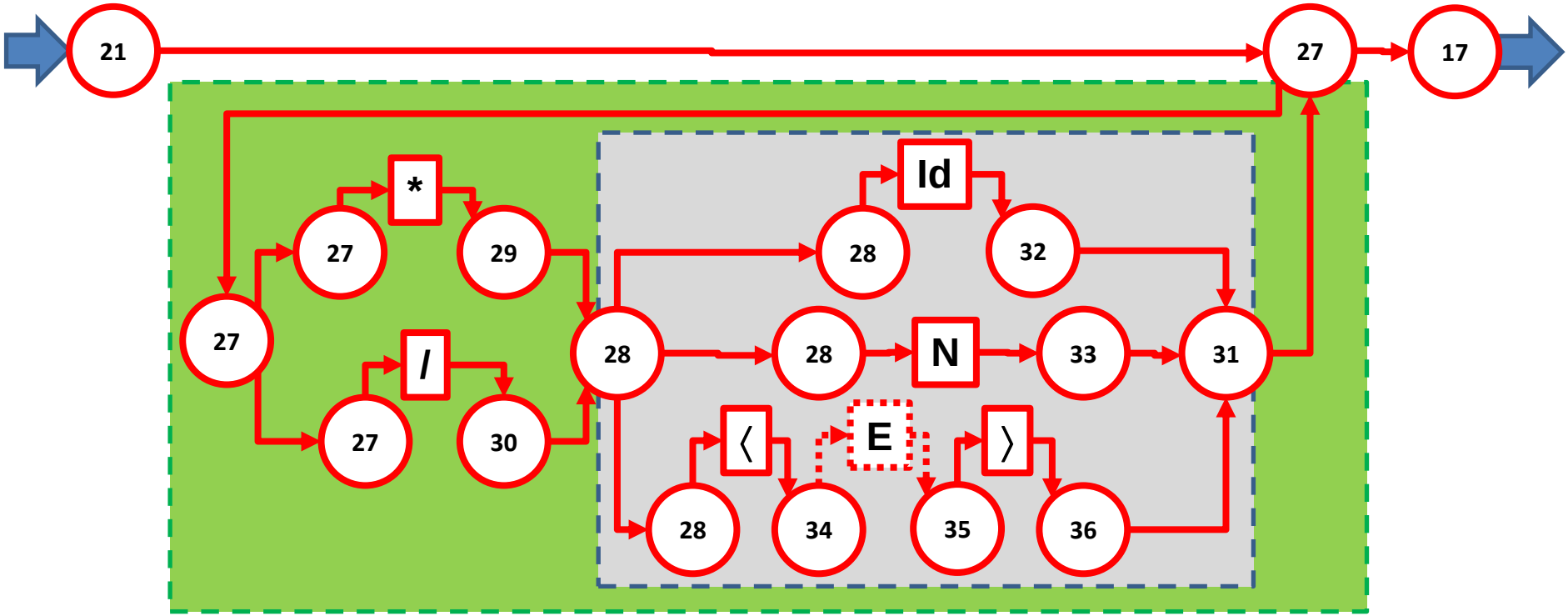
expressão

. { . (. + . | . - .) . (. Id . | . N . | . ⟨ . E . ⟩ .) .
7 17 17 19 17 20 18 18 22 18 23 18 24 25 26 21

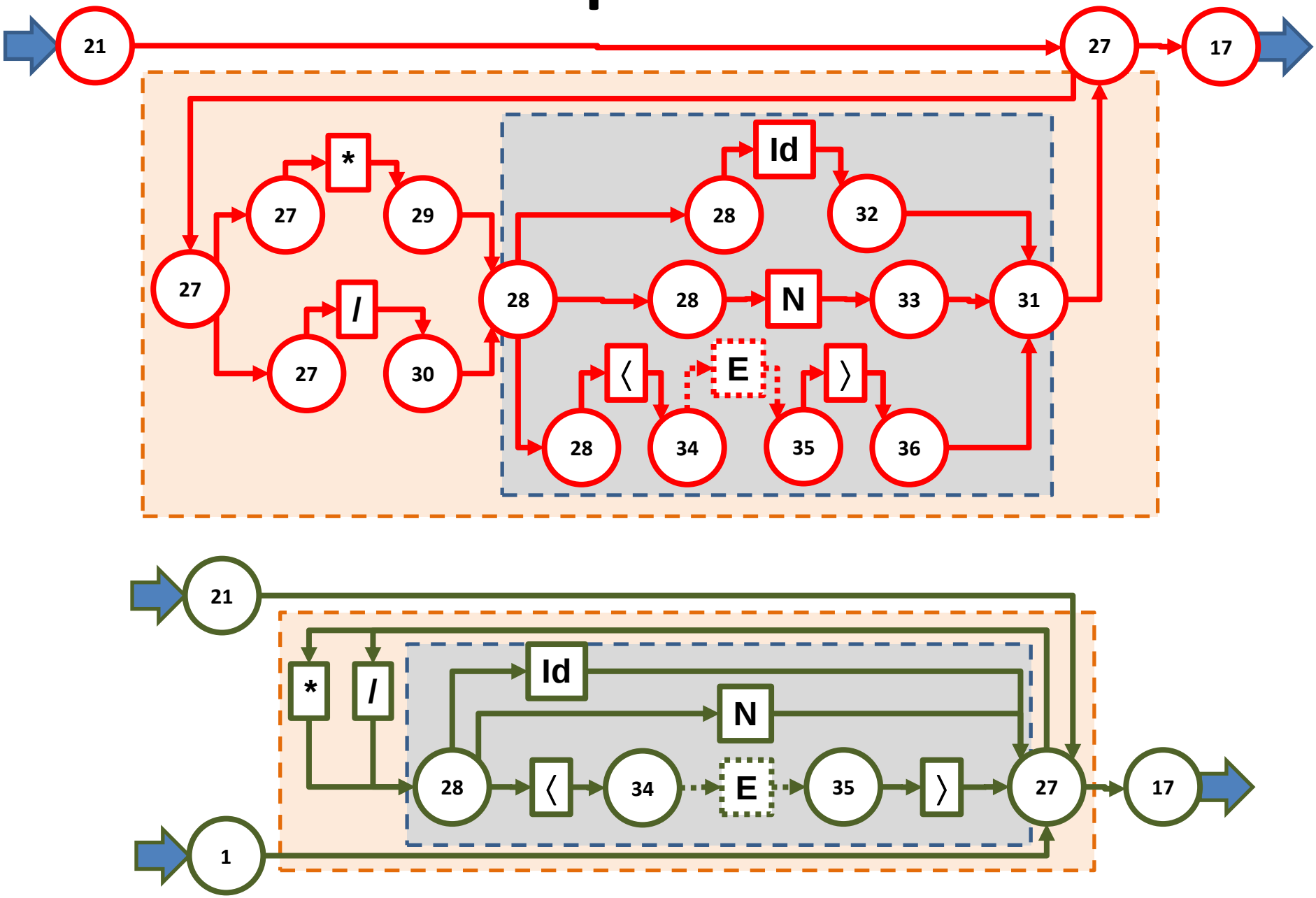


expressão

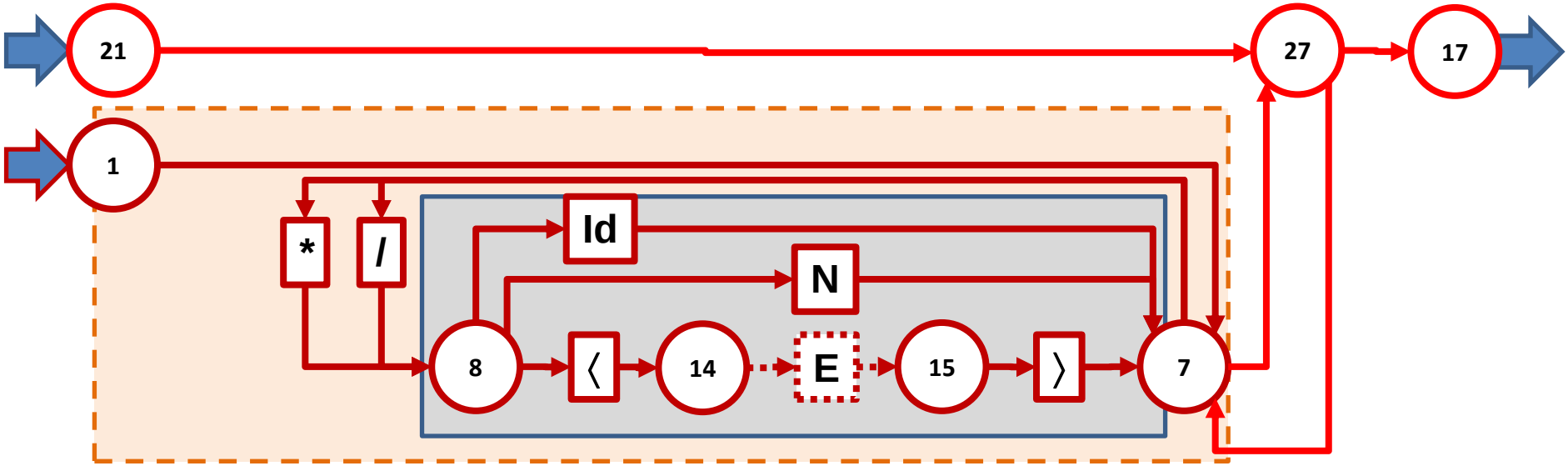
. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } . }
21 27 27 29 27 30 28 28 32 28 33 28 34 35 36 31 27 17

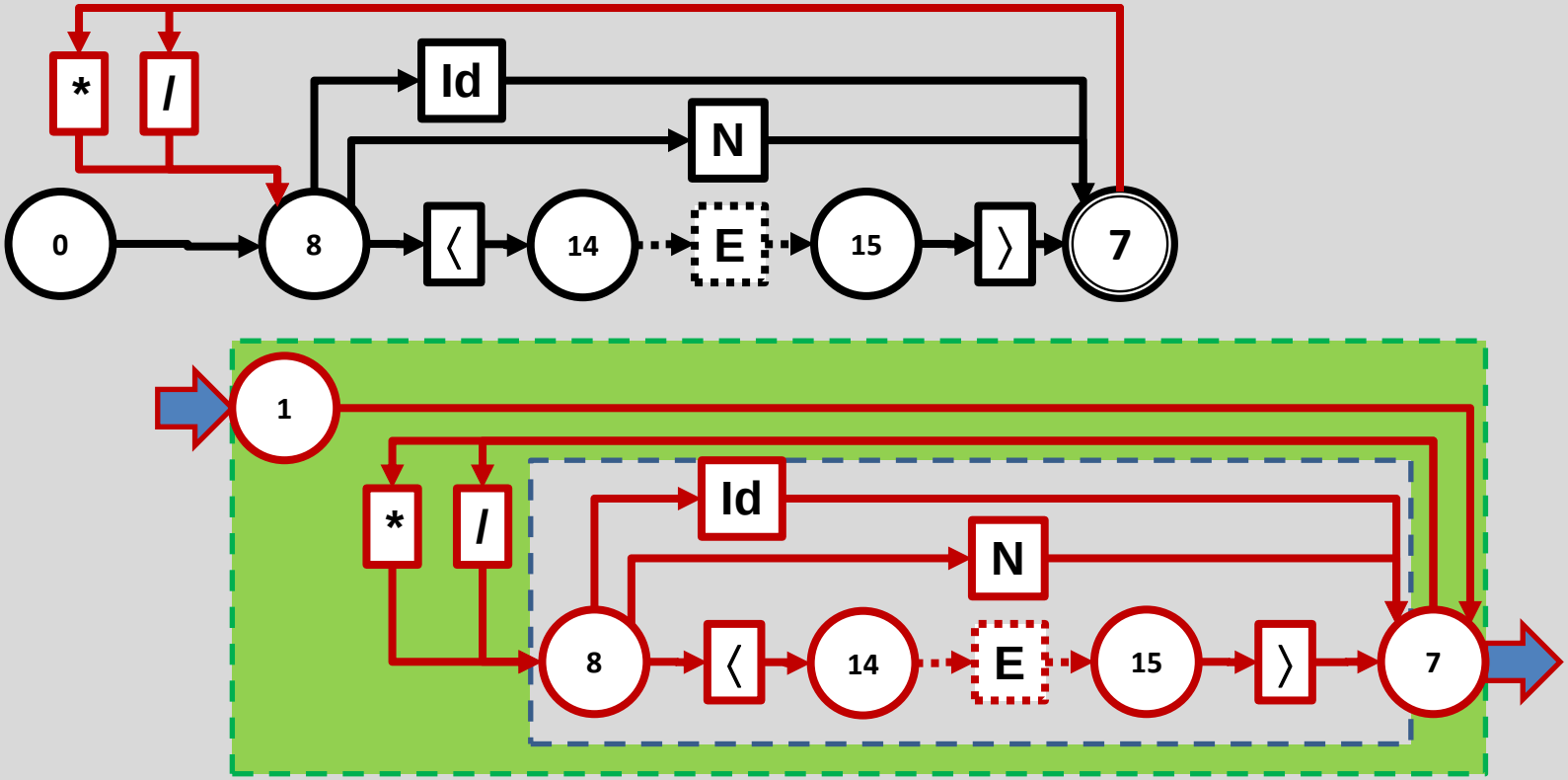


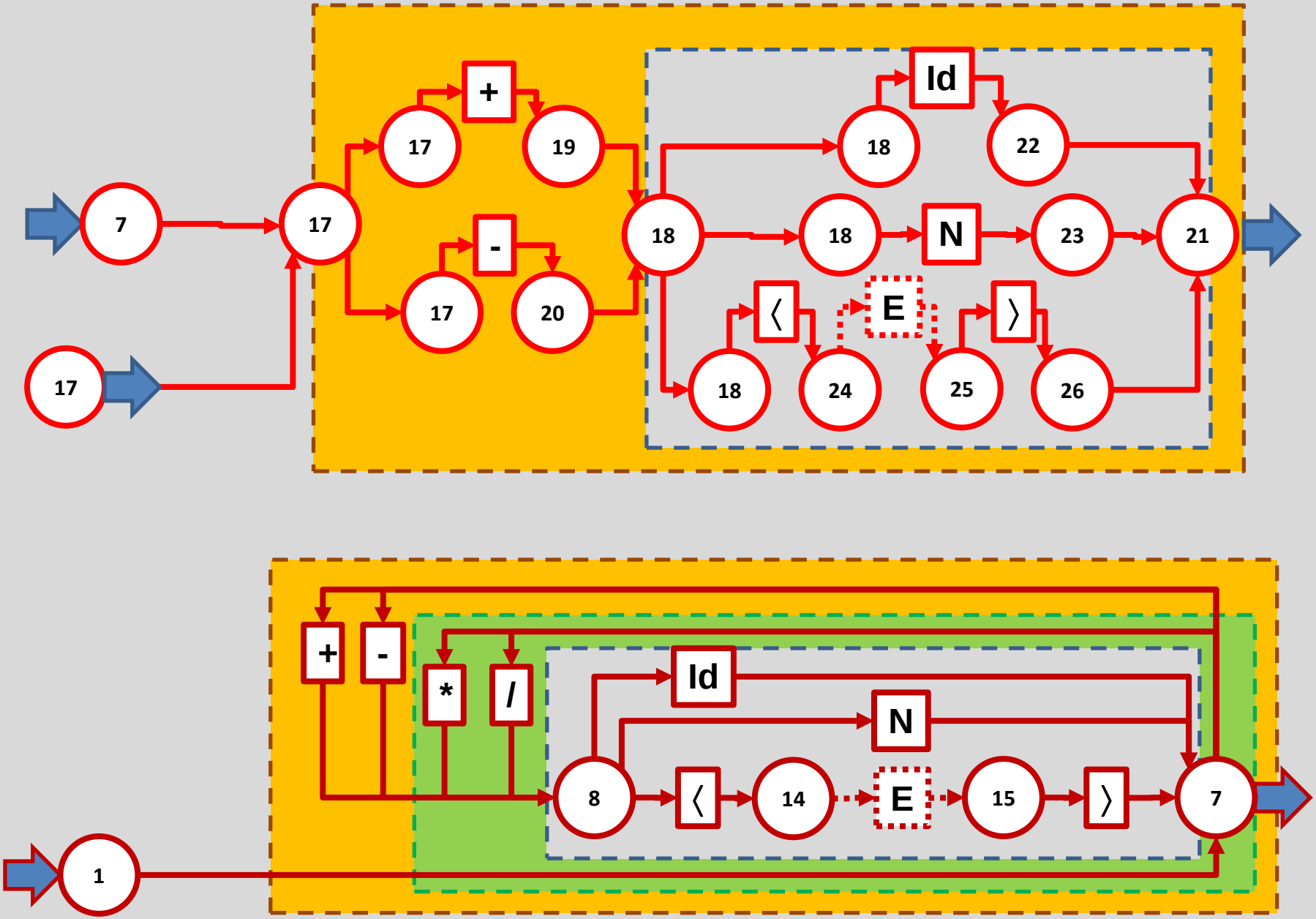
expressão

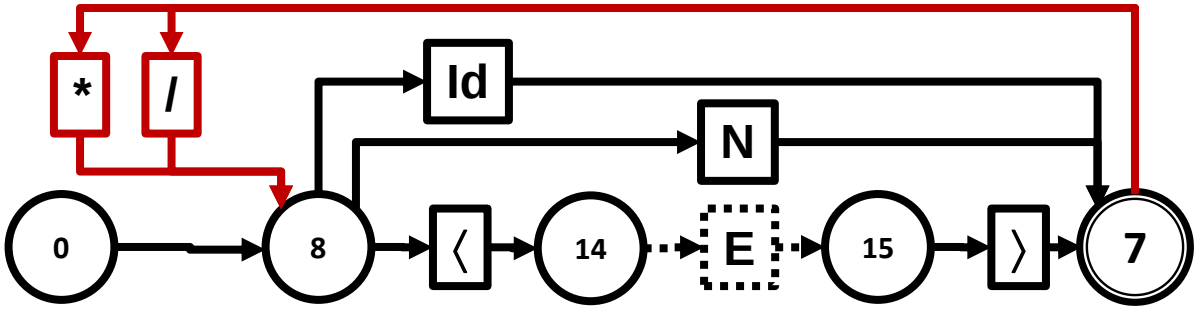

21
27
17
*
/
27
29
27
30
27
28
Id
28
32
28
N
33
31
⟨
E
⟩
28
34
35
36
21
1
*
/
Id
N
28
⟨
34
E
35
⟩
27
17

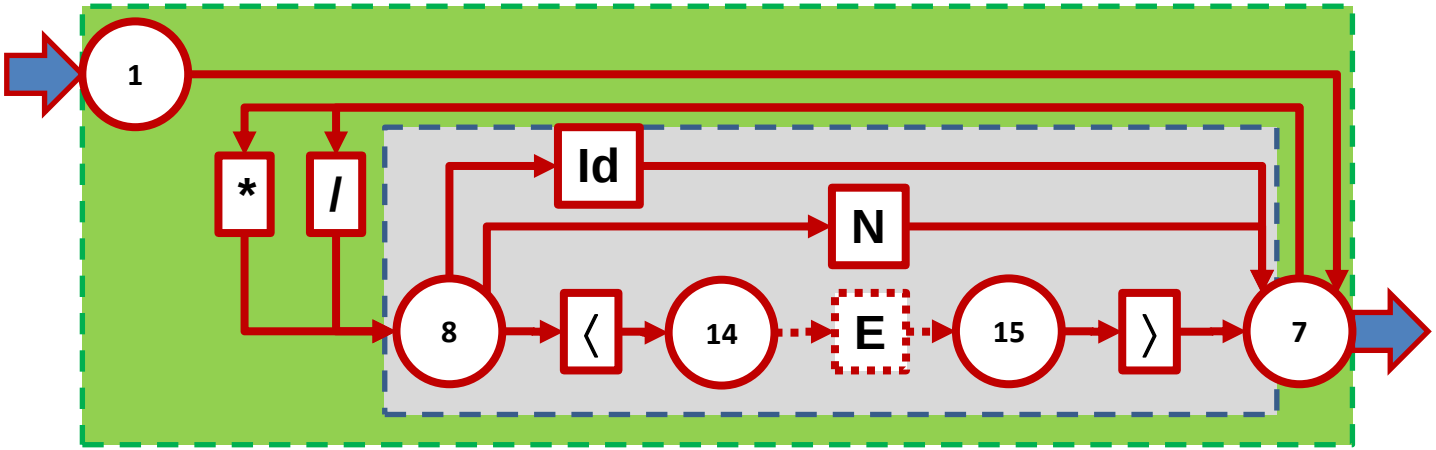
expressão

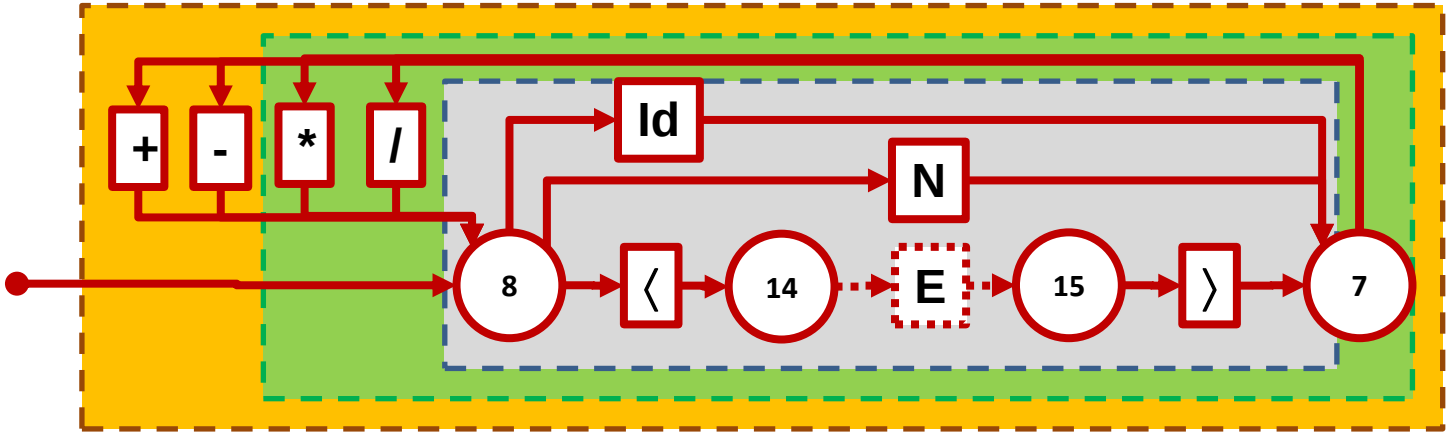


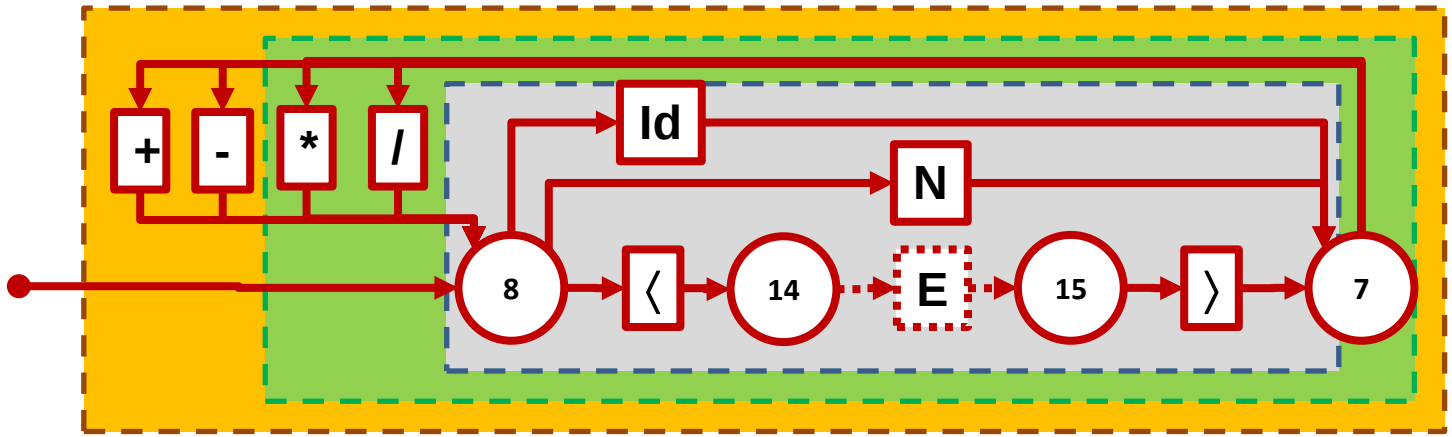

*
/
Id
N
0
8
⟨
14
E
15
⟩
7
1
*
/
Id
N
8
⟨
14
E
15
⟩
7


7
17
17
+
-
17
19
17
20
18
Id
18
22
18
N
23
21
18
⟨
E
⟩
18
24
25
26
1
+
-
*
/
Id
N
8
⟨
14
E
15
⟩
7


*
/
Id
N
0
8
⟨
14
E
15
⟩
7


1
*
/
Id
N
8
⟨
14
E
15
⟩
7

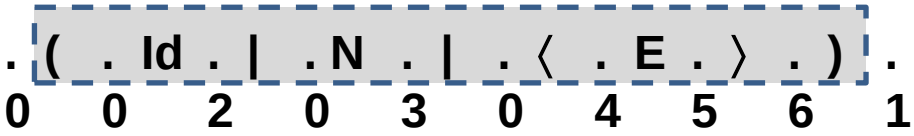

+
-
*
/
Id
N
8
⟨
14
E
15
⟩
7

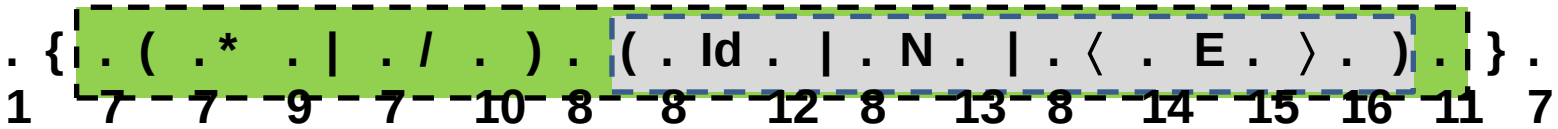


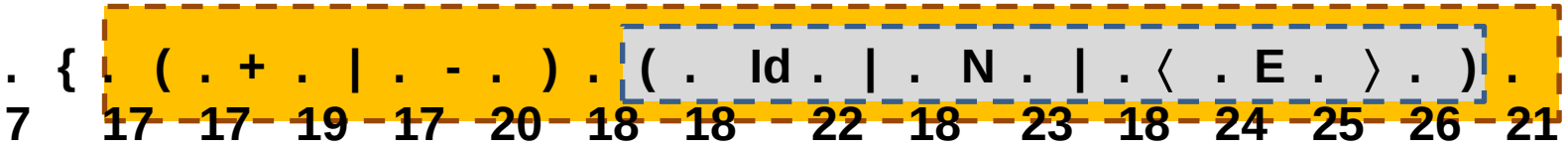
	id	N	<	E	>	+	-	*	/	outros
→ 8	7	7	14							
14				15						
15					7					
7→						8	8	8	8	

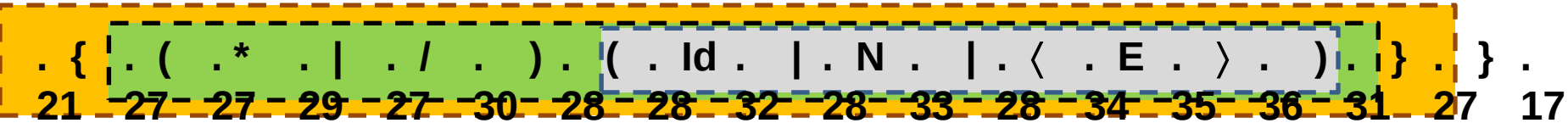
OUTRA FORMA DE OBTENÇÃO DO DIAGRAMA DE ESTADOS PARA EXPRESSÃO

expressão

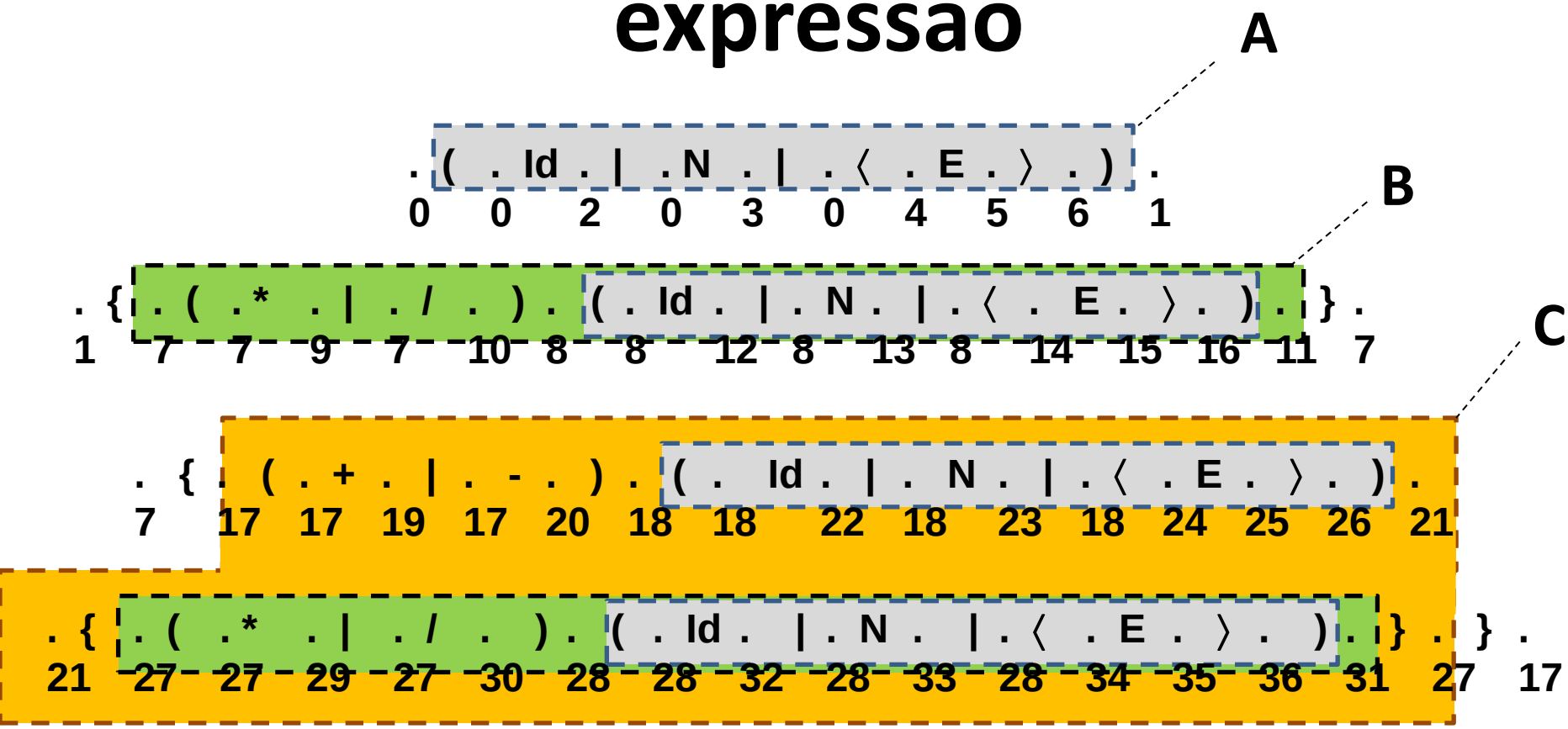

. (. Id . | . N . | . ⟨ . E . ⟩ .) .
0 0 2 0 3 0 4 5 6 1


. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } .
1 7 7 9 7 10 8 8 12 8 13 8 14 15 16 11 7


. { . (. + . | . - .) . (. Id . | . N . | . ⟨ . E . ⟩ .) .
7 17 17 19 17 20 18 18 22 18 23 18 24 25 26 21


. { . (. * . | . / .) . (. Id . | . N . | . ⟨ . E . ⟩ .) . } . } .
21 27 27 29 27 30 28 28 32 28 33 28 34 35 36 31 27 17

expressão



C = + A { * A }

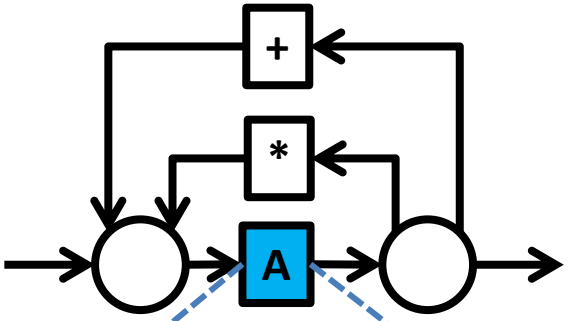
B = { * A }

Expressão = A {B} {C} = A { * A } { + B } = A { * A } { + A { * A } }

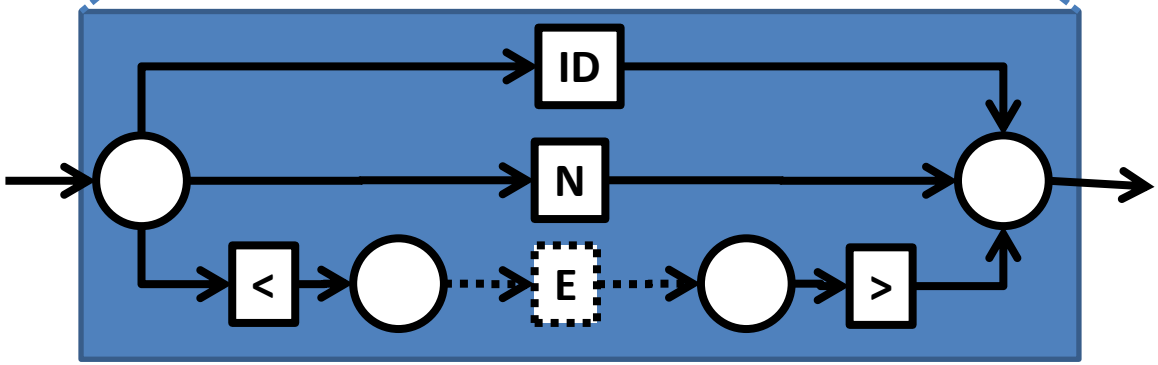
expressão

Expressão = A {B} {C} = A B { + B } = A { * A } { + A { * A } }

Expressão = A { * A } { + A { * A } } .



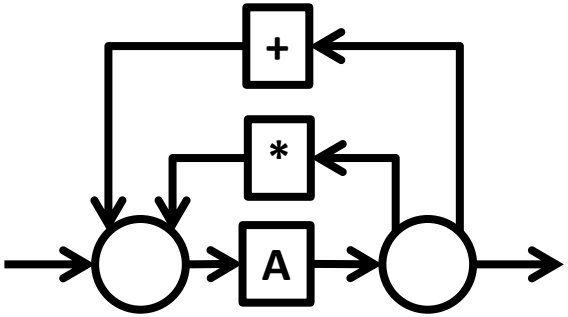
A = ID | N | < E > .

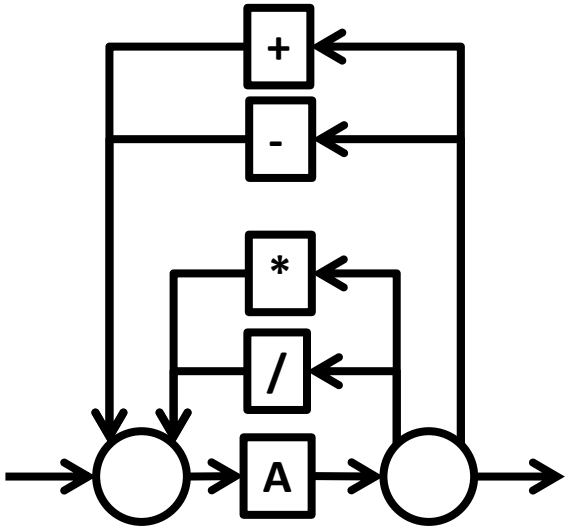


expressão

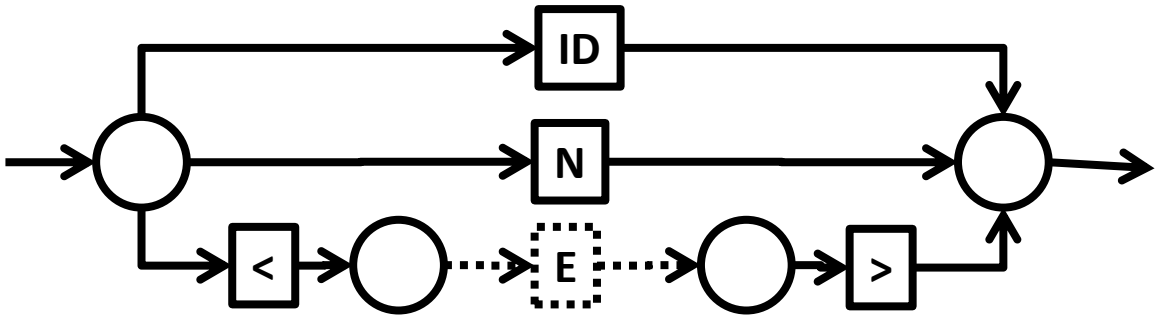
Expressão = A {B} {C} = A B { + B } = A { * A } { + { * A } }

Expressão = A { * A } { + { * A } } .

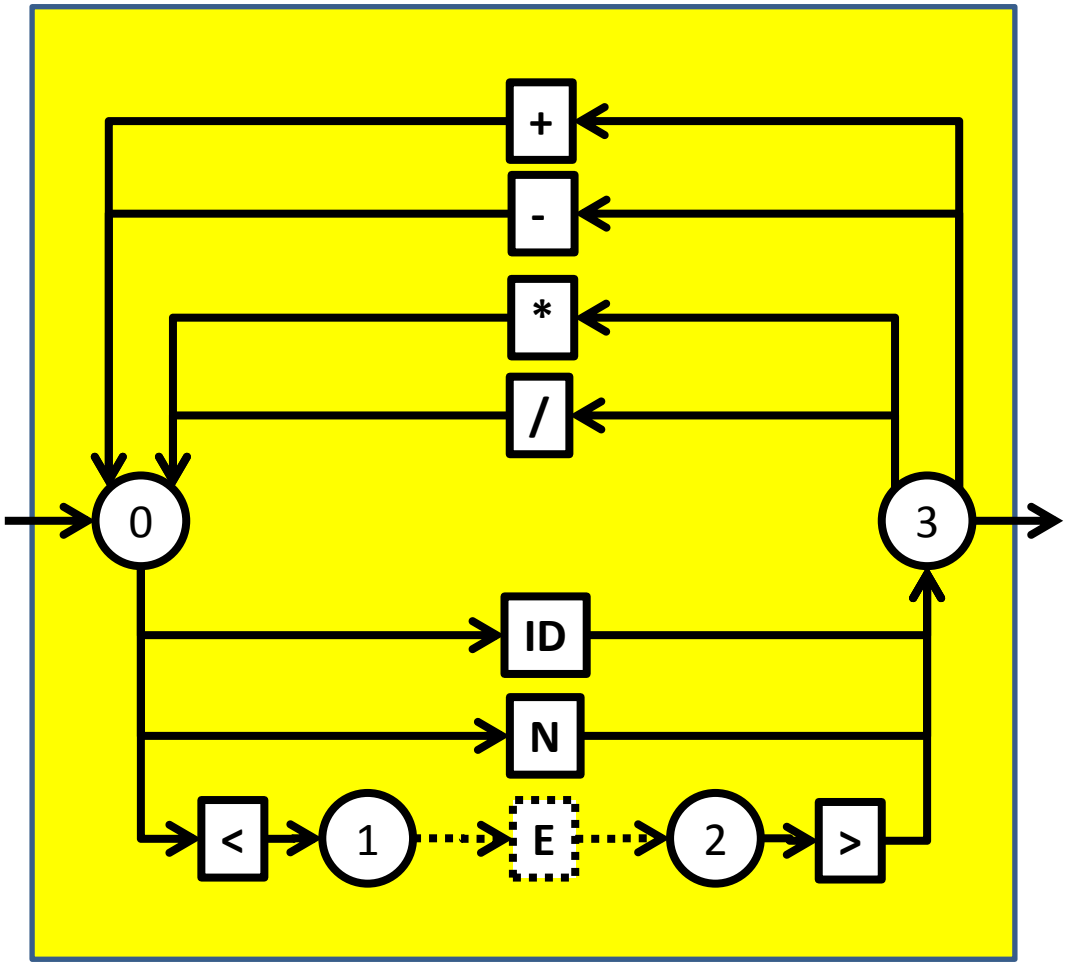




A = ID | N | < E > .



expressão



expressão

	id	N	<	E	>	+	-	*	/	outros
→ 0	3	3	1							
1				2						
2					3					
3 →						0	0	0	0	

expressão

Como afirma a teoria, o autômato produzido é exatamente o mesmo que foi produzido anteriormente, a menos dos nomes dos estados:

Tabela anterior:

	id	N	<	E	>	+	-	*	/	outros
→ 8	7	7	14							
14				15						
15					7					
7→						8	8	8	8	

Tabela recém-obtida:

	id	N	<	E	>	+	-	*	/	outros
→ 0	3	3	1							
1				2						
2					3					
3 →						0	0	0	0	

LINGUAGEM DE SAÍDA

Instruções da Linguagem de saída

Instrução	Interpretação
LDA X	Obtém o conteúdo da posição X, e deposita no acumulador.
STA X	Armazena o conteúdo do acumulador na posição X.
ADA X	Soma ao acumulador o conteúdo da posição X.
SUB X	Subtrai, no acumulador, o conteúdo da posição X.
MUL X	Multiplica, no acumulador, o conteúdo da posição X.
DIV X	Divide o acumulador pelo conteúdo da posição X.
CALL X	Chama a subrotina X (da biblioteca de execução).
BRU X	Desvia incondicionalmente para a posição X.
BRN X	Desvia para a posição X se o conteúdo do acumulador for negativo.
BRZ X	Desvia para a posição X se o conteúdo do acumulador for zero.
BRP X	Desvia para a posição X se o conteúdo do acumulador for positivo.

Pseudos da Linguagem de Saída

Pseudo-instrução	Interpretação
LBL X	Atribui ao endereço da próxima instrução o rótulo X.
DS X	Define posição de memória com conteúdo inicial igual ao número X.
END	Define o final físico do programa-fonte.
EXT X	Define o rótulo X como nome de rotina externa, chamada pelo código-objeto (rotina de biblioteca).

ROTINAS SEMÂNTICAS

Comando vazio

Nada é gerado

Nada é feito

Declarações de rótulos

Identificador$_1$: ... Identificador$_n$: comando

LBL identificador$_1$;
...
LBL identificador$_n$;
Código referente ao* comando*;

Desvio incondicional

GO TO identificador

BRU identificador;

Implementação de Rótulos e Comandos de Desvio Incondicional

Usando uma lógica similar à utilizada em montadores de um passo para o tratamento de rótulos, é possível implementar de modo trivial os desvios, gerando instruções de desvio equivalentes para os rótulos correspondentes.

`GO TO X`
`...`
`X:<comando>`
	`BRU X`
	`...`
	`X <código para o comando>`

Desvio condicional múltiplo

GO TO identificador$_0$ OF identificador$_1$, ..., identificador$_n$

```
LDA     identificador0;
BRZ     identificador1;
SUB     = 1;
...

SUB     = 1;
BRZ     identificadorn;
```

Atribuição

LET identificador := expressão

...
***código-objeto referente à* expressão**
(retorna o resultado no acumulador) ;

STA identificador;

...

Leitura sem parâmetros

READ

CALL #READ;

Leitura de lista de variáveis

READ identificador$_1$, ... , identificador$_n$

```
        CALL    #READ;
        STA     identificador1;
        CALL    #READ;
        STA     identificador2;
        ...

        CALL    #READ;
        STA     identificadorn;
```

Impressão sem parâmetros

PRINT

CALL #NEWLİNE;

Impressão de lista de valores

PRINT expressão$_1$, ... , expressão$_n$

código relativo à **expressão$_1$*;***
CALL #CONVERT;
CALL #NEWLINE;
...

código relativo à **expressão$_n$*;***
CALL #CONVERT;
CALL #NEWLINE;

Decisão (if-then-else)
1 – comparação de expressões

IF expressão$_1$ comparação expressão$_2$

THEN comando$_1$ ELSE comando$_2$

código referente à* expressão$_1$*;

STA #TEMP;

código referente à* expressão$_2$*;

SUB #TEMP;

(Continua...)

Decisão (if-then-else)
2 - Operações de comparação

Comparação > :

```
BRZ     #ELSE_k;
BRN     #ELSE_k;
BRU     #THEN_k;
                (Continua...)
```

Comparação =:

```
BRZ     #THEN_k;
BRU     #ELSE_k;
                (Continua...)
```

Comparação <:

```
BRN     #THEN_k;
BRU     #ELSE_k;
                (Continua...)
```

Decisão (if-then-else)
3 - comandos e desvios associados

LBL #THEN$_k$;

***código referente ao* comando$_1$;**

BRU #FIM$_k$;

LBL #ELSE$_k$;

***código referente ao* comando$_2$;**

LBL #FIM$_k$;

INSERÇÃO DAS ROTINAS SEMÂNTICAS NOS AUTÔMATOS

Observações

- Nos slides seguintes, as rotinas semânticas referentes a cada submáquina serão indicadas por meio de um nome numérico (p/ex. 1, 2, 3, etc.), e sua associação às transições do autômato serão denotadas na tabela de transições incluindo-se o número da rotina semântica à direita do estado-destino nas células da tabela de transições, separados por barra.
- Embora constituam esboços bastante completos, as rotinas semânticas aqui apresentadas em pseudo-código obviamente não estão prontas para serem utilizadas, ficando como exercício sua implementação definitiva na linguagem de programação escolhida para a codificação do projeto.

Expressões
1 – operações aritméticas

Topo da Pilha de operadores	Topo da Pilha de operandos	2ª posição da pilha de operandos	código-objeto a ser gerado
+	A	B	LDA B; ADA A; STA #TEMP$_i$;
-	A	B	LDA B; SUB A; STA #TEMP$_i$;
*	A	B	LDA B; MUL A; STA #TEMP$_i$;
/	A	B	LDA B; DIV A; STA #TEMP$_i$;

Expressões
2 – tabela de transições com ações

A título de exemplo, mostra-se abaixo a tabela de transições da sub-máquina de **expressão**. Nos slides seguintes, detalham-se os pseudo-códigos das rotinas semânticas a ...m, as quais, naturalmente, estão ainda por codificar.
A seta à esquerda de 0 indica que 0 é o estado inicial da submáquina.
A seta à direita de 3 indica que 3 é um estado de aceitação da submáquina.
As ações de saída são executadas no caso de ser atingida uma célula vazia ou de não ser possível executar nenhuma transição.
A ação j não fornece mensagem de erro, pois 3 é um estado de aceitação.

	id	N	<	E	>	+	-	*	/	ação
→ 0	3/a	3/b	1/c							k
1				2/d						l
2					3/e					m
3 →						0/f	0/g	0/h	0/i	j

Expressões
3 – Rotinas semânticas (1ªparte)

a	empilha (pilha de operandos, identificador encontrado)
b	empilha (pilha de operandos, número encontrado)
c	empilha (pilha de operadores, "(")
d	nada executa
e	X5: consulta (pilha de operadores, Y);
	Se Y ≠ "(" : executa GERACÓDIGO, detalhada adiante; GO TO X5;
	Se Y = "(" : desempilha (pilha de operadores, Y);
f	X6: consulta (pilha de operadores, Y);
	Se Y for "+", "-", "*" ou "/":
	executa GERACÓDIGO, detalhada adiante; GO TO X6;
	Caso contrário: empilha (pilha de operadores, "+");
g	X7: consulta (pilha de operadores, Y);
	Se Y for "+", "-", "*", ou "/":
	executa GERACÓDIGO, detalhada adiante; GO TO X7;
	Caso contrário: empilha (pilha de operadores, "-");

Expressões
4 – rotinas semânticas (2ªparte)

h	X8: consulta (pilha de operadores, Y);
	Se Y for "*" ou "/": executa GERACÓDIGO, detalhada adiante; GO TO X8;
	Caso contrário: empilha (pilha de operadores, "*");
i	X9: consulta (pilha de operadores, Y);
	Se Y for "*" ou "/": executa GERACÓDIGO, detalhada adiante; GO TO X9;
	Caso contrário: empilha (pilha de operadores, "/");
j	(* Esta rotina é associada ao final do reconhecimento da expressão *)
	X10: consulta (pilha de operadores, Y);
	Se Y não for "1": executa GERACÓDIGO, detalhada adiante, GO TO X10;
k	ERRO ("esperava-se identificador, número ou '(' neste ponto").
l	ERRO ("esperava-se uma expressão correta neste ponto").
m	ERRO ("esperava-se ')' neste ponto").

Expressões
5 – rotinas auxiliares (1ªparte)

GERACÓDIGO:

n	Desempilha (pilha de operadores, Y);
o	Desempilha (pilha de operandos, B);
p	Desempilha (pilha de operandos, A);
q	Gera ("LD", A);
r	Se Y = "+": gera (“+", B);
s	Se Y = "-": gera (“-", B);
t	Se Y = "*": gera (“*", B);
u	Se Y = "/": gera (“/", B);
v	Incrementacontador (CONTATEMP);
w	Gera (“MM", "#TEMP", CONTATEMP);
x	empilha (pilha de operandos, "#TEMP", CONTATEMP);

Programa
1 – código principal

comando$_1$; ... comando$_n$ END

código referente ao* comando$_1$*;
...
código referente ao* comando$_n$*;
CALL #STOP;

Programa
2 – área das variáveis declaradas

LBL identificador$_1$;
DS 0;
LBL identificador$_2$;
DS 0;
...
LBL identificador$_n$;
DS 0;

Programa
3 – área dos temporários utilizados

```
LBL    #TEMP;
DS     0;
LBL    #TEMP1;
DS     0;
LBL    #TEMP2;
DS     0;
...

LBL    #TEMPn;
DS     0;
```

Programa
4 – rotinas externas e fim do programa

```
EXT     #NEWLINE;
EXT     #CONVERT;
EXT     #READ;
EXT     #STOP;
END;
```

Construção das Rotinas do Ambiente de Execução

Um simples levantamento das variáveis, constantes e rotinas de biblioteca do ambiente de execução chamadas nesses exemplos dá uma boa idéia do conteúdo do ambiente de execução necessário para esse compilador.

ASPECTOS DE IMPLEMENTAÇÃO E INTEGRAÇÃO

O programa principal se limita a executar três procedimentos, em seqüência:

```
Programa principal: iniciação;
                    análise sintática (PROGRAMA);
                    finalização;
```

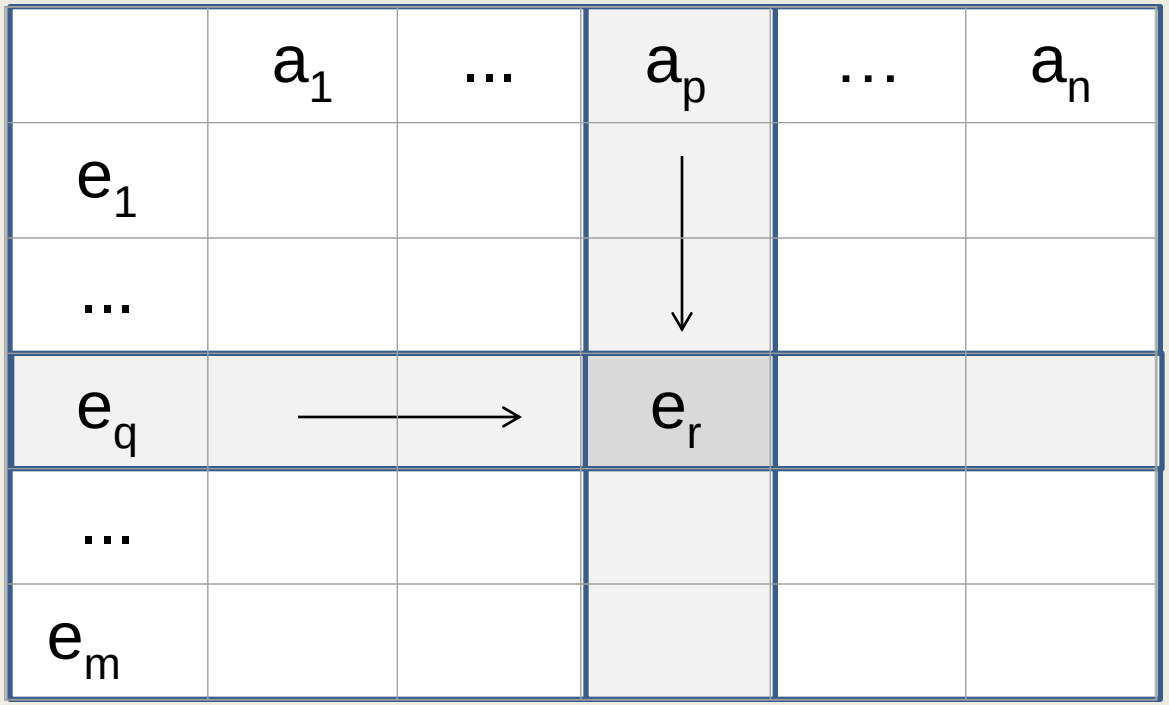
Bases de dados utilizadas

SUBMÁQUINA	**indica a submáquina correntemente em uso.**
ESTADO	**indica o estado corrente da submáquina em uso**
PILHA SINTÁTICA	**estrutura organizada em pilha, cujos elementos são pares (submáquina, estado) e registram os estados de retorno durante o reconhecimento sintático.**
TOPO SINTÁTICA	**aponta para o elemento de PILHA SINTÁTICA mais recentemente colocado nesta estrutura (topo da pilha)**
TRANSIÇÕES PROGRAMA	**tabela de transições da submáquina correspondente ao não-terminal programa.**
TRANSIÇÕES EXPRESSÃO	**idem, expressão**
TRANSIÇÕES COMANDO	**idem, comando**
AÇÕES PROGRAMA	**tabela de ações semânticas correspondentes a TRANSIÇÕES PROGRAMA**
AÇÕES EXPRESSÃO	**idem, TRANSIÇÕES EXPRESSÃO**
AÇÕES COMANDO	**idem, TRANSIÇÕES COMANDO**
ÁTOMOS PROGRAMA	**vetor de átomos e submáquinas com que transita TRANSIÇÕES PROGRAMA**
ÁTOMOS EXPRESSÃO	**idem, TRANSIÇÕES EXPRESSÃO**
ÁTOMOS COMANDO	**idem, TRANSIÇÕES COMANDO**

Códigos associados aos não-terminais

código	submáquina
1	PROGRAMA
2	EXPRESSÃO
3	COMANDO

Tabelas de transições



Codificação das células

ZERO	**indica que não há transição prevista nesta célula. Se o estado correspondente for estado final, indica que deve ser efetuado um retorno para a submáquina chamadora, ou então que terminou o processamento. Para estados não finais, isto corresponde a um erro de sintaxe.**
POSITIVO	**o número contido na célula é interpretado como o número do estado para onde a submáquina deve evoluir nesta transição.**

Pseudo-código do Reconhecedor Sintático

```
Reconhecimento Sintático (S):
        Salvar inicialmente ESTADO e SUBMÁQUINA na PILHA SINTÁTICA:
        Fazer ESTADO: =1; SUBMÁQUINA: = S;
LOOP:   Chamar o Analisador LÉXICO (TIPO, INFORMAÇÃO);
        Buscar TIPO entre os átomos desta submáquina, associando-a assim à coluna da tab.de transições;
        (* se COLUNA = 0, isso indica que não encontrou o átomo *)
        Obter CÉLULA: = tabela de transições corrente [ESTADO, COLUNA]
        Se COLUNA ≠ 0
            então Se CÉLULA ≠ 0
                        então executar rotina indicada em tabela de ações semânticas [ESTADO, COLUNA];
                              fazer ESTADO: = CÉLULA;
                              executar a ação sintática associada à transição, se existir
                        caso contrário,  Se tabela de ações semânticas [ESTADO, 0] indicar um estado final,
                                              então retornar (* à submáquina chamadora *)
                                              se não, emitir mensagem de erro e terminar o processamento.
            se não (*COLUNA = 0),
                        não consumir o átomo;
                        se o estado for final, então retornar à submáquina chamadora.
                                               se não, emitir mensagem de erro e terminar o processamento.
```

FIM