Construção Automática de Autômatos de Pilha Estruturados a partir de gramáticas em Notação de Wirth tradicional

PCS 3566 / PCS 3866 – Aula 06

Projeto de um meta-reconhecedor sintático para linguagens livres de contexto

Apresentação

- Esta apresentação complementa a anterior, acrescentando ao reconhecedor sintático da Notação de Wirth as rotinas semânticas que faltam para convertê-lo em um meta-reconhecedor, ou seja, em um gerador de reconhecedores sintáticos para linguagens definidas através de gramáticas em Notação de Wirth que lhe são apresentadas como entrada.

- Recomenda-se o exame e estudo de um material complementar que disponibiliza uma execução passo a passo do algoritmo desenvolvido adiante, nesta aula, bem como a sua aplicação à geração de um reconhecedor sintático particular, e um pequeno conjunto de exemplos de utilização passo a passo deste reconhecedor automaticamente gerado.

Desenvolvimento de meta-reconhecedor para linguagens livres de contexto

Projeto de uma ferramenta
para a obtenção automática de
Autômatos de Pilha Estruturados
a partir de Gramáticas em Notação de Wirth

Gramática de Wirth que descreve a Notação de Wirth tradicional

- A sintaxe da Notação de Wirth tradicional, adotada para este projeto, está descrita abaixo, em Notação de Wirth:

Sintaxe

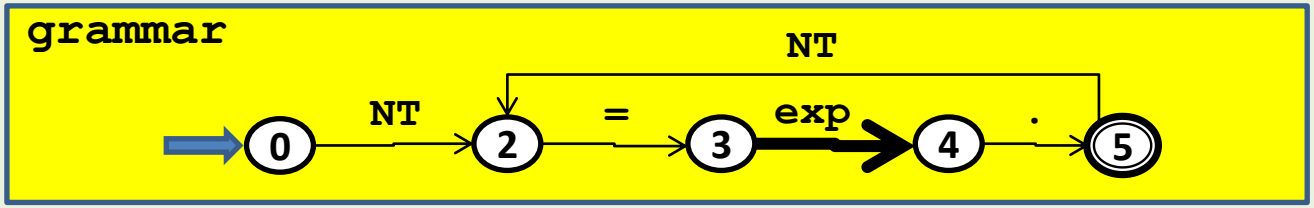
```
grammar = rule { rule } .
rule = NT "=" exp "." .
exp = term { "|" term } .
term = factor { factor } .
factor = NT | T | “ε” | "(" exp ")"
       | "[" exp "]" | "{" exp "}" .
```

Gramática Léxica

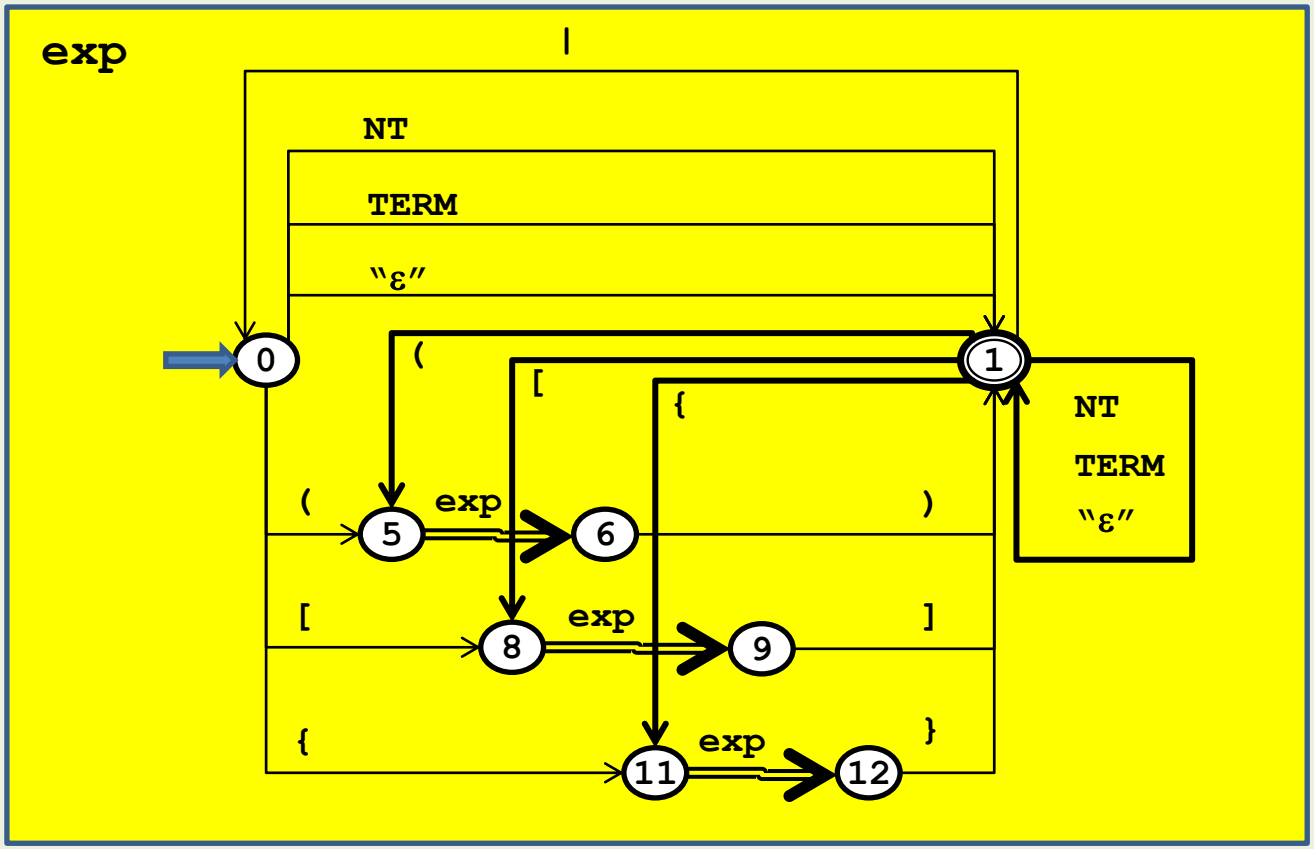
```
NT = letter { letter | digit } .
T = """ { any_character } """ .
```

Autômato final:

- O autômato de pilha estruturado construído na aula anterior a partir da gramática sintática que define a Notação de Wirth é o seguinte:
- Sub-máquina **grammar**:



- Sub-máquina **exp**:



Projeto de um Meta-Reconhecedor

- **Objetivo**: Neste projeto procura-se obter, de forma automática, um reconhecedor sintático para linguagens livres de contexto formalmente definidas através de gramáticas expressas na Notação de Wirth tradicional.

- **Método**: Para isso, o reconhecedor da Notação de Wirth é instrumentado com rotinas semânticas que deverão gerar o autômato correspondente às gramáticas por ele reconhecidas, seguindo o mesmo método que vem sendo aplicado manualmente.

Observação Importante

- O método escolhido associa uma rotina semântica (de tratamento) a cada uma das transições do autômato reconhecedor, de modo que, a cada vez que a **transição** ocorrer, a correspondente **rotina semântica** seja executada.
- É imediato o paralelo que se pode fazer entre a ocorrência de cada uma dessas **transições** e um **evento**, e entre a execução da **rotina semântica** associada à transição e a **rotina de tratamento** associada à ocorrência desse evento.
- Assim sendo, é evidente que na **prática a implementação desse meta-reconhecedor,** já identificado como um sistema reativo guiado por eventos, **seja feita utilizando como base um motor de eventos**: tomando como eventos as transições do autômato, e como rotinas de tratamento, as rotinas semânticas. O mesmo raciocínio se aplica a outros processadores de linguagens: interpretadores e compiladores.

Elementos a serem manipulados

- Uma **Pilha** (cada elemento = par ordenado)
- Ponteiro para a cadeia de entrada da gramática (aponta o **símbolo de entrada** em análise)
- **Contador de estados** já atribuídos (um número de série, inteiro)
- **Autômato de saída** (na forma de conjunto de estados e transições)

Algoritmo de geração

- Implementa-se um conjunto de ações semânticas que, à medida que os *tokens* de uma gramática em Notação de Wirth são utilizados, vai executando o processo de mapeamento da gramática no autômato equivalente.
  - **terminais** geram transições com consumo de átomo.
  - **não-terminais** geram chamadas de sub-máquinas.
  - **finais de regra** geram retornos de sub-máquina.
  - **agrupamentos** entre parênteses ou colchetes impõem o mesmo estado de entrada, e designam um só estado de saída para onde afluem os estados que finalizam suas opções
  - **agrupamentos entre chaves** designam um estado de saída comum, usado também para (re)iniciar as iterações.

Variáveis de trabalho

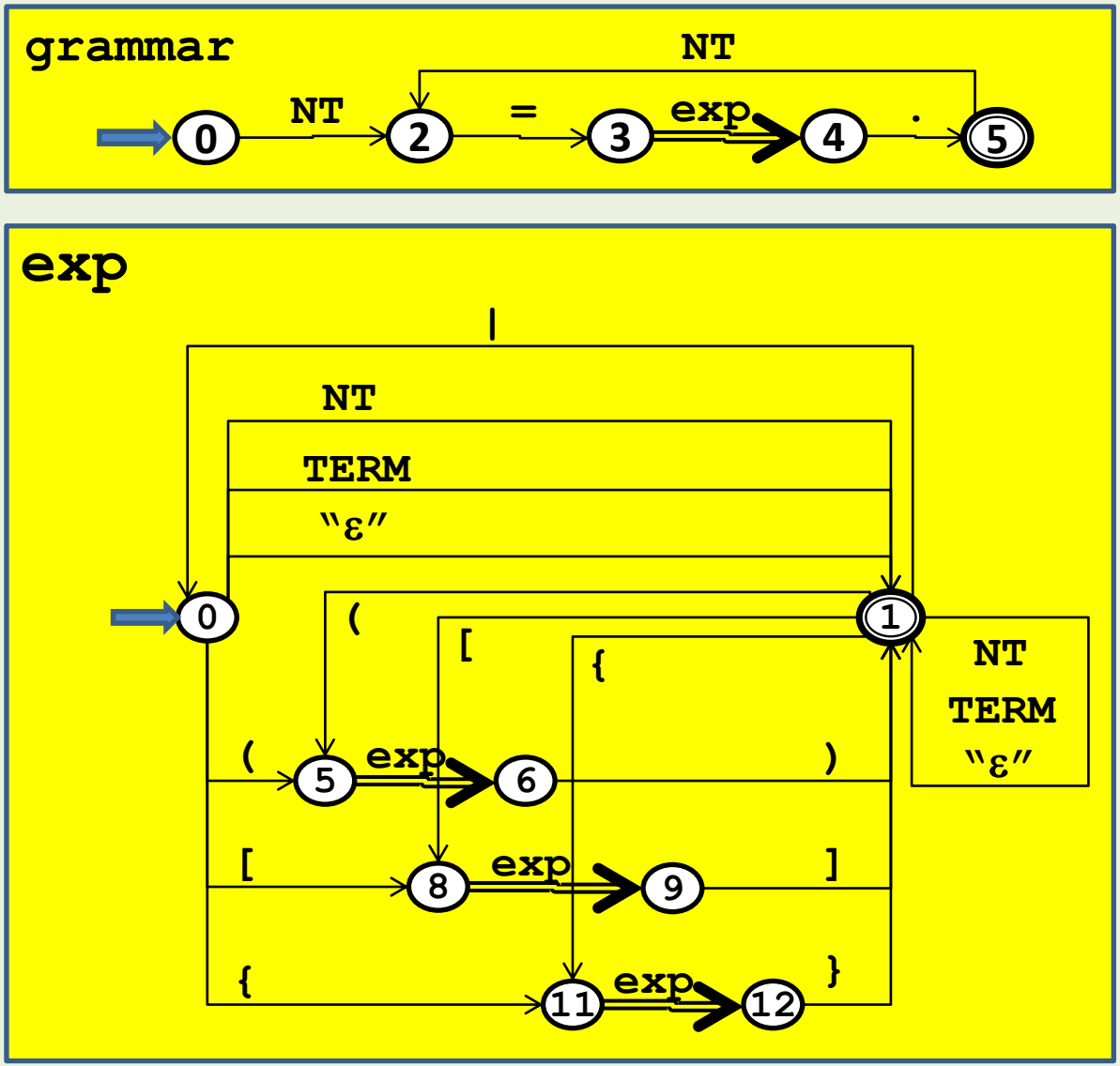
- A variável **Estado Corrente** indica a qual estado do autômato gerado corresponde o ponto em análise da expressão de Wirth.
- A variável **Próximo Estado** associa um novo estado a algum ponto da expressão. É um número de série, que deve ser incrementado a cada nova designação de estado à saída de um agrupamento, ou ao estado-destino no caso de consumo de átomo ou de chamada de submáquina.
- Uma **Pilha** associa, a cada agrupamento em análise, um par (estado corrente, próximo estado) que lhe caracteriza o escopo.

Método adotado

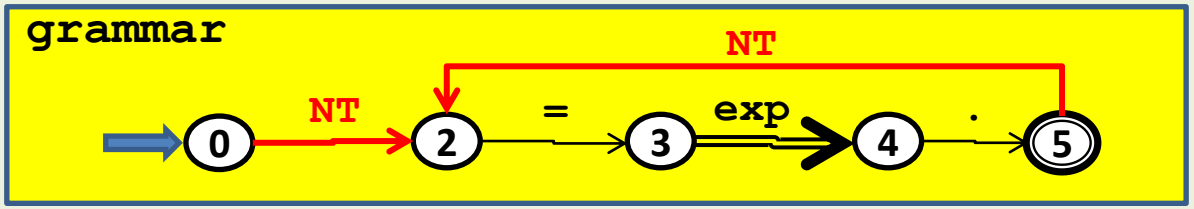
- A cada transição sobre a expressão de Wirth, associa-se ao ponto analisado um **Estado Corrente** , de acordo com o contexto em análise.
- Cada terminal ou não-terminal cria uma transição entre o **Estado Corrente** e um novo **Próximo Estado**
- Identificado na gramática de entrada o **início de um agrupamento** (entre parênteses, colchetes ou chaves), um novo escopo deve ser definido, devendo-se antes salvar em uma pilha o par de estados associado ao escopo corrente.
- Identificado na gramática de entrada o **final de um agrupamento**, o escopo a ele correspondente deve ser encerrado, retornando-se ao escopo anterior, no qual estava aninhado. Para isso, a partir da informação previamente salva no topo da pilha, restaura-se o par de estados que caracteriza esse escopo anterior, que volta então a ser o escopo corrente.

- Este autômato de pilha estruturado é capaz de reconhecer gramáticas denotadas na notação de Wirth.
- Notar a obrigatoriedade de ao menos uma regra gramatical
- A submáquina `grammar` é a submáquina principal
- A submáquina `exp` reconhece expressões escritas na Notação de Wirth.
- Nos slides seguintes, são acrescentadas sucessivas rotinas semânticas a essas transições do autômato, de acordo com o que se deseja gerar como saída.

(sub-máquinas `grammar` e `exp`)



Rotinas semânticas (**A**)



foi encontrado o nome do não-terminal a ser definido - criar nova sub-máquina

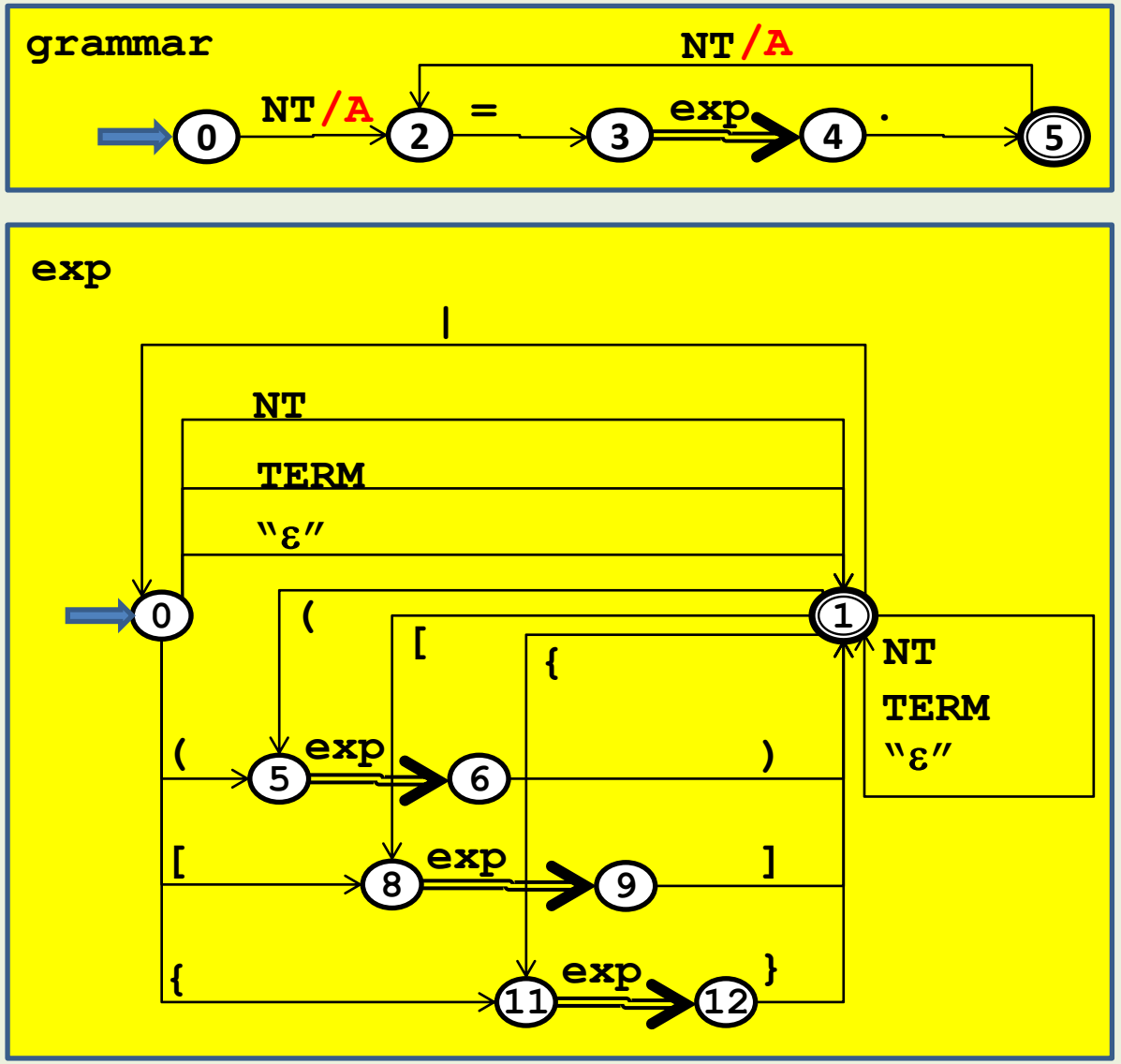
- associar às duas transições acima (0-2 e 5-2) uma mesma rotina semântica **A** , responsável por:

A:
- esvaziar a pilha
- iniciar o estado corrente em 0 e o contador em 1
- gerar uma transição em vazio de retorno de submáquina, a partir do estado 1

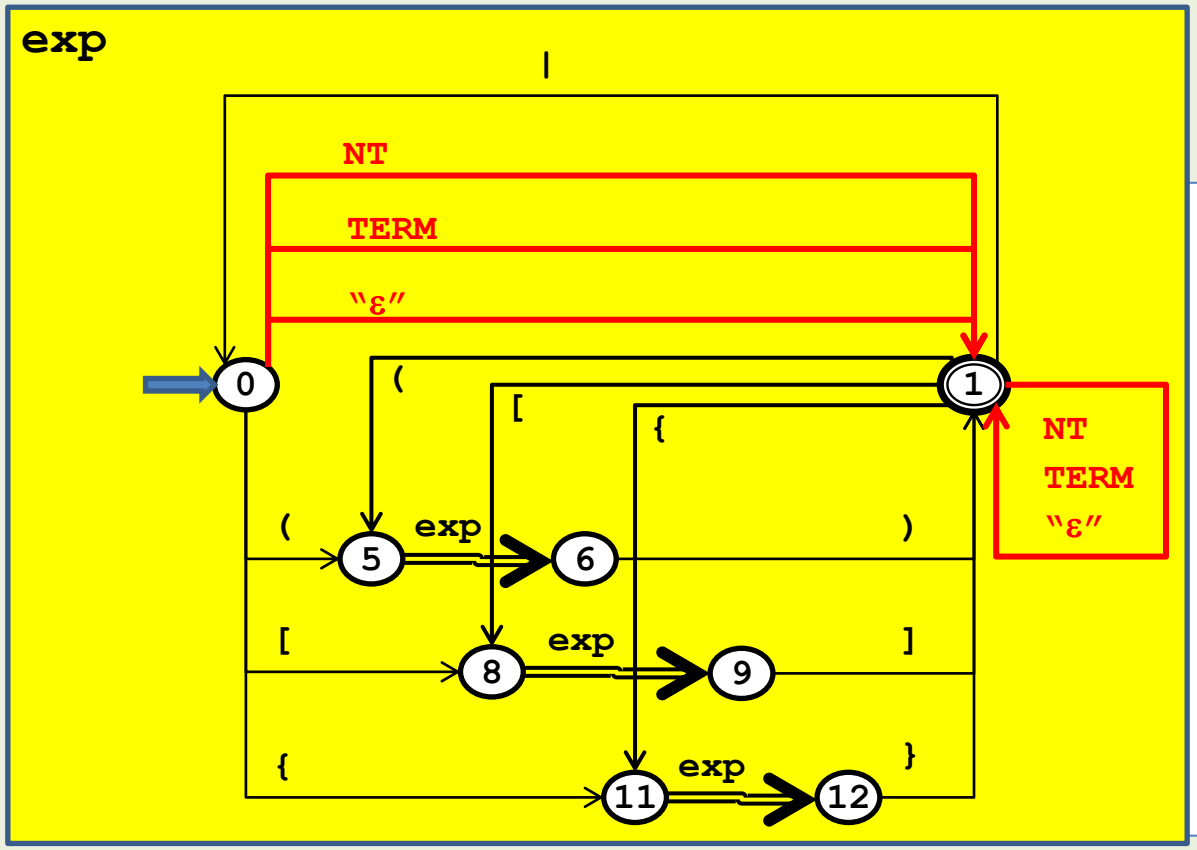
(obs.: em tempo de execução, o endereço de retorno a ser usado nessa transição se encontra no topo da pilha do autômato, e é desempilhado ao ser aplicada essa transição)

Evolução do meta-reconhecedor

Até aqui foi incluída a rotina semântica **A**



Rotinas semânticas (B)



encontrado um elemento terminal, não-terminal e símbolo de vazio - gerar a transição correspondente

- associar às duas transições acima (0-1 e 1-1) uma mesma rotina semântica **B**, responsável por:

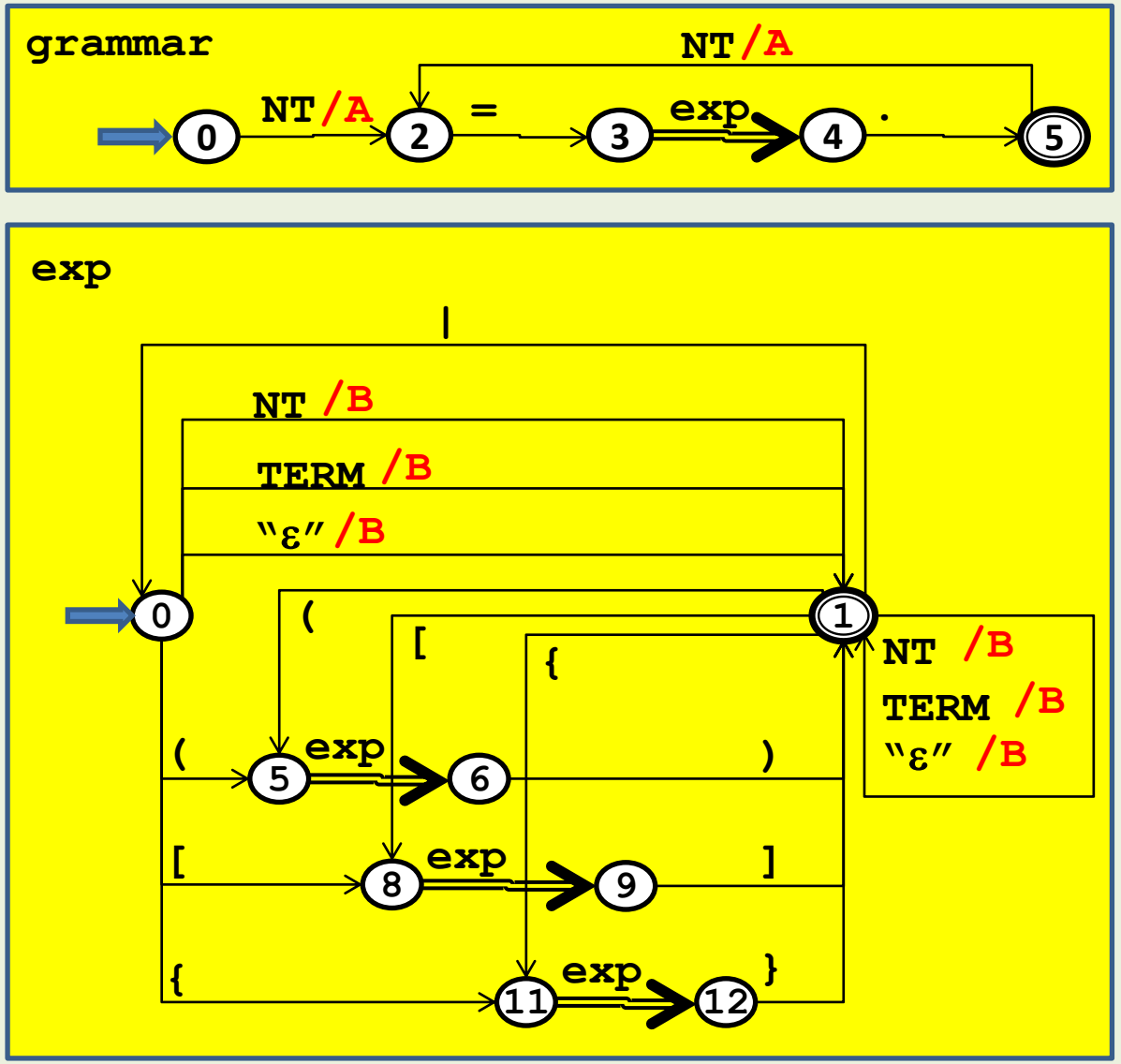
B:

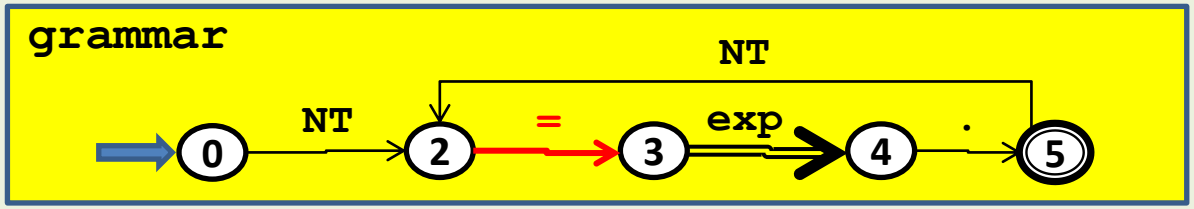
- gerar uma transição rotulada com o terminal (consumo de átomo), não-terminal (chamada de sub-máquina) ou vazio (transição em vazio), partindo do estado corrente para um novo estado, indicado pelo contador.

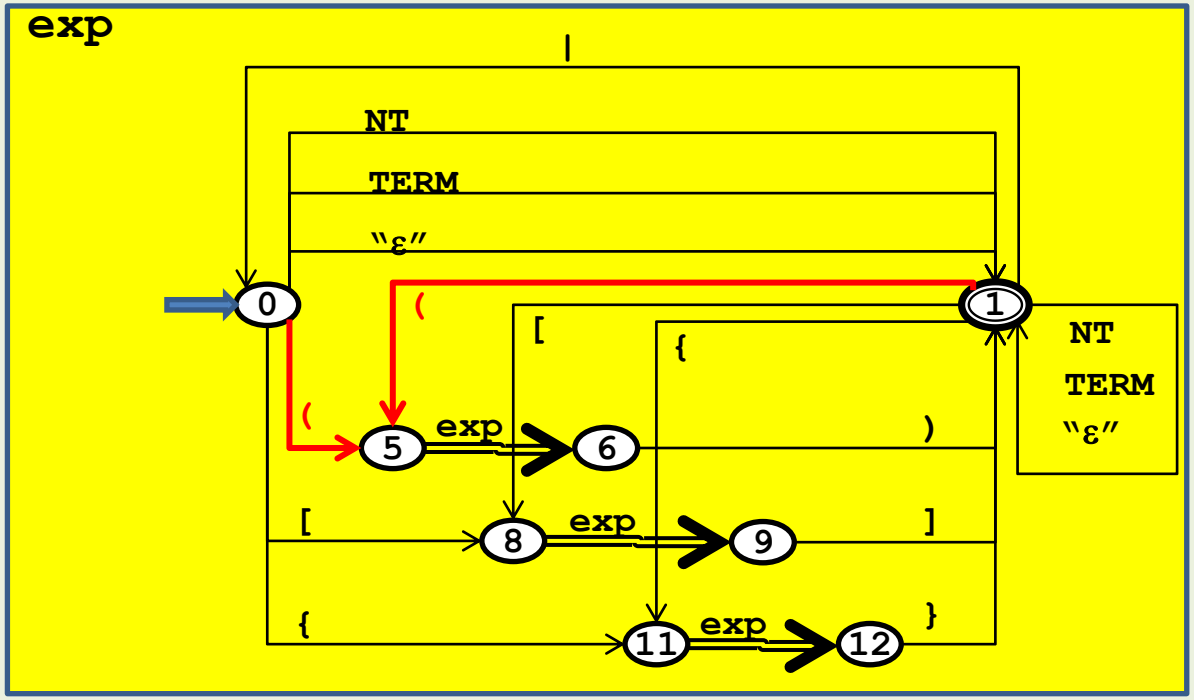
  Obs.: no caso de chamada de submáquina, em tempo de execução haverá o empilhamento do estado de retorno, e o desvio para o estado inicial da submáquina chamada.
- Atualizar o estado corrente com o valor do contador.
- Incrementar o contador.

Evolução do meta-reconhecedor

Até aqui foram incluídas as rotinas semânticas **A** e **B**







Rotinas semânticas (C)

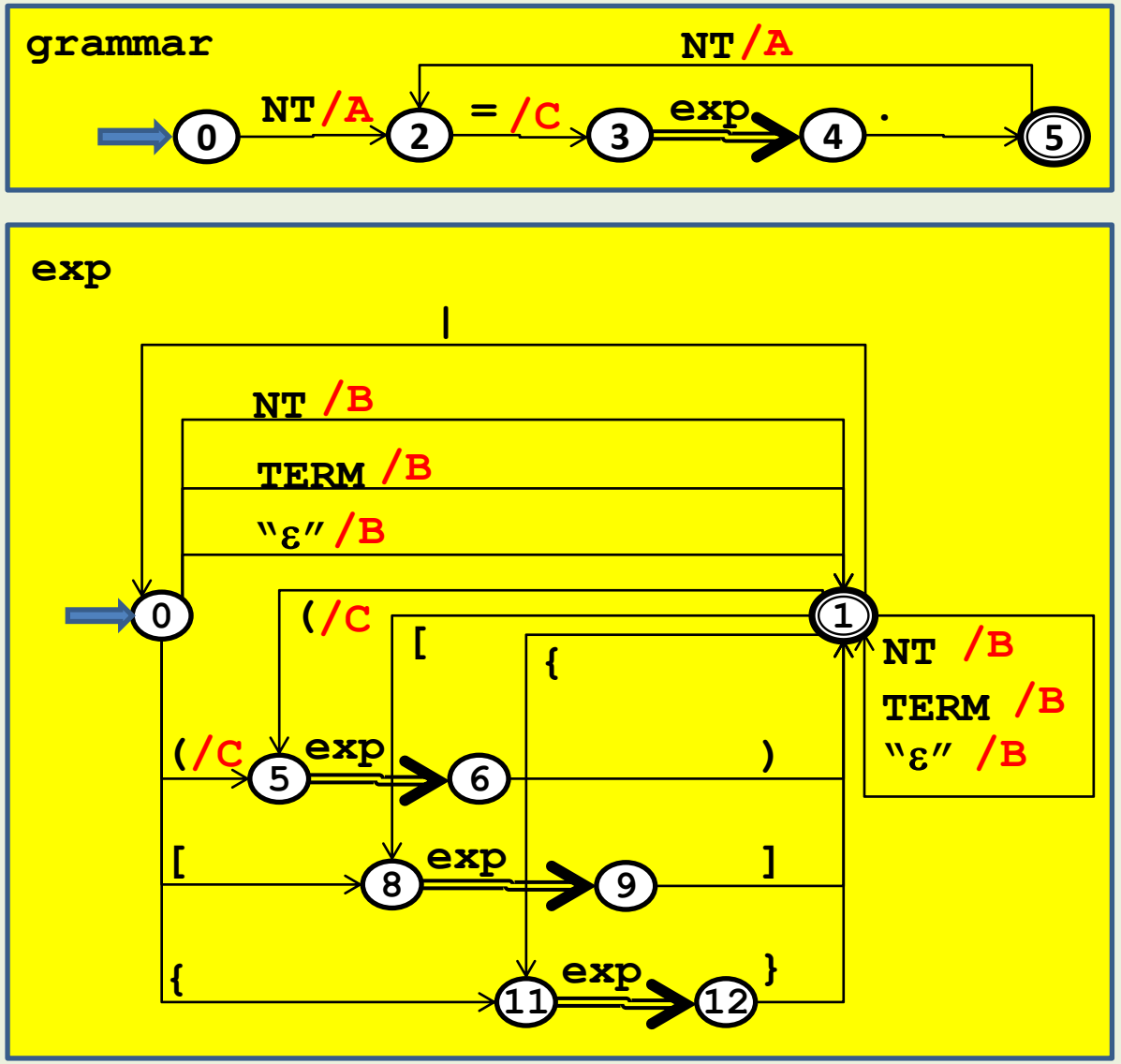
Abertura de parênteses ou início de definição (=) - início de novo escopo de elementos agrupados

- Ao ser encontrado um início de agrupamento (0-5 e 1-5):

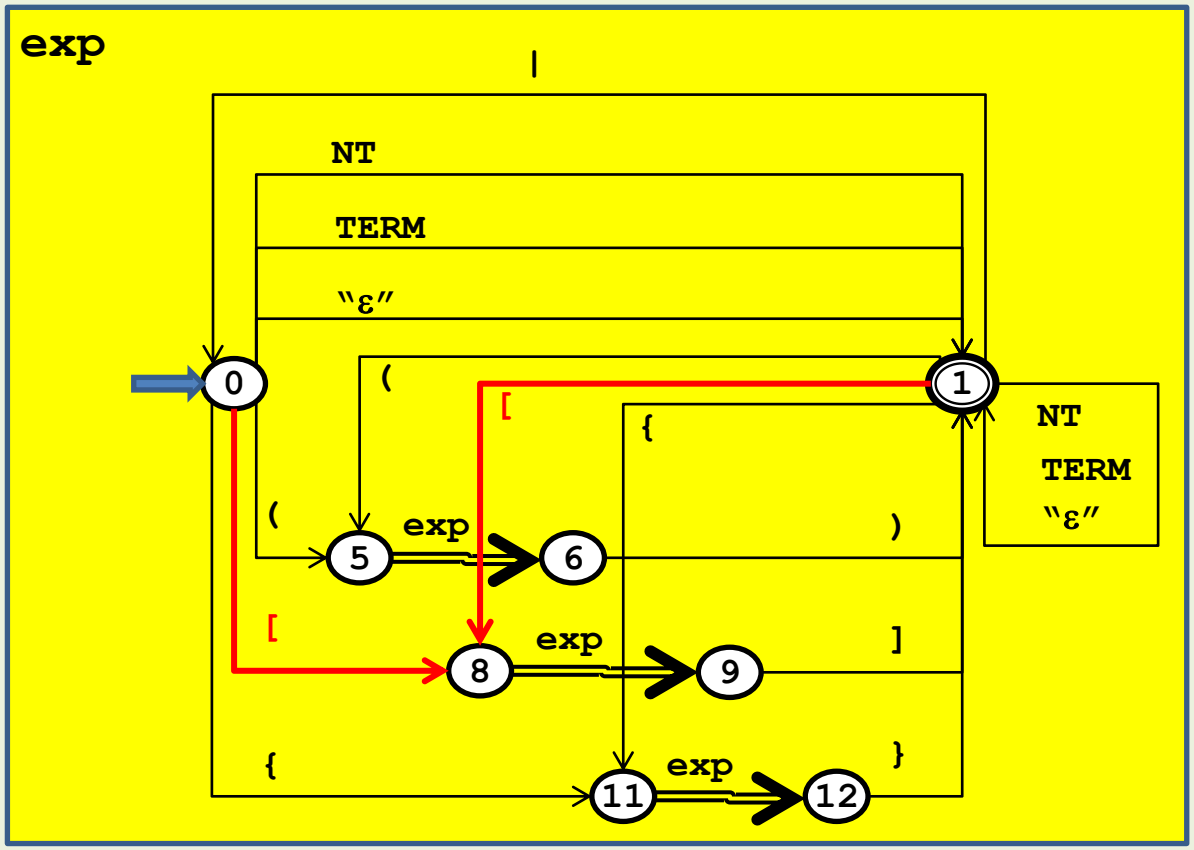
C :
- manter o estado corrente
- empilhar o par de informações
  (estado corrente , contador)
- incrementar o contador

Evolução do meta-reconhecedor

Até aqui foram incluídas as rotinas semânticas **A**, **B** e **C**



Rotinas semânticas (D)



Abertura de colchetes - início de novo escopo de elementos agrupados

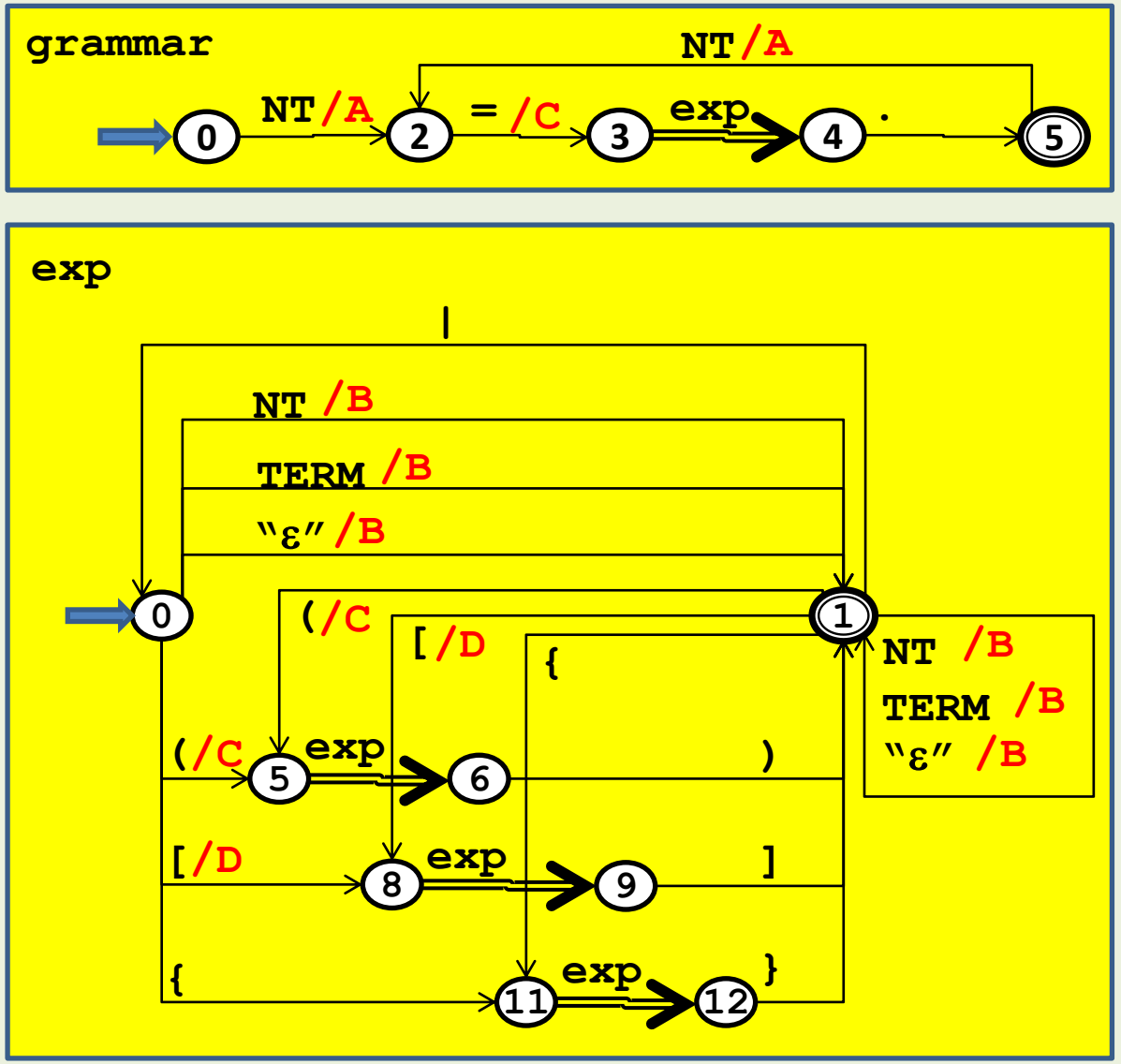
- Ao ser encontrada uma abertura de colchetes (0-8 e 1-8):

D :

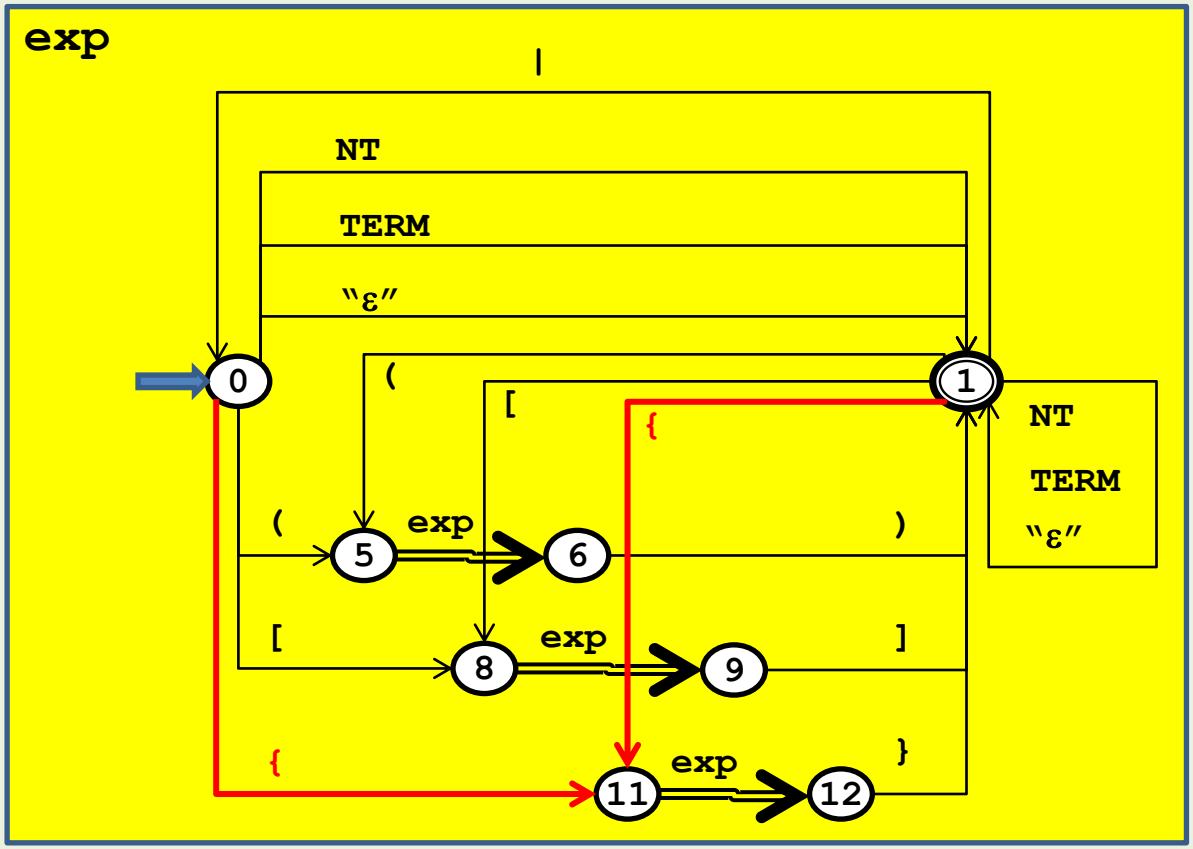
- manter o estado corrente
- gerar uma transição em vazio do estado corrente para o representado pelo contador
- empilhar o par (estado corrente , contador)
- incrementar o contador

Evolução do meta-reconhecedor

Até aqui foram incluídas as rotinas semânticas **A**, **B**, **C** e **D**



Rotinas semânticas (E)



Abertura de chaves - início de novo escopo de elementos agrupados

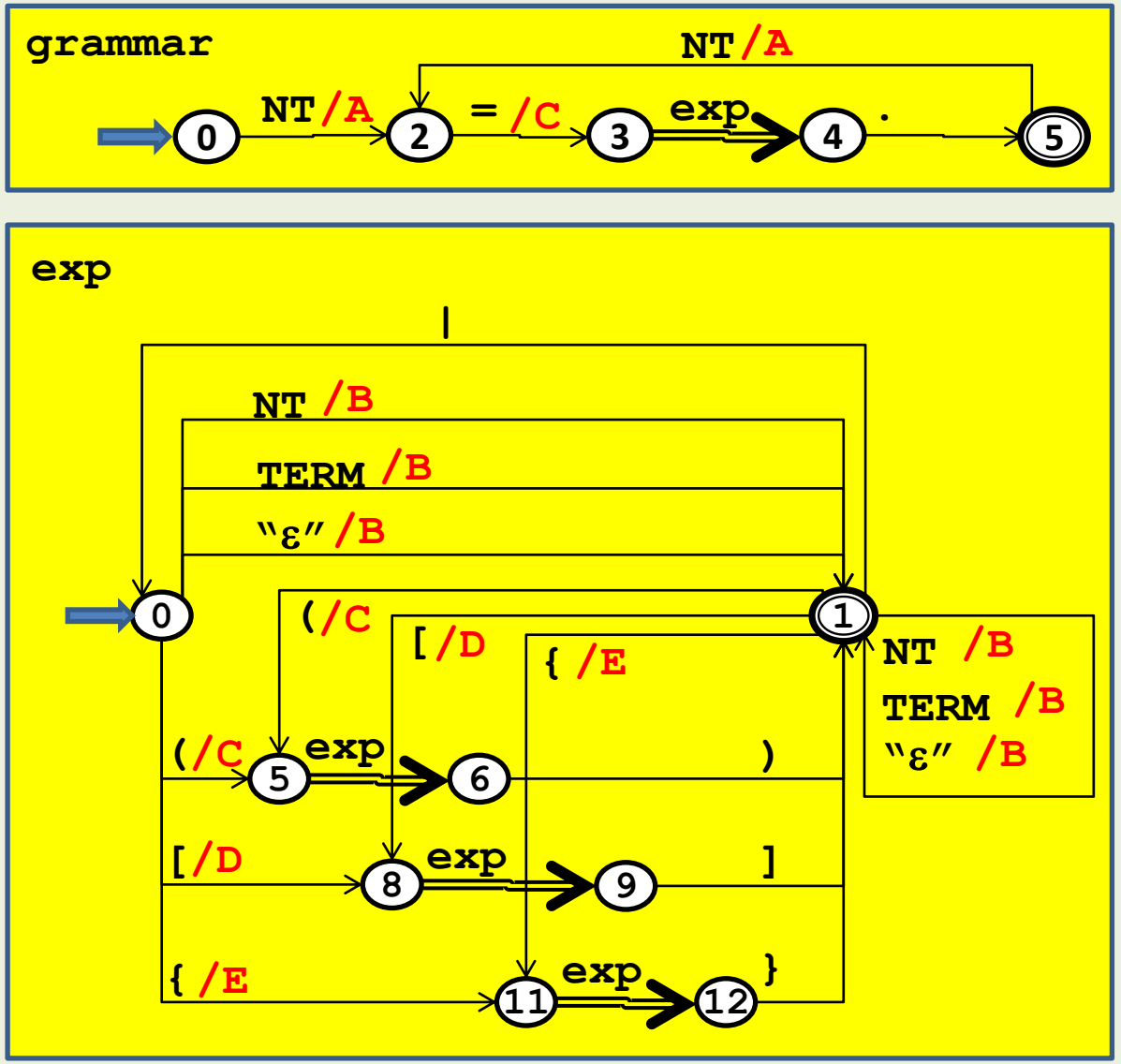
- Ao ser encontrada uma abertura de chaves (0-11 e 1-11):

E :

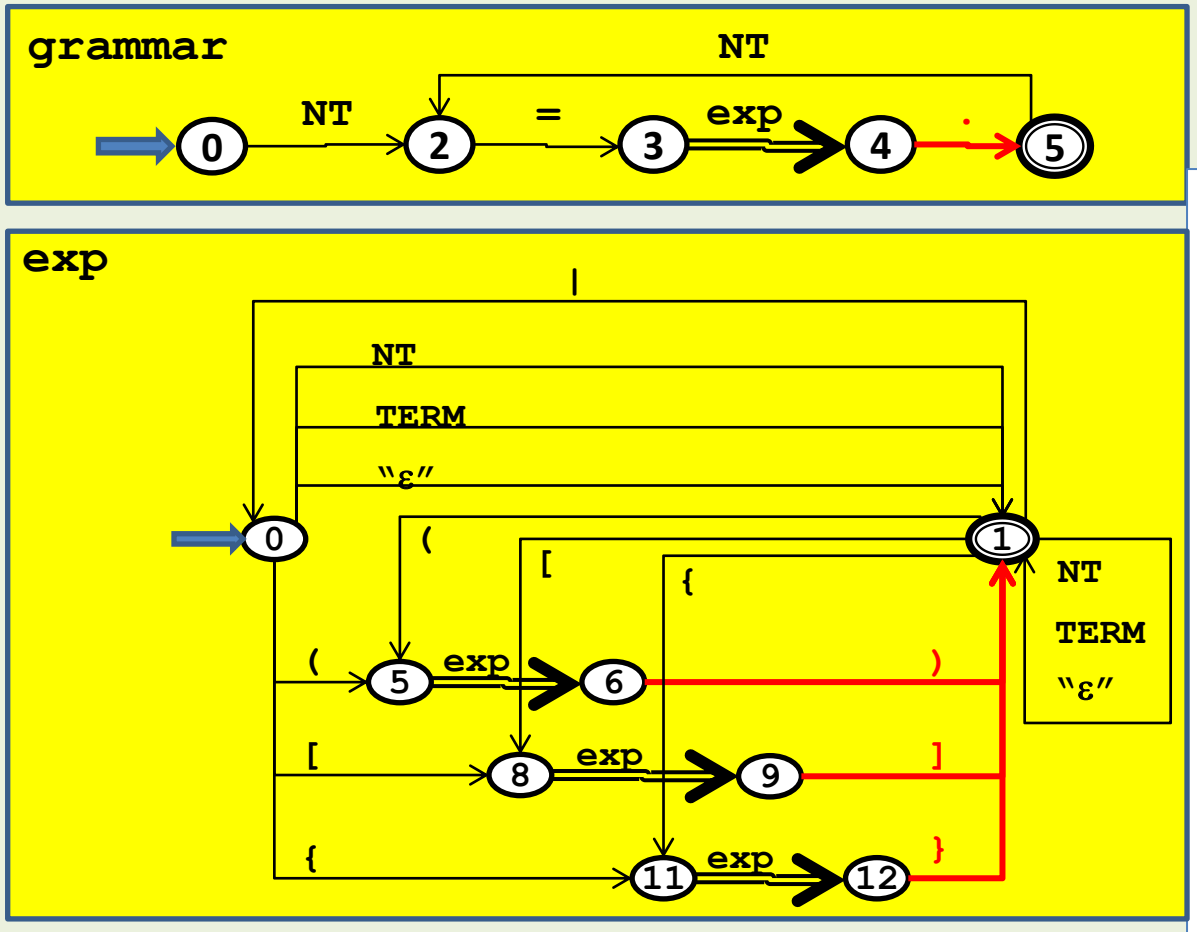
- gerar uma transição em vazio do estado corrente para o estado representado pelo contador
- alterar o estado corrente para o estado representado pelo contador
- empilhar o par (contador , contador)
- incrementar o contador

Evolução do meta-reconhecedor

Até aqui foram incluídas as rotinas semânticas **A**, **B**, **C**, **D** e **E**



Rotinas semânticas (F)



Fechamento de parênteses, colchetes ou chaves, ou final de definição (.) - final do escopo do agrupamento corrente.

(no caso da transição 4-5, o par de delimitadores é (= , .)

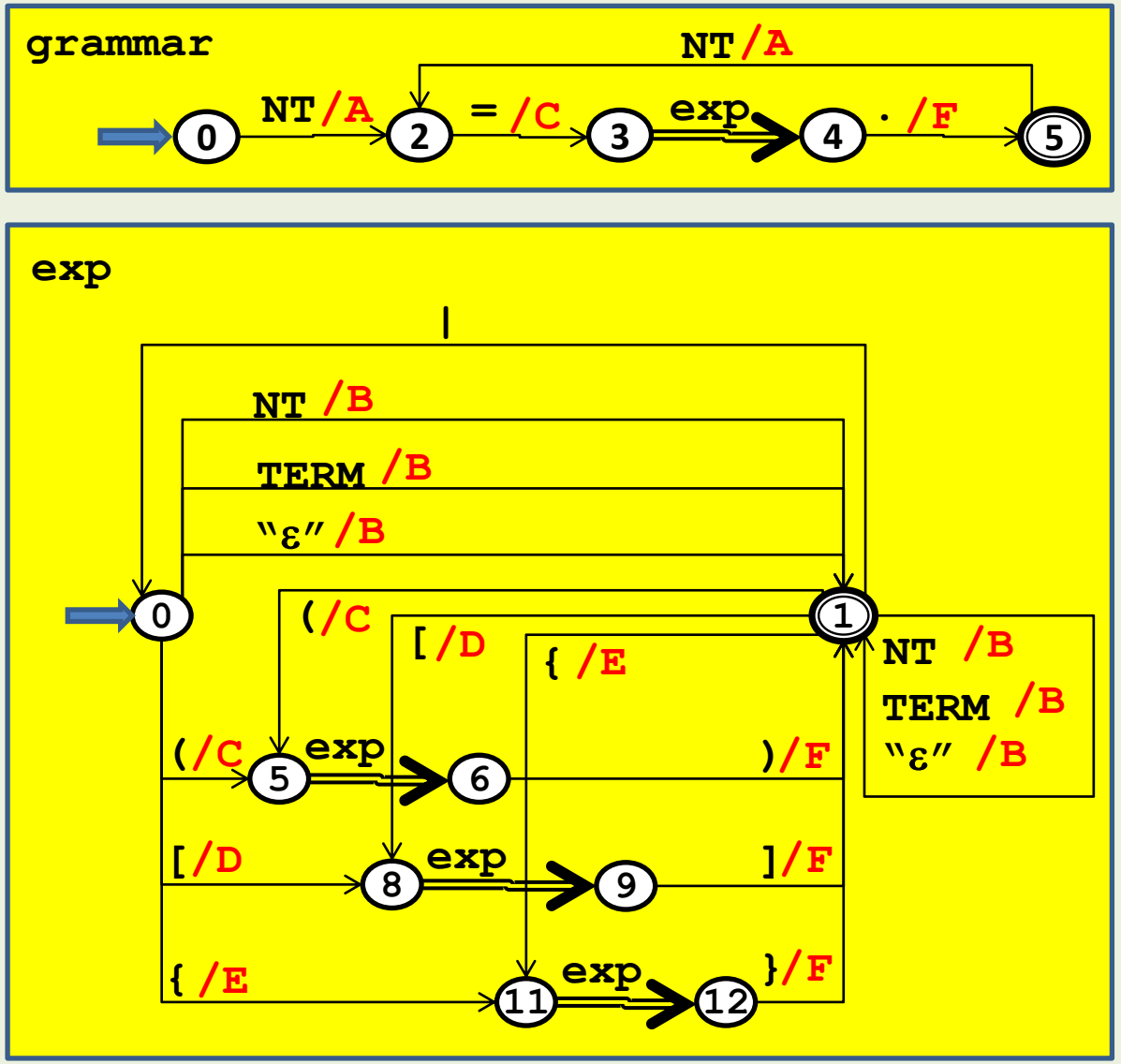
- Ao final do escopo de qualquer agrupamento (6-1,9-1,12-1,4-5):

F :

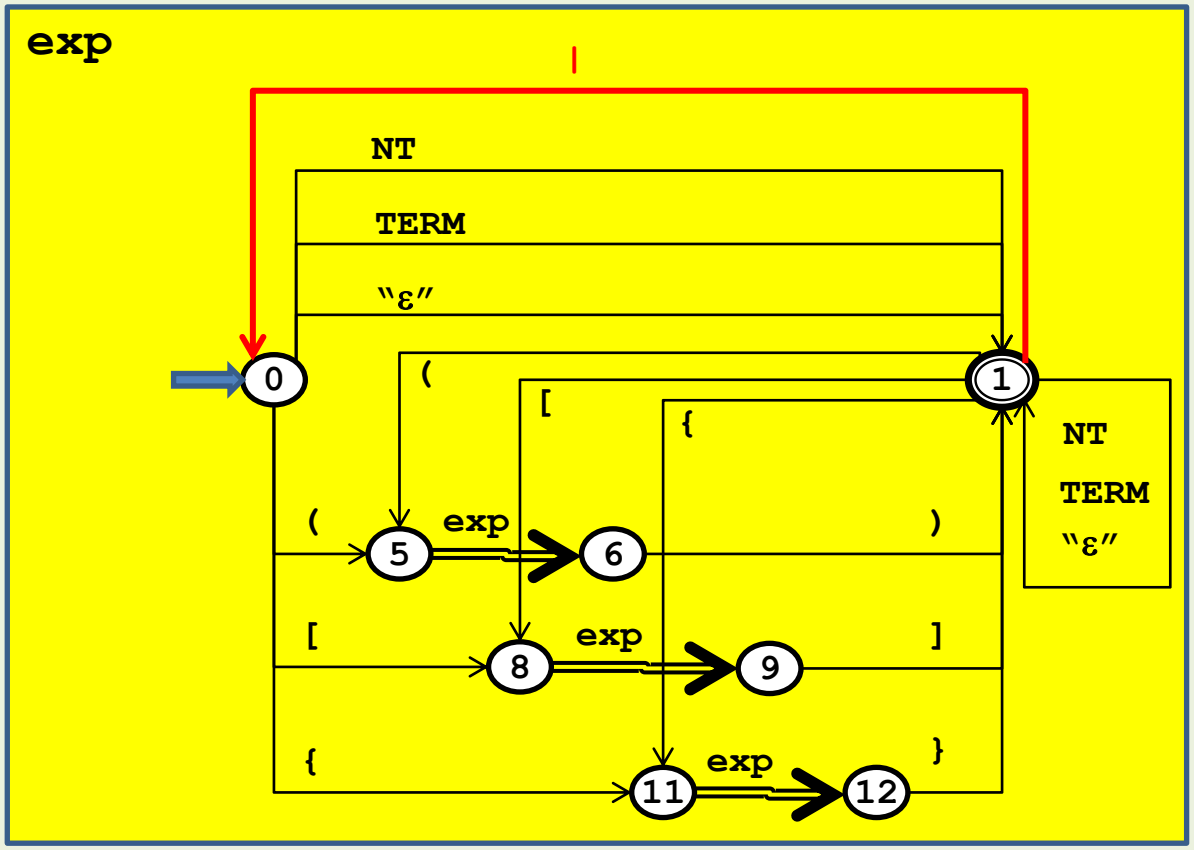
- criar uma transição em vazio do estado corrente para o estado correspondente ao elemento da direita do par ordenado presente no topo da pilha
- fazer desse estado o estado corrente
- desempilhar o par de estados do topo da pilha

Evolução do meta-reconhecedor

Até aqui foram incluídas as rotinas semânticas **A**, **B**, **C**, **D**, **E** e **F**



Rotinas semânticas (G)



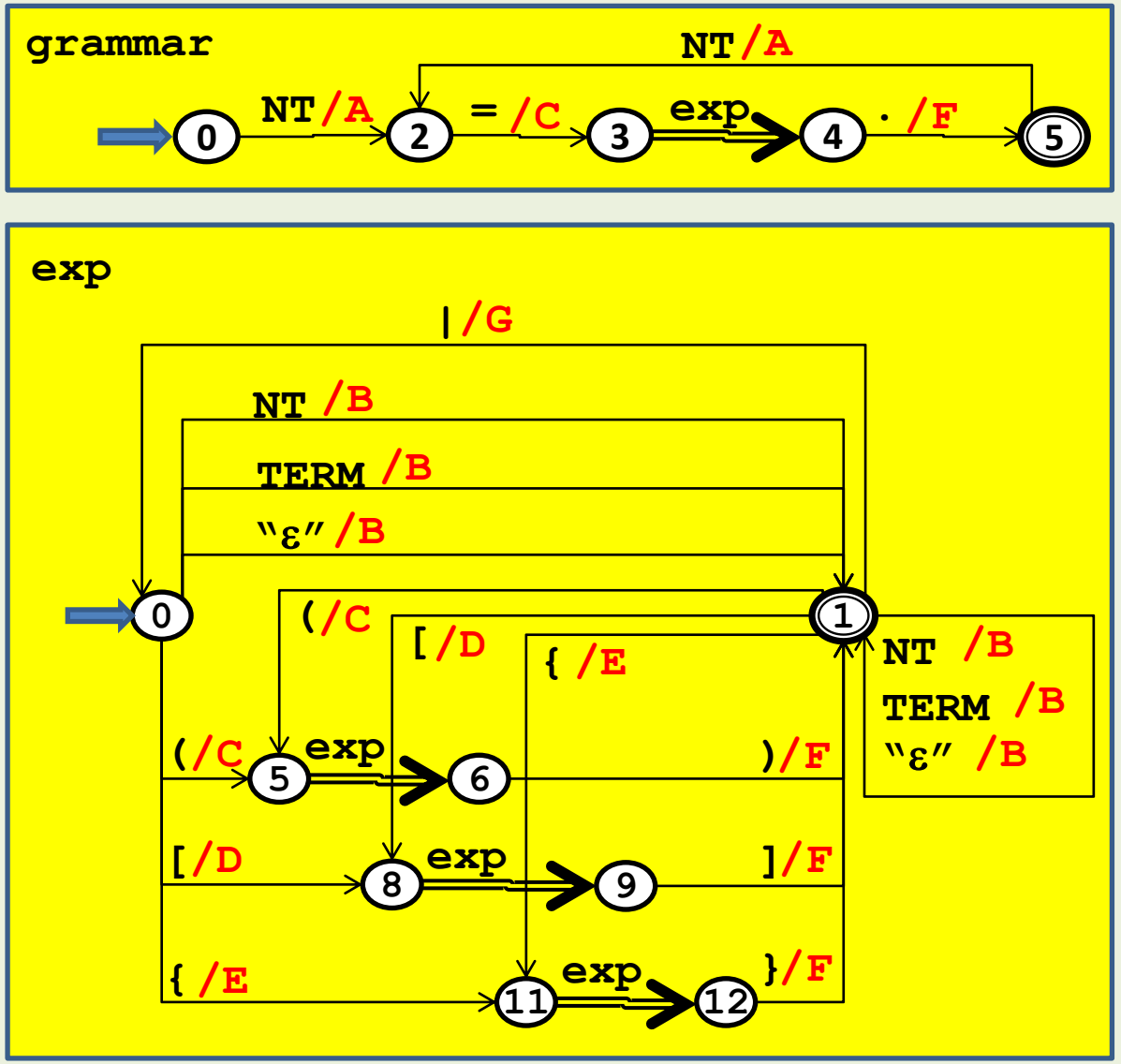
Fim de opção - reinício do escopo corrente

- Ao final de opção em um dado escopo (1-0):

G:
- criar uma transição em vazio do estado corrente para o estado correspondente ao elemento da direita do par ordenado presente no topo da pilha
- fazer o estado corrente igual ao estado que corresponde ao elemento da esquerda do par ordenado presente no topo da pilha

Evolução do meta-reconhecedor

Até aqui foram incluídas todas as rotinas semânticas **A**, **B**, **C**, **D**, **E**, **F** e **G**

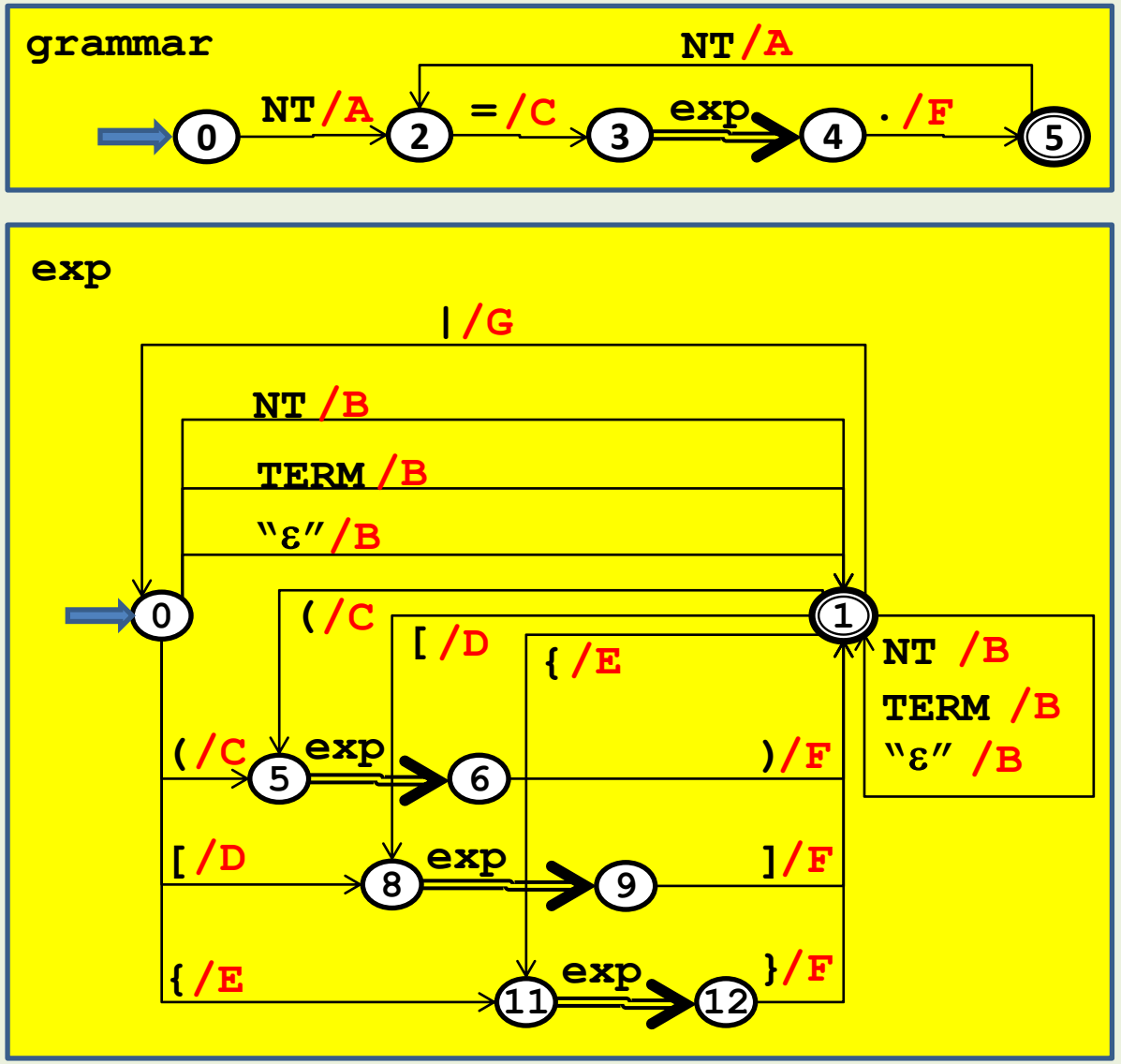


O meta-reconhecedor final

Desse desenvolvimento, resultou, portanto, o meta-reconhecedor desejado, realizado pelo autômato de pilha estruturado ao lado, decorado com todas as rotinas semânticas

A, **B**, **C**, **D**, **E**, **F** e **G**

as quais, por conveniência, estão transcritas no slide a seguir.

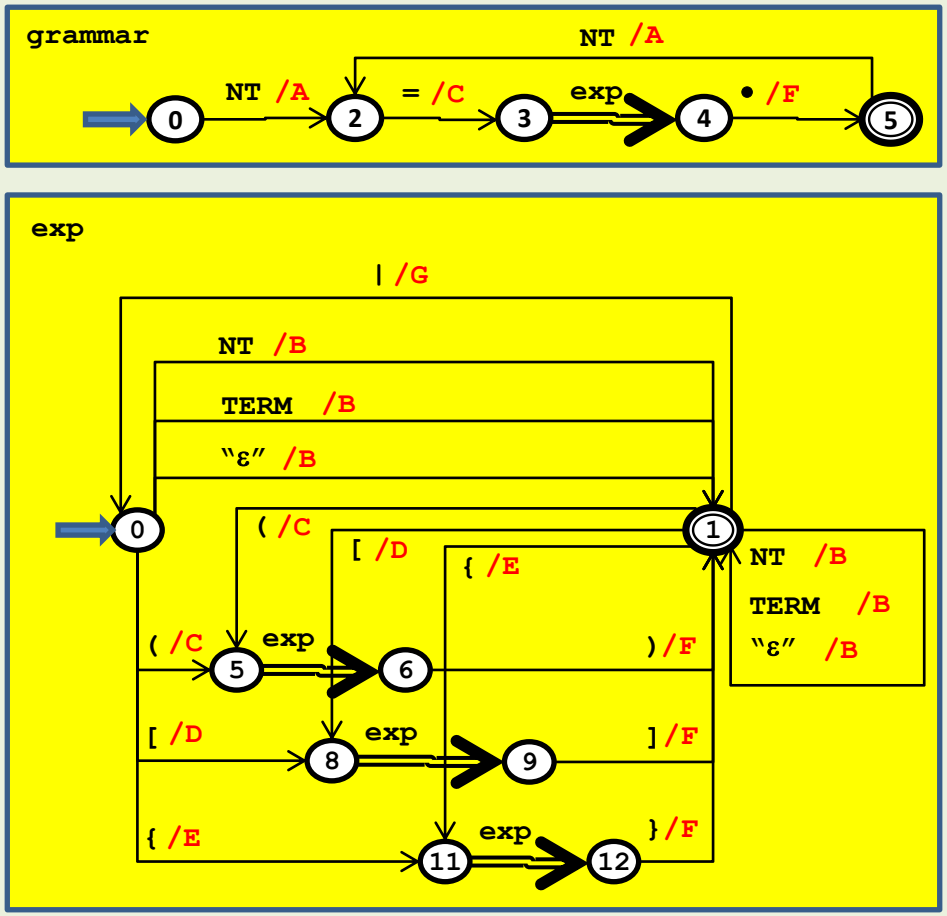


Rotinas semânticas do meta-reconhecedor

A	esvaziar a pilha; iniciar o estado corrente em 0 e o contador em 1; gerar uma transição em vazio de retorno de submáquina, a partir do estado 1
B	gerar uma transição rotulada com o terminal (consumo de átomo), não-terminal (chamada de sub-máquina) ou vazio (transição em vazio), partindo do estado corrente para um novo estado, indicado pelo contador; Atualizar o estado corrente com o valor do contador; Incrementar o contador.
C	manter o estado corrente; empilhar o par (estado corrente, contador); incrementar o contador
D	manter o estado corrente; gerar uma transição em vazio do estado corrente para o representado pelo contador; empilhar o par (estado corrente, contador); incrementar o contador
E	gerar uma transição em vazio do estado corrente para o estado representado pelo contador; alterar o estado corrente para o estado representado pelo contador; empilhar o par (contador, contador); incrementar o contador
F	criar uma transição em vazio do estado corrente para o estado correspondente ao elemento da direita do par ordenado presente no topo da pilha; fazer desse estado o estado corrente; desempilhar o par de estados contido no topo da pilha
G	criar uma transição em vazio do estado corrente para o estado correspondente ao elemento da direita do par ordenado presente no topo da pilha; fazer o estado corrente igual ao estado que corresponde ao elemento da esquerda do par ordenado presente no topo da pilha

Transição	Origem	Ação	Semântica	Destino
1	>G 0	NT	A	G 2
2	G 2	=	C	G 3
3	G 3	CALL E 0		G 4
4	G 4	•	F	G 5
5	G 5	NT	A	G 2
6	G 5	STOP		
7	>E 0	NT	B	E 1
8	>E 0	TERM	B	E 1
9	>E 0	"ε"	B	E 1
10	>E 0	(	C	E 5
11	>E 0	[	D	E 8
12	>E 0	{	E	E 11
13	E 1	|	G	E 0
14	E 1	NT	B	E 1
15	E 1	TERM	B	E 1
16	E 1	"ε"	B	E 1
17	E 1	(	C	E 5
18	E 1	[	D	E 8
19	E 1	{	E	E 11
20	E 1	RETURN		
21	E 5	CALL E 0		E 6
22	E 6	)	F	E 1
23	E 8	CALL E 0		E 9
24	E 9	]	F	E 1
25	E 11	CALL E 0		E 12
26	E 12	}	F	E 1

←Tabela das transições do meta-reconhecedor



CALL ESTADO = chamada de submáquina
RETURN = retorno de submáquina
STOP = final da execução, com reconhecimento

Pseudo-codificação do meta-reconhecedor

- Para a obtenção de um pseudocódigo para o meta-reconhecedor que estamos construindo, basta efetuar uma transformação simples sobre a tabela de transições do slide anterior.
- Há as seguintes tipos de transição na tabela:
  - Transições internas com consumo de átomo
  - Transições de chamada de submáquina
  - Transições de retorno de submáquina
  - Transição de término de reconhecimento, com sucesso
  - Transição de término de reconhecimento, com erro
- Vamos estudar a seguir o pseudo-código apropriado para cada uma dessas transições.
- Antes, convém agrupar as transições que têm origem em um mesmo estado, para simplificar o pseudo-código a ser gerado.

Transições internas com consumo de átomo e Transição de término de reconhecimento, com erro

Por exemplo:

Transição	Origem	Ação	Semântica	Destino
2	G 2	=	C	G 3

- A transição acima pode ser trivialmente transliterada para a forma de pseudo-código da seguinte maneira:

  G2: if átomo = "=" then {call C; go to G3};

- Se esta for a única ou a última das transições com origem em G2, pospõe-se ao seu pseudo-código uma notificação de erro, a ser executada caso a aplicação desta transição não tenha tido sucesso:

  call ERRO;

Transições de chamada de submáquina

Transição	Origem	Ação	Semântica	Destino
21	E 5	CALL E 0		E 6

- A transição do exemplo acima tem origem no estado E5, e efetua a chamada da submáquina cujo estado inicial é E0. Terminada a execução dessa submáquina, o retorno deve ser feito para o estado E6. Assim, E6 deve ser empilhado para futuro retorno, e E0 deve ser o estado para onde a execução do autômato deve prosseguir. O pseudo-código a ser construído para esta transição deve ser portanto o seguinte:

  E5: call PUSH (E6); goto E0;

Transições de retorno de submáquina

Transição	Origem	Ação	Semântica	Destino
20	E 1	RETURN		

- A transição do exemplo acima efetua um retorno de submáquina. E1 é o estado final dessa submáquina, tendo ela conseguido finalizar com êxito o reconhecimento do seu texto de entrada.
- Na ocasião da chamada dessa submáquina, foi empilhado o estado para onde deve ser desviado o processamento nesta ocasião. Assim, o pseudo-código a ser construído para transliterar esta transição será simplesmente o seguinte, a ser posicionado após todas as demais eventuais transições que tiverem como origem o estado E1:

```
call RETURN;
```

Transição de término de reconhecimento, com sucesso

Transição	Origem	Ação	Semântica	Destino
6	G 5	STOP		

- Análoga à transição de retorno de submáquina, a transição de final de reconhecimento com sucesso gera um pseudocódigo igualmente trivial.
- A transição do exemplo acima translitera-se imediatamente para o seguinte pseudo-código, que deve ser posicionado após o de todas as demais transições que tenham G5 como estado de origem:

```
call STOP;
```

Construção de um pseudocódigo para o meta-reconhecedor

Transição	Origem	Ação	Semântica	Destino
1	>G 0	NT	A	G 2
2	G 2	=	C	G 3
3	G 3	PUSH G 4		E 0
4	G 4	•	F	G 5
5	G 5	NT	A	G 2
6	G 5	STOP		
7	>E 0	NT	B	E 1
8	>E 0	TERM	B	E 1
9	>E 0	"ε"	B	E 1
10	>E 0	(	C	E 5
11	>E 0	[	D	E 8
12	>E 0	{	E	E 11
13	E 1	|	G	E 0
14	E 1	NT	B	E 1
15	E 1	TERM	B	E 1
16	E 1	"ε"	B	E 1
17	E 1	(	C	E 5
18	E 1	[	D	E 8
19	E 1	{	E	E 11
20	E 1	RETURN		
21	E 5	PUSH E 6		E 0
22	E 6	)	F	E 1
23	E 8	PUSH E 9		E 0
24	E 9	]	F	E 1
25	E 11	PUSH E12		E 0
26	E 12	}	F	E 1

G0: if átomo = NT then {call AVANÇA; call A; goto G2};
call ERRO;
G2: if átomo = "=" then {call AVANÇA; call C; goto G3};
call ERRO;
G3: call PUSH (G4); goto E0;
G4: if átomo = "•" then {call AVANÇA; call F; goto G5};
call ERRO;
G5: if átomo = NT then {call AVANÇA; call A; goto G2};
call STOP;
E0: if átomo = NT then {call AVANÇA; call B; goto E1};
if átomo = TERM then {call AVANÇA; call B; goto E1};
if átomo = **"ε"** then {call AVANÇA; call B; goto E1};
if átomo = "(" then {call AVANÇA; call C; goto E5};
if átomo = "[" then {call AVANÇA; call D; goto E8};
if átomo = "}" then {call AVANÇA; call E; goto E11};
call ERRO;

CONTINUA >>>

Transição	Origem	Ação	Semântica	Destino
1	>G 0	NT	A	G 2
2	G 2	=	C	G 3
3	G 3	PUSH G 4		E 0
4	G 4	•	F	G 5
5	G 5	NT	A	G 2
6	G 5	STOP		
7	>E 0	NT	B	E 1
8	>E 0	TERM	B	E 1
9	>E 0	“ε”	B	E 1
10	>E 0	(	C	E 5
11	>E 0	[	D	E 8
12	>E 0	{	E	E 11
13	E 1	|	G	E 0
14	E 1	NT	B	E 1
15	E 1	TERM	B	E 1
16	E 1	“ε”	B	E 1
17	E 1	(	C	E 5
18	E 1	[	D	E 8
19	E 1	{	E	E 11
20	E 1	RETURN		
21	E 5	PUSH E 6		E 0
22	E 6	)	F	E 1
23	E 8	PUSH E 9		E 0
24	E 9	]	F	E 1
25	E 11	PUSH E12		E 0
26	E 12	}	F	E 1

Construção de um pseudocódigo para o meta-reconhecedor

```
E1: if átomo = “|” then {call AVANÇA; call G; goto E0};
      if átomo = NT then {call AVANÇA; call B; goto E1};
      if átomo = TERM then {call AVANÇA; call B; goto E1};
      if átomo = “ε” then {call AVANÇA; call B; goto E1};
      if átomo = “(“ then {call AVANÇA; call C; goto E5};
      if átomo = “[“ then {call AVANÇA; call D; goto E8};
      if átomo = “}” then {call AVANÇA; call E; goto E11};
      call RETURN;
E5: call PUSH (E6); goto E0;
E6: if átomo = “)” then {call AVANÇA; call F; goto E1};
      call ERRO;
E8: call PUSH (E9); goto E0;
E9: if átomo = “]” then {call AVANÇA; call F; goto E1};
      call ERRO;
E11: call PUSH (E12); goto E0;
E12: if átomo = “}” then {call AVANÇA; call F; goto E1};
      call ERRO;
```

UMA UTILIZAÇÃO DO META-RECONHECEDOR

Exercício

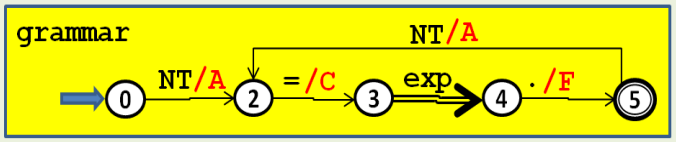
- A título de ilustração para a matéria desta aula, são apresentados a seguir alguns slides mostrando as principais passagens da utilização completa desta ferramenta que acaba de ser projetada.
- Para isso, parte-se de uma gramática, e acompanha-se passo a passo a construção do seu reconhecedor, de acordo com os efeitos da aplicação das ações determinadas pela ferramenta.
- Como exercício, para cada movimento apresentado nos slides adiante, faça o acompanhamento passo a passo das rotinas semânticas executadas, dos valores das variáveis e do conteúdo da pilha, e confira/corrija os resultados aqui antecipados.

Gramática

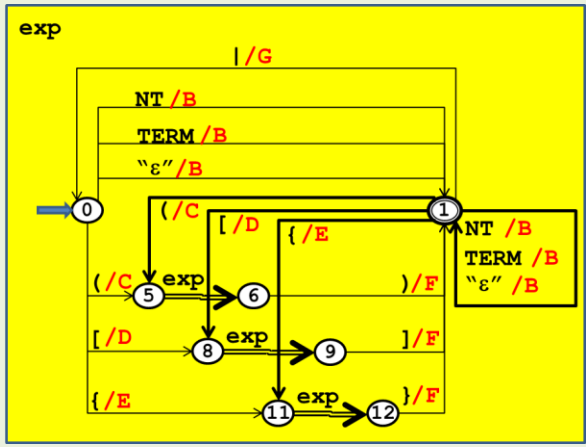
- Usaremos nesta ilustração uma pequena linguagem que representa um único comando de atribuição do valor de uma expressão aritmética bastante simplificada (só numérica; somas e produtos apenas) a uma variável:

```
Assignment = id "=" Expression .
Expression = Term { "+" Term } .
Term = Factor { "*" Factor } .
Factor =  num | "(" Expression ")" .
```

- Esta gramática é um caso ainda mais reduzido que o daquela que foi proposta como exercício na aula 05.



Factor



0 2 3(4) .

0 2 3(4) .

0 2 3(4) .

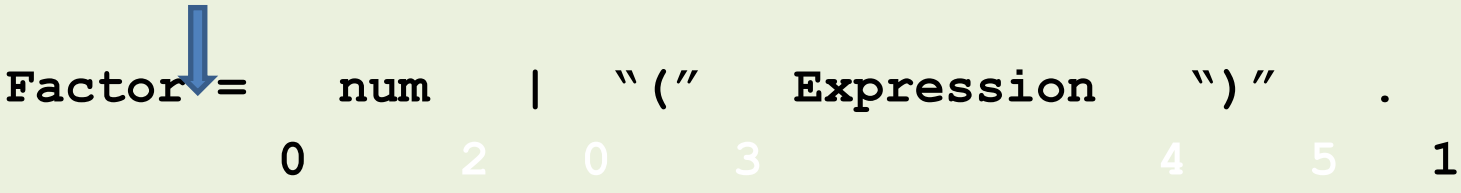
- **Factor = num | "(" Expression ")" .**

. .

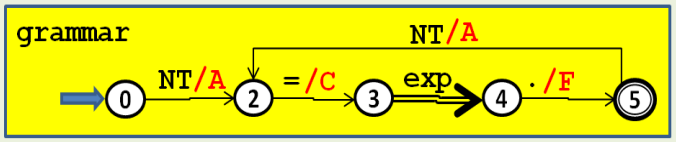
Situação inicial das variáveis:

Pilha	Entrada	Corrente	Contador	Saída	Devolvidos	Estado	Token
(Vazia)	(início)		0	(Vazia)		Grammar 0	

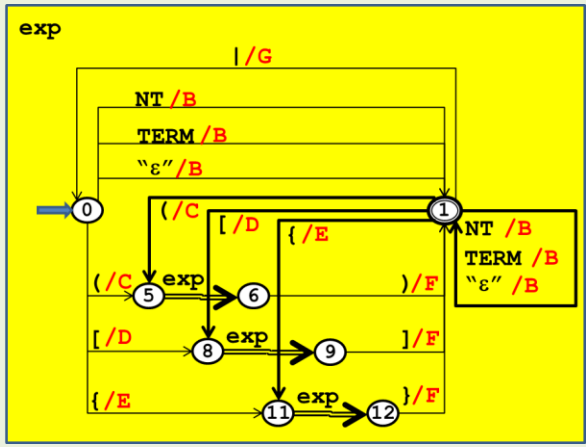
Factor



Factor



Factor



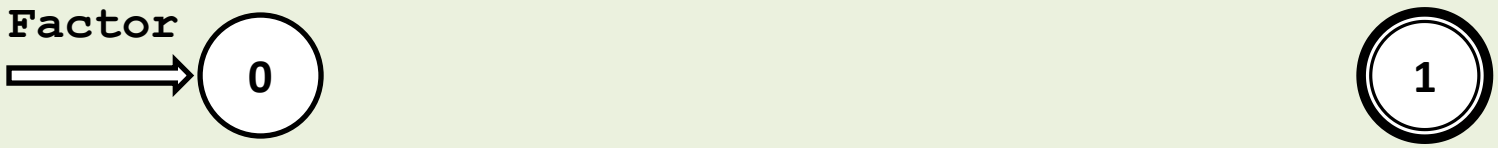

```
0          2    3(4)                                          .
0          2    3(4)                                          .
0          2    3(4)                                          .
  Factor   =    num    |   "("    Expression    ")"    .
  NT       =    NT     |   TERM   NT            TERM   .
. A                                                          .
```

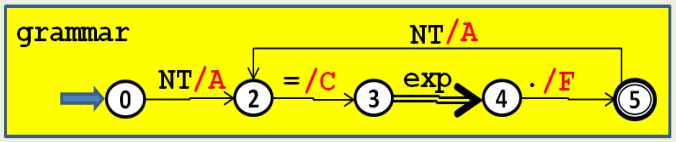
A	esvaziar a pilha; iniciar o estado corrente em 0; Iniciar o contador em 1; gerar uma transição em vazio de retorno de submáquina, com origem no estado 1; Incrementar contador

Pilha	Entrada	Corrente	Contador	Saída	Devolvidos	Estado	Token
(Vazia)	Factor	0	2	Retorno no estado 1	-	2	NT

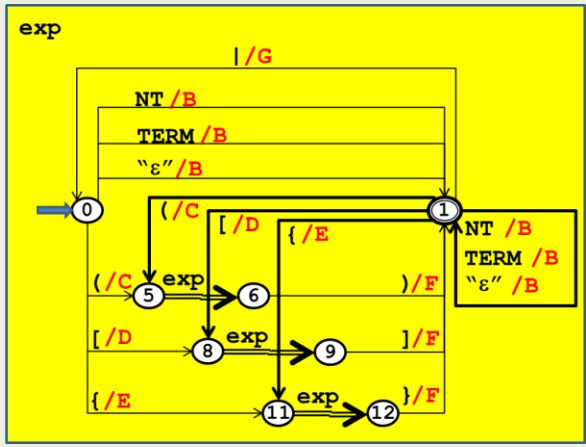
Factor

```
Factor =   num   |  "("   Expression   ")"   .
           0     2   0    3              4    5   1
```





Factor

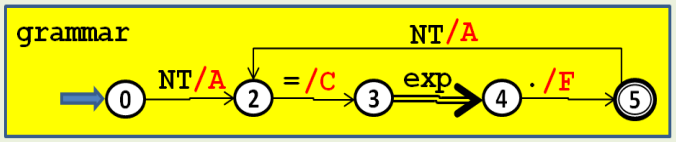



```
0          2    3(4)                                          .
0          2(4)                                               .
0          2    3(4)                                          .
  Factor    =    num    |   "("    Expression    ")"    .
  NT        =     NT    |   TERM   NT            TERM   .
. A         C                                                .
```

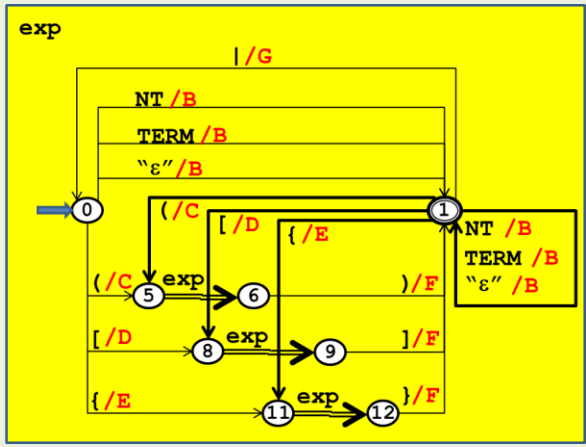
C manter o estado corrente;
empilhar o par (estado corrente, contador);
incrementar o contador

Mostrar o movimento das pilhas léxica e sintática para o estabelecimento do novo escopo

Pilha	Entrada	Corrente	Contador	Saída	Devolvidos	Estado	Token
(0,1)	=	0	2	Retorno no estado 1	-	3	=



Factor

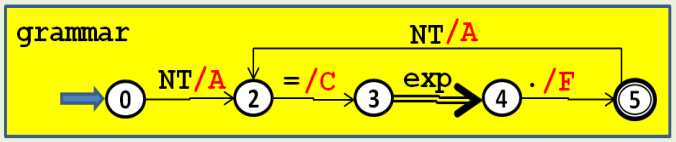



```
0          2    3(4)                                                    .
0          2(4)0                                    1                  .
0          2    3(4)                                                    .
  Factor    = ● num    |  "("   Expression   ")"    .
  NT          =   NT     |  TERM  NT           TERM   .
. A           C                                                        .
```

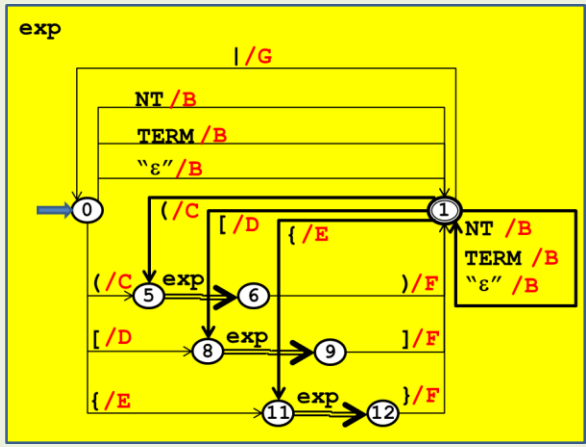
Ações léxicas de devolução de token
Ações sintáticas de chamada de submáquina

Mostrar a situação das pilhas léxica e sintática

Pilha	Entrada	Corrente	Contador	Saída	Devolvidos	Estado	Token	Est.retorno
(0,1)	num	0	2	Retorno no estado 1	num	Expression 0	num	4



Factor



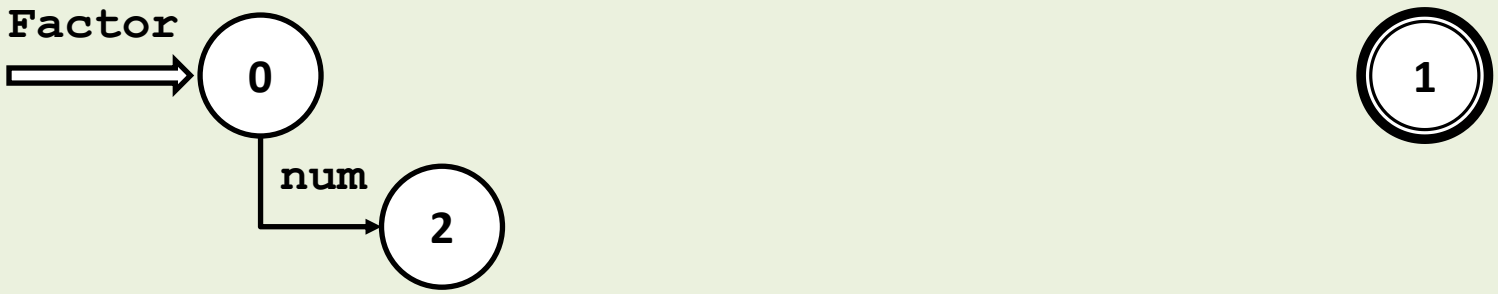

```
0        2   3(4)                                          .
0        2(4)0     1                          1           .
0        2   3                                            .
  Factor   =   num  |  "("   Expression   ")"   .
  NT       =   TERM |  TERM  NT             TERM  .
. A        C   B                                         .
```

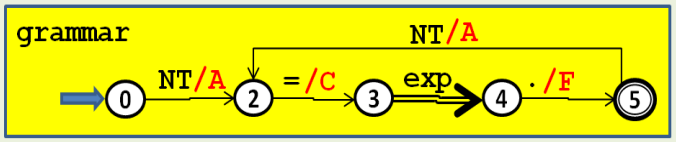
B gerar uma transição rotulada com o terminal (consumo de átomo), partindo do estado corrente para um novo estado, indicado pelo contador;
Atualizar o estado corrente com o valor do contador;
Incrementar o contador.

Obs.: num é considerado um terminal, pois é saída do classificador léxico.

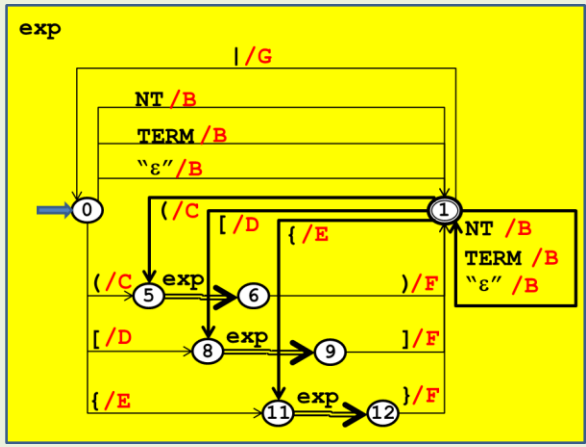
Pilha	Entrada	Corrente	Contador	Saída	Devolvidos	Estado	Token
(0,1)	num	0	3	(0,num)->2	-	1	TERM

```
Factor =   num   |  "("   Expression   ")"   .
          0     2   0    3                4    5   1
```





Factor

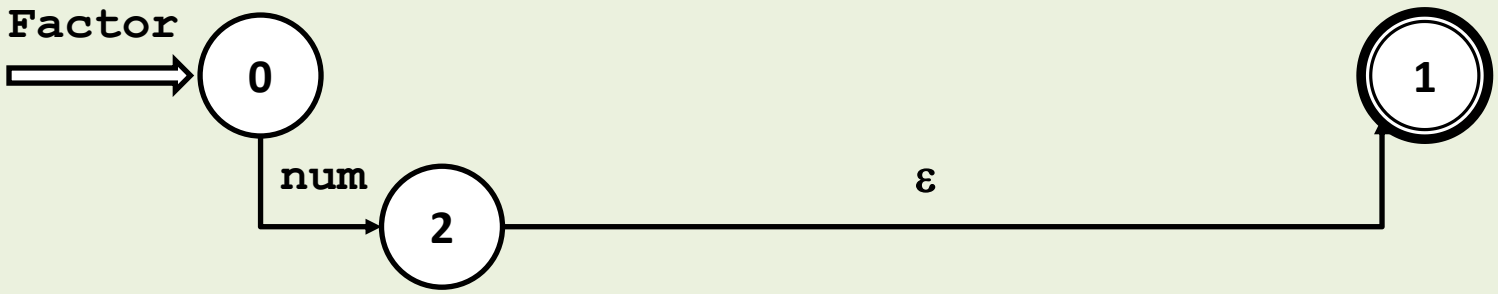


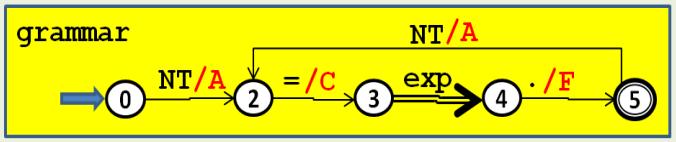

```
0          2    3(4)                                          .
0          2(4)0       1   0                          1      .
0          2    3                                            .
   Factor     =     num     | "("    Expression     ")"     .
   NT         =     NT      |  TERM  NT             TERM    .
.  A          C     B       G                                .
```

G criar uma transição em vazio do estado corrente para o estado correspondente ao elemento da direita do par ordenado presente no topo da pilha;
fazer o estado corrente igual ao estado que corresponde ao elemento da esquerda do par ordenado presente no topo da pilha

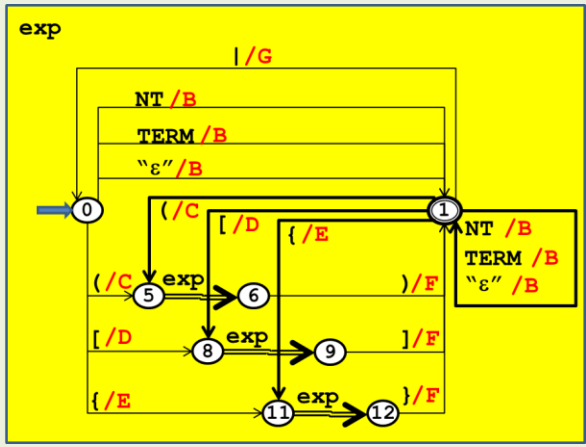
Mostrar que isto restabelece o escopo mais externo ao iniciar-se nova opção sintática.

```
Factor =   num   |  “(”   Expression   “)”   .
          0     2   0    3                4    5   1
```





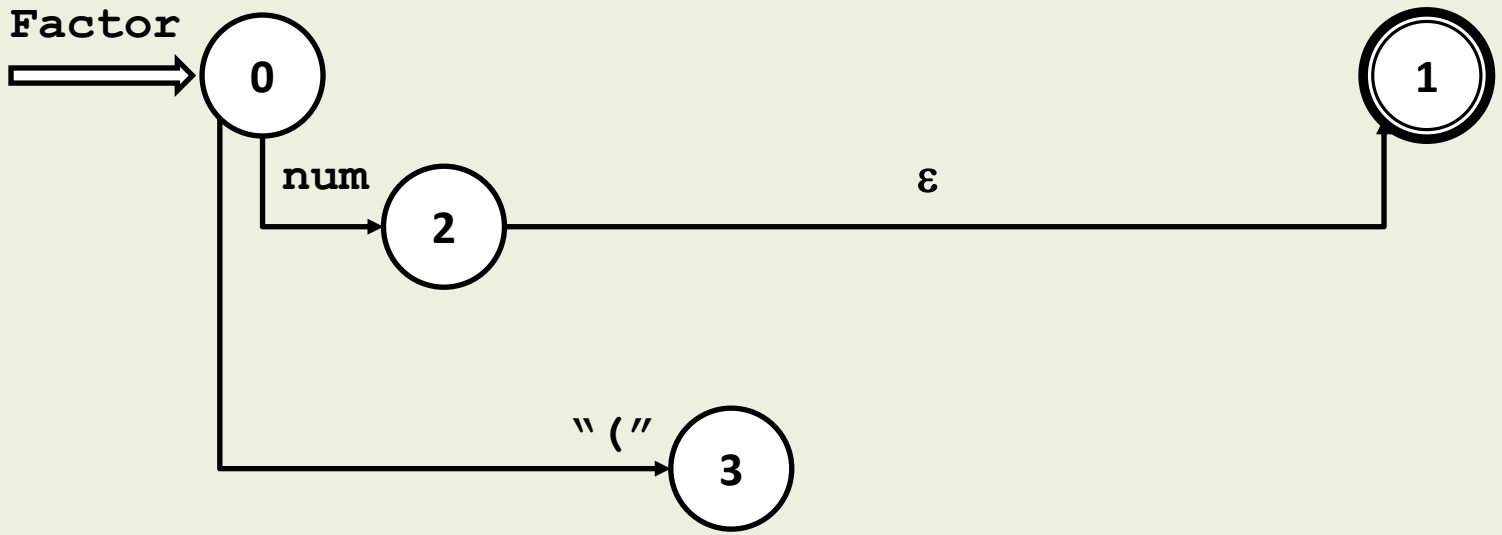
Factor

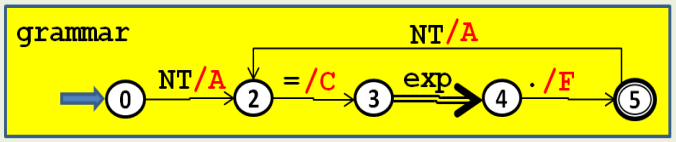



```
0         2   3(4)                                                .
0         2(4)0     1   0     1                          1        .
0         2   3                                                   .
  Factor    =   num    |  "(" Expression    ")"    .
  NT        =   NT     |  TERM  NT              TERM   .
. A         C   B      G  B                                      .
```

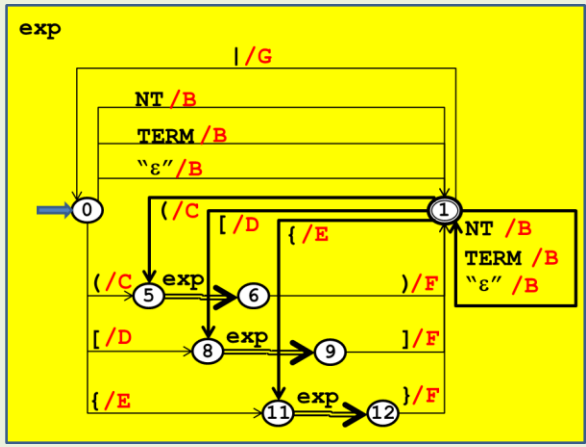
B	**gerar uma transição rotulada com o terminal (consumo de átomo), partindo do estado corrente para um novo estado, indicado pelo contador;** **Atualizar o estado corrente com o valor do contador;** **Incrementar o contador.**

```
Factor =   num   |  "("   Expression   ")"   .
           0     2   0    3                4    5   1
```





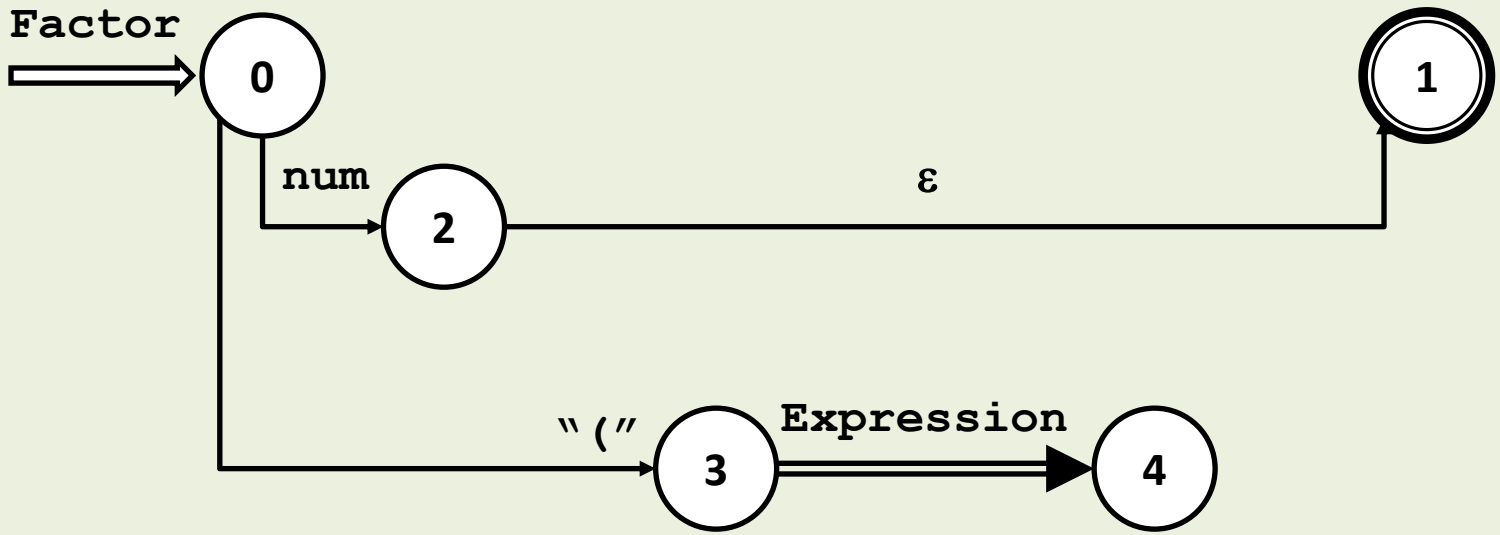
Factor

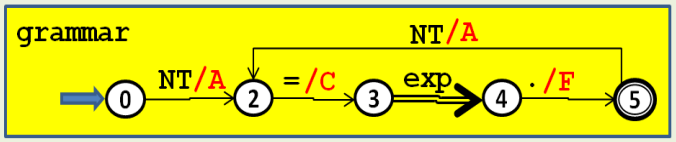



```
0        2   3(4)                                                .
0        2(4)0     1  0     1            1          1            .
0        2   3                                                   .
  Factor   =   num   |  "("   Expression  ●  ")"   .
  NT       =   NT    |  TERM  NT              TERM   .
. A        C   B     G  B     B                                  .
```

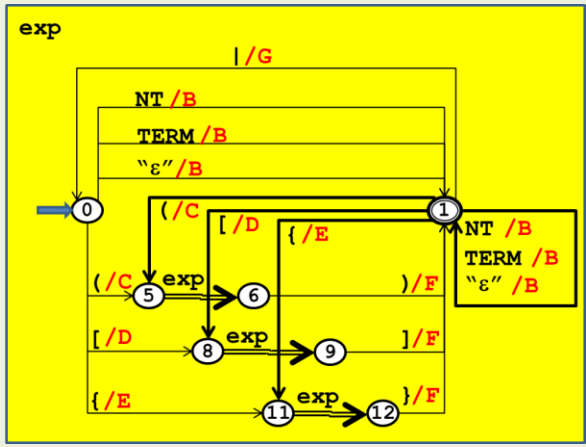
B	**gerar uma transição rotulada com o não-terminal (chamada de sub-máquina), partindo do estado corrente para um novo estado, indicado pelo contador;** **Atualizar o estado corrente com o valor do contador;** **Incrementar o contador.**

```
Factor =   num   |  "("   Expression   ")"   .
           0     2   0    3              4    5   1
```





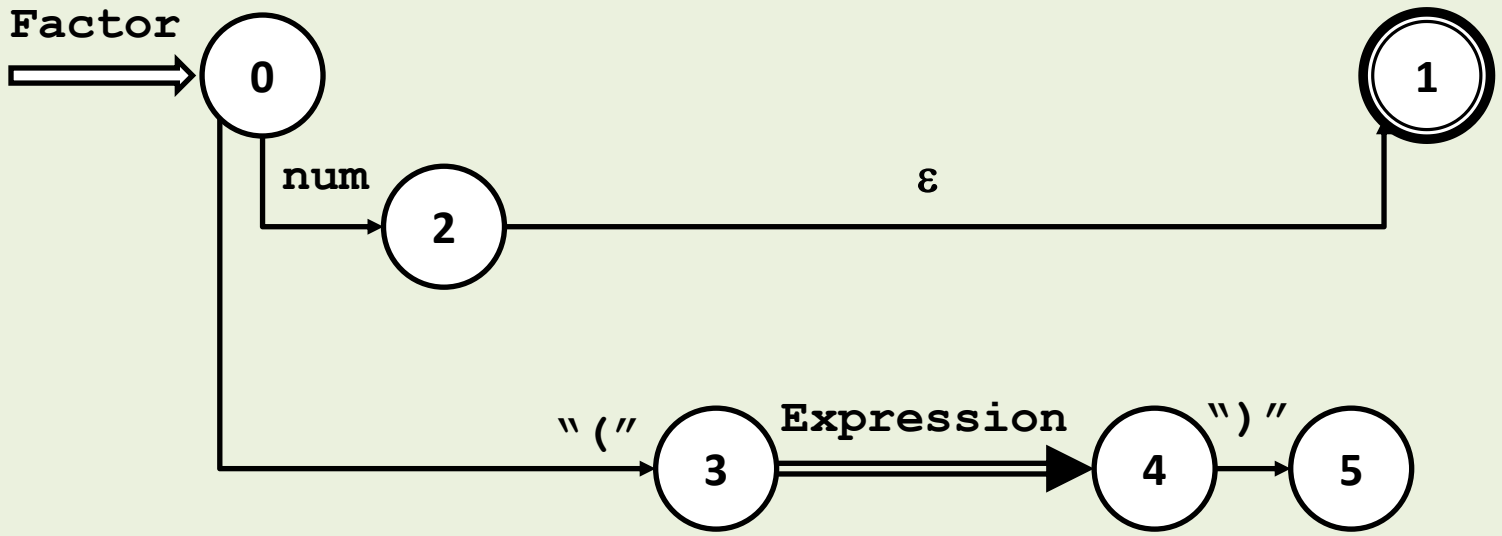
Factor

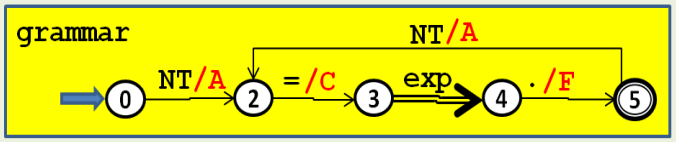



```
0        2   3(4)                                                .
0        2(4)0     1  0     1              1     1  1             .
0        2   3                                                   .
  Factor   =   num   |  "("   Expression   ")"  .
  NT       =   NT    |  TERM  NT           TERM  .
. A        C   B     G  B     B            B                    .
```

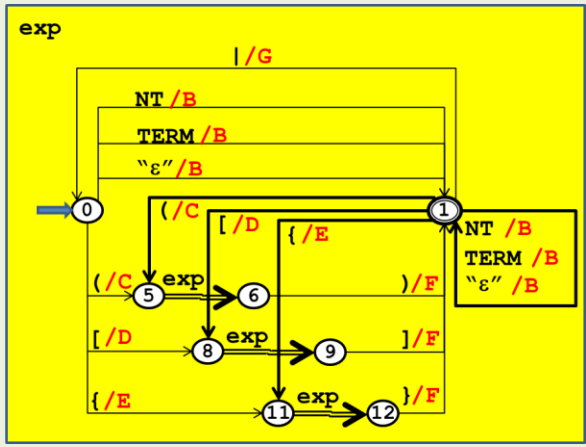
B	**gerar uma transição rotulada com o terminal (consumo de átomo), partindo do estado corrente para um novo estado, indicado pelo contador;** **Atualizar o estado corrente com o valor do contador;** **Incrementar o contador.**

```
Factor =   num   |  "("   Expression   ")"   .
           0     2   0    3              4    5   1
```





Factor

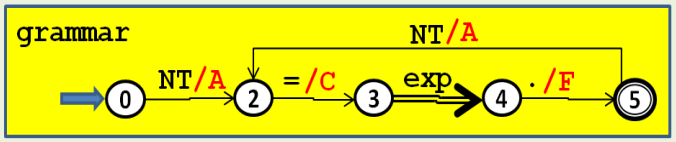



```
0       2   3(4)                                                .
0       2(4)0     1   0     1             1     1   1           .
0       2   3                             4
  Factor   =   num   |  "("   Expression   ")"  .
  NT       =   NT    |  TERM  NT           TERM  .
. A        C   B     G  B     B            B                   .
```

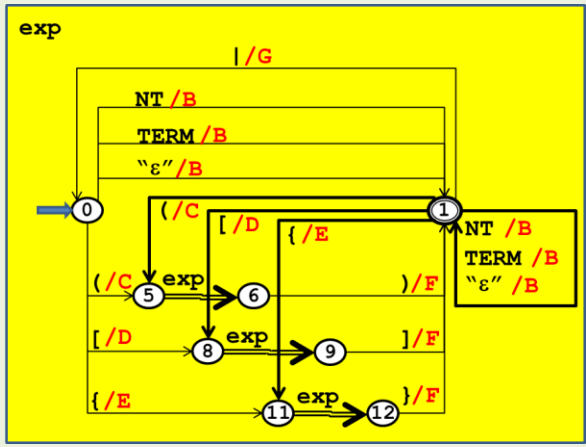
Mostrar a pilha léxica (devolução do ponto)

Mostrar o movimento da pilha sintática (retorno de submáquina).

Mostrar o retorno da submáquina Expression para a submáquina grammar.



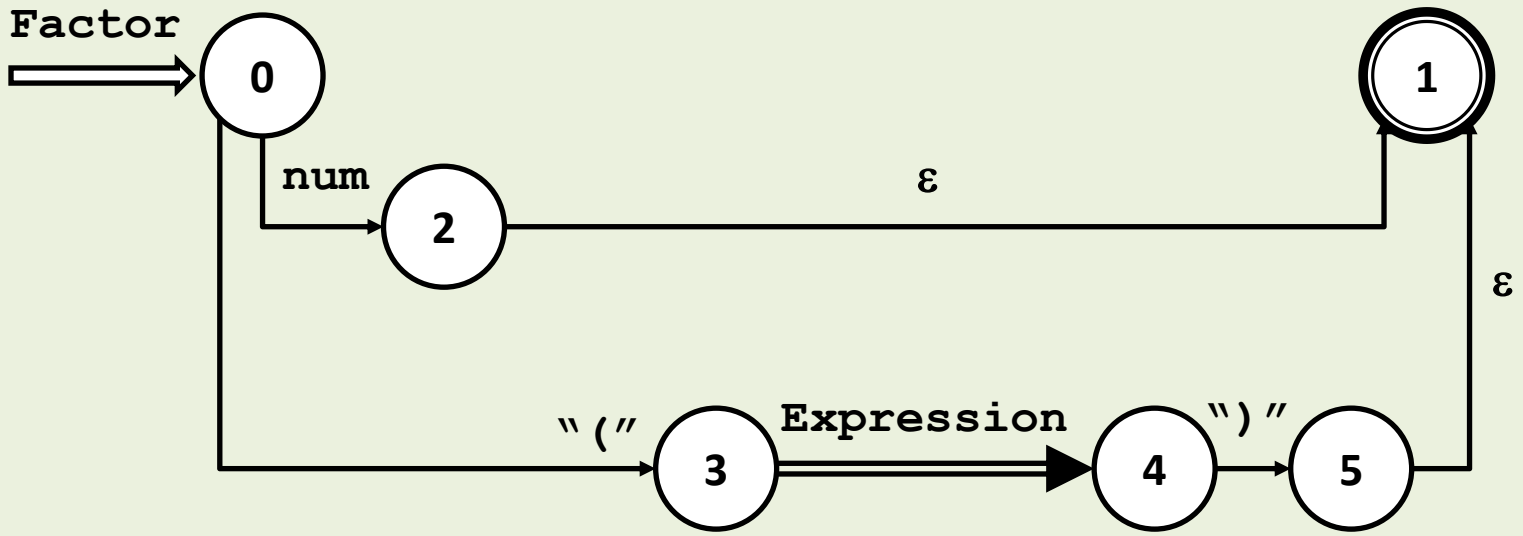
Factor




```
0        2   3(4)                                                .
0        2(4)0     1  0     1              1     1                 .
0        2   3                                 4   5
  Factor   =   num   |  "("   Expression   ")"   .
  NT       =   NT    |  TERM  NT           TERM  .
. A        C   B     G  B     B            B     F                 .
```

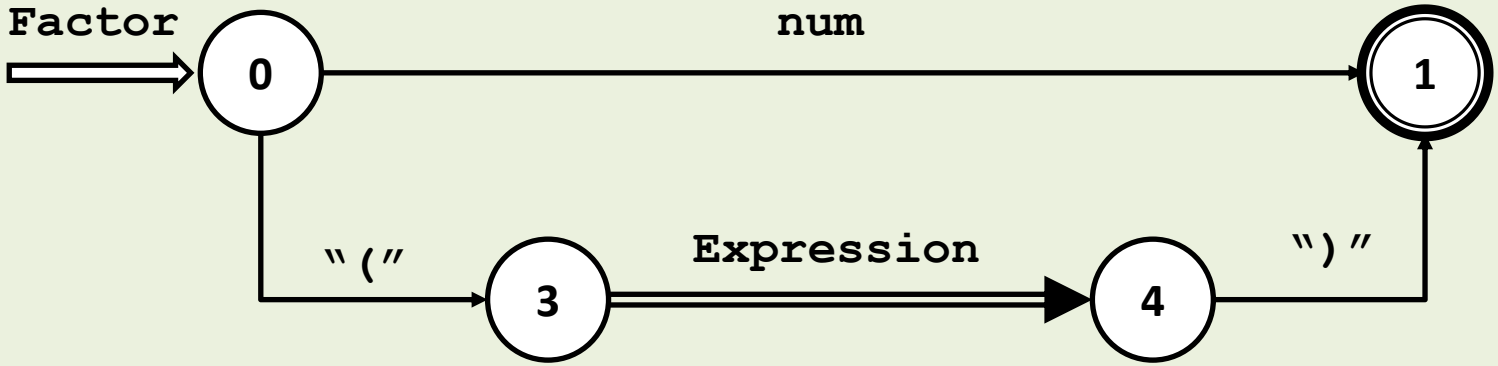
F	**criar uma transição em vazio do estado corrente para o estado correspondente ao elemento da direita do par ordenado presente no topo da pilha; fazer desse estado o estado corrente; desempilhar o par de estados do topo da pilha**

```
Factor =   num   |  "("   Expression   ")"   .
           0     2   0    3              4    5   1
```



```
Factor =   num   |  "("   Expression   ")"   .
          0     2   0    3              4    5   1
```

Eliminando as transições em vazio e redesenhando o autômato:

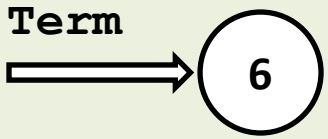


Rotinas semânticas do meta-reconhecedor

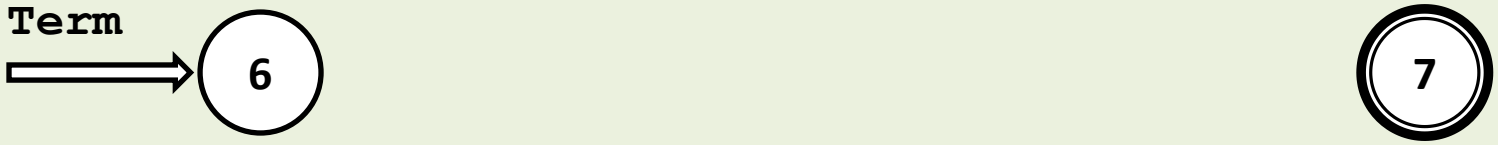
A	esvaziar a pilha; iniciar o estado corrente em 0 e o contador em 1; gerar uma transição em vazio de retorno de submáquina, a partir do estado 1
B	gerar uma transição rotulada com o terminal (consumo de átomo), não-terminal (chamada de sub-máquina) ou vazio (transição em vazio), partindo do estado corrente para um novo estado, indicado pelo contador; Atualizar o estado corrente com o valor do contador; Incrementar o contador.
C	manter o estado corrente; empilhar o par (estado corrente, contador); incrementar o contador
D	manter o estado corrente; gerar uma transição em vazio do estado corrente para o representado pelo contador; empilhar o par (estado corrente, contador); incrementar o contador
E	gerar uma transição em vazio do estado corrente para o estado representado pelo contador; alterar o estado corrente para o estado representado pelo contador; empilhar o par (contador, contador); incrementar o contador
F	criar uma transição em vazio do estado corrente para o estado correspondente ao elemento da direita do par ordenado presente no topo da pilha; fazer desse estado o estado corrente; desempilhar o par de estados contido no topo da pilha
G	criar uma transição em vazio do estado corrente para o estado correspondente ao elemento da direita do par ordenado presente no topo da pilha; fazer o estado corrente igual ao estado que corresponde ao elemento da esquerda do par ordenado presente no topo da pilha

Term

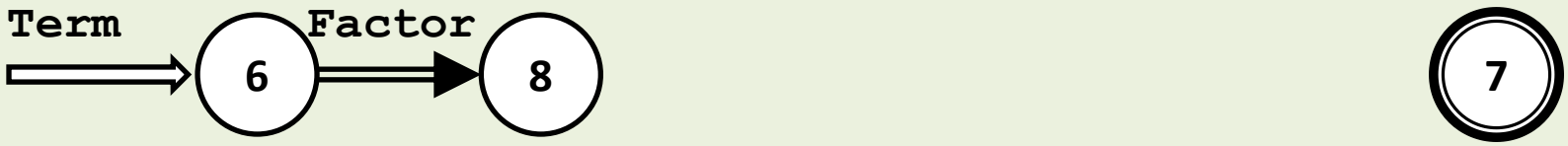
```
Term   =   Factor   {   "*"    Factor   }   .
         6          8    9    10         11  8    7
```



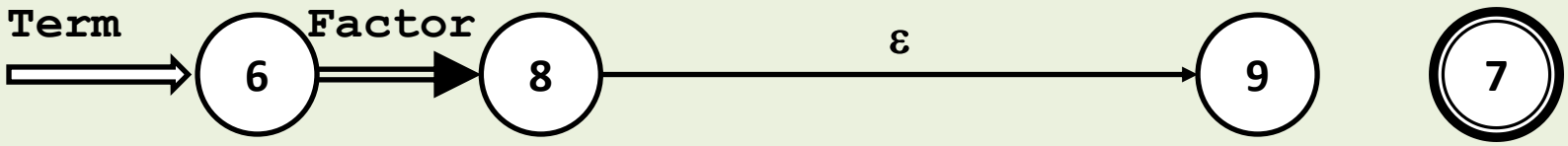
```
Term   =   Factor   {   "*"    Factor   }   .
         6          8    9    10         11  8    7
```



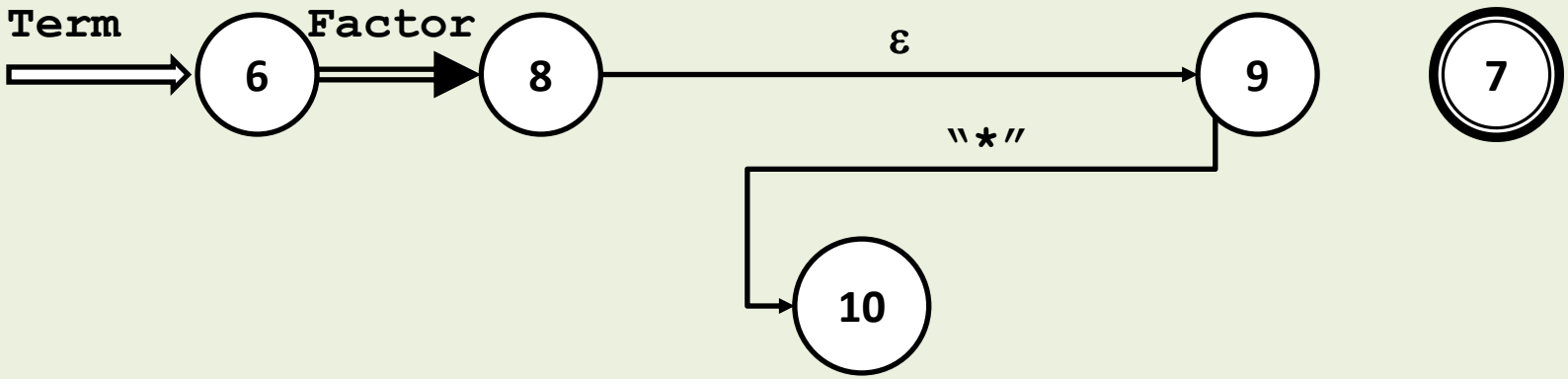
```
Term   =   Factor   {   "*"    Factor   }      .
         6          8     9     10          11  9         7
```



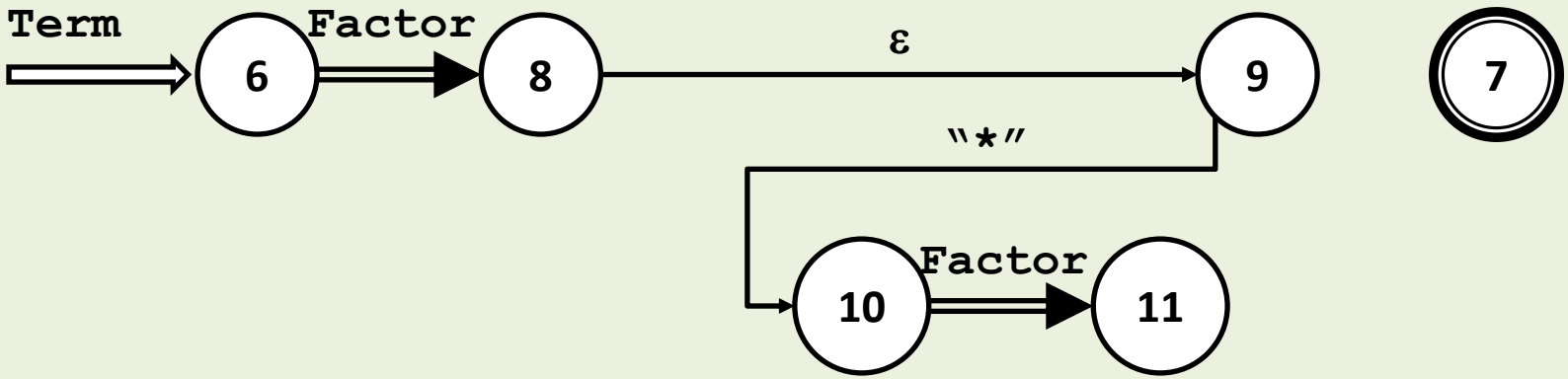
```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9        7
```



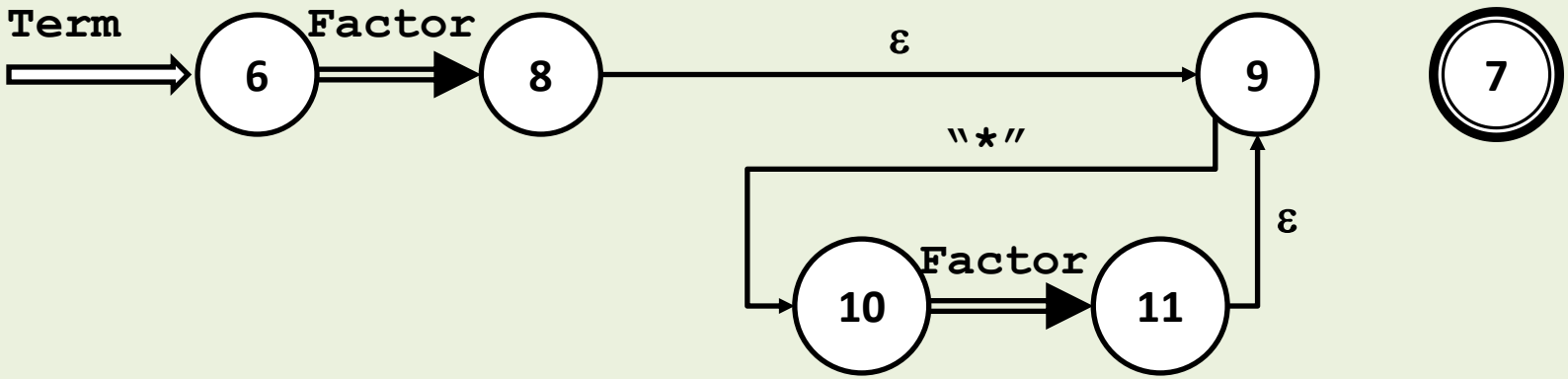
```
Term   =   Factor   {   "*"    Factor   }      .
         6          8    9    10         11  9        7
```



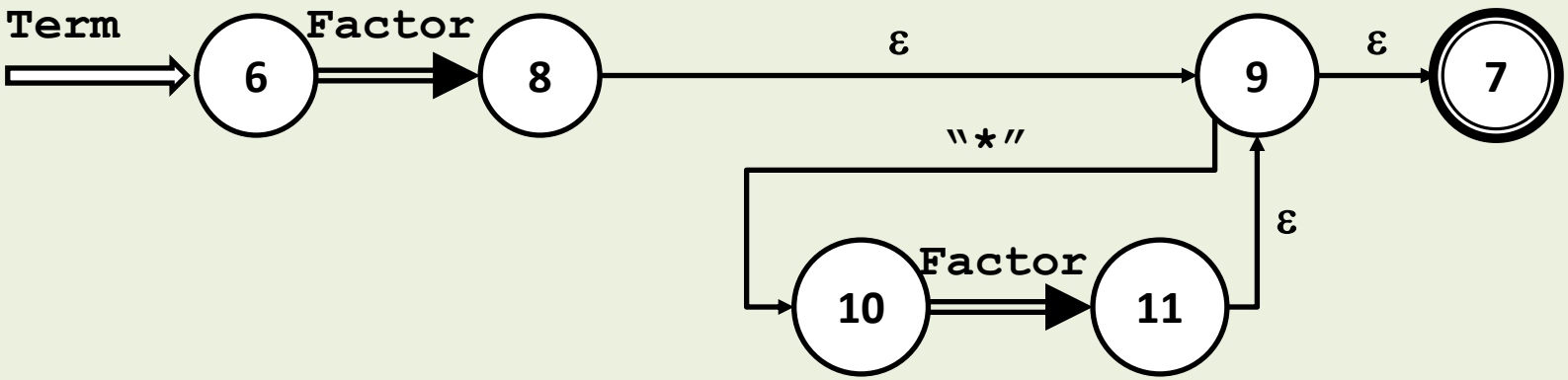
```
Term   =   Factor   {   "*"    Factor   }      .
         6          8    9    10         11  9        7
```



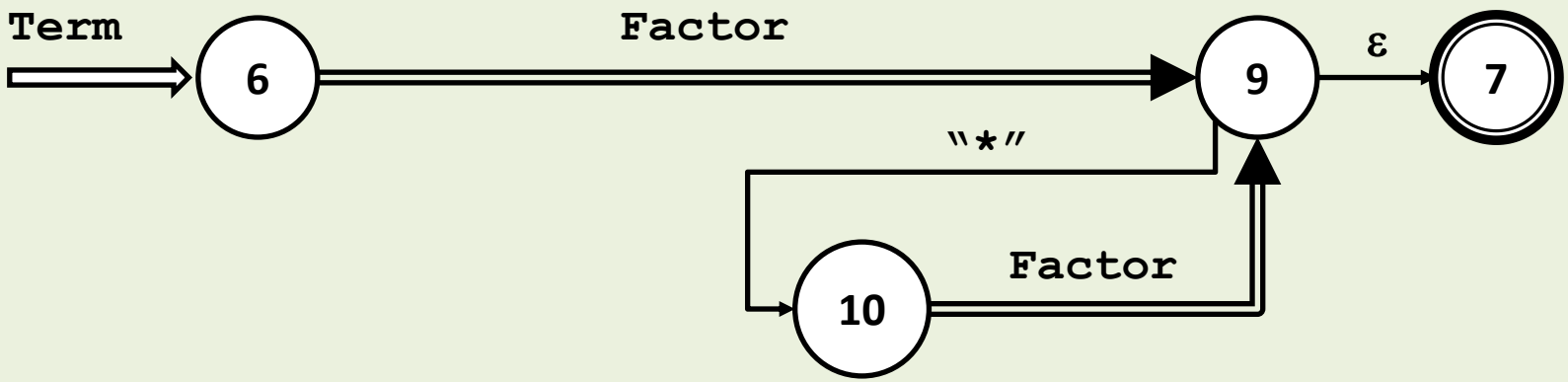
```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9       7
```



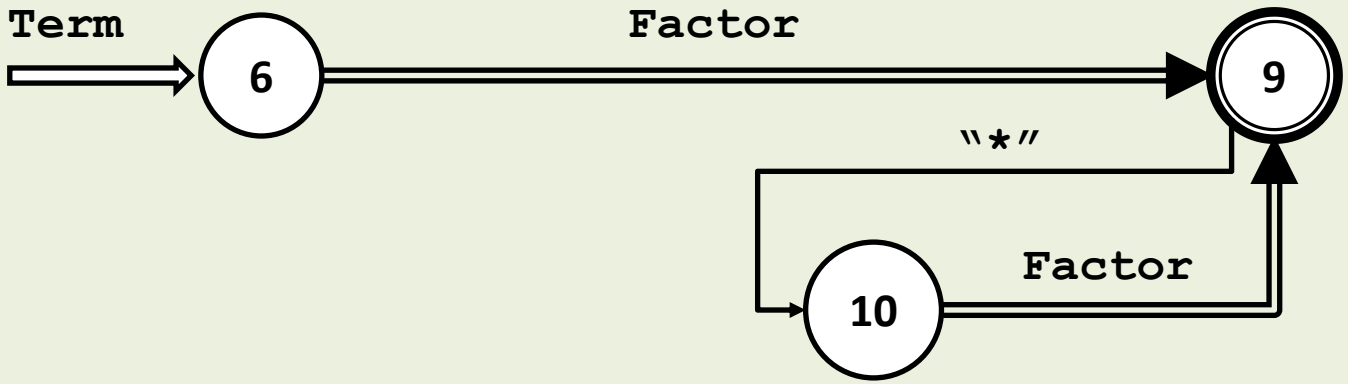
```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9        7
```



```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9        7
```

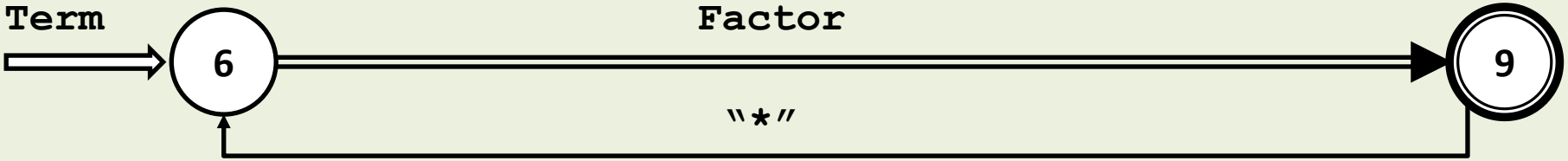


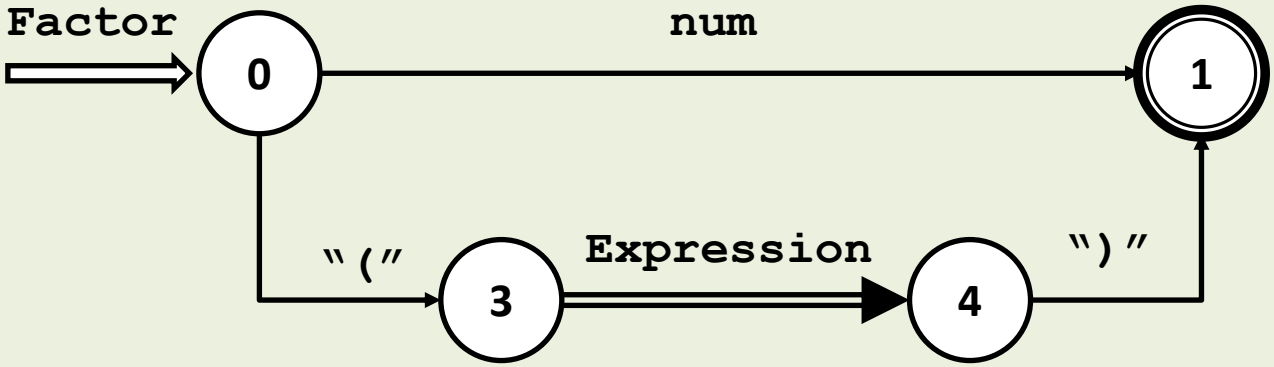
```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9       7
```



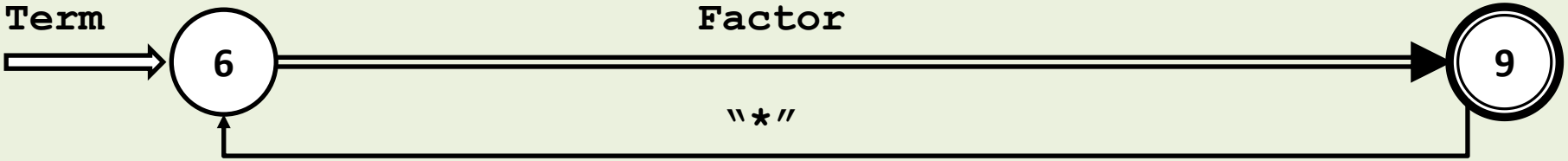
```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9       7
```

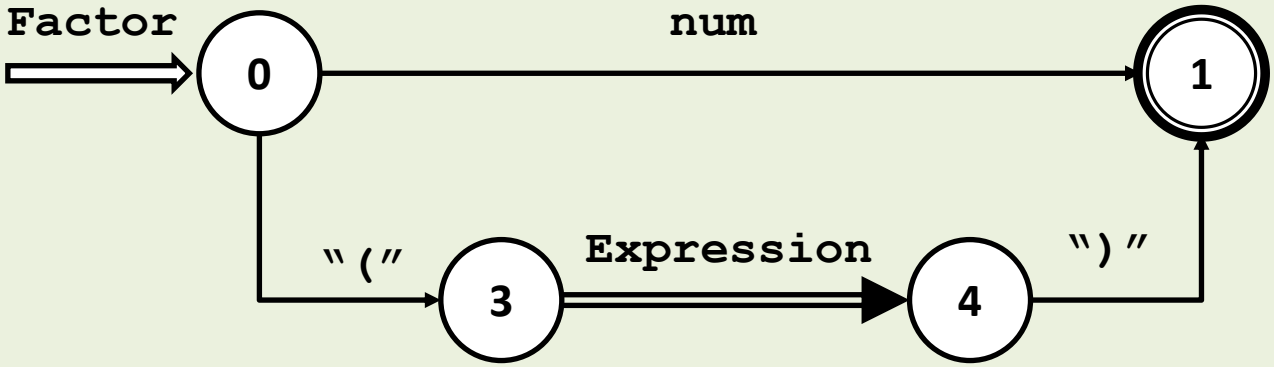
```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9       7
```



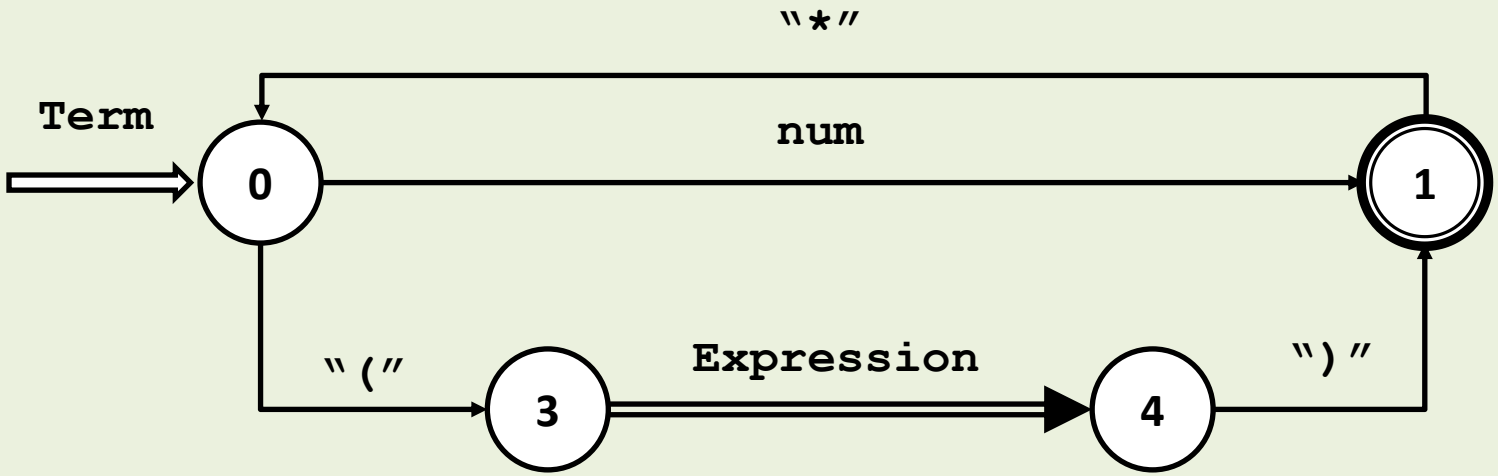


```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9       7
```





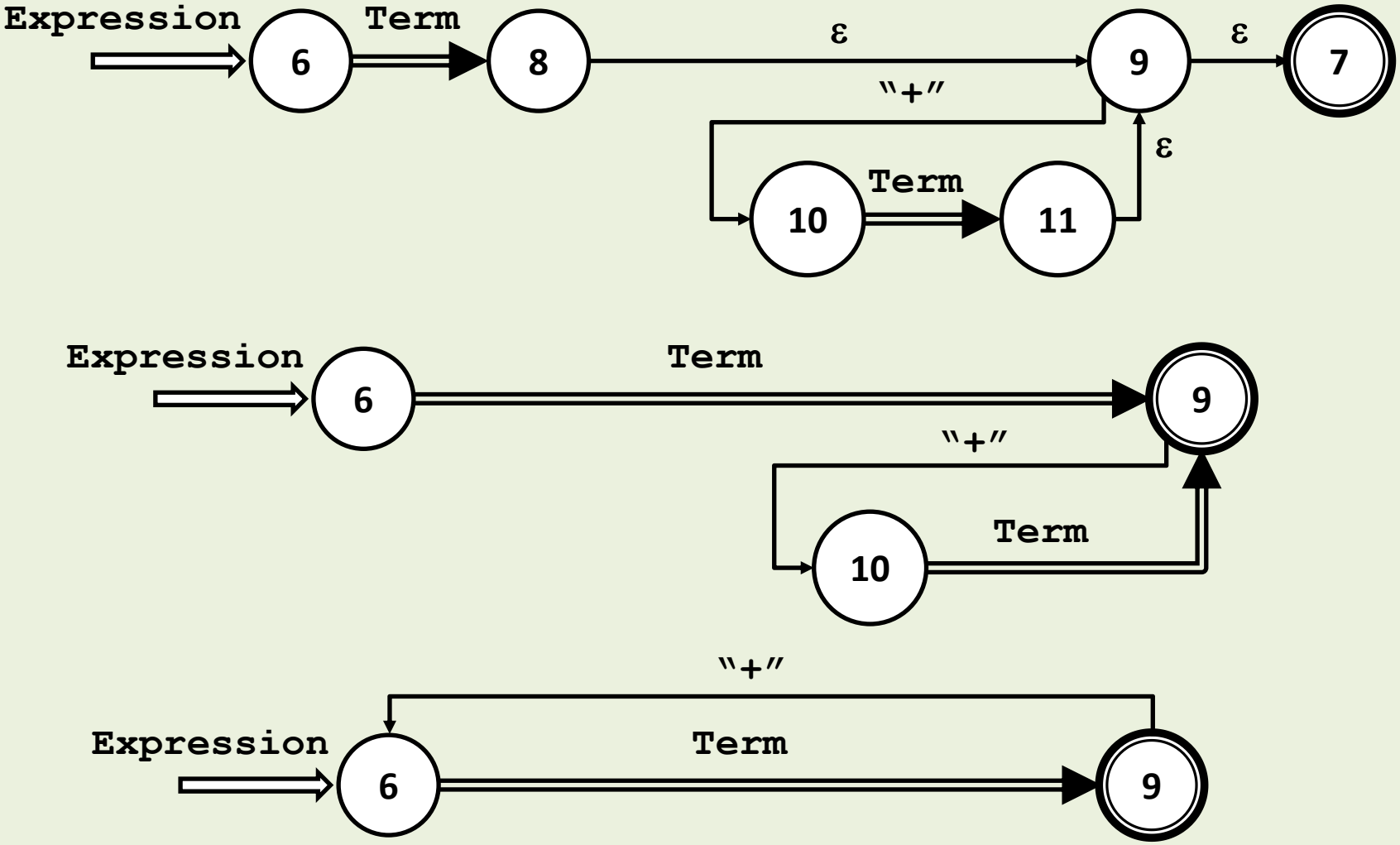
```
Term   =   Factor   {   "*"    Factor   }      .
         6         8    9    10         11  9        7
```



Expression

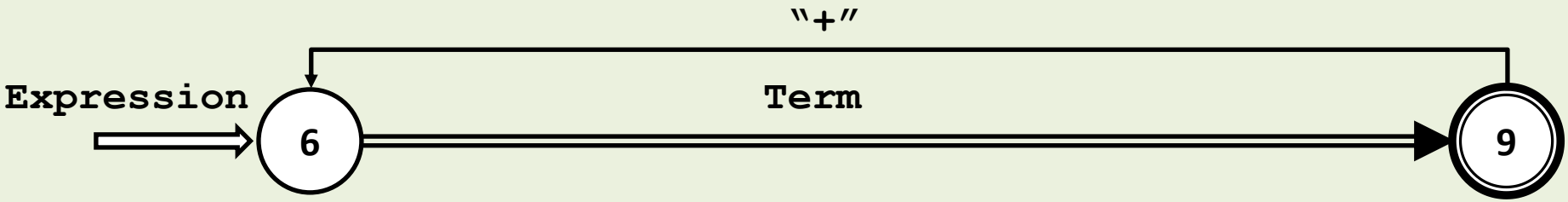
```
Expression = Term { "+" Term } .
```

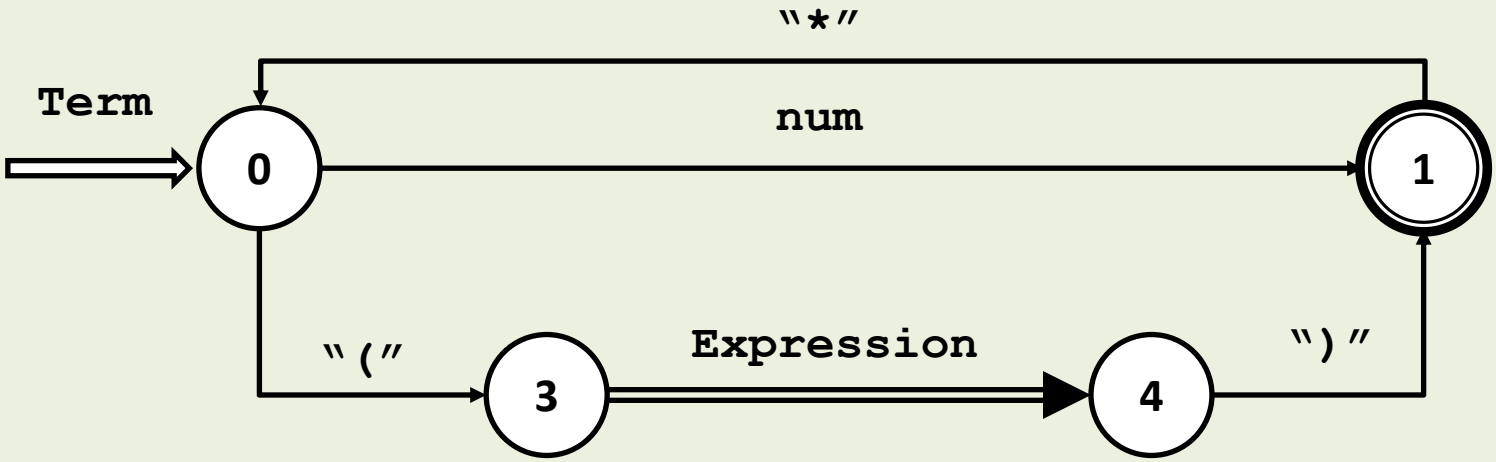
- Analogamente ao que foi feito com Term:



Expression

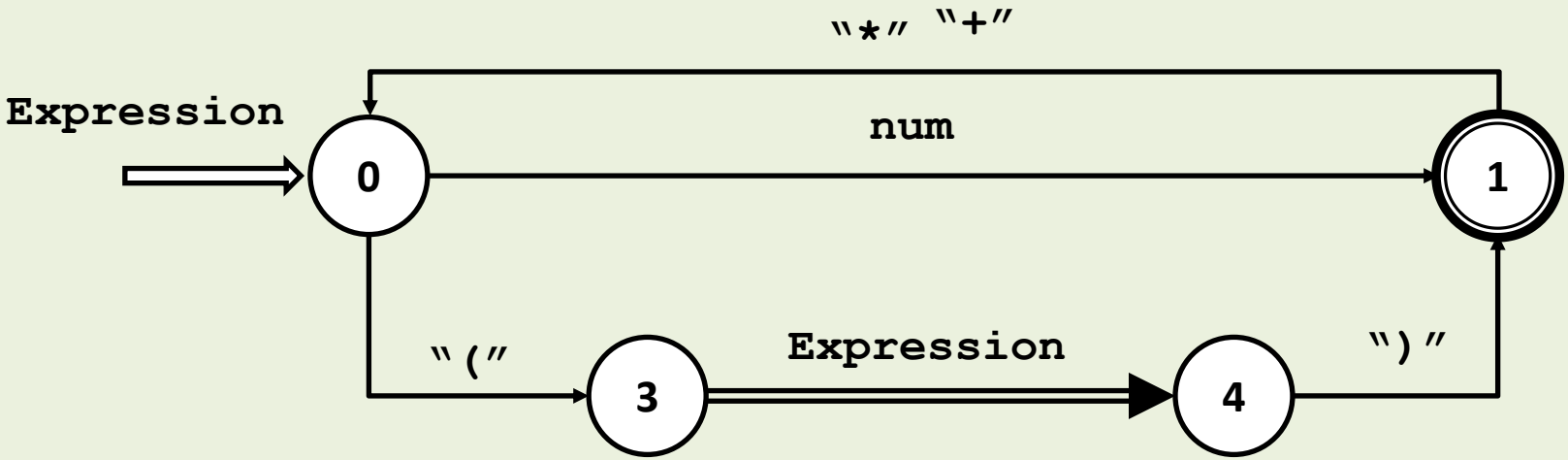
```
Expression = Term { "+" Term } .
```





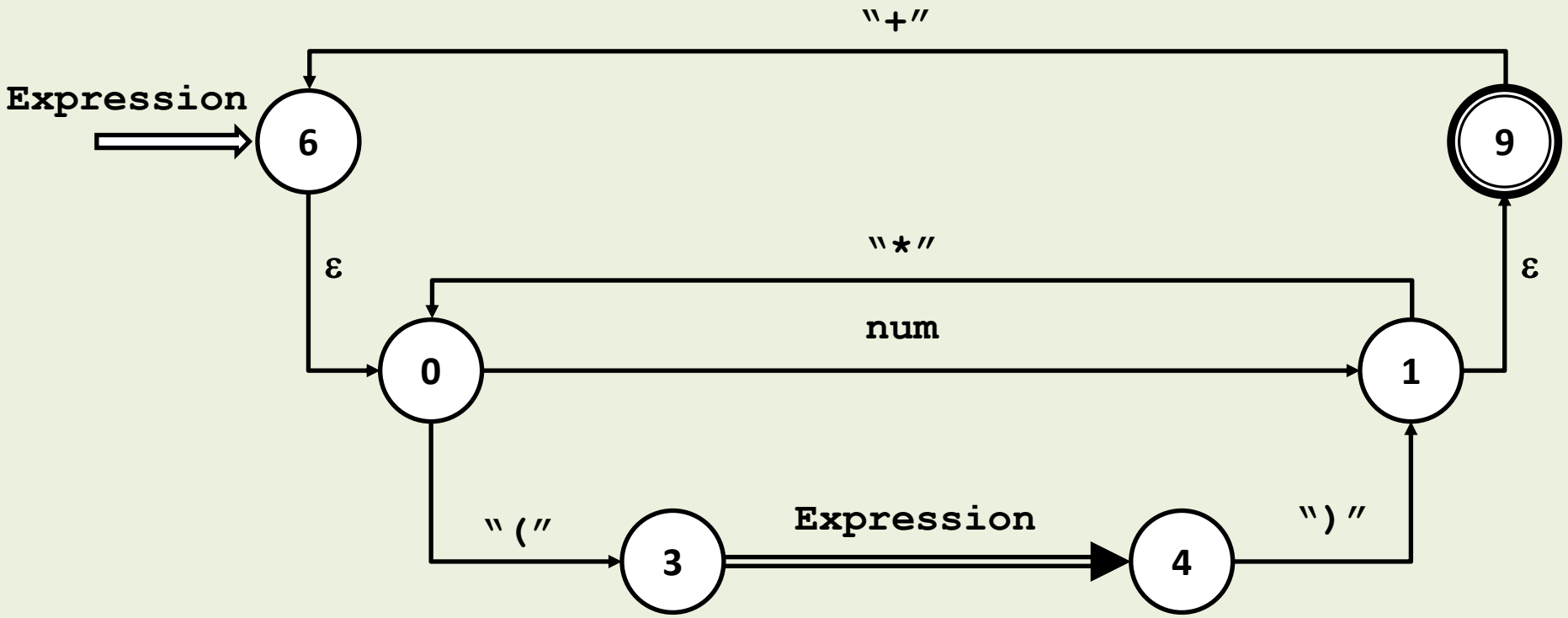
Expression

- Por simples observação, é possível acrescentar diretamente ao diagrama de Term uma transição adicional que consome "+", obtendo-se diretamente o seguinte diagrama:

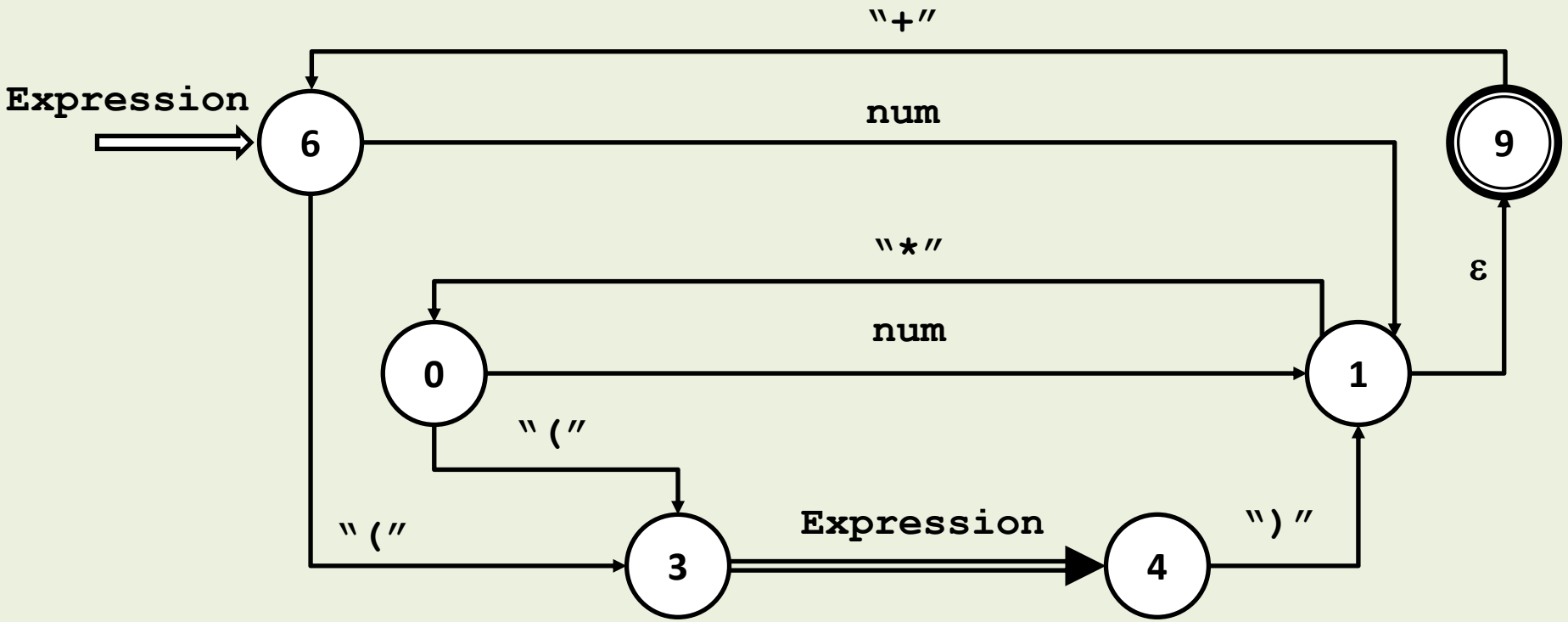


- Pode-se chegar a esse mesmo diagrama seguindo outro caminho, mais longo mas equivalente:

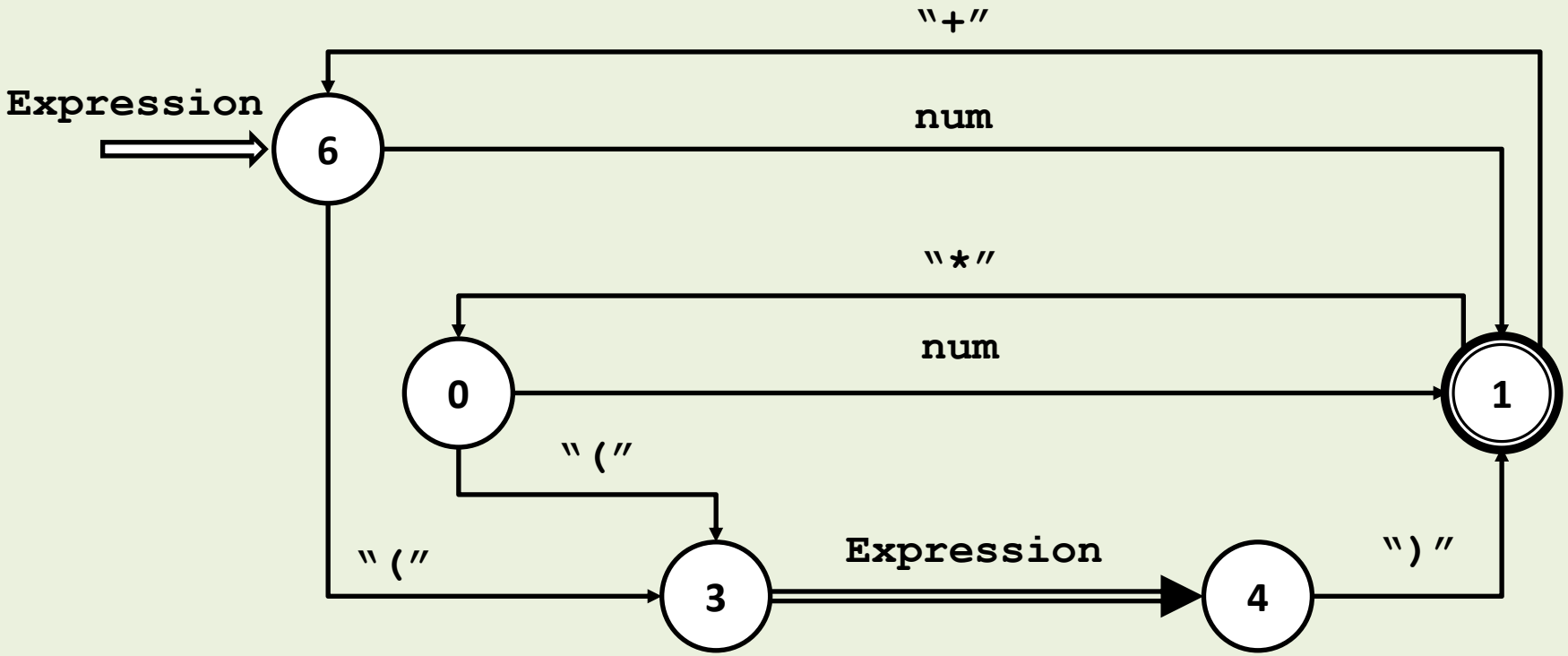
- Analogamente ao que foi feito com Term:



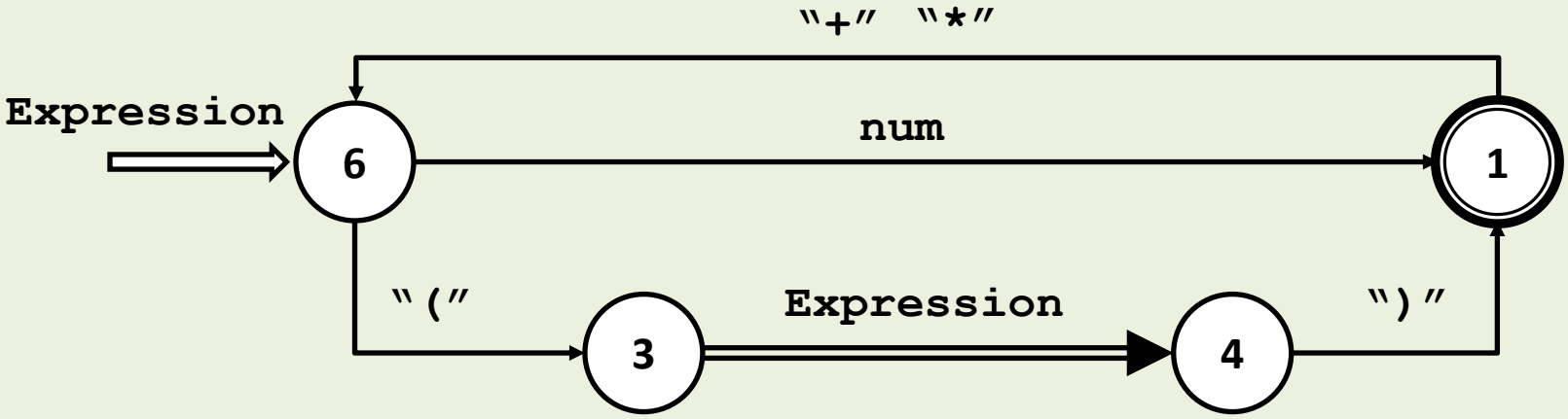
- Eliminando a transição em vazio 6-0:



- Eliminando a transição em vazio 1-9:



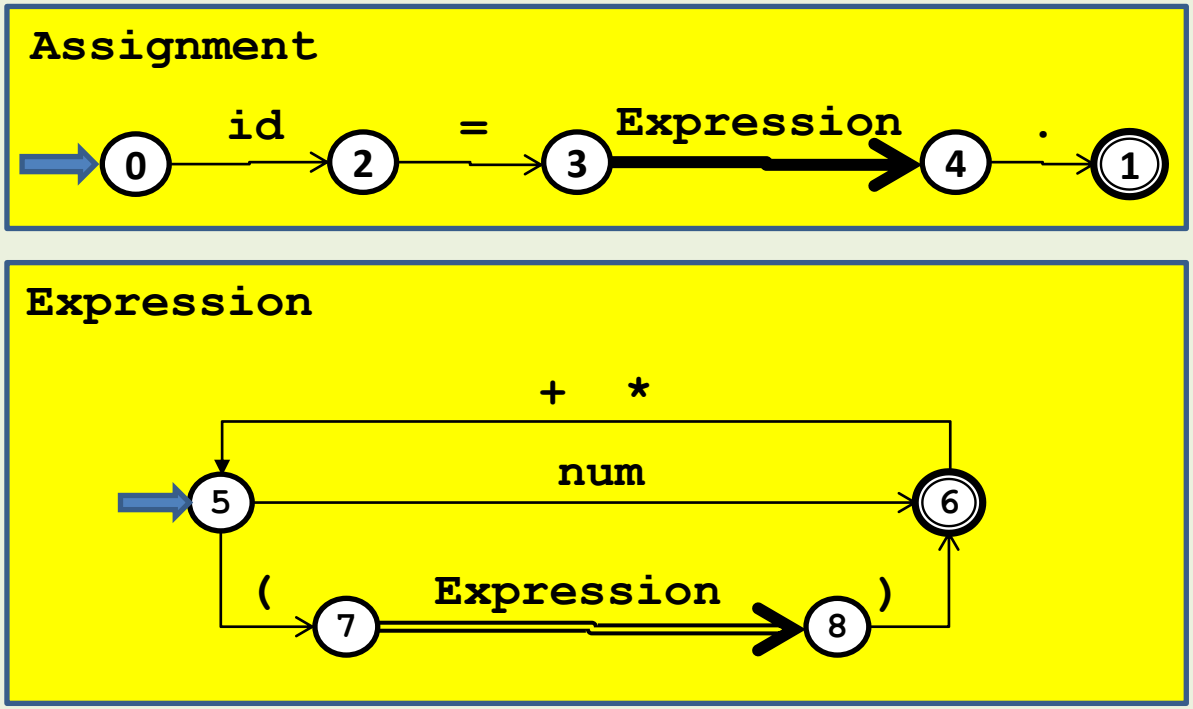
- Observando a equivalência dos estados 0 e 6, pode-se descartar um deles, por exemplo, o estado 0, obtendo-se finalmente esta versão mínima da submáquina **`Expression`** , idêntica à já apresentada anteriormente :



A partir da gramática apresentada, foi construído o reconhecedor abaixo, na forma de autômato de pilha estruturado:

- **Este autômato de pilha estruturado é capaz de reconhecer um comando de atribuição de valor de uma expressão muito simples a uma variável.**
  - **A submáquina `Assignment` é a sua submáquina principal**
  - **A submáquina `Expression` reconhece expressões restritas, que contenham apenas números e as operações de soma e de multiplicação.**
- **Por conveniência, foram evitados nomes iguais para estados diferentes.**

(sub-máquinas `Assignment` e `Expression`)



Exercícios

- Use o material, disponibilizado como complemento a esta aula, para compreender melhor o processo de geração automática com o qual você está trabalhando.
- Utilize-o também para verificar o funcionamento dos programas que você desenvolver neste e em outros exercícios, assim como para ajudar a localizar e remover eventuais erros remanescentes.
- Para alguma gramática que defina uma linguagem relativamente simples, aplique manualmente o algoritmo do transdutor aqui desenvolvido.
- Registre passo a passo o funcionamento do autômato gerado, para um pequeno programa-fonte fornecido na linguagem definida pela gramática do exercício anterior.
- Implemente o transdutor apresentado usando a técnica dos motores de eventos, e faça-o funcionar no seu computador.
- Use esse transdutor para gerar automaticamente o reconhecedor da linguagem que você desenvolveu no projeto em andamento.

Uso no projeto

- Utilize o meta-reconhecedor construído em duas grandes aplicações:
  - *Bootstrap*, reconstruindo automaticamente o reconhecedor da Notação de Wirth tradicional, a partir da sua própria gramática.
  - Construindo automaticamente o reconhecedor do Dartmouth BASIC (e outras linguagens que desejar) a partir de suas gramáticas, expressas na Notação de Wirth tradicional.