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

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

PCS 3446 – Aula 05

Problema a resolver

- Na aula anterior estudou-se informalmente um método para a obtenção manual de um reconhecedor sintático para linguagens livres de contexto.
- Nesta aula, propõe-se um exercício para criar familiaridade com esse método, enquanto proporciona a obtenção de mais uma ferramenta útil para quem deseja desenvolver reconhecedores, analisadores, interpretadores e compiladores para essas linguagens livres de contexto.
- Em particular, a real implementação de alguma solução para o problema proposto nesta aula propicia disponibilizar, se não toda, ao menos boa parte do *front-end* de um ambiente de geração automática de reconhecedores sintáticos para linguagens livres de contexto, formalizadas por meio de gramáticas expressas na Notação de Wirth.

Proposta de solução

- Como sabemos, para se poder fazer um reconhecedor sintático operante é preciso dispor de um categorizador, na forma de analisador léxico, nos moldes lógicos e estruturais vistos anteriormente.
- Todavia, no presente caso o analisador deverá extrair itens léxicos aderentes à linguagem da Notação de Wirth, pouco diferentes, portanto, daqueles tratados em aulas anteriores.
- Portanto, o analisador léxico necessário neste caso difere ligeiramente do estudado para o BASIC, exigindo um pequeno esforço adicional, para se obter o analisador léxico desejado.
- Para completar o gerador automático desejado, será preciso ainda criar um autômato que reconheça a sintaxe da Notação de Wirth, e decorá-lo com as rotinas semânticas adequadas.
- A cada passo do processamento de uma gramática de Wirth, será então executada a rotina semântica correspondente.

Operação

- A sequência resultante de execuções dessas rotinas destina-se especialmente a gerar o conjunto de regras de transição do autômato equivalente à gramática fornecida, e à gravação de uma representação dessas regras, em um arquivo de saída.
- A implementação dessa solução segue o método adotado nos demais módulos do compilador, usando:
  - Átomos da Notação de Wirth, operando como eventos
  - Uma instância do motor de eventos, operando como programa principal de um simulador de sistema reativo guiado por eventos
  - Um conjunto de procedimentos semânticos , que operam como rotinas de tratamento dos diversos tipos de eventos (reação), cada qual responsável por tratar um átomo da Notação de Wirth a que se refere, para a construção das transições do autômato, associadas ao elemento da gramática que estiver sendo então processado.

Gramática Wirth da Notação de Wirth

GRAMÁTICA SINTÁTICA

```
syntax = { rule } .
rule = identifier "=" expression "." .
expression = term { "|" term } .
term = factor { factor } .
factor = identifier | quoted_symbol | "(" expression ")"
        | "[" expression "]" | "{" expression "}" .
```

GRAMÁTICA LÉXICA

```
identifier = letter { letter | digit } .
quoted_symbol = """ { any_character } """ .
```

Gramática de Wirth que descreve a Notação de Wirth tradicional

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 } """ .
```

Análise léxica

- O tratamento dos identificadores que representam não-terminais em gramáticas na Notação de Wirth é idêntico ao já estudado para os identificadores das linguagens de programação.
- Os símbolos entre aspas, que se referem aos terminais, têm uma lei de formação um pouco diferente, porém similar ao formato das *strings* usualmente utilizadas nas linguagens de programação.

- Espaçadores (brancos, tabulações, fins de linha, comentários) são ignorados.
- Os demais símbolos úteis na gramática são os metassímbolos da Notação de Wirth:

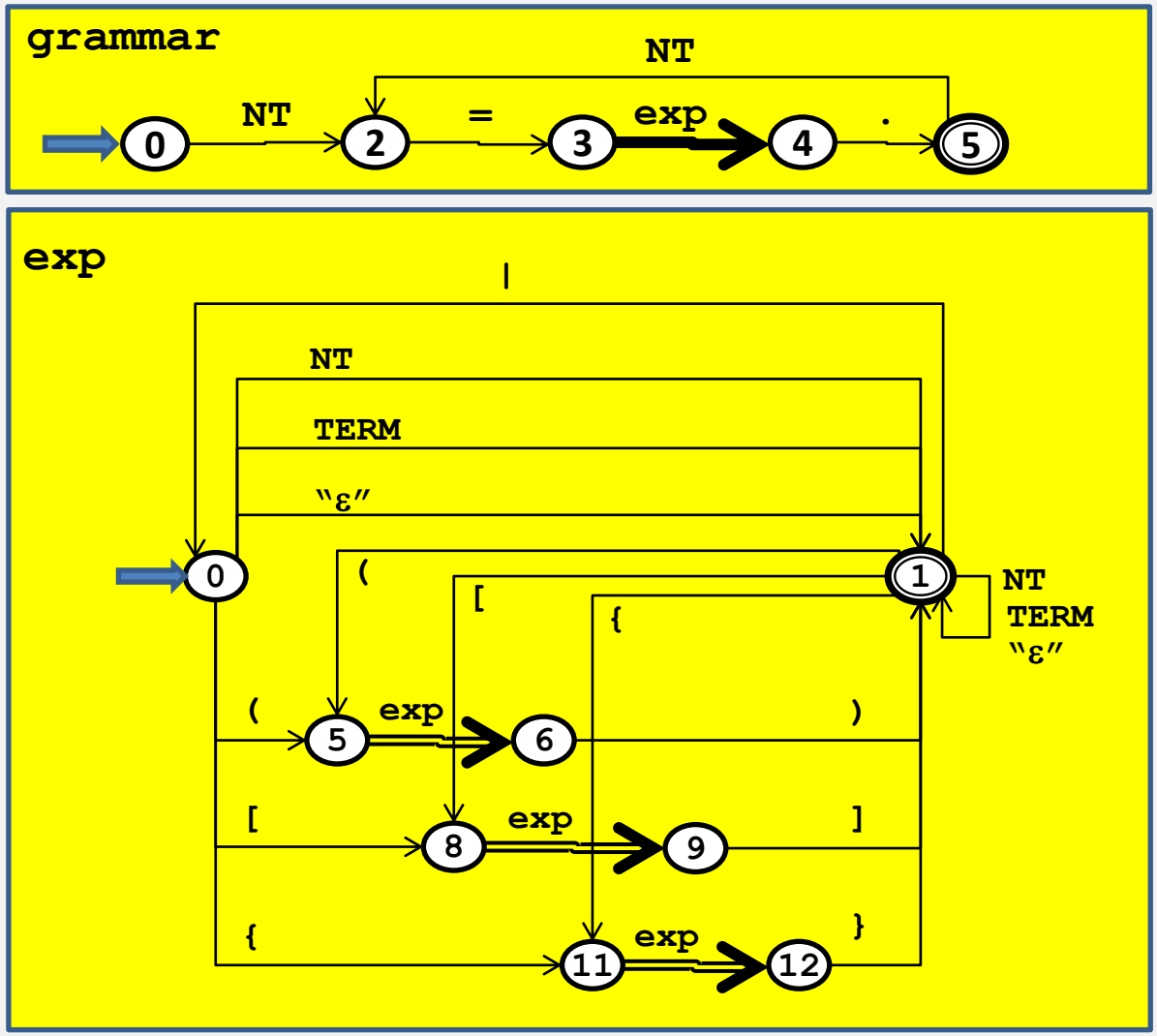
  { } [ ] ( ) ε . " = |
- A ocorrência de qualquer um dos demais caracteres ASCII fora de um *quoted symbol* deve ser considerada como uma situação de erro de sintaxe, cometido no uso da Notação de Wirth.

- Como átomos de saída deve-se portanto esperar que o analisador léxico extraia: *identifier, quoted symbol,* {, }, [,], (,), **"**, ., =, **ε**, |, σ, onde σ é qualquer caractere ASCII não explicitamente listado anteriormente, inclusive brancos, finais de linha e tabulações.
- O valor semântico associado a cada um deles é a respectiva cadeia de caracteres que lhe deu origem.
- A elaboração deste analisador léxico é trivial e fica como exercício.

Reconhecimento sintático

- Um autômato que efetua o reconhecimento sintático de gramáticas denotadas na Notação de Wirth está apresentado no slide seguinte.
- Sua construção é descrita adiante, passo a passo, a partir da **Gramática de Wirth que descreve a Notação de Wirth tradicional,** incluída em um slide anterior, nesta mesma apresentação.

Diagrama de um reconhecedor sintático de gramáticas em Notação de Wirth



Implementação

- Usando a técnica de simulação guiada por eventos, implemente um analisador léxico, de forma idêntica ao que foi feito na parte anterior do projeto, gerando como saídas os itens léxicos já mencionados: ***identifier, quoted symbol*, {, }, [,], (,), ", ., =, ε, |, σ**
- Implemente, também usando a técnica da simulação guiada por eventos, o reconhecedor de gramáticas de Wirth do slide anterior, para a qual os eventos são as saídas do analisador léxico.

Testes

- Use como dados para teste diversas gramáticas disponíveis em Notação de Wirth.
- Depois que todas elas forem devidamente aceitas pelo reconhecedor, use também como arquivo de dados para teste a gramática de Wirth que você está utilizando neste projeto.
- Aproveite a oportunidade para analisar melhor, completar, corrigir algum detalhe ou melhorar a gramática, se for o caso.

Para entregar, nesta parte do trabalho

- Espera-se, nesta fase do projeto:
  - Implementação (manual), usando motores de eventos, de um reconhecedor operante de gramáticas livres de contexto, expressas na Notação de Wirth, definida:
    - pelas especificações da análise léxica
    - pelo diagrama de estados completo do reconhecedor sintático
  - Verificação da sintaxe das gramáticas fornecidas no site, e eventuais indicações e correções dos erros encontrados.
  - Verificação e correção da gramática adotada para o projeto.
  - Listagens dos rastros de execução do reconhecedor para algumas gramáticas de entrada que utilizem, em várias situações diferentes, todos os recursos da Notação de Wirth.

Objetivos

- A meta principal do assunto tratado nesta e na próxima aula é tripla:
  - Desenvolver a intuição do uso de reconhecedores sintáticos como base para a construção de **processadores de linguagens** dirigidos pela sintaxe
  - Exercitar a construção de rotinas semânticas voltadas à **automatização dos fenômenos ligados ao processamento de linguagens** de programação
  - Fornecer subsídios para a **extensão [“semântica”] de um reconhecedor sintático** de forma que se obtenha um transdutor, um compilador, ou um interpretador da linguagem

Projeto de um meta-reconhecedor para linguagens livres de contexto

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

- **Gramáticas**, fornecidas como textos expressos na Notação de Wirth, representam uma descrição formal da sintaxe das linguagens que descrevem.
- **Meta-reconhecedores** são quaisquer programas que, a partir de informações que formalizam a sintaxe de alguma particular linguagem, sejam capazes de gerar um reconhecedor sintático para essa linguagem.

- Programas dessa natureza necessitam, portanto, ler e manipular a **Notação de Wirth** para terem acesso às informações, contidas nas gramáticas, acerca da sintaxe das linguagens por elas descritas.
- Sendo assim, podemos iniciar construindo um reconhecedor sintático para gramáticas expressas na Notação de Wirth.

- Considerando como eventos que se sucedem os movimentos desse reconhecedor, é possível associar a cada um desses movimentos uma ação, como se faz na programação guiada por eventos.
- Assim, toda vez que **ocorrer uma transição (= evento)**, a rotina a ela associada **(= rotina de tratamento do evento)** será também executada.
- Começamos pela construção do reconhecedor sintático da Notação de Wirth.

CONSTRUÇÃO DO RECONHECEDOR SINTÁTICO DA NOTAÇÃO DE WIRTH

- Devemos em primeiro lugar construir um reconhecedor da linguagem de entrada do meta-reconhecedor desejado, ou seja, da Notação de Wirth.
- Vamos aplicar em seguida o mesmo método previamente utilizado para obter um autômato a partir de uma gramática, expressa em Notação de Wirth.
- Partimos portanto de uma descrição formal da sintaxe da Notação de Wirth, representada por uma gramática denotada também, ela mesma, na Notação de Wirth.

Gramática de Wirth que descreve a Notação de Wirth tradicional

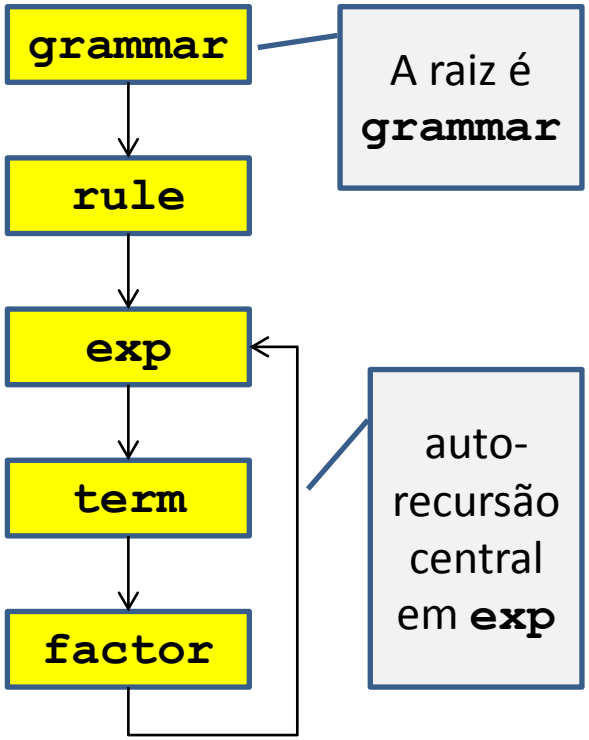
Regras gramaticais da Sintaxe:

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

Gramática Léxica (só NT e T, omitidas as triviais):

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

Grafo de dependências dos não-terminais da gramática:



Observações

- Notar que no slide anterior a linguagem foi descrita em duas partes:
  - Gramática sintática, que descreve as grandes estruturas da Notação de Wirth
  - Gramática léxica, que fornece a lei de formação dos elementos compostos de diversos caracteres, utilizados na gramática sintática como elementos indivisíveis (os terminais **`T`** e os não-terminais **`NT`**)
- Como é usual, estão omitidas na gramática léxica (ficando, portanto, subentendidas) as regras gramaticais triviais, que descreveriam *tokens* compostos de um único caractere ASCII.
- Por questão de simplicidade, está omitido, na maior parte dos diagramas de estados apresentados adiante, o rótulo **ε** nas transições em vazio.

Lema de Arden

- Da teoria, sabe-se (Lema de Arden) que em dois casos é possível eliminar regras gramaticais auto-recursivas:
  - Regras **X = X a | b** se resolvem como **X = b {a}.**
  - Regras **Y = a Y | b** se resolvem como **Y = {a} b.**
- Sabe-se também que, no casos geral, não-terminais auto-recursivos centrais não são elimináveis :
  - Linguagens regulares compõem suas sentenças usando apenas concatenações, uniões de conjuntos e fechos de Kleene, e podem ser reconhecidas por autômatos finitos.
  - Linguagens livres de contexto não regulares apresentam não-terminais com auto-recursões centrais, não elimináveis: Regras **Z = a Z b | c** se resolvem como **$Z = a^n c b^n$. $n \geq 0$** (e **não** como ~~**Z = {a} c {b}.**~~)
  - Linguagens livres de contexto não regulares não podem ser reconhecidas por autômatos finitos, mas precisam de autômatos de pilha para o seu reconhecimento.

Determinação dos não-terminais essenciais

- Montando-se o grafo de dependências dos não-terminais envolvidos na sintaxe, observa-se que são essenciais:
  - **`grammar`** – por ser a raiz da gramática
  - **`exp`** – por ser auto-recursivo central

  Os demais não-terminais podem ser eliminados.
- Note-se que não há nesta gramática definições auto-recursivas à direita nem à esquerda, que possam ser eliminadas aplicando-se o Lema de Arden.
- Assim, todas as eliminações podem ser feitas apenas por substituição.

Gramática léxica

Gramática léxica

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

Outros átomos:

```
"=" "." "|" "ε" "(" ")" "[" "]" "{" "}"
```

O analisador léxico para um reconhecedor da Notação de Wirth deverá extrair tokens das formas **T** e **NT**, além de símbolos como igual, ponto, barra vertical, epsilon, abre e fecha parênteses, colchetes e chaves.

Não-terminal `grammar`

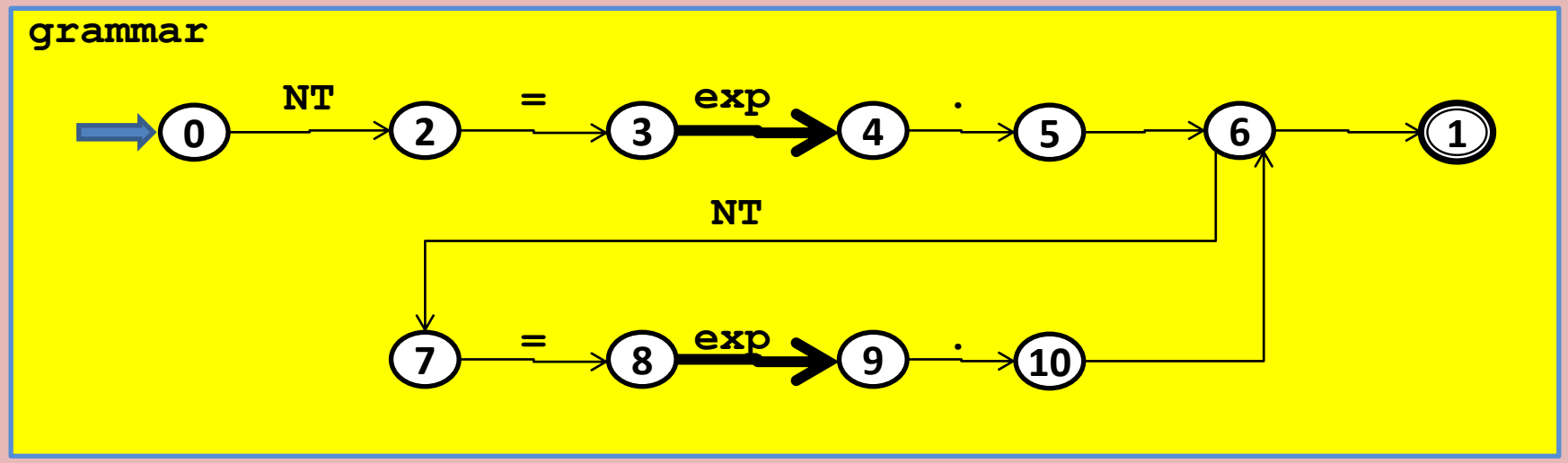
```
grammar = rule { rule } .
rule = NT "=" exp "." .
```

Eliminando `rule` por substituição:

```
grammar =  NT "=" exp "." { NT "=" exp "." } .
```

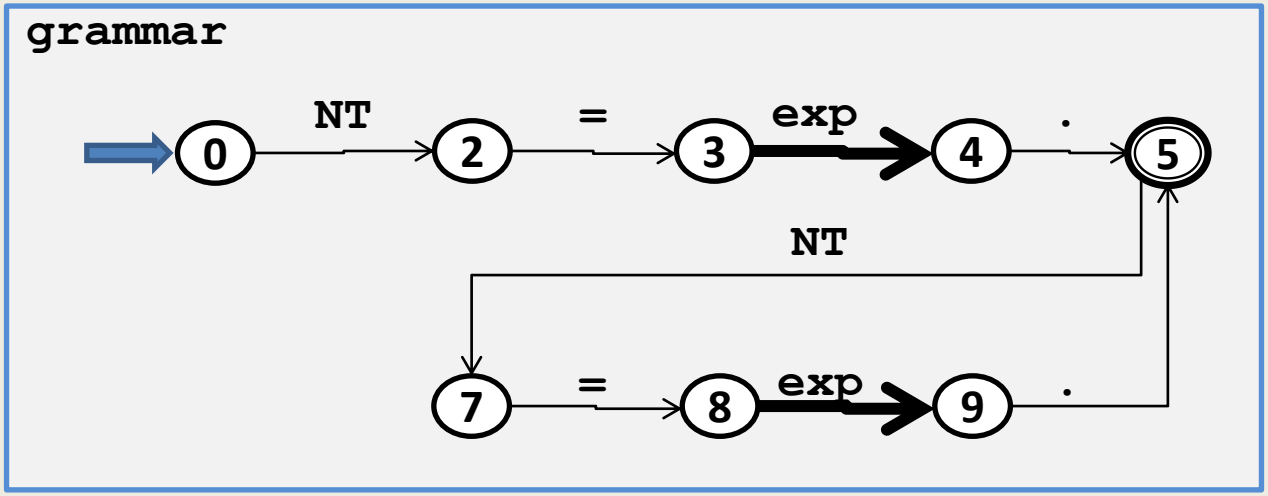
Sub-máquina grammar:

```
grammar =  NT "=" exp "." { NT "=" exp "."  } .
           0   2    3    4   5  6   7    8    9   10  6  1
```



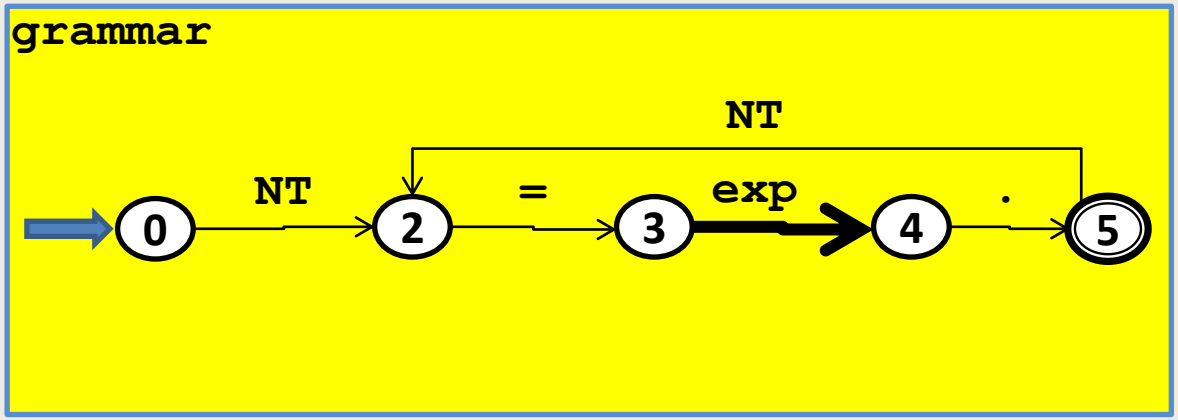
Sub-máquina `grammar`:

- Eliminando as transições em vazio fica assim:



Sub-máquina `grammar`:

- Eliminando transições redundantes:



Não-terminal `exp`

```
exp = term { "|" term } .
term = factor { factor } .
```

Eliminando `term` por substituição:

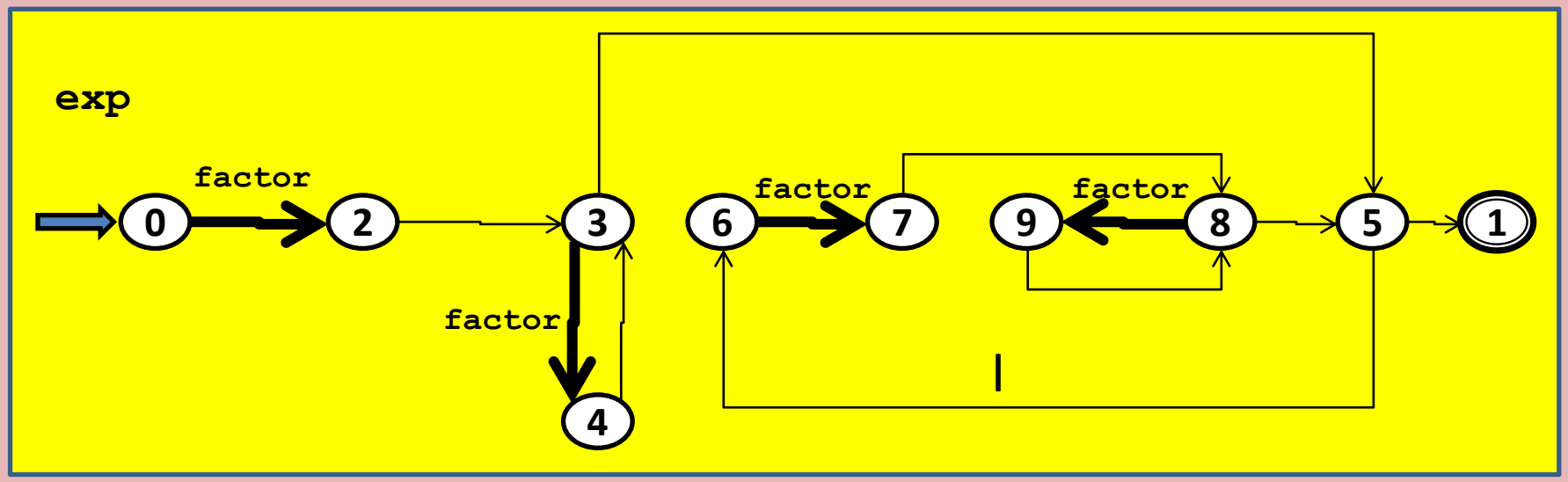
```
exp = factor { factor } { "|" factor { factor } } .
```

```
exp = factor
    { factor }
{ "|" factor
    { factor } } .
```

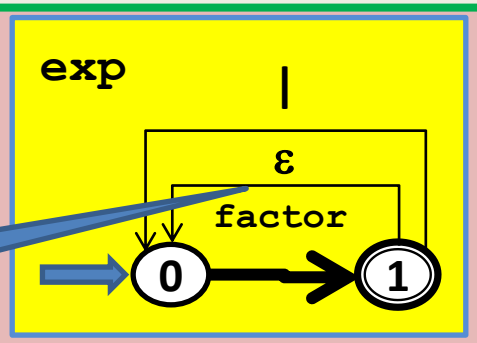
A ocorrência quádrupla de `factor` sugere que não se faça sua substituição nesta ocasião, evitando assim repetidas e desnecessárias manipulações extensas.

Sub-máquina exp

```
exp = factor { factor } { "|" factor { factor } } .
      0       2 3       4 3 5   6       7 8       9 8 5 1
```



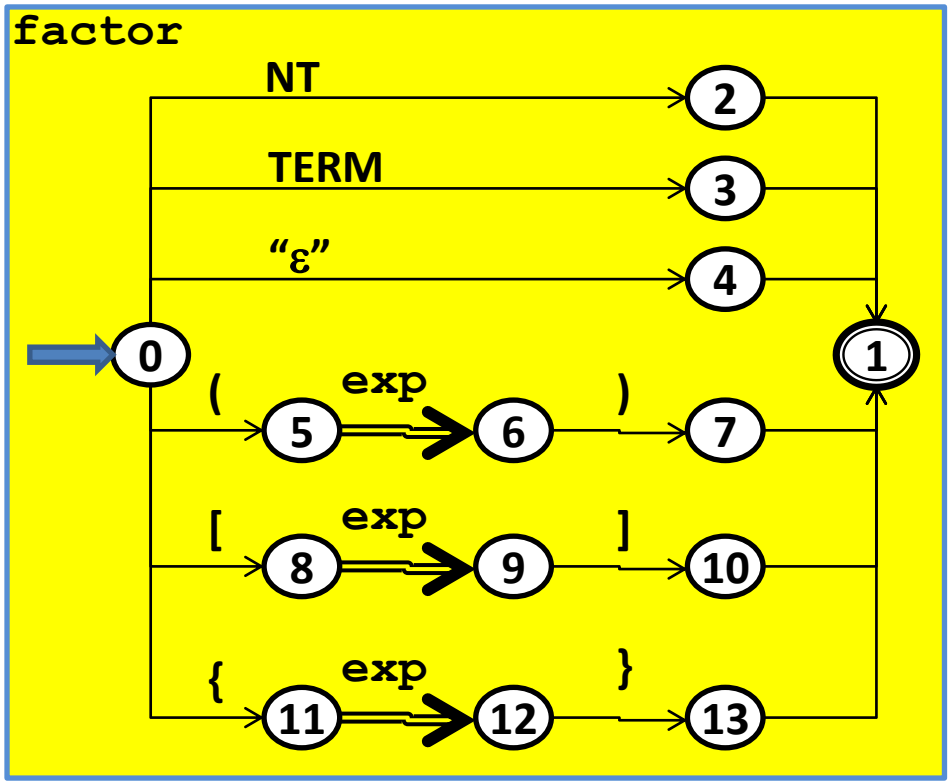
Simplificando (faça você, como exercício, passo a passo, a eliminação de transições em vazio e a redução dos estados equivalentes). Obtém-se ao final o seguinte diagrama:



Note que esta é uma transição em vazio...

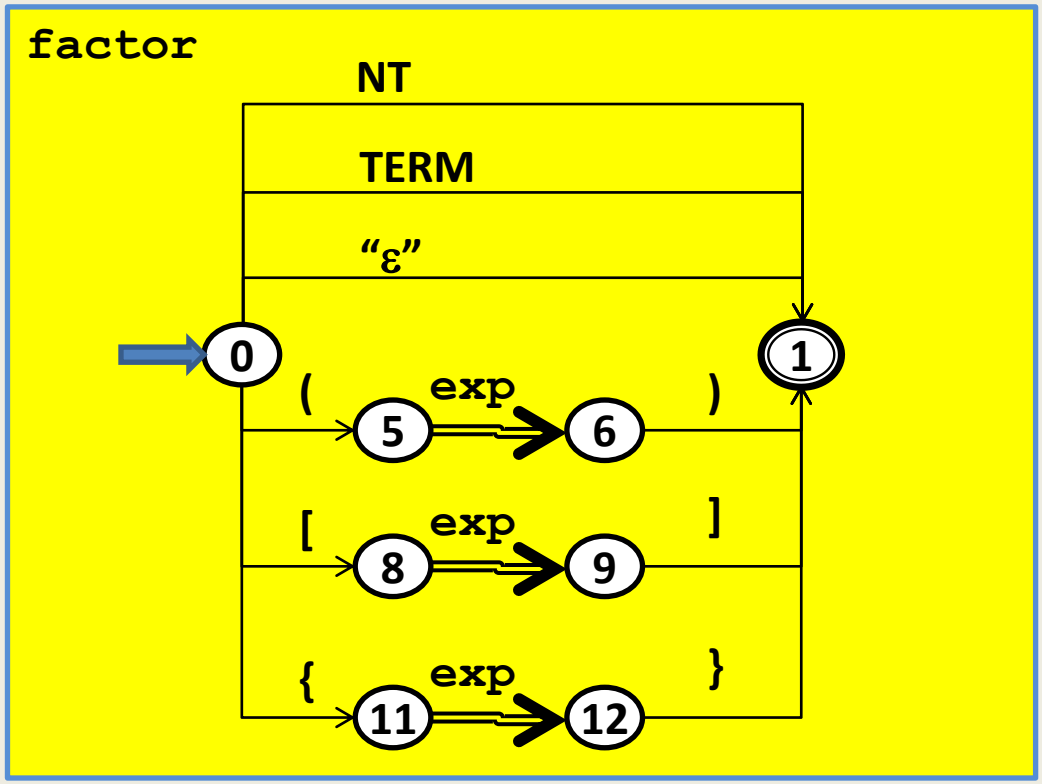
Sub-máquina `factor`

```
factor =  NT | TERM | "ε" | "(" exp ")" | "[" exp "]" | "{" exp "}"  .
          0   2 0    3 0   4  0   5    6   7  0   8    9  10 0   11   12  13  1
```

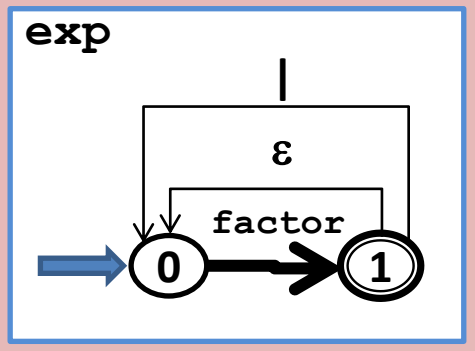


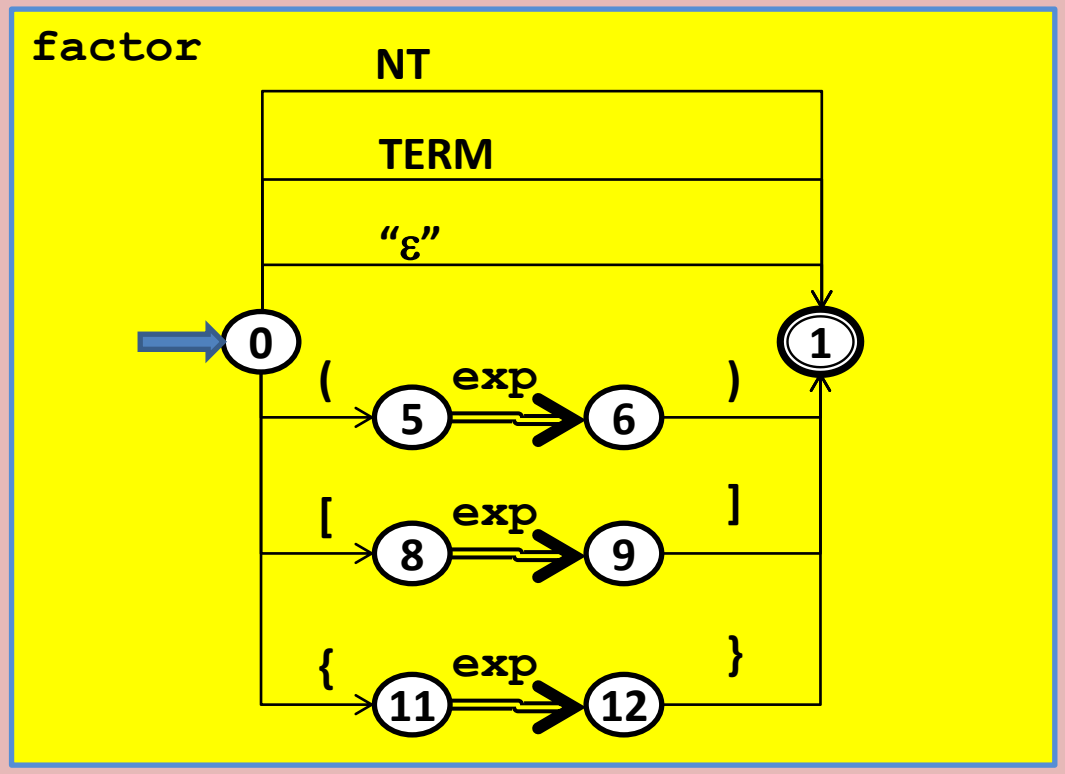
Sub-máquina `factor`

- Eliminando transições em vazio fica assim:



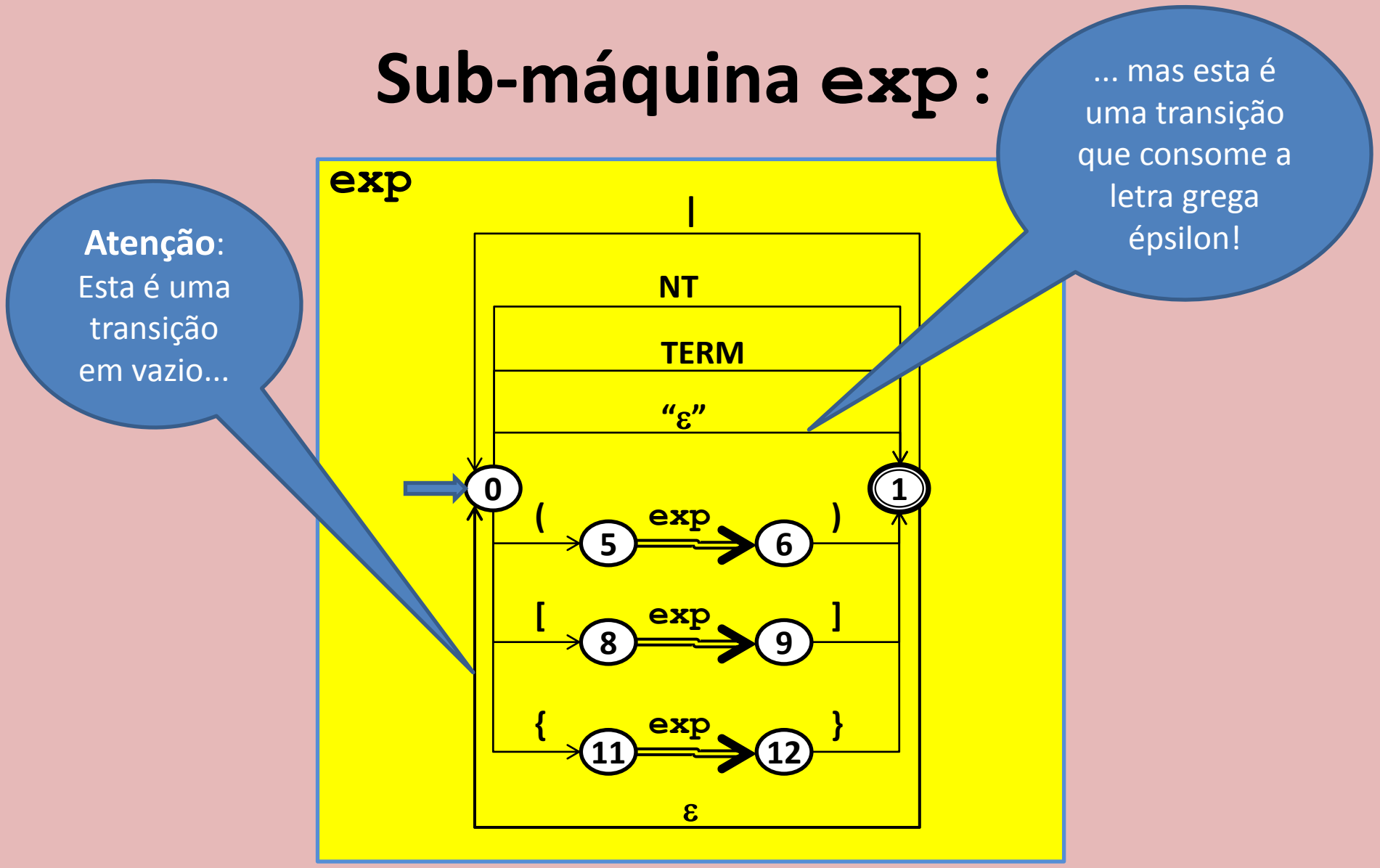
Sub-máquina exp :





- Expandindo o segundo diagrama (**factor**) no primeiro (**exp**):

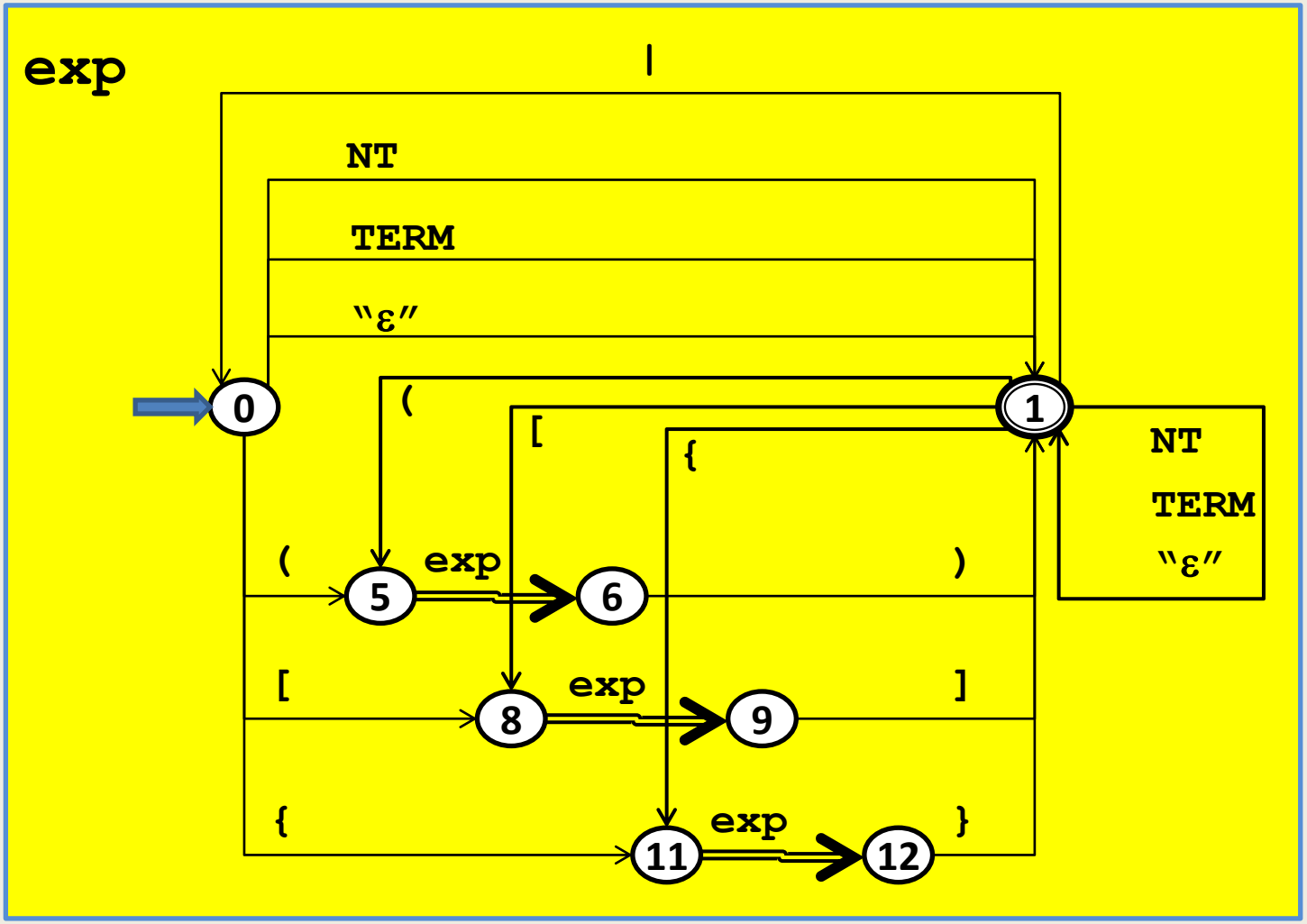
Sub-máquina exp :



- Eliminando a transição em vazio do estado 1 para o estado 0,

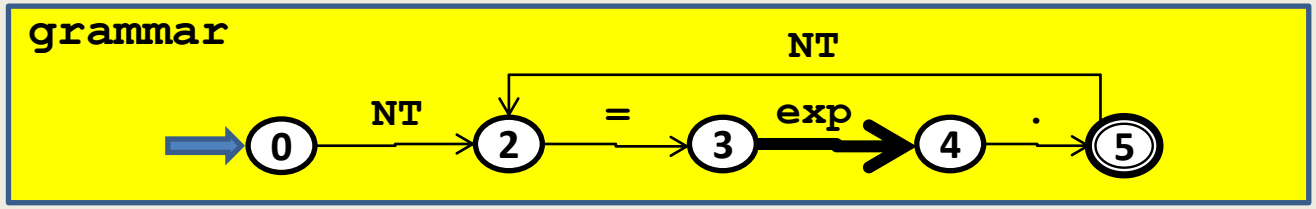
Sub-máquina exp :

- Após a simplificação, obtém-se para **exp** o diagrama de estados abaixo.
- Obs.: Notar que este diagrama tem mais transições que o anterior!

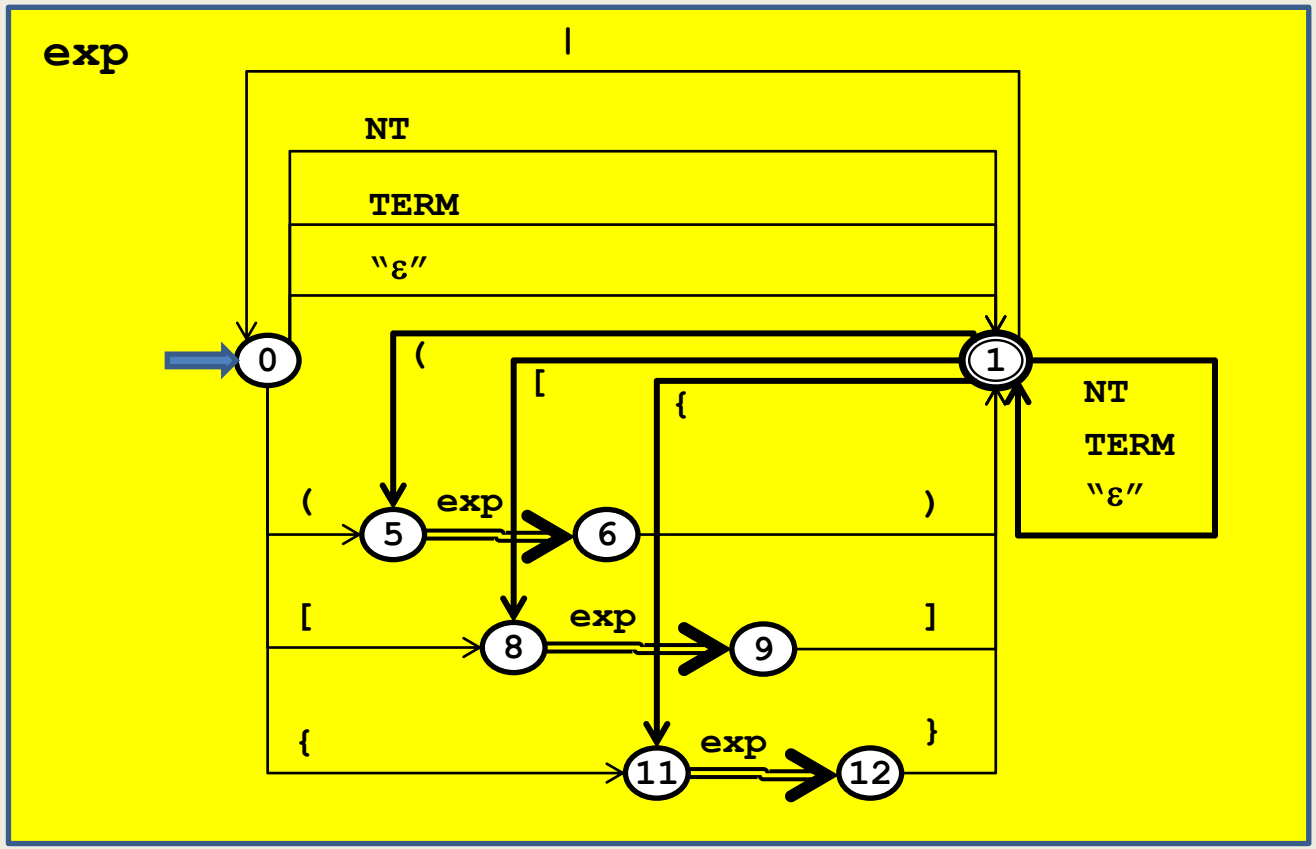


Autômato final:

- Notar que o autômato assim obtido é formado de duas submáquinas, a primeira **grammar** (raiz), e a segunda **exp**, auto-recursiva independente:
- Sub-máquina **grammar**:



- Sub-máquina **exp**:



Próximos Passos

- Antes de prosseguir, convém explorar, neste ponto, a similaridade sintática que existe entre as expressões da Notação de Wirth e as expressões aritméticas das linguagens usuais de programação.
- Para isso, propõe-se como exercício a construção, passo a passo, do reconhecedor sintático de uma linguagem de programação envolvendo expressões aritméticas.
- Tendo a gramática do enunciado a seguir como referência, isole as correspondentes regras da gramática do Basic para resolver este exercício.
- O passo seguinte a este pequeno exercício é incorporar rotinas semânticas, pauta da sexta aula.

Exercício

- Utilizando a técnica de programação guiada por eventos, **implemente** o reconhecedor da Notação de Wirth, cujo projeto foi apresentado nos slides anteriores.
- Teste-o alimentando o reconhecedor obtido com a gramática completa do Dartmouth BASIC fornecida na segunda aula, e também com as pequenas gramáticas utilizadas nesta aula. Todas essas gramáticas devem ser aceitas pelo reconhecedor como sintaticamente corretas.
- Introduza nessas gramáticas diversos tipos de erros de sintaxe e submeta-as ao reconhecedor. Ele deve rejeitá-las.
- Guarde o reconhecedor assim validado, para transformá-lo em um meta-reconhecedor, conforme será estudado na sequência, nesta disciplina.

Exercício

- Dada a seguinte gramática em Notação de Wirth:

```
Program = Assignment { ";" Assignment } .
Assignment = "LET" id "=" Expression .
Expression = Term { ( "+" | "-" ) Term } .
Term = Factor { ( "*" | "/" ) Factor } .
Factor =  id | num | "(" Expression ")" .
```

- Seguindo os passos mostrados para a construção do reconhecedor para a Notação de Wirth, acrescentar às regras acima uma gramática léxica e, a partir da gramática completa, construir um autômato de pilha estruturado que reconheça a linguagem assim formalizada.
- Para o programa abaixo, construa manualmente um "log" passo a passo (informando em cada passo o estado corrente do autômato, o símbolo corrente de entrada, em tratamento, e o conteúdo da pilha) de todo o processo de reconhecimento sintático:

```
LET a = a / 2 + ( b - 3 ) * ( 3 + b ) ;
LET a = ( ( ( b ) ) * ( 2 - c ) ) ;
LET a = b + ( ( 2 * c ) / ( d - e ) + 1 )
```