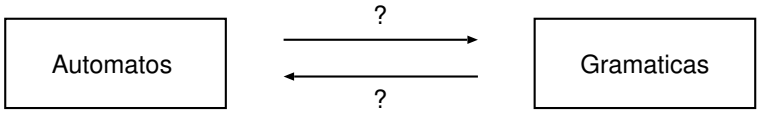
Teoria dos Autômatos e Linguagens Formais

aula 23: Gramáticas e autômatos

1 Introdução

Na aula de hoje, nós vamos começar a investigar a relação entre os autômatos e as gramáticas



Mas, antes de entrar nessa discussão, é útil considerar um fato um tanto surpreendente:

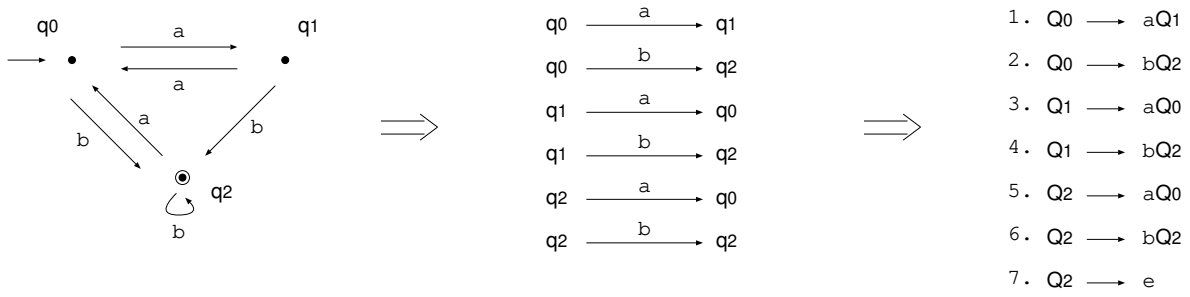
- *Todo autômato finito é uma gramática (disfarçada)!*

Quer dizer, a única diferença entre eles é a maneira como as regras são apresentadas.

No autômato as regras são apresentadas na forma de um diagrama de estados e transições, e na gramática elas são apresentadas na forma de uma lista.

Portanto, para obter a gramática que gera a linguagem reconhecida por um autômato finito, basta desmontar o seu diagrama.

Exemplo 1: um autômato finito e a sua gramática



Note que os estados do autômato correspondem às variáveis da gramática, e que as transições do diagrama correspondem às regras de substituição.

Uma consequência imediata dessa construção é que *caminhos* no diagrama correspondem a *derivações* na gramática.

Por exemplo, o caminho

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_2 \xrightarrow{a} q_0 \xrightarrow{b} q_2 \xrightarrow{b} q_2$$

corresponde à seguinte derivação

$$Q_0 \xrightarrow{(1)} aQ_1 \xRightarrow{(3)} aaQ_0 \xRightarrow{(2)} aabQ_2 \xRightarrow{(5)} aabaQ_0 \xRightarrow{(2)} aababQ_2 \xRightarrow{(6)} aababbQ_2$$

Observe que a sequência de símbolos que é lida pelo autômato durante o caminho aparece como o segmento inicial da palavra que está sendo construída pela derivação.

Além disso, para indicar na gramática que $\mathtt{q}_2$ é um estado final, e que se a computação parar nesse ponto a palavra é aceita pelo autômato, nós adicionamos a seguinte regra

$$\mathtt{Q}_2 \to \mathtt{e}$$

Dessa maneira, a derivação também pode parar nesse ponto, de modo que `aababb` é uma palavra da linguagem gerada pela gramática.

◇

Examinando a gramática que foi obtida no exemplo acima, nós observamos a seguinte peculiaridade: todas as suas regras tem a forma

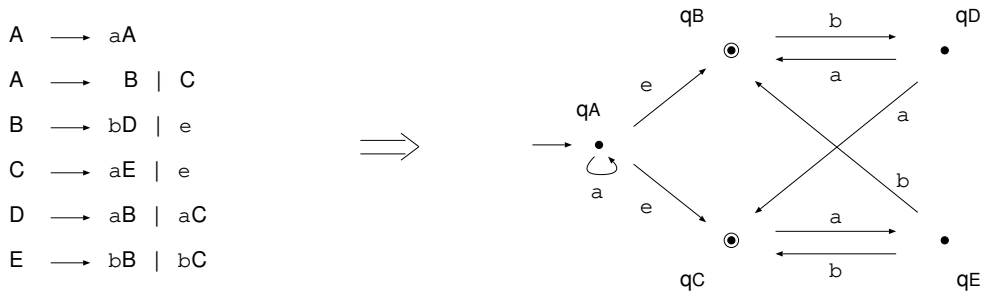
$$S \;\to\; \mathtt{s}\,T \qquad\qquad \text{ou} \qquad\qquad S \;\to\; \mathtt{e} \tag{1}$$

A questão natural que surge nesse ponto é:

- *Se todas as regras de uma gramática possuem esse formato simples,*
  será que é possível transformá-la em um autômato que reconhece a mesma linguagem?

A resposta é: Sim, basta montar um diagrama de estados e transições com as suas regras.

Exemplo 2: uma gramática simples e o seu autômato



Note que o autômato obtido é não-determinístico, mas nós já sabemos que isso não é um problema.

◇

Na verdade, as regras da gramática não precisam ser tão simples assim.

Basta que o lado direito de cada regra contenha no máximo uma variável, e que essa variável sempre apareça depois (ou antes) que todos os outros símbolos no lado direito da regra.

Por exemplo,

$$A \;\to\; \mathtt{ba}\,B \qquad\qquad \text{e} \qquad\qquad B \;\to\; \mathtt{aab}$$

As gramáticas que satisfazem essa condição são conhecidas como *gramáticas regulares.*

E o resultado interessante que nós acabamos de ver é que

- a linguagem gerada por uma gramática regular é sempre uma linguagem regular
- e toda linguagem regular pode ser gerada por uma gramática regular.

2 Gramáticas e autômatos de pilha

Como se pode imaginar, a relação entre as gramáticas e os autômatos de pilha é um pouco mais complicada.

E na aula de hoje nós vamos ver apenas o lado mais fácil da história:

- O fato de que toda linguagem gerada por uma *gramática livre de contexto* pode ser reconhecida por um *autômato de pilha*.

Quer dizer, as gramáticas que nós temos encontrado até o momento também tem uma restrição:

→ O lado esquerdo da regra sempre contém apenas uma variável (e nenhum outro símbolo)

As gramáticas que satisfazem essa condição são conhecidas como gramáticas *livres de contexto*[1].

Mas, o lado direito das regras pode ter qualquer formato.

E é isso o que complica as coisas ...

Nós vamos começar examinando um caso que ainda não envolve grandes problemas.

Considere a regra

$$A \quad \rightarrow \quad \texttt{aa}\, B\, \texttt{ba}$$

Caso não houvesse o fragmento `ba` no final do lado direito, nós poderíamos interpretá-la como a transição de $\mathsf{q_A}$ para $\mathsf{q_B}$ em um autômato que lê o fragmento `aa` na palavra de entrada.

Mas desta vez nós estamos trabalhando com os autômatos de pilha.

Então, nós podemos pensar em uma transição que lê o fragmento `aa` e empilha o fragmento `ba`

$$\mathsf{q_A} \xrightarrow{\texttt{aa[e,ba]}} \mathsf{q_B}$$

Agora, considere o par de regras

$$A \quad \rightarrow \quad \texttt{aa}\, B\, \texttt{ba} \qquad\qquad B \quad \rightarrow \quad \texttt{b}\, A\, \texttt{a}$$

e a seguinte derivação

$$A \quad \Rightarrow \quad \texttt{aa}\, B\, \texttt{ba} \quad \Rightarrow \quad \texttt{aa b}\, A\, \texttt{a ba} \quad \Rightarrow \quad \texttt{aa b aa}\, B\, \texttt{ba a ba}$$

A ideia é que essa derivação corresponde ao caminho de computação

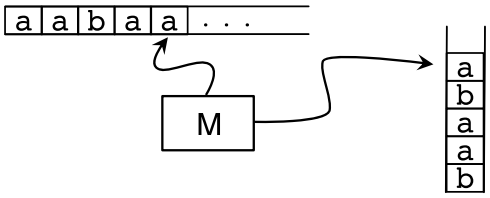
$$\mathsf{q_A} \xrightarrow{\texttt{aa[e,ba]}} \mathsf{q_B} \xrightarrow{\texttt{b[e,a]}} \mathsf{q_A} \xrightarrow{\texttt{aa[e,ba]}} \mathsf{q_B}$$

que lê a porção inicial `aabaa` da palavra de entrada, e empilha a sequência de símbolos `baaba`.

[1] O termo *livre de contexto* vem da observação de que uma regra como

$$X \quad \rightarrow \quad \texttt{a}\, Y\, \texttt{b}$$

permite que você substitua a variável X pelo termo que se encontra no lado direito, não importa onde ela apareça — i.e., em qualquer contexto.



Nesse ponto, se nós aplicamos a regra

$$B \quad \to \quad \mathtt{e}$$

a derivação chega ao fim

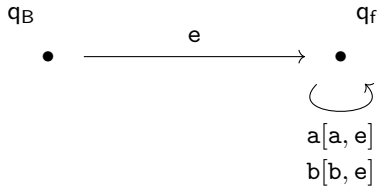
$$A \quad \Rightarrow \quad \mathtt{aa}\,B\,\mathtt{ba} \quad \Rightarrow \quad \mathtt{aa\,b}\,A\,\mathtt{a\,ba} \quad \Rightarrow \quad \mathtt{aa\,b\,aa}\,B\,\mathtt{ba\,a\,ba} \quad \Rightarrow \quad \mathtt{aa\,b\,aa\,ba\,a\,ba}$$

tendo produzido a palavra `aabaabaaba`.

Mas, o autômato de pilha ainda precisa terminar o seu trabalho.

Quer dizer, ele ainda tem uma sequência de símbolos na pilha que precisam ser comparados com o restante da palavra de entrada.

E a ideia aqui é que isso pode ser feito realizando uma transição para um estado final $\mathtt{q_f}$



Portanto, se a nossa gramática só contém regras da forma

$$A \quad \to \quad \alpha\,B\,\gamma \qquad\qquad B \quad \to \quad \beta \qquad\qquad B \quad \to \quad \mathtt{e}$$

onde α, γ, β são sequências de símbolos, nós sempre podemos converte-la para um autômato de pilha que reconhece a mesma linguagem.

Abaixo nós temos um exemplo concreto.

Exemplo 3: a gramática de PAL e o seu autômato de pilha

$$S \quad \to \quad \mathtt{a}S\mathtt{a} \;\mid\; \mathtt{b}S\mathtt{b}$$
$$S \quad \to \quad \mathtt{a} \;\mid\; \mathtt{b} \;\mid\; \mathtt{e}$$

A variável S pode ser representada no autômato pelo estado $\mathtt{q_0}$, e as duas primeiras regras correspondem às transições

$$\mathtt{q_0} \xrightarrow{\mathtt{a[e,a]}} \mathtt{q_0} \qquad \text{e} \qquad \mathtt{q_0} \xrightarrow{\mathtt{b[e,b]}} \mathtt{q_0}$$

As regras

$$S \quad \to \quad \mathtt{a} \;\mid\; \mathtt{b} \;\mid\; \mathtt{e}$$

indicam que a derivação chegou ao fim.

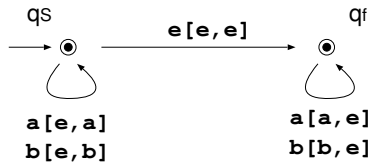
Mas, o autômato ainda precisa verificar que o resto da palavra corresponde ao conteúdo da pilha. Para fazer isso, nós introduzimos um novo estado $\mathtt{q_1}$ e as transições:

$$\mathtt{q_0} \xrightarrow{\mathtt{a,b,e}} \mathtt{q_1}$$

e

$$\mathtt{q_1} \xrightarrow{\mathtt{a[a,e]}} \mathtt{q_1} \qquad \text{e} \qquad \mathtt{q_1} \xrightarrow{\mathtt{b[b,e]}} \mathtt{q_1}$$

O resultado que nós obtemos é basicamente o autômato que nós vimos na aula 17:



◇

2.1 Regras com duas variáveis

O próximo passo é considerar regras da forma

$$A \quad \to \quad \mathtt{aa}\, B\, C\, \mathtt{ba}$$

Como antes, a ideia é que um autômato que estivesse no estado $\mathtt{q_A}$

- já poderia ler o fragmento `aa` na palavra de entrada, fazendo a transição para $\mathtt{q_B}$
- e colocaria o fragmento `ba` na pilha para compará-lo com o final da palavra de entrada

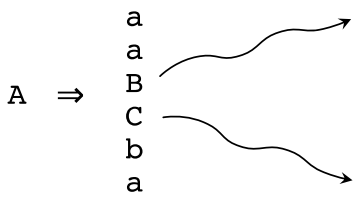
Mas, e a variável C?

Bom, pensando de novo em termos de gramáticas, nós observamos que uma derivação a partir de

$$A \quad \Rightarrow \quad \mathtt{aa}\, B\, C\, \mathtt{ba}$$

deve continuar aplicando regras de substituição sobre as variáveis B e C.

De fato, nós podemos pensar que nesse ponto a derivação continua em duas linhas paralelas de desenvolvimento



Como indicado no esquema, as duas linhas de desenvolvimento acabam produzindo as palavras `x` e `y`.

E o resultado final da derivação completa a partir de A é

$$A \quad \Rightarrow \quad \mathtt{aa}\, B\, C\, \mathtt{ba} \quad \Rightarrow \quad \ldots \quad \Rightarrow \quad \mathtt{aa\,x\,y\,ba}$$

Mas, como é que um autômato de pilha pode fazer esse tipo de coisa?

A resposta é fácil.

O autômato pode começar fazer a transição para o estado $\mathsf{q_B}$, para acompanhar a linha de derivação a partir de B

$$\mathsf{q_A} \xrightarrow{\mathtt{aa[e,ba]}} \mathsf{q_B}$$

Daí, quando essa derivação acaba, o autômato precisa acompanhar a linha de derivação a partir de C.

Mas, como é que o autômato sabe disso?

A resposta mais uma vez é fácil: basta que essa informação esteja na pilha.

Quer dizer, no momento em que o autômato faz a transição para o estado $\mathsf{q_B}$, ele deixa um lembrete na pilha

$$\mathsf{q_A} \xrightarrow{\mathtt{aa[e,C\ ba]}} \mathsf{q_B}$$

para recordar mais tarde (no estado $\mathsf{q_f}$) que ele ainda precisa realizar uma computação a partir de $\mathsf{q_C}$

$$\mathsf{q_f} \xrightarrow{\mathtt{aa[C,e]}} \mathsf{q_C}$$

Vejamos um exemplo concreto.

Exemplo 4: a gramática de DOB-a e o seu autômato de pilha

Na aula ??, nós apresentamos a seguinte gramática para a linguagem DOB-a

$$\begin{aligned} S &\to \mathtt{a}S\mathtt{a}S\mathtt{b} \mid \mathtt{a}S\mathtt{b}S\mathtt{a} \mid \mathtt{b}S\mathtt{a}S\mathtt{a} \\ S &\to S\,S \\ S &\to \mathtt{e} \end{aligned}$$

Como essa gramática só possui uma variável, nós vamos construir um autômato de pilha com apenas dois estados: $\mathsf{q_S}$ e q_f.

Começando com a regra

$$S \to \mathtt{a}S\mathtt{a}S\mathtt{b}$$

nós observamos que ela corresponde à seguinte transição

$$\mathsf{q_S} \xrightarrow{\mathtt{a[e,a\ S\ b]}} \mathsf{q_S}$$

As outras duas regras são análogas

$$S \to \mathtt{a}S\mathtt{b}S\mathtt{a} \quad \Rightarrow \quad \mathsf{q_S} \xrightarrow{\mathtt{a[e,b\ S\ a]}} \mathsf{q_S}$$

$$S \to \mathtt{b}S\mathtt{a}S\mathtt{a} \quad \Rightarrow \quad \mathsf{q_S} \xrightarrow{\mathtt{b[e,a\ S\ a]}} \mathsf{q_S}$$

A regra

$$S \to S\,S$$

corresponde simplesmente a

$$\mathsf{q_S} \xrightarrow{\mathtt{e[e,\,S]}} \mathsf{q_S}$$

Quer dizer, ela inicia uma computação a partir de $\mathsf{q_S}$ imediatamente, e deixa um lembrete na pilha para iniciar uma outra mais tarde.

Finalmente, a regra

$$S \quad \to \quad \mathtt{e}$$

corresponde a uma passagem para o estado $\mathsf{q_f}$

$$\mathsf{q_S} \xrightarrow{\mathtt{e}} \mathsf{q_f}$$

Mas, ainda faltam as transições do estado $\mathsf{q_f}$.

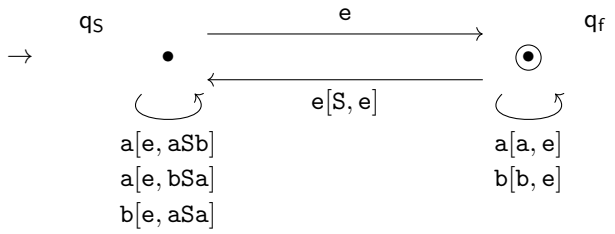
O primeiro tipo de transição são aquelas que comparam um símbolo da palavra de entrada com o símbolo no topo da pilha

$$\mathsf{q_f} \xrightarrow{\mathtt{a[a,e]}} \mathsf{q_f} \qquad\qquad \mathsf{q_f} \xrightarrow{\mathtt{b[b,e]}} \mathsf{q_f}$$

E a outra é aquela que trata os lembretes, para iniciar novas computações a partir de $\mathsf{q_S}$

$$\mathsf{q_f} \xrightarrow{\mathtt{e[S,e]}} \mathsf{q_S}$$

Abaixo nós temos o diagrama completo do autômato que reconhece a linguagem DOB-a



◇

3 Método sistemático

As observações que nós acabamos de fazer já são suficientes para obter um método que converte gramáticas livres de contexto em autômatos de pilha de maneira sistemática.

E o mais interessante é que esse autômato sempre tem 3 estados apenas.

Quer dizer, a ideia é fazer a divisão de tarefas de maneira ligeiramente diferente

- nós vamos ter um estado $\mathsf{q_f}$ que é responsável por ler símbolos na palavra de entrada (que estão indicados na pilha)
- e um outro estado $\mathsf{q_R}$ que trata os lembretes que forma deixados na pilha, aplicando as regras da gramática

O terceiro estado apenas dá início ao processo, colocando o símbolo da variável inicial S na pilha.

Vejamos um exemplo concreto.

Exemplo 5: a gramática de BAL e o seu autômato de pilha

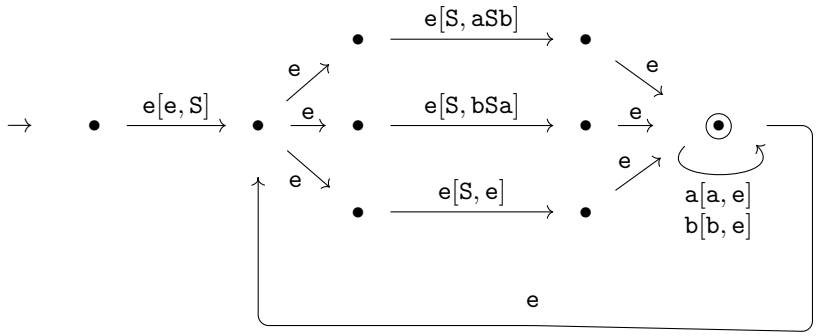
Abaixo nós temos a gramática de BAL

$$S \quad \rightarrow \quad \mathtt{a}S\mathtt{b} \;\mid\; \mathtt{b}S\mathtt{a} \;\mid\; SS \;\mid\; \mathtt{e}$$

Como indicado acima, a ideia é ter

- um estado $\mathsf{q_R}$ com uma transição para cada regra da gramática
- um estado $\mathsf{q_f}$ que lê símbolos da palavra de entrada e os compara com os símbolos da pilha
- e o estado inicial do autômato

Abaixo nós temos o diagrama do autômato de pilha que reconhece a linguagem BAL



◇

Abaixo nós temos mais um exemplo, que aplica o método sistemático sobre a gramática de EXPR.

Exemplo 6: a gramática de EXPR e o seu autômato de pilha

Abaixo nós temos a gramática de EXPR

$$\begin{array}{lcl} E & \rightarrow & (\, E \,+\, E \,) \;\mid\; (\, E \,*\, E \,) \;\mid\; N \\ N & \rightarrow & D\, N \;\mid\; D \\ D & \rightarrow & 0 \;\mid\; \ldots \;\mid\; 9 \end{array}$$

E abaixo nós temos o autômato de pilha que reconhece EXPR ◇

4 Gramáticas e autômatos recursivos

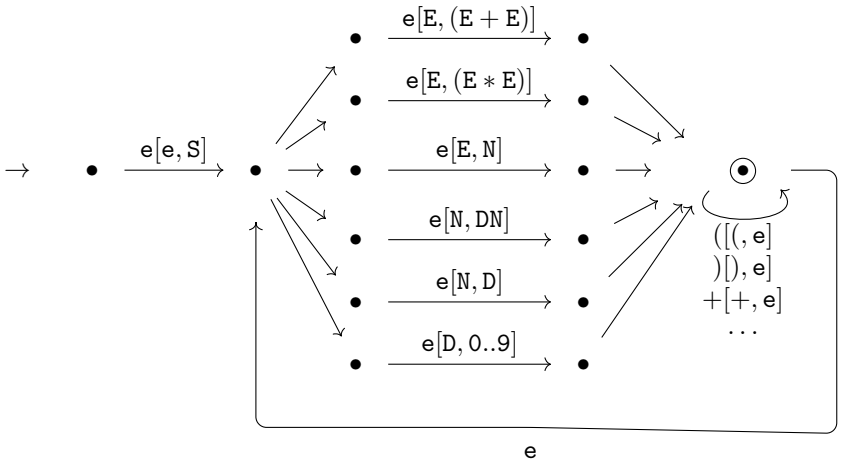
Na Introdução, nós chamamos atenção para o fato de que

- *Autômatos finitos e gramáticas regulares são essencialmente a mesma coisa*

Agora, nós vamos fazer uma observação semelhante

- *Autômatos recursivos e gramáticas livres de contexto são essencialmente a mesma coisa*

Quer dizer, para obter a gramática que gera a linguagem reconhecida por um autômato recursivo, basta desmontar o diagrama e apresentar as transições no formato de lista.



E para obter um autômato recursivo que reconhece a linguagem gerada por uma gramática livre de contexto, basta montar as suas regras na forma de um diagrama de estados e transições.

O segundo caso é bem fácil de ver, e ele já apareceu na aula passada.

Quer dizer, a ideia é

- → definir um módulo para cada variável da gramática
- → e um caminho no módulo para cada regra associada a essa variável

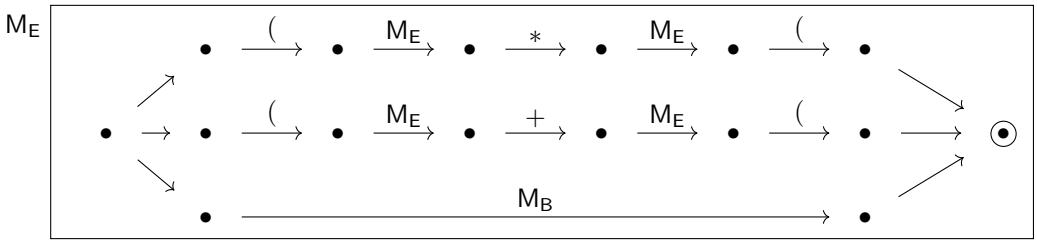
Vejamos como a coisa funciona na forma de um exemplo concreto.

Exemplo 7: a gramática de BOOL e o seu autômato recursivo

Abaixo nós temos a gramática de BOOL

$$E \rightarrow (\ E\ +\ E\ )\ \mid\ (\ E\ *\ E\ )\ \mid\ B$$
$$B \rightarrow 0\ \mid\ 1$$

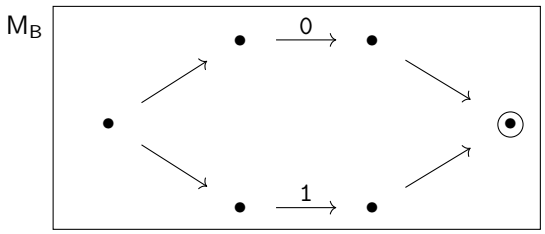
E abaixo nós temos o autômato recursivo que reconhece a linguagem BOOL



◇

E o primeiro caso também é bem fácil de ver.

Quer dizer, um autômato recursivo é basicamente um autômato finito que realiza chamadas recursivas.



Nós já sabemos como tratar as transições do autômato finito

$$q_i \xrightarrow{a} q_j \qquad \Rightarrow \qquad Q_i \rightarrow a Q_j$$

E as chamadas recursivas podem ser tratadas da seguinte maneira

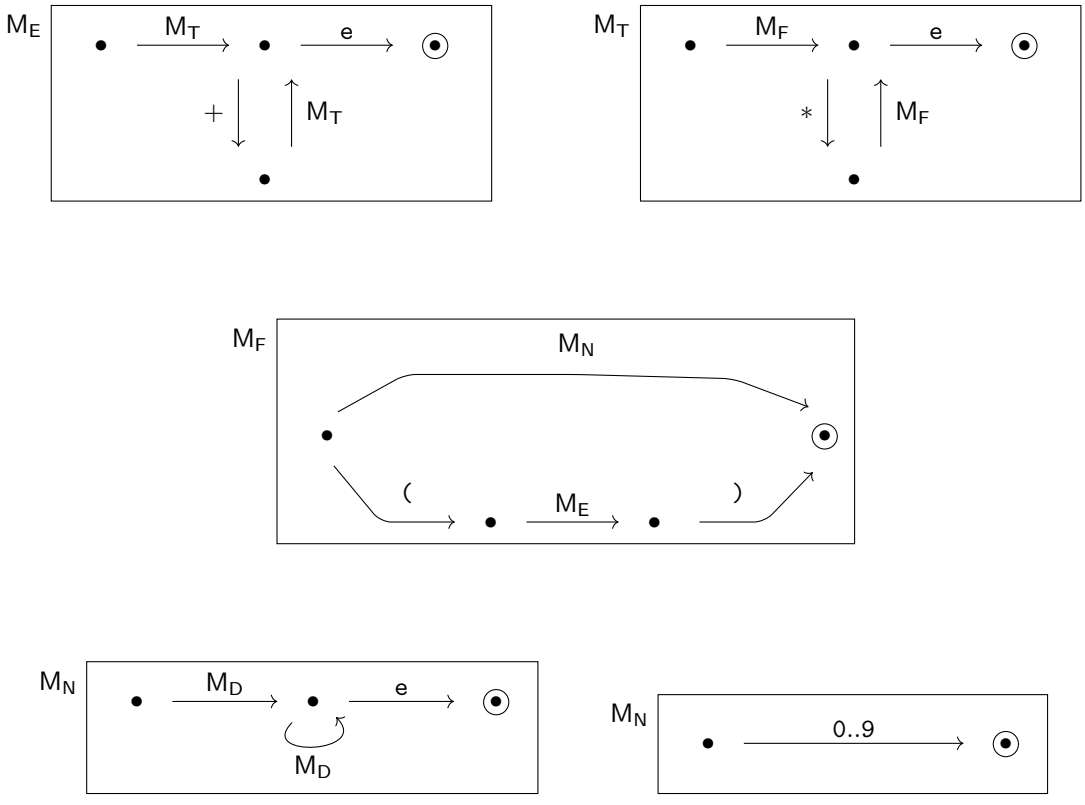
$$q_i \xrightarrow{M} q_j \qquad \Rightarrow \qquad Q_i \rightarrow M\, Q_j$$

Vejamos como a coisa funciona na forma de um exemplo concreto.

Exemplo 8: o autômato recursivo de EXPR-n e a sua gramática livre de contexto

Lembre que EXPR-n é o conjunto de expressões aritméticas (não necessariamente completamente parentizadas) envolvendo apenas as operações + e ∗.

Abaixo nós temos os módulos que definem um autômato recursivo que reconhece EXPR-n



◇