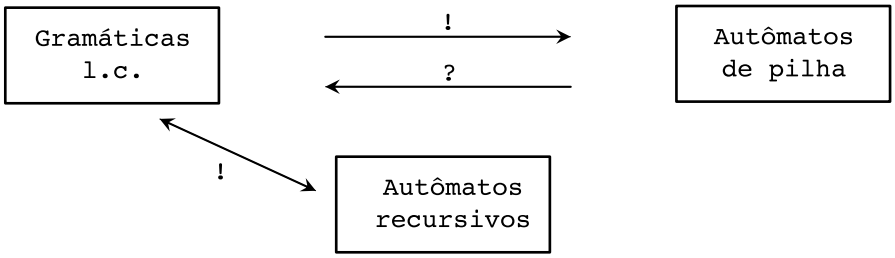
Teoria dos Autômatos e Linguagens Formais

aula 24: Autômatos de pilha e gramáticas

1 Introdução

O esquema abaixo mostra o que nós vimos na semana passada



Quer dizer, nós vimos um método sistemático para converter gramáticas livres de contexto em autômatos de pilha.

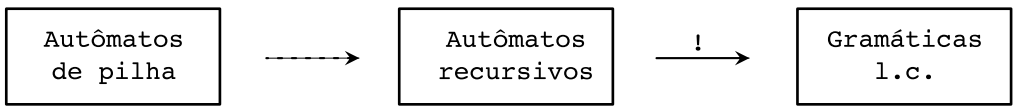
Nós vimos que as gramáticas livres de contexto e os autômatos recursivos são apenas representações diferentes da mesma ideia.

E nós deixamos em aberto a questão sobre a possibilidade de converter autômatos de pilha em gramáticas livres de contexto (de maneira sistemática).

Nessa aula, nós vamos resolver essa questão.

2 Chamadas de função no autômato de pilha

A estratégia que nós vamos utilizar é uma abordagem indireta



Quer dizer, nós vamos mostrar primeiro que todo autômato de pilha pode ser convertido em um autômato recursivo.

E daí, nós já sabemos que a conversão do autômato recursivo em uma gramática livre de contexto é uma coisa fácil.

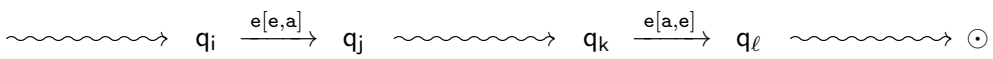
Certo.

A nossa tarefa, então, é visualizar chamadas de função e retornos dessas funções na execução de um autômato de pilha comum.

E a observação chave é a seguinte

→ *Todo símbolo colocado na pilha deve ser removido antes que a computação acabe*
— *(para que o autômato aceite a palavra de entrada)*

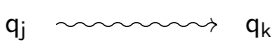
O esquema abaixo ilustra essa ideia



Quer dizer, aqui nós temos uma linha de execução do autômato que aceita a palavra de entrada.

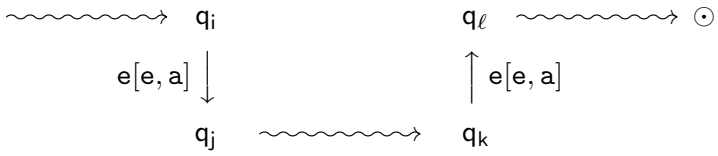
E nós estamos assumindo que o símbolo que é colocado na pilha pela transição $q_i \to q_j$ é o símbolo que é removido da pilha pela transição $q_k \to q_l$.

Mas, se isso é o caso, então todo símbolo que é colocado na pilha durante o trecho



é removido da pilha durante esse mesmo trecho — (*para que o nosso símbolo* a *esteja novamente no topo em* q_k)

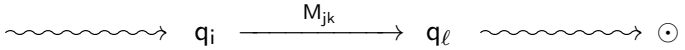
A ideia então é ver esse trecho como a execução de uma função, de modo que a computação passe a ser vista assim

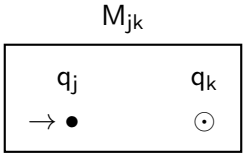


Quer dizer, a ideia é ver

- as transíções que colocam um símbolo na pilha como chamadas de função
- as transições que removem um símbolo da pilha como retornos de funções

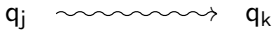
Ora, mas se nós vamos ver as coisas assim, então nós podemos representar a computação dessa maneira





Quer dizer, o módulo M_{jk} é uma cópia do autômato original onde q_j foi designado como o estado inicial, e q_k foi designado como o (único) estado final.

É fácil ver que o trecho de computação

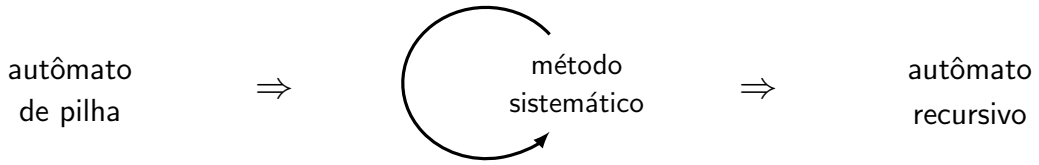


corresponde a uma execução completa do módulo M_{jk} — (*i.e., uma execução que termina em* q_k *com a pilha vazia*)

Logo, o trecho $q_j \rightsquigarrow q_k$ pode ser substituído por uma chamada ao módulo M_{jk}.

3 Método sistemático

E agora é só repetir



Mas antes, é preciso resolver dois pequenos problemas.

O primeiro deles é que em geral as transições fazem mais de uma coisa

$$q_i \xrightarrow{a[b,c]} q_j$$

Mas isso é fácil de resolver, decompondo a transição assim

$$q_i \xrightarrow{e[b,e]} \bullet \xrightarrow{a} \bullet \xrightarrow{e[e,c]} q_j$$

Quer dizer, primeiro acontece o retorno de alguma função, depois a leitura do símbolo na palavra de entrada, e depois a chamada de uma função — (*mas qualquer outra ordem também funciona — porque?*)

O caso de operações na pilha com múltiplos símbolos também não apresenta maiores problemas, pois nós sempre podemos decompor a transição

$$q_i \xrightarrow{a[bb,cc]} q_j$$

como um caminho

$$q_i \xrightarrow{e[b,e]} \bullet \xrightarrow{e[b,e]} \bullet \xrightarrow{a} \bullet \xrightarrow{e[e,c]} \bullet \xrightarrow{e[e,c]} q_j$$

O segundo problema é o seguinte.

Quando nós fizemos o raciocínio na seção anterior, nós estávamos trabalhando com uma *linha de execução* específica do autômato.

Mas, a conversão precisa funcionar para todas as execuções possíveis.

Em particular, o símbolo que é colocado na pilha pela transição

$$q_i \xrightarrow{e[e,a]} q_j$$

pode ser removido da pilha por transições diferentes em execuções diferentes do autômato

$$q_k \xrightarrow{e[a,e]} q_l \qquad q_m \xrightarrow{e[a,e]} q_n \qquad q_u \xrightarrow{e[a,e]} q_v$$

Quer dizer, para interpretar a transição

$$q_i \xrightarrow{e[e,a]} q_j$$

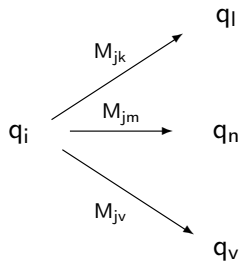
como uma chamada de função, é preciso saber já no momento da chamada qual é o lugar de retorno (ou o estado que vai remover o símbolo da pilha), para saber qual é o módulo que está sendo chamado.

No exemplo acima, nós teríamos 3 possibilidades

$$q_i \xrightarrow{M_{jk}} q_l \qquad q_i \xrightarrow{M_{jm}} q_n \qquad q_i \xrightarrow{M_{ju}} q_v$$

Mas, nós podemos deixar essa questão ser resolvida pelo não-determinismo (em tempo de execução).

Quer dizer, nós podemos substituir as transições que manipulam a pilha pela coleção de chamadas de função possíveis



Pronto.

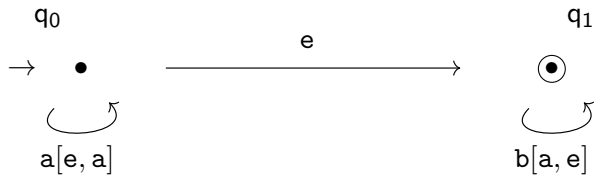
Esses eram os problemas que precisavam ser resolvidos.

E agora nós podemos considerar alguns exemplos concretos.

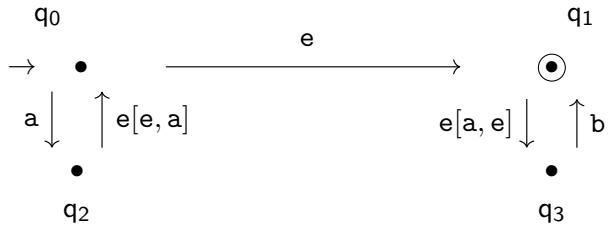
Exemplos

a. A linguagem $a^n b^n$

Abaixo nós temos o autômato de pilha que reconhece a linguagem $a^n b^n$



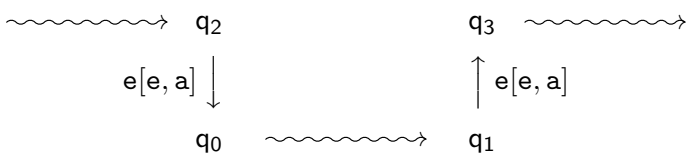
Para fazer a conversão para um autômato recursivo, o primeiro passo é decompor as transições que realizam múltiplas operações em múltiplas transições que realizam uma única operação



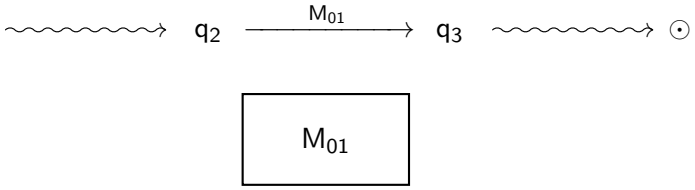
Daí, nós observamos que só existe uma transição que coloca o símbolo a na pilha, e uma transição

que remove o símbolo a da pilha.

Logo, a chamada que é feita pela primeira é retornada pela segunda



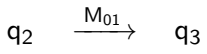
Ou, na notação dos autômatos recursivos



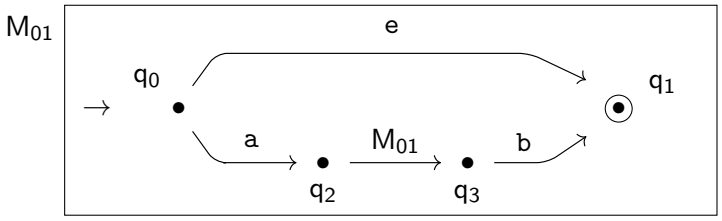
Para implementar essa mudança no diagrama dos autômatos, nós

- removemos as transições que manipulam a pilha

- e adicionamos a transição que realiza a chamada de função



Reorganizando o diagrama, nós obtemos o seguinte resultado



Em princípio, nós também precisaríamos adicionar o módulo M_{01} ao autômato.

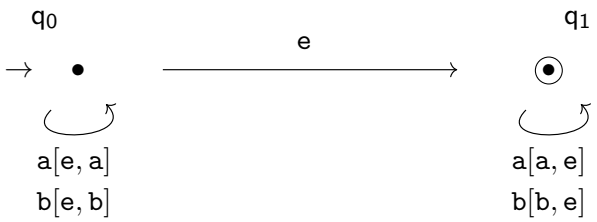
Mas, dessa vez, esse é o próprio módulo acima.

Quer dizer, na prática, o diagrama está realizando uma chamada recursiva a si mesmo.

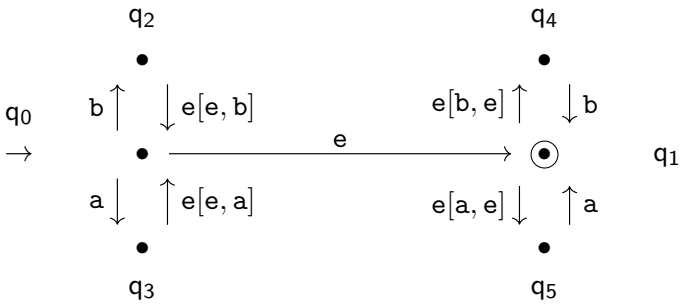
◇

b. A linguagem PAL

Abaixo nós temos o autômato de pilha que reconhece a linguagem PAL



Para fazer a conversão, nós separamos e decompomos as transições umas das outras



A seguir, nós identificamos os pares de transições que colocam e removem o mesmo símbolo da pilha

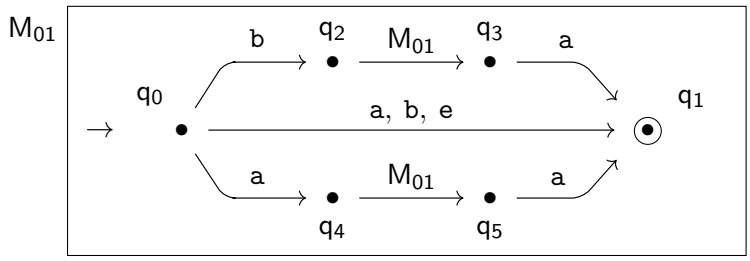
$$q_2 \xrightarrow{e[e,a]} q_0 \qquad q_1 \xrightarrow{e[a,e]} q_4$$

$$q_3 \xrightarrow{e[e,b]} q_0 \qquad q_1 \xrightarrow{e[b,e]} q_5$$

A seguir, nós substituímos esses pares de transição por transições que realizam uma chamada de função

$$q_2 \xrightarrow{M_{01}} q_4 \qquad q_3 \xrightarrow{M_{01}} q_5$$

E reorganizamos o diagrama para obter

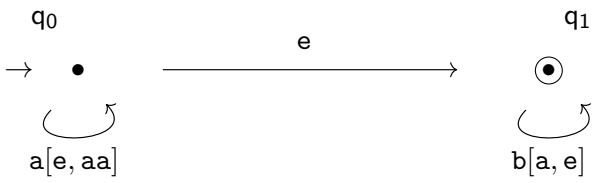


Mais uma vez, as chamadas de função que aparecem são chamadas ao módulo principal, e por isso não é necessário adicionar novos módulos ao autômato.

◇

c. A linguagem $a^n b^{2n}$

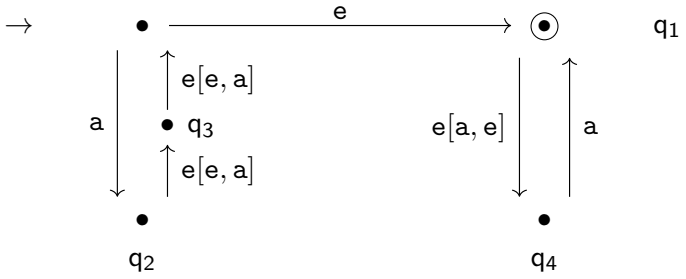
Esse exemplo fica mais interessante se nós consideramos o seguinte autômato de pilha que reconhece a linguagem $a^n b^{2n}$



Quer dizer, nesse exemplo nós temos uma transição que coloca dois símbolos na pilha, o que vai corresponder a duas chamadas de função consecutivas.

Vejamos como a coisa funciona.

Primeiro nós decompomos as transições



Depois, nós identificamos os pares de transições que colocam e tiram o mesmo símbolo da pilha

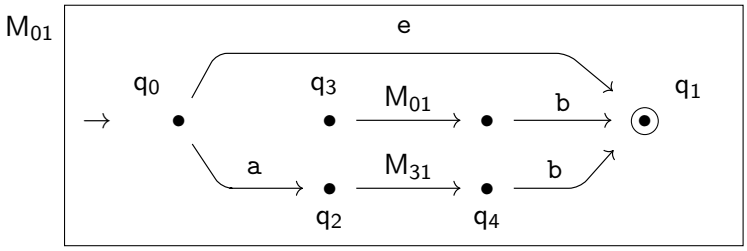
$$q_2 \xrightarrow{e[e,a]} q_3 \qquad q_1 \xrightarrow{e[a,e]} q_4$$

$$q_3 \xrightarrow{e[e,a]} q_0 \qquad q_1 \xrightarrow{e[a,e]} q_4$$

E daí nós substituímos esses pares de transição por transições que fazem uma chamada de função

$$q_2 \xrightarrow{M_{31}} q_4 \qquad q_3 \xrightarrow{M_{01}} q_4$$

Reorganizando o diagrama, nós obtemos

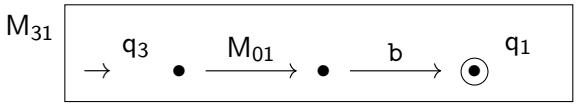


E aqui aparecem duas novidades.

A primeira é que o caminho intermediário não pode ser alcançado a partir do estado inicial q_0, e pode ser removido do diagrama.

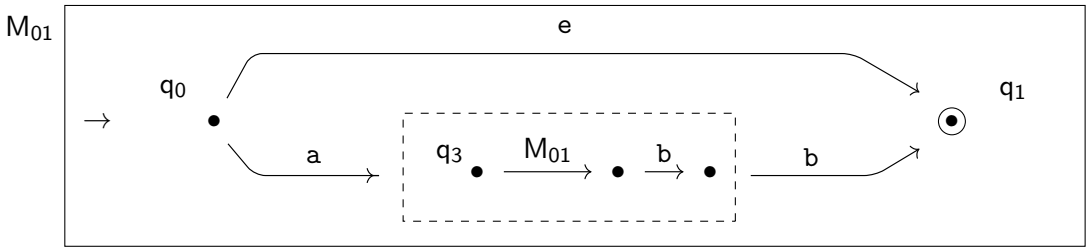
Mas, esse caminho vai fazer parte do módulo M_31 que é chamado pelo caminho de baixo.

E essa é a segunda novidade: dessa vez nós precisamos adicionar um novo módulo à especificação do autômato recursivo



Esse autômato recursivo ficou um pouco diferente daquele que nós construímos na aula ??.

Mas, substituindo a chamada ao módulo $\mathsf{M_{31}}$ por uma cópia do seu diagrama, nós voltamosa obter uma construção familiar

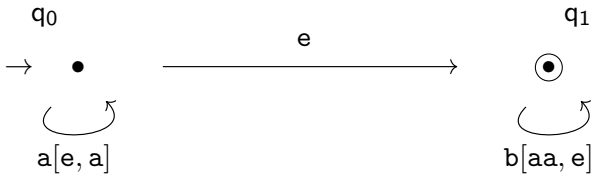


O que comprova que a nossa construção inicial estava correta.

◇

d. $\mathtt{a}^{2n}\mathtt{b}^{n}$

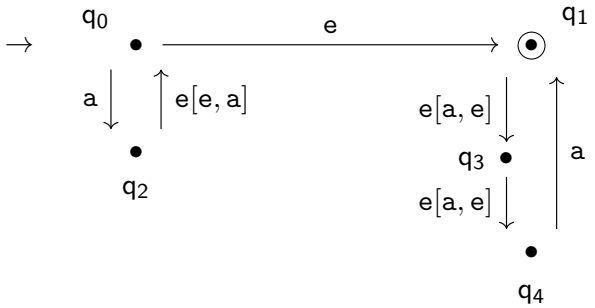
Esse exemplo fica mais interessante se nós consideramos o seguinte autômato de pilha



Quer dizer, nesse exemplo nós temos uma transição que remove dois símbolos da pilha, o que vai corresponder a dois retornos de função consecutivos.

Vejamos como a coisa funciona.

Primeiro nós decompomos as transições



Depois nós identificamos os pares de transições que colocam e removem o mesmo símbolo da pilha

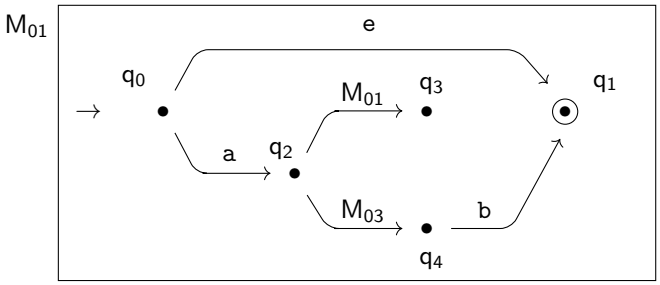
$$\mathsf{q_2} \xrightarrow{\mathtt{e[e,a]}} \mathsf{q_0} \qquad\qquad \mathsf{q_1} \xrightarrow{\mathtt{e[a,e]}} \mathsf{q_3}$$

$$\mathsf{q_2} \xrightarrow{\mathtt{e[e,a]}} \mathsf{q_0} \qquad\qquad \mathsf{q_3} \xrightarrow{\mathtt{e[a,e]}} \mathsf{q_4}$$

E daí nós substituímos esses pares de transição por transições que realizam uma chamada de função

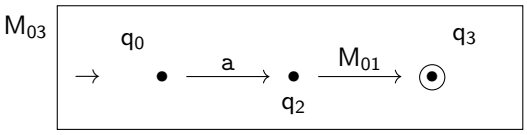
$$q_2 \xrightarrow{M_{01}} q_3 \qquad\qquad q_2 \xrightarrow{M_{03}} q_4$$

Reorganizando o diagrama, nós obtemos



Como no exemplo anterior, aqui apareceu uma chamada a um módulo que não é o módulo principal: M_{03}.

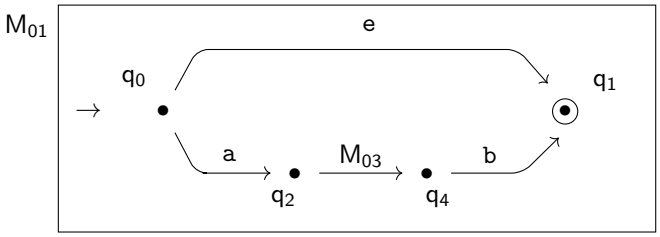
Logo, nós devemos adicionar esse módulo à especificação do autômato recursivo



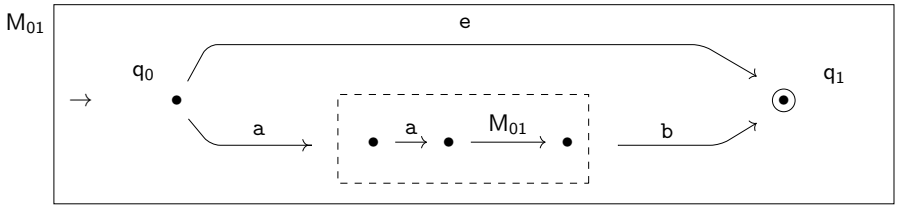
Esse módulo é uma cópia do módulo principal (M_{01}), onde o estado q_3 foi designado como o estado final.

Mas, note que todos os estado que não podem alcançar o estado final q_3 foram removidos (para simplificar a construção).

Da mesma forma, nós podemos remover a transição $q_2 \rightarrow q_3$ do módulo principal, uma que que a partir de q_3 não se vai a lugar nenhum



Agora, para nos certificarmos de que a construção está correta, bas substituir a chamada ao módulo M_{03} por uma cópia do seu diagrama



◇