Teoria dos Autômatos e Linguagens Formais

aula 11: Raciocinando sobre linguagens

1 Introdução

Na aula de hoje nós vamos apresentar um ponto de vista diferente: a ideia um pouco mais abstrata de que o conjunto de palavras aceitas por um autômato é uma *linguagem*.

Quer dizer, esse ponto de vista não é realmente abstrato ...

Todos nós temos um autômato dentro da nossa cabeça.

Que reconhece as palavras do português.

Mas o português não tem um padrão repetitivo — (*ou será que tem?*)

Porque existe só um número finito de palavras no português — (*ou será que não?*)

Certo.

A ideia é que as linguagens dos autômatos são conjuntos — (*que possuem um padrão repetitivo, e por isso são infinitas ...*)

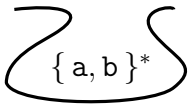
Daí, raciocinando com conjuntos nós podemos definir conjuntos mais complicados — (*por meio das operações de união, interseção, complemento, etc*)

E daí, a ideia é construir autômatos mais complicados que reconhecem essas linguagens complicadas.

2 Raciocinando sobre linguagens

Imagine que nós colocamos todas as palavras que podem ser formadas com as letras `a` e `b` dentro de um saquinho.

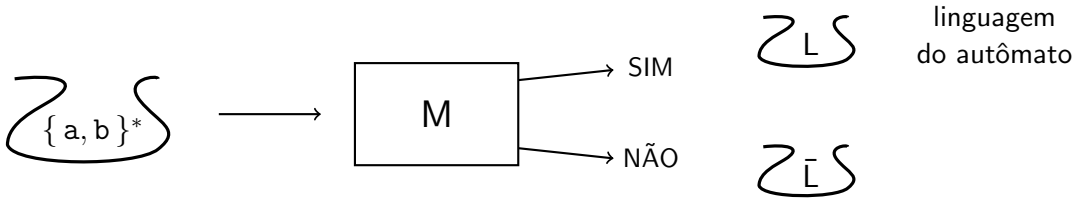
Isso é uma quantidade infinita de palavras, mas não importa



Daí, imagine que nós fornecemos todas essas palavras como entrada para o autômato.

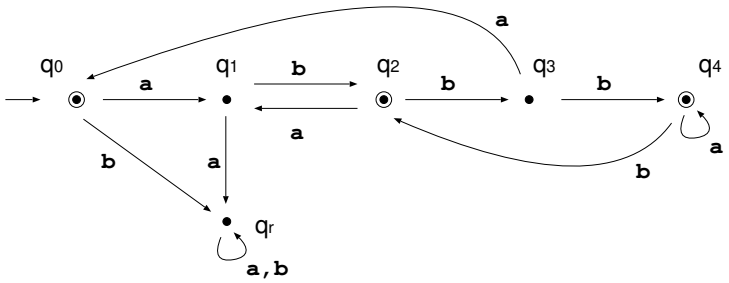
E separamos do outro lado aquelas que dão a resposta SIM daquelas que dão a resposta NÃO.

A linguagem do autômato é o subconjunto de palavras que dão a resposta SIM



Mas, vejamos um exemplo concreto.

Considere o autômato abaixo



Esse autômato é um tanto complicado.

Mas nesse momento não interessa saber exatamente o que ele faz.

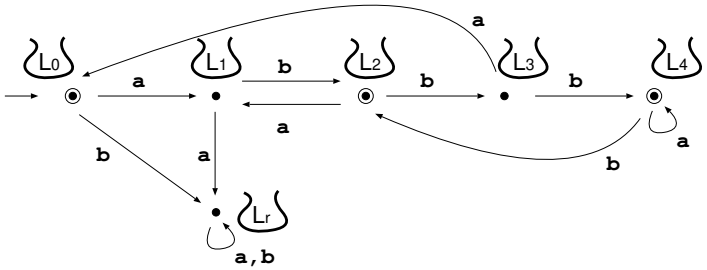
Só o que importa é que ele aceita qualquer palavra que faça a computação terminar em q_0, q_2 ou q_4 — (*i.e., os estados finais*)

Essa observação já é suficiente para concluir que a linguagem do autômato pode ser decomposta assim

$$L \quad = \quad L_0 \ \cup \ L_2 \ \cup \ L_4$$

onde L_0, L_2 e L_4 são os subconjuntos de palavras que levam o autômato do estado inicial q_0 para os estados q_0, q_2 e q_4, respectivamente.

Quer dizer, a ideia aqui é que cada estado q_j do autômato está associado a uma pequena linguagem L_j

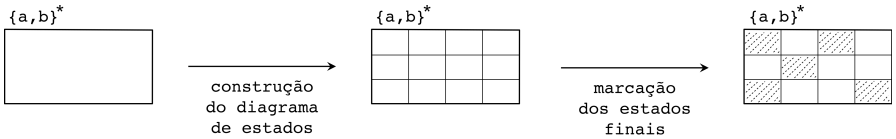


Daí que, na prática, ao marcar os estados finais do autômato, nós estamos selecionando algumas dessas pequenas linguagens para formar a linguagem L do autômato.

Em outras palavras, a construção de um autômato pode ser vista como um processo de duas etapas

1. a construção do diagrama, que particiona o universo de todas as palavras $\{a, b\}^*$ em uma coleção de pequenas linguagens
2. a marcação de estados finais, que define a linguagem reconhecida pelo autômato

A figura abaixo ilustra essa ideia.

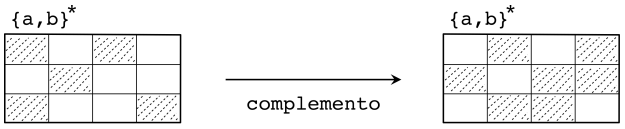


Essa observação já dá o primeiro resultado da aula de hoje

Corolário: Suponha que o autômato M reconhece a linguagem L.

Então, para obter um autômato que reconhece o complemento $\bar{\mathsf{L}}$, basta inverter os estados finais e não finais de M.

Prova:



□

A gente já sabia disso.

Mas será que existe alguma coisa parecida para as expressões regulares ?

A resposta é: *Não que eu saiba ...*

Quer dizer, imagine que nós queremos descobrir o complemento de uma expressão regular.

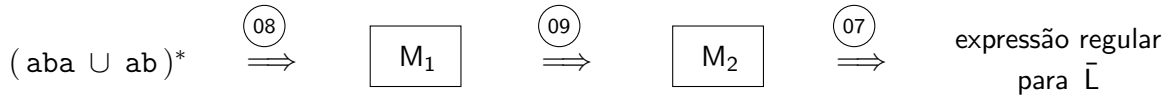
Por exemplo,

$$L \;=\; (\mathtt{aba} \cup \mathtt{ab})^* \qquad\qquad \Rightarrow \qquad\qquad \bar{L} \;=\; ?$$

O ponto é que não existe nenhuma regra simples que produz a expressão regular de $\bar{\mathsf{L}}$ a partir da expressão regular de L — (*até onde eu saiba ...*)

Mas isso não significa que essa expressão não existe.

Quer dizer, aplicando os métodos sistemáticos das aulas 07, 08 e 09, nós podemos fazer o seguinte



Fazendo isso, nós obtemos a seguinte expressão regular

$$\mathtt{a}\big(\mathtt{b}(\mathtt{ab})^*\mathtt{aa}\big)^*\Big(\mathtt{e} \cup \big(\mathtt{a} \cup \mathtt{b}(\mathtt{ab})^*\mathtt{b}\big)(\mathtt{a}\cup\mathtt{b})^*\Big) \;\cup\; \mathtt{b}(\mathtt{a}\cup\mathtt{b})^*$$

Legal, não é?

2.1 A operação de união

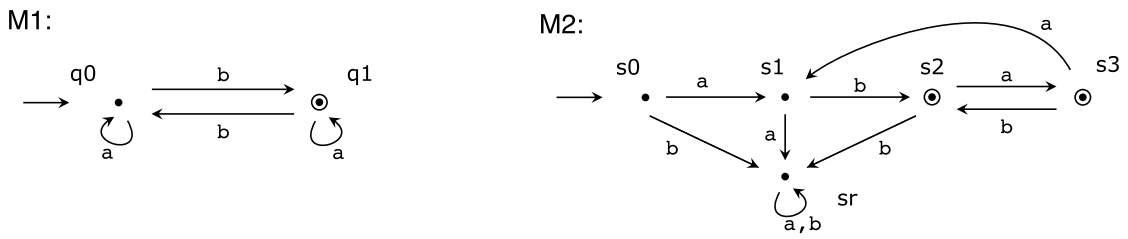
As coisas são ao contrário quando a gente considera a operação de união.

Por exemplo, considere as linguagens

L_1 = palavras com um número ímpar de b's

L_2 = concatenações não-vazias de blocos ab e aba

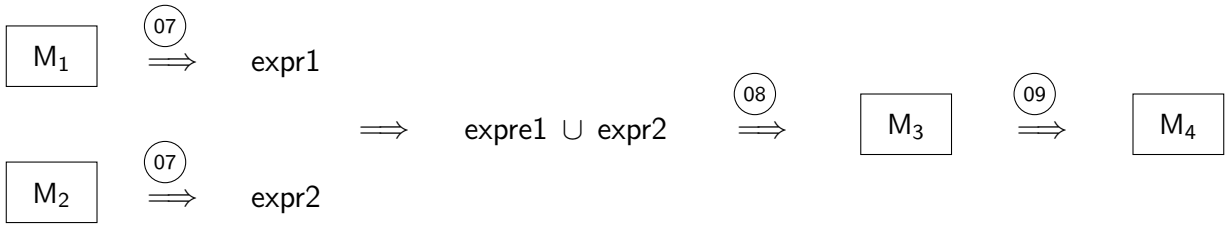
E considere dois autômatos M_1 e M_2 que reconhecem essas linguagens



A pergunta é: *Será que existe alguma maneira simples de obter um autômato (determinístico) que reconhece a linguagem* $\mathsf{L}_1 \cup \mathsf{L}_2$ *?* — (i.e., diretamente a partir de M_1 e M_2)

E a resposta é: *Não que eu saiba ...*

Mas se a gente quiser esse autômato, nós podemos fazer o seguinte



2.2 A operação de interseção

Agora, as coisas ficam mais interessantes quando a gente considera a operação de interseção.

Porque daí,

→ *Não existe uma regra simples que produz a interseção de dois autômatos*
e não existe uma regra simples que produz a interseção de duas expressões regulares

Mas daí, a gente pode lembrar que

$$\mathsf{L}_1 \ \cap \ \mathsf{L}_2 \quad = \quad \overline{\left(\overline{\mathsf{L}}_1 \ \cup \ \overline{\mathsf{L}}_2 \right)}$$

Daí que, a gente sempre pode obter uma expressão regular para a interseção de duas expressões regulares.

3 O autômato produto

Mas, essa não é a única maneira de fazer as coisas.

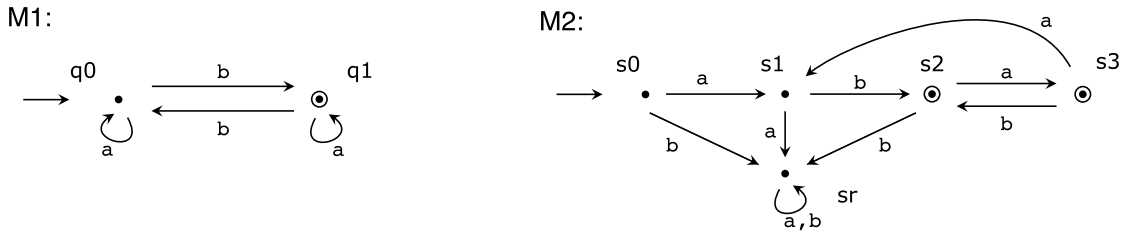
Quer dizer, a gente também pode raciocinar diretamente com autômatos.

Considere mais uma vez as linguagens

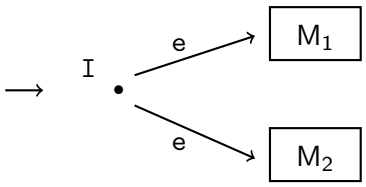
L_1 = palavras com um número ímpar de b's

L_2 = concatenações não-vazias de blocos ab e aba

E considere os autômatos M_1 e M_2 que reconhecem essas linguagens



E agora, imagine que nós construímos o seguinte autômato



Mas, como é que esse negócio funciona ?

Bom, no começo da computação, o autômato passa do estado inicial $\mathtt{I}$ para (q_0, s_0).

Daí, a gente lembra que M_1 e M_2 são autômatos determinísticos.

Isso significa que, a partir de agora, nós sempre vamos estar em um estado de M_1 e um estado de M_2.

Por exemplo,

$$\mathtt{I} \xrightarrow{e} (q_0, s_0) \xrightarrow{a} (q_0, s_1) \xrightarrow{b} (q_1, s_2) \xrightarrow{a} (q_1, s_3) \longrightarrow \cdots$$

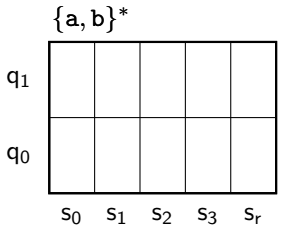
Quer dizer, o que nós temos aqui é uma máquina que executa M_1 e M_2 em paralelo.

Mas, o que é que essa máquina faz ?

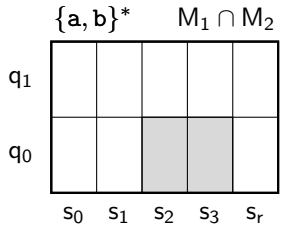
Bom, aqui nós podemos raciocinar em termos de linguagens.

Quer dizer, os diagramas de M_1 e M_2 particionam o conjunto de todas as palavras assim

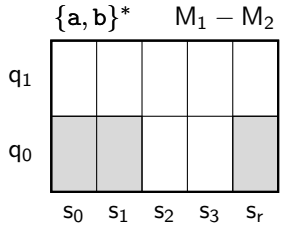
Daí que, quando nós executamos M_1 e M_2 em paralelo, nós obtemos a seguinte partição



Finalmente, se a gente quer o autômato que reconhece $\mathsf{L}_1 \cap \mathsf{L}_2$, basta marcar os pares que contém estados finais de M_1 e M_2 como finais



Por outro lado, se a gente quiser um autômato que reconhece a linguagem $\mathsf{L}_1 - \mathsf{L}_2$, basta marcar os estados finais assim



Legal, não é?

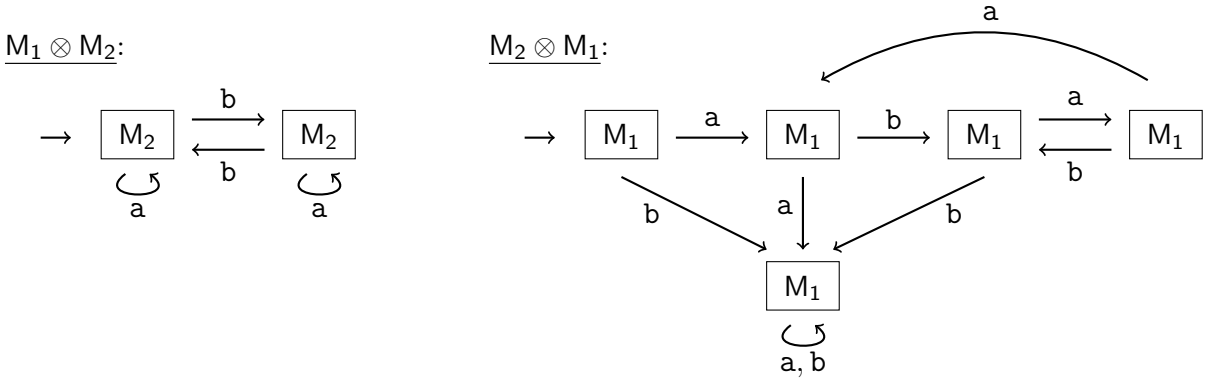
Sim, mas tem uma coisa ainda mais legal.

Quer dizer, imagine que nós queremos um autômato determinístico que reconhece a linguagem $\mathsf{L}_1 \cap \mathsf{L}_2$.

Daí, uma ideia seria aplicar o método sistemático da aula 09.

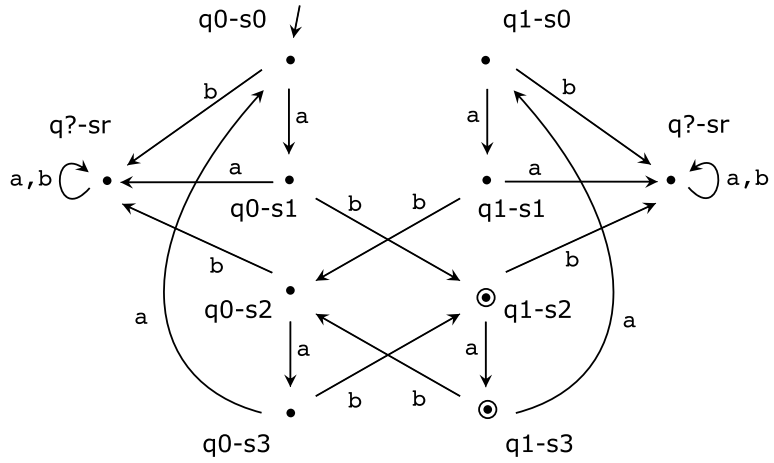
Mas, nós também podemos construir o *autômato produto* de M_1 e M_2.

Quer dizer, existem duas maneiras de fazer isso



A ideia é que nós substituímos os estados de um autômato por cópias do outro autômato.

E agora, nós podemos expadir o primeiro deles assim



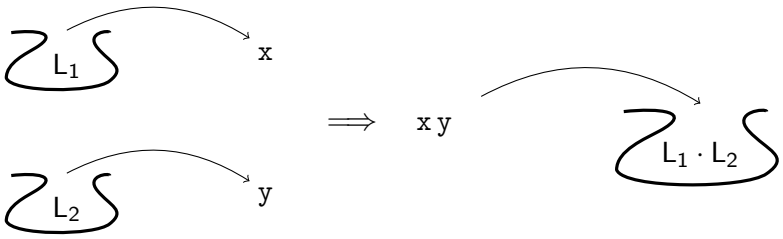
4 Programando com autômatos

Agora imagine que L_1 e L_2 são duas linguagens quaisquer.

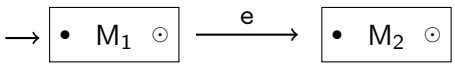
Imagine que M_1 e M_2 são dois autômatos que reconhecem essas linguagens.

E imagine que nós queremos um autômato que reconhece a linguagem $\mathsf{L}_1 \cdot \mathsf{L}_2$ — (*i.e., a concatenação de* L_1 *e* L_2)

Quer dizer, essa linguagem contém as palavras que podem ser formadas pela concatenação de uma palavra de L_1 com uma palavra de L_2



Mas, é fácil ver que o seguinte autômato reconhece a linguagem $\mathsf{L}_1 \cdot \mathsf{L}_2$

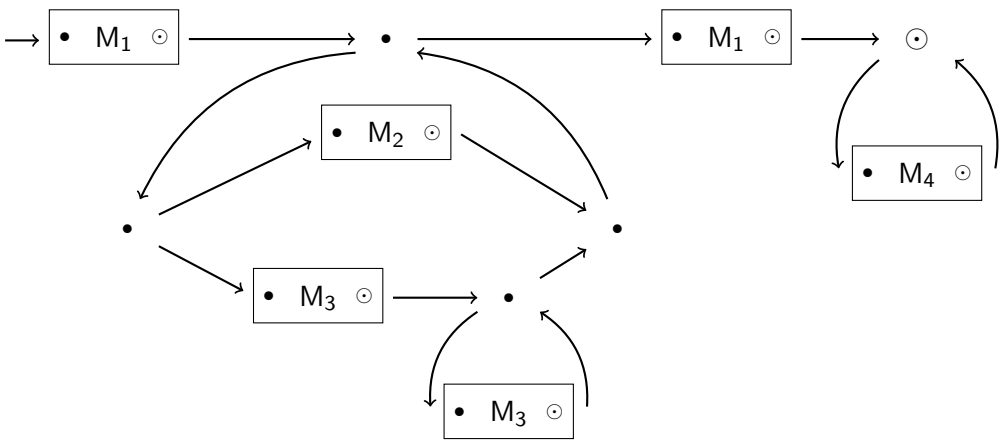


Quer dizer, sempre que a computação se encontra em um estado final de M_1, a máquina pode advinhar que a primeira parte da palavra já chegou ao fim, e que está na hora de ler a parte que pertence a L_2.

Quando a gente vê isso, não é difícil pensar que agora nós podemos escrever linguagens complicadas como

$$\mathsf{L}_1 \left(\mathsf{L}_2 \cup \mathsf{L}_3^* \right)^* \mathsf{L}_1 \mathsf{L}_4^*$$

E daí, nós podemos construir um autômato complicado que reconhece essa linguagem, da seguinte maneira

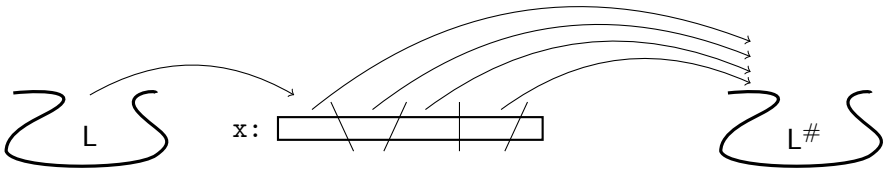


4.1 Pedaços, prefixos, sufixos e mais

Agora imagine que L é a linguagem reconhecida pelo autômato M.

Imagine que nós pegamos as palavras de L e quebramos em pedaços de todas as maneiras possíveis.

E imagine que nós formamos uma nova linguagem $L^{\#}$ com todos esses pedaços



A questão é

- *Será que nós conseguimos construir um autômato que reconhece a linguagem* $L^{\#}$ *?*

Hmm ...

A intuição parece ser a seguinte

- $L^{\#}$ foi construída com os pedaços das palavras de L

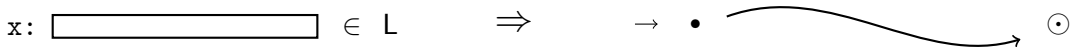
  logo, $M^{\#}$ deve ser construído com os pedaços de M

Hmm ...

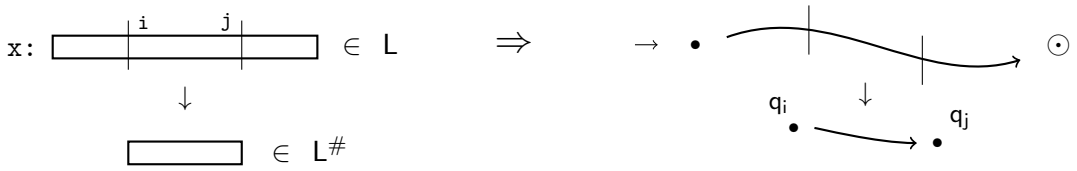
Mas ainda não é isso.

Quer dizer, a ideia é a seguinte: *os caminhos do autômato* $M^{\#}$ *devem ser construídos com pedaços dos caminhos de* M.

O que nós temos em mente aqui é que toda palavra de L corresponde a um caminho de aceitação no autômato M



Logo, um pedaço dessa palavra corresponde a ume pedaço do caminho



O que nós queremos é que esse pedaço de caminho seja um caminho de aceitação em $M^{\#}$ — (*porque o pedaço da palavra está em* $L^{\#}$)

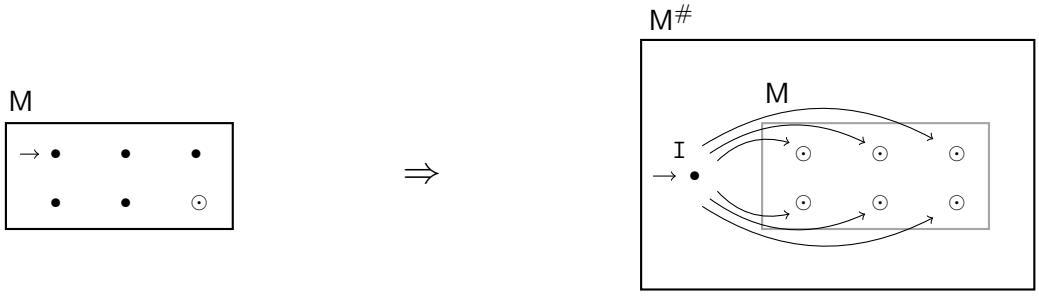
Bom, para isso basta fazer com que q_i seja o estado inicial e que q_j seja um estado final de $M^{\#}$.

Mas, q_i e q_j são dois estados quaisquer no meio do caminho.

Logo, o que nós precisamos é que

→ *todos os estados de* M *sejam estados iniciais de* $M^{\#}$
e todos os estados de M *sejam estados finais de* $M^{\#}$

Quer dizer,



Na verdade, não são realmente todos os estados de M que viram estados iniciais e finais de $M^{\#}$.

Quer dizer, são apenas os estados que aparecem em algum caminho de aceitação de M.

Por exemplo, esse não é o caso do estado de rejeição.

Vejamos outros exemplos.

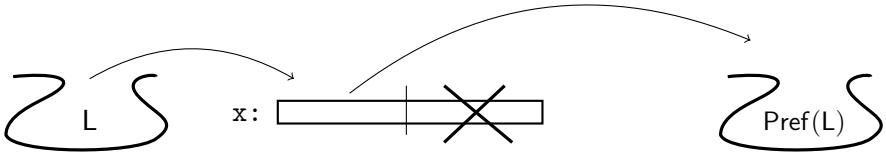
Exemplos

a. Prefixos e sufixos

Imagine outra vez que L é a linguagem reconhecida pelo autômato M.

Imagine que nós pegamos as palavras de L e quebramos em um ponto qualquer.

E imagine que nós formamos a linguagem Pref(L) com a primeira parte de todas esses cortes



Quer dizer, $\mathsf{Pref}(\mathsf{L})$ é a linguagem formada pelos prefixos das palavras da linguagem L.

E nós podemos definir a linguagem dos sufixos $\mathsf{Suf}(\mathsf{L})$ de maneira análoga.

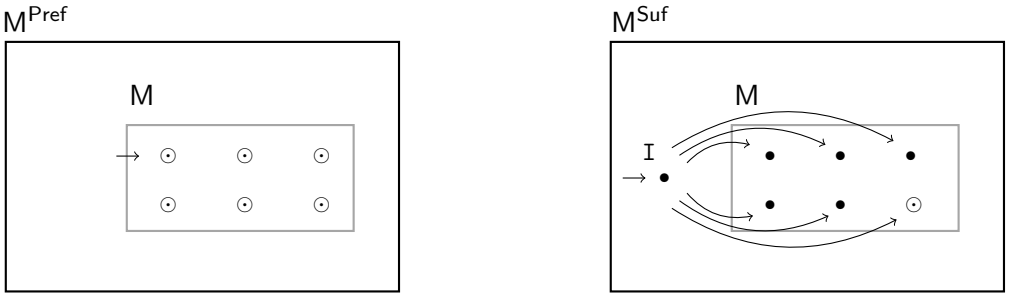
O problema consiste em

→ *Construir autômatos que reconhecem as linguagens* $\mathsf{Pref}(\mathsf{L})$ *e* $\mathsf{Suf}(\mathsf{L})$

Mas, agora isso já não é difícil.

Quer dizer, basta usar os truques que a gente acabou de ver — (*um para cada autômato*)

A coisa fica assim



◇

b. Pref-Suf

Agora imagine que L_1, L_2 são as linguagens reconhecidas pelos autômatos M_1, M_2.

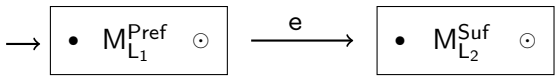
E imagine que nós queremos construir um autômato para a seguinte linguagem

$$\mathsf{Pref}(\mathsf{L}_1) \cdot \mathsf{Suf}(\mathsf{L}_2)$$

Quer dizer, essas são as palavras formadas pela concatenação do prefixo de uma palavra de L_1 com o sufixo de uma palavra de L_2.

Ora, mas agora isso é bem fácil.

Quer dizer, basta fazer o seguinte

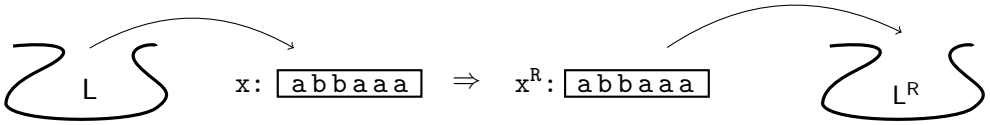


◇

c. Reverso

Imagine mais uma vez que L é a linguagem reconhecida pelo autômato M.

E imagine que nós formamos a linguagem L^{R} virando as palavras de L ao contrário



O problema consiste em

→ *Construir um autômato que reconhece a linguagem* L^{R}

E aqui é a hora para uma pequena esperteza.

Quer dizer, a ideia é que basta virar uma expressão regular ao contrário, para obter a expressão regular do reverso da linguagem original.

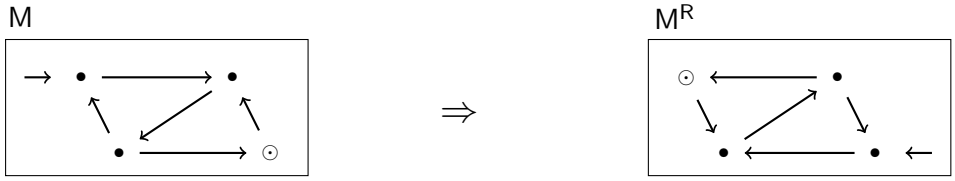
Por exemplo,

$$\mathsf{L} \quad = \quad \mathtt{a}\,(\mathtt{ab} \cup \mathtt{bab})^*\mathtt{ab}^* \qquad \Rightarrow \qquad \mathsf{L}^{\mathsf{R}} \quad = \quad \mathtt{b}^*\mathtt{a}\,(\mathtt{bab} \cup \mathtt{ba})^*\mathtt{a}$$

Mas, nós também podemos fazer o mesmo truque com os autômatos.

Quer dizer, para obter o autômato M^{R}

- nós invertemos a direção das transições
- transformamos o estado inicial em estado final
- e transformamos os estados finais em estados iniciais



Legal, não é?

$\diamond$