

# Capítulo 12: Autômatos

O nosso modelo de referência para computação mecânica é a máquina de Turing. Uma máquina Turing tem apenas dois componentes, uma CPU e uma fita. Vamos agora tirar a fita e estudar a CPU sozinha.

Dito de outra maneira, enquanto uma máquina de Turing tem memória ilimitada, os dispositivos que usamos todos os dias não têm. Podemos perguntar que tarefas podem ser feitas por uma máquina com memória limitada.

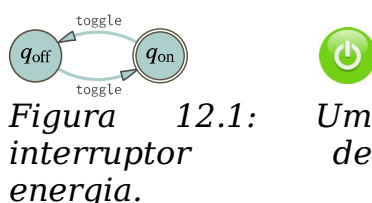
## 12.1. Máquinas de estados finitos

Produzimos um novo modelo de computação, modificando a definição da Máquina de Turing. Eliminaremos a capacidade de escrever, alterando a cabeça da fita de leitura/escrita para somente leitura. Isto dar-nos-á uma visão do que pode ser feito apenas com estados. Descobriremos que este tipo de máquina pode fazer muitas coisas, mas não tantas como uma máquina Turing.

No nosso novo modelo, utilizaremos o mesmo tipo de tabelas de transição e grafos de transição que fizemos com as máquinas de Turing.

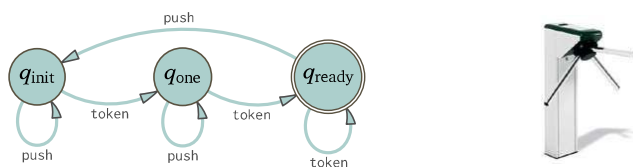
### Exemplo 12.1.1

Um interruptor de energia tem dois estados,  $q_{\text{off}}$  e  $q_{\text{on}}$  e o seu alfabeto de entrada tem um símbolo, toggle. (Figura 12.1)



### Exemplo 12.1.2

Opere esta catraca, colocando duas fichas (token) e depois passando (push) por ela. Ela possui três estados e o seu alfabeto de entrada é  $\Sigma = \{ \text{token}, \text{push} \}$ .



Como vimos com as máquinas de Turing, os estados são uma forma limitada de memória. Por exemplo,  $q_{\text{one}}$  é, no exemplo acima, como a catraca “se lembra” de ter recebido até agora uma ficha.

### Exemplo 12.1.3

Esta máquina de venda automática dispensa artigos que custam 30 centavos. Ela aceita apenas moedas de 5, 10 e 25 centavos. A figura é complexa, por isso vamos mostrá-la em três camadas. A primeira é das setas para as moedas de 5 centavos (que chamamos de  $n$ , de *nickel* em inglês) e o apertar do botão de distribuição.

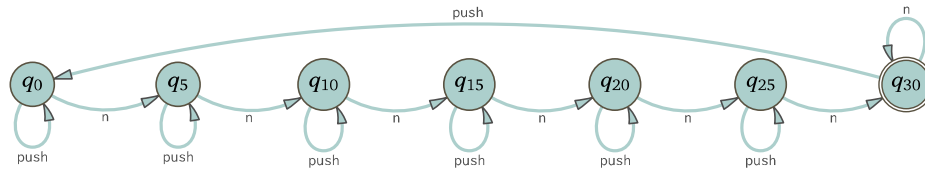


Figura 12.3: Máquina de vendas. Camada para as moedas de 5 centavos.

Após receber 30 centavos e receber outra moeda de 5 centavos, esta máquina faz algo não muito sensato: fica em  $q_{30}$ . Na prática, uma máquina teria mais estados para acompanhar os excessos, para que pudéssemos dar troco, mas aqui ignoramos isso. A seguir vêm as setas para as moedas de 10 centavos (que chamamos de  $d$ , de *dime* em inglês)

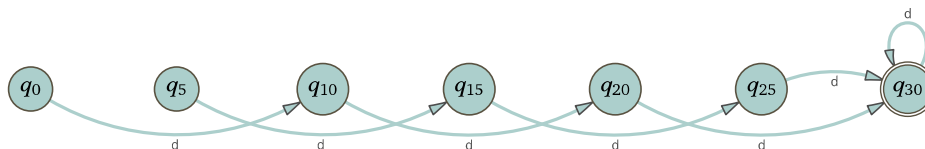


Figura 12.4: Máquina de vendas. Camada para as moedas de 10 centavos.

e para as moedas de 25 centavos (que chamamos de  $q$ , de *quarter* em inglês).

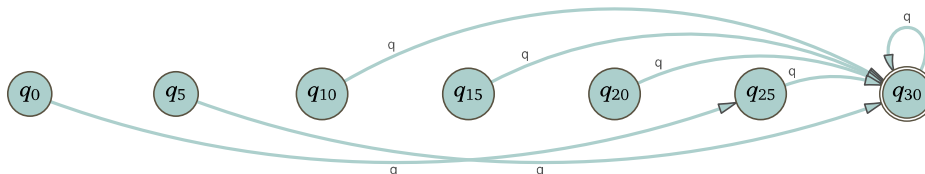


Figura 12.5: Máquina de vendas. Camada para as moedas de 25 centavos.

### Exemplo 12.1.4

Esta máquina, quando iniciada no estado  $q_0$  e alimentada com cadeias de bits, manterá o registro do resto módulo 4 da quantidade de 1's.

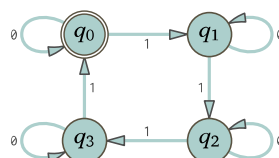


Figura 12.6: Quantidade de 1's módulo 4

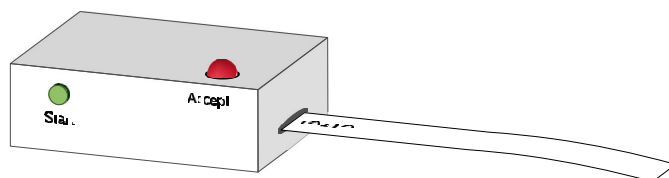
## Definição

Uma **máquina de estados finitos**, ou **autômato de estados finitos**, é composta por cinco coisas,  $\mathcal{M} = \langle Q, q_{\text{start}}, F, \Sigma, \Delta \rangle$ . São elas: um **conjunto finito de estados**  $Q$ , um dos quais é o **estado inicial**  $q_{\text{start}}$ , um subconjunto  $F \subseteq Q$  de **estados de aceitação** ou **estados finais**, um conjunto **alfabeto de entrada** finito  $\Sigma$ , e uma **função do próximo estado** ou **função de transição**  $\Delta : Q \times \Sigma \rightarrow Q$ .

Isto pode não parecer imediatamente como a nossa definição de uma Máquina de Turing. Parte disso deve-se ao fato de já termos definido os termos ‘alfabeto’ e ‘função de transição’. As outras diferenças decorrem do fato de que as máquinas de estados finitos não podem escrever. Em primeiro lugar, por não poderem escrever, as máquinas de estados finitos não precisam mover a fita para o trabalho de rascunho, por isso abandonamos os símbolos de ações de fita L e R.

A outra diferença entre as máquinas de estados finitos e as máquinas de Turing é a presença dos estados de aceitação. Considere, na máquina de venda automática do Exemplo 12.1.3, o estado  $q_{30}$ . Ele é um estado de aceitação, o que significa que a máquina viu na entrada o que procura. O mesmo se aplica ao estado  $q_{\text{ready}}$  do Exemplo 12.1.2 e ao estado  $q_{\text{on}}$  do interruptor de alimentação do Exemplo 12.1.1. Embora possamos conceber uma Máquina de Turing para indicar uma escolha, organizando de modo que, para cada entrada, a máquina parará e a única coisa na fita será um 1 ou 0, uma máquina de estados finitos fornece uma decisão, terminando num destes estados designados. Podemos imaginar que os estados de aceitação são ligados a uma luz vermelha para que saibamos quando uma computação é bem-sucedida. Nos grafos de transição denota-se os estados finais com círculos duplos e nas tabelas de funções de transição marcamos-os com um '+’.

Para trabalhar com uma máquina de estados finitos, colocamos a entrada de comprimento finito na fita e pressionamos Start.



A máquina consome a entrada, em cada passo apagando o caractere anterior da fita e depois lendo o seguinte. Podemos rastrear os passos quando a máquina de módulo 4 do Exemplo 12.1.4 recebe a entrada 10110.

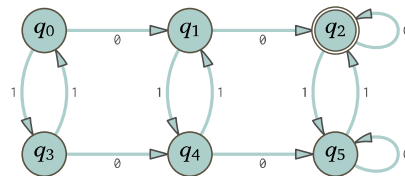
Passo	Configuração	Passo	Configuração
0	<div>1 0 1 1 0</div> <div>q<sub>0</sub></div>	3	<div>1 0</div> <div>q<sub>2</sub></div>
1	<div>0 1 1 0</div> <div>q<sub>1</sub></div>	4	<div>0</div> <div>q<sub>3</sub></div>
2	<div>1 1 0</div> <div>q<sub>1</sub></div>	5	<div></div> <div>q<sub>3</sub></div>

Consequentemente, não há problema da parada para as máquinas de estados finitos — elas sempre param após um número de passos igual ao comprimento da entrada.

No final, ou a luz Accept está acesa ou não está. Se estiver acesa, então dizemos que a máquina **aceita** a cadeia de entrada, caso contrário **rejeita** a cadeia.

### Exemplo 12.1.5

Esta máquina aceita uma string se e somente se contiver pelo menos dois 0's, bem como um número par de 1's. (O '+' ao lado de  $q_2$  marca-o como um estado de aceitação).

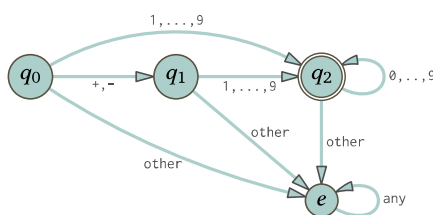


$\Delta$	0	1
$q_0$	$q_1$	$q_3$
$q_1$	$q_2$	$q_4$
+ $q_2$	$q_2$	$q_5$
$q_3$	$q_4$	$q_0$
$q_4$	$q_5$	$q_1$
$q_5$	$q_5$	$q_2$

Esta máquina ilustra o segredo para conceber máquinas de estado finito, que cada estado tenha um significado intuitivo. O estado  $q_4$  significa “até agora, a máquina viu um 0 e um número ímpar de 1's”. E  $q_5$  significa “até agora, a máquina viu dois 0's mas um número ímpar de 1's”. O grafo traz à tona este princípio. A sua primeira linha tem estados que até agora têm visto um número par de 1's, enquanto os estados da segunda linha viram um número ímpar. A sua primeira coluna contém estados que não viram 0's, a segunda coluna contém estados que viram um zero, e a terceira coluna tem estados que viram dois 0's.

### Exemplo 12.1.6

Esta máquina aceita strings que são válidas como representações decimais de números inteiros. Assim, ela aceita '21' e '-707' mas não aceita '501-'. Tanto o grafo de transição como a tabela agrupam algumas entradas quando resultam na mesma ação. Por exemplo, quando no estado  $q_0$  esta máquina faz a mesma coisa, quer a entrada seja + ou -, ou seja, passa para  $q_1$ .



$\Delta$	+, -	0, ... 9	else
$q_0$	$q_1$	$q_2$	$e$
$q_1$	$e$	$q_2$	$e$
+ $q_2$	$e$	$q_2$	$e$
$e$	$e$	$e$	$e$

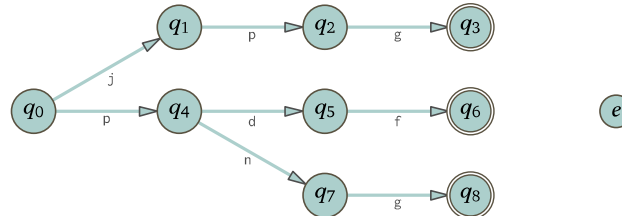
Qualquer caractere de entrada incorreto envia a máquina para o estado de erro,  $e$ , que é um estado de sumidouro, o que significa que a máquina nunca sai desse estado.

As nossas descrições de máquina de estado finito assumirão geralmente que o alfabeto é claro a partir do contexto. Por exemplo, o exemplo anterior apenas diz “else” (ou no grafo, “other”). Na prática, tomamos o alfabeto como sendo o conjunto de caracteres que alguém poderia possivelmente digitar, incluindo letras como a e A ou caracteres como ponto de exclamação ou abre parênteses. Assim, a concepção de uma

máquina de estados finitos de acordo com um padrão moderno poderia utilizar todo o Unicode. Mas para os exemplos e exercícios aqui, vamos utilizar pequenos alfabetos.

### Exemplo 12.1.7

Esta máquina aceita cadeias que são membros do conjunto  $\{ \text{jpg}, \text{pdf}, \text{png} \}$  de extensões de nomes de arquivo. Note que ela tem mais de um estado final.

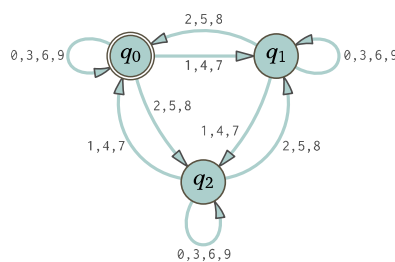


Esse desenho omite muitas arestas, as que envolvem o estado de erro  $e$ . Por exemplo, do estado  $q_0$  qualquer caractere de entrada que não seja  $j$  ou  $p$  é um erro. (Colocar todas as arestas faria uma bagunça. Casos como este são aqueles em que a tabela de transição é melhor do que a figura do grafo. Mas a maioria das nossas máquinas são pequenas e têm apenas alguns estados e arestas, por isso, normalmente preferimos a figura).

Este exemplo ilustra que para qualquer linguagem finita existe uma máquina de estado finito que aceita uma cadeia se e somente se for um membro da linguagem. A ideia é colocar as cadeias em ordem alfabética, e para aquelas que têm prefixos comuns, a máquina passa pelas partes partilhadas em conjunto, como aqui com  $\text{pdf}$  e  $\text{png}$ .

### Exemplo 12.1.8

Embora não tenham memória de rascunho, as máquinas de estados finitos podem realizar trabalhos úteis, como alguns tipos de aritmética. Esta máquina aceita cadeias representando um número natural que é um múltiplo de três, tais como 15 e 5013.

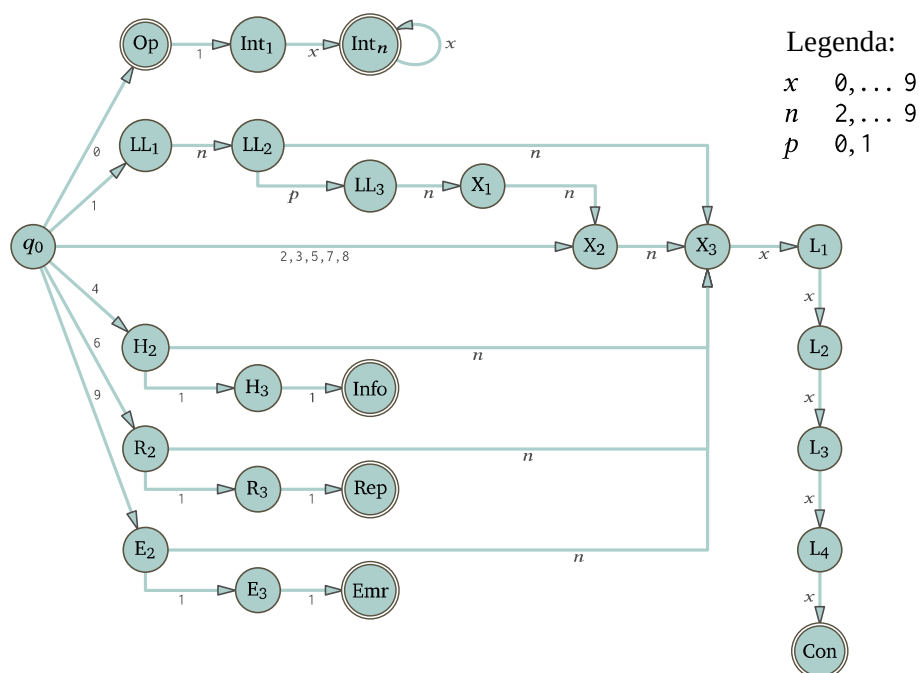


Como  $q_0$  é um estado de aceitação, esta máquina aceita a cadeia vazia.

### Exemplo 12.1.9

Esta é uma versão simplificada de como os números de telefone costumavam ser tratados na América do Norte. Considere o número 1-802-555-0101. O 1 inicial significa que a chamada deve deixar a central local para ir para as linhas de longa distância. O 802 é um código de área; o sistema pode perceber isso porque o seu segundo dígito é 0 ou 1, portanto não é uma central local da mesma área. A seguir o sistema processa o número de central local, o 555, encaminhando a chamada para uma central

local física específica. Essa central processa o número de linha 0101, e faz a conexão.



Hoje em dia, o quadro é muito mais complicado. Por exemplo, já não é necessário que os códigos de área tenham um dígito central 0 ou 1. Esta complicação adicional é possível porque em vez de comutar com dispositivos físicos, fazemo-lo agora em software.

Após a definição da máquina de Turing, demos uma descrição formal da ação dessas máquinas. A seguir, fazemos o mesmo aqui.

Uma **configuração** de uma máquina de estados finitos é um par  $C = \langle q, \tau \rangle$ , onde  $q$  é um estado,  $q \in Q$ , e  $\tau$  é uma cadeia (possivelmente vazia),  $\tau \in \Sigma^*$ . Iniciamos uma máquina com alguma cadeia de **entrada**  $\tau_0$  e dizemos que a configuração inicial é  $C_0 = \langle q_0, \tau_0 \rangle$ .

Uma máquina de estados finitos atua por uma sequência de **transições** de uma configuração para outra. Para  $s \in \mathbb{N}^+$  a configuração da máquina após a  $s$ -ésima transição é a sua configuração no **passo**  $s$ ,  $C_s$ .

Esta é a regra para fazer uma transição (dizemos por vezes que é uma transição **permitida** ou **legal**, para enfatizar). Suponha que a máquina está na configuração  $C_s = \langle q, \tau_s \rangle$ . No caso de  $\tau_s$  não ser vazia, apague o símbolo inicial da cadeia,  $c$ . Ou seja, onde  $c = \tau_s[0]$ , pegue  $\tau_{s+1} = \langle \tau_s[1], \dots, \tau_s[k] \rangle$  para  $k = |\tau_s| - 1$ . Então, o próximo estado da máquina é  $\hat{q} = \Delta(q, c)$  e a sua próxima configuração é  $C_{s+1} = \langle \hat{q}, \tau_{s+1} \rangle$ . Denote esta relação antes-depois entre as configurações por  $C_s \vdash C_{s+1}$ .<sup>1</sup>

O outro caso é quando a string  $\tau_s$  é vazia. Esta é a **configuração de parada**  $C_h$ . Nenhuma transição se segue a partir da configuração de parada.

<sup>1</sup> Assim como para as Máquinas de Turing, leia o símbolo  $\vdash$  como “deriva em um passo”.

A cada transição, o comprimento da string da fita diminui de um, de modo que cada computação eventualmente atinge uma configuração de parada  $C_h = \langle q, \varepsilon \rangle$ . Uma computação de máquina de estados finitos é uma sequência  $C_0 \vdash C_1 \vdash C_2 \vdash \dots \vdash C_h$ . Podemos abreviar essa sequência por  $\vdash^*$ , como em  $C_0 \vdash^* C_h$ .<sup>2</sup>

Se o estado ao se chegar na configuração de parada for um estado final,  $q \in F$ , então a máquina aceita a entrada  $\tau_0$ , caso contrário ela rejeita  $\tau_0$ .

Note que, tal como no formalismo para as máquinas de Turing, o cerne das definições é a função de transição  $\Delta$ . Ela faz a máquina mover-se passo a passo, de configuração em configuração, em resposta à entrada.

#### Exemplo 12.1.10

A máquina de múltiplos de três do Exemplo 12.1.8 fornece a computação.  $\langle q_0, 5013 \rangle \vdash \langle q_2, 013 \rangle \vdash \langle q_2, 13 \rangle \vdash \langle q_0, 3 \rangle \vdash \langle q_0, \varepsilon \rangle$ . Uma vez que  $q_0$  é um estado de aceitação, a máquina aceita 5013.

#### Definição

O conjunto de cadeias aceitas por uma máquina de estados finitos  $\mathcal{M}$  é a **linguagem dessa máquina**,  $\mathcal{L}(\mathcal{M})$ , ou a linguagem **reconhecida**, ou **decidida**, (ou **aceita**), pela máquina.

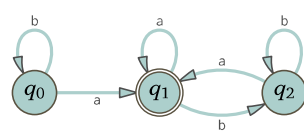
Para as máquinas de estados finitos, decidir uma linguagem é equivalente a reconhecê-la. ‘Reconhecer’ é o termo mais comum do que ‘Decidir’ aqui.

#### Definição

Para qualquer máquina de estados finitos com função de transição  $\Delta : Q \times \Sigma \rightarrow Q$ , a **função de transição estendida**  $\hat{\Delta} : \Sigma^* \rightarrow Q$  fornece o estado em que a máquina terminará após ter começado no estado inicial e consumido a cadeia dada.

#### Exemplo 12.1.11

A função de transição estendida  $\hat{\Delta}$  desta máquina



$\Delta$	a	b
$q_0$	$q_1$	$q_0$
$+ q_1$	$q_1$	$q_2$
$q_2$	$q_1$	$q_2$

estende a sua função de transição ordinária  $\Delta$  na medida em que repete a primeira linha da tabela de  $\Delta$ .

$$\hat{\Delta}(a) = q_1 \quad \hat{\Delta}(b) = q_0$$

<sup>2</sup> Leia o símbolo  $\vdash^*$  como “deriva eventualmente”, ou simplesmente “deriva”.

(Ignoramos a diferença entre os caracteres de entrada  $\Delta$  e as cadeias de caracteres de comprimento um de  $\hat{\Delta}$ ). Aqui está o efeito de  $\hat{\Delta}$  nas cadeias de caractere de comprimento dois.

$$\hat{\Delta}(aa) = q_1 \quad \hat{\Delta}(ab) = q_2 \quad \hat{\Delta}(ba) = q_1 \quad \hat{\Delta}(bb) = q_0$$

Note que isso requer determinismo, sem o qual  $\hat{\Delta}$  não estaria bem definida;  $\Delta$  tem um estado seguinte para todas as configurações de entrada e assim, por indução, para todas as cadeias de entrada,  $\hat{\Delta}$  tem um estado de saída ao final.

Finalmente, note a semelhança entre  $\hat{\Delta}$  e  $\phi_e$ , a função computada pela máquina de Turing  $\mathcal{P}_e$ . Ambas tomam como entrada o conteúdo da fita inicial da sua máquina, e ambas fornecem como saída o resultado da sua máquina.

## 12.2. Não-determinismo

Tanto as máquinas de Turing quanto as máquinas de estados finitos têm a propriedade de que o estado seguinte é completamente determinado pelo estado e caractere atuais. Uma vez que se estabelece uma fita inicial e se aperta o Start, então simplesmente se caminha através dos passos como, bem... como um autômato. Consideramos agora máquinas não determinísticas, nas quais, a partir de qualquer configuração, a máquina poderia mover-se para mais de um estado seguinte, ou apenas para um, ou mesmo para nenhum estado.

**Motivação.** Imagine uma gramática com algumas regras e símbolo de início. É-nos fornecida uma cadeia e nos é perguntado se existe uma derivação. O desafio para estes problemas é que por vezes é preciso adivinhar qual o caminho que a derivação deve tomar. Por exemplo, se tivermos  $S \rightarrow aS \mid bA$  então a partir de  $S$  podemos fazer duas coisas diferentes; qual delas funcionará?

Nos exemplos de derivações com gramáticas, esperamos que uma pessoa inteligente tivesse o discernimento para adivinhar o caminho correto. No entanto, se em vez disso você estivesse escrevendo um programa, então poderia tentar cada caso; poderia fazer uma busca em largura na árvore de todas as derivações, até obter um sucesso.

O filósofo estadunidense e famoso jogador de baseball Y Berra disse: “Quando chegar a uma bifurcação na estrada, pegue-a”. Essa é uma forma natural de atacar este problema: quando se deparar com múltiplas possibilidades, bifurque um processo filho para cada uma. Assim, a rotina pode começar com o estado inicial  $S$  e para cada regra que possa ser aplicada ela desencadeia um processo filho, derivando uma cadeia que é uma substituição desde o início. Depois disso, cada filho encontra cada regra que poderia aplicar à sua cadeia e gera os seus próprios filhos, cada um dos quais possui agora uma cadeia que é duas substituições desde o início. Continue até aparecer a cadeia desejada  $\sigma$ , se é que alguma vez ela vai aparecer.

O exemplo prototípico é o famoso problema do Caixeiro Viajante, o de encontrar o circuito mais curto para todas as cidades numa lista. Comece com um mapa das estradas nos vinte e seis estados do Brasil. Queremos saber se existe uma viagem que visite cada capital de estado e regresse ao local onde começou em, digamos, menos de 16.000 quilômetros. Vamos começar em Recife, a capital de Pernambuco. A partir



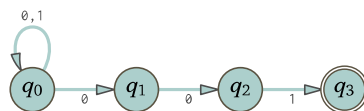
daí, para cada potencial próxima capital, podemos bifurcar um processo, fazendo vinte e cinco novos processos. O processo que é atribuído a João Pessoa, Paraíba, por exemplo, saberia que a viagem até agora é de 122 quilômetros. Na iteração seguinte, cada filho bifurcaria os seus próprios processos filhos, vinte e quatro deles. Por exemplo, o processo que, depois de Recife foi atribuído a João Pessoa, teria um filho atribuído a Natal, Rio Grande do Norte, e saberia que até agora a viagem é de 288 quilômetros. No final, se houver uma viagem de menos de 16.000 quilômetros, então algum processo saberá disso. Haverá muitos processos e muitos deles terão falhado em encontrar uma viagem curta, mas se mesmo um apenas for bem-sucedido, então consideramos a busca global como um sucesso.

Esta computação é não determinística, na medida em que enquanto está acontecendo a máquina está simultaneamente em muitos estados diferentes. Imagine uma máquina sem limites de paralelismo, onde sempre que houver um trabalho para um agente computacional adicional, uma CPU, você pode alocar uma.<sup>3</sup> Pense numa tal máquina como angelical, na medida em que sempre que ela quiser mais recursos computacionais, tais como a possibilidade de alocar novos filhos, esses recursos simplesmente aparecem.

Esta seção considera as máquinas de estados finitas não determinísticas (máquinas de Turing não determinísticas apareceram no capítulo dez). Teremos duas formas de pensar sobre o não-determinismo, dois modelos mentais.<sup>4</sup> O primeiro foi introduzido acima: quando uma máquina deste tipo é apresentada com múltiplos estados seguintes possíveis, então a máquina bifurca, de modo a estar em todos eles simultaneamente. O exemplo seguinte ilustra isso.

### Exemplo 12.2.1

A máquina de estados finitos abaixo é não determinística porque saindo de  $q_0$  há duas setas rotuladas como 0. Ela também tem estados com um deficit de arestas; por exemplo, nenhuma seta 1 sai de  $q_1$ , portanto se ela estiver nesse estado e ler essa entrada então não passa para nenhum estado.



A figura a seguir mostra o que acontece com a entrada 00001. Representamos o rastro da computação como uma árvore. Por exemplo, no primeiro 0, a computação divide-se em duas.

<sup>3</sup> Isto ecoa a nossa experiência com computadores quotidianos, quando estamos escrevendo um e-mail numa janela enquanto vemos um vídeo noutra. A máquina parece estar em múltiplos estados simultaneamente, embora possa, na realidade, estar “fatiando o tempo”, ou seja, executando cada processo em sucessão por alguns tiques.

<sup>4</sup> Embora estes modelos sejam úteis para aprender e pensar sobre o não-determinismo, eles não fazem parte das definições e provas formais.

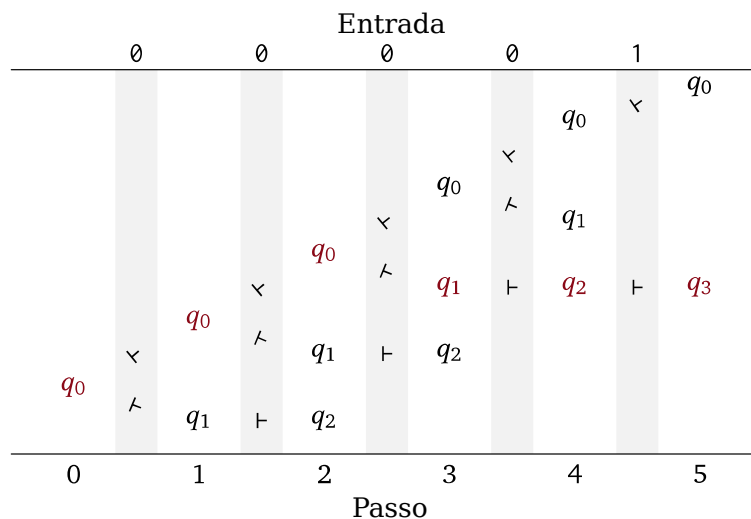


Figura 12.7: Árvore para a computação da máquina (não-determinística).

Quando consideramos a abordagem de bifurcação para derivações de strings ou para o Caixeiro Viajante, observamos que se existisse uma solução, então algum processo filho a encontraria. O mesmo acontece aqui; há um ramo da árvore da computação que aceita a cadeia de entrada. Há também ramos que não são bem-sucedidos. O da parte inferior morre após o passo 2 porque quando o estado atual é  $q_2$  e a entrada é 0, esta máquina passa para o não-estado.<sup>5</sup> Outro é o ramo da parte superior, que nunca morre mas também não aceita a entrada. No entanto, não nos importamos com ramos mal sucedidos, apenas nos importamos que haja um ramo bem-sucedido. Por isso, vamos definir que uma máquina não determinística aceita uma entrada se houver pelo menos um ramo na árvore da computação que aceite a entrada.

A máquina no exemplo acima aceita uma cadeia se ela terminar em dois 0's e um 1. Quando lhe fornecemos a entrada 00001, o problema que a máquina enfrenta é: quando deve parar de percorrer o loop de  $q_0$ 's e partir para a direita? A nossa definição diz que a máquina aceita esta entrada de modo que a máquina resolveu este problema — visto de fora, poderíamos dizer, talvez um pouco fantasiosamente, que a máquina adivinhou corretamente. Este é o nosso segundo modelo para o não-determinismo. Vamos imaginar a programação chamando uma função, alguma  $\text{amb}(S, R_0, R_1, \dots, R_{n-1})$ , e o computador conseguindo, de alguma forma, adivinhar uma sequência bem-sucedida.

Dizer que a máquina está adivinhando é chocante. Com base nas aulas de programação, a intuição de uma pessoa pode ser que “adivinhar” não é mecanicamente realizável. Em vez disso, podemos imaginar que a resposta é fornecida à máquina (“dê a volta duas vezes, depois vá para a direita”) e só temos que verificá-la. Este modelo mental de não determinismo parece demoníaco porque o fornecedor da resposta assemelha-se a um ser sobrenatural, um demônio, que de alguma forma conhece respostas que de outra forma não podem ser encontradas, mas não é confiável e portanto deve-se verificar que a resposta não é um truque. Sob este modelo, um cálculo não-determinístico aceita a entrada se existir um ramo da árvore de cálculo que uma máquina determinística, se lhe for dito que ramo tomar, poderia verificar.

<sup>5</sup> O não-estado não pode ser um estado de aceitação, uma vez que não é sequer um estado.

Abaixo descreveremos o não determinismo usando ambos os paradigmas: como uma máquina que está em múltiplos estados ao mesmo tempo, e como uma máquina que adivinha. Como mencionado acima, aqui faremos isso para as máquinas de estados finitos, mas isso também poderia ser feito no contexto das máquinas de Turing.

### Definição

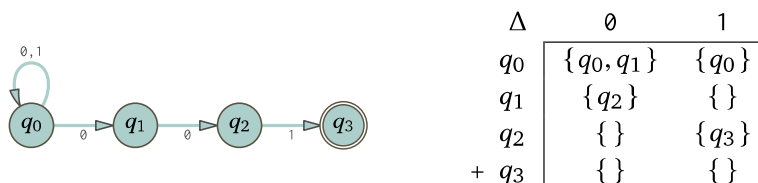
A função do próximo estado de uma máquina de estados finitos não determinística não produz estados únicos, mas sim conjuntos de estados.

Uma **máquina de estados finitos não determinística**  $\mathcal{M} = \langle Q, q_{\text{start}}, F, \Sigma, \Delta \rangle$  consiste num conjunto finito de estados  $Q$ , um dos quais é o estado inicial  $q_{\text{start}}$ , um subconjunto  $F \subseteq Q$  de estados de aceitação ou estados finais, um conjunto alfabeto de entrada finito  $\Sigma$ , e uma função do próximo estado  $\Delta : Q \times \Sigma \rightarrow \mathbb{P}(Q)$ .

Vamos utilizar estas máquinas de três maneiras. Primeiro, com elas descobrimos o não-determinismo, que é de extrema importância na análise de complexidade computacional. Em segundo lugar, elas também são úteis na prática; nos exemplos a seguir veremos tarefas que são mais facilmente resolvidas desta forma. Finalmente, vamos usá-los para demonstrar o Teorema de Kleene (FIXME ref).

### Exemplo 12.2.2

Esta é a máquina de estados finitos não determinística do Exemplo 12.2.1, juntamente com a sua função de transição.



Note que numa máquina não determinística, os valores das células da tabela de transição não são estados, são conjuntos de estados.

As máquinas não determinísticas podem parecer conceitualmente confusas, por isso as formalidades são uma ajuda. Contraste estas definições com as de máquinas determinísticas.

Uma **configuração** é um par  $C = \langle q, \tau \rangle$ , onde  $q \in Q$  e  $\tau \in \Sigma^*$ . Uma máquina começa com uma configuração inicial  $C_0 = \langle q_0, \tau_0 \rangle$ . A string  $\tau_0$  é a **entrada**.

Dada a configuração inicial, pode haver uma ou mais sequências de **transições**. Suponha que existe uma configuração de máquina  $C_s = \langle q, \tau_s \rangle$ . Para  $s \in \mathbb{N}^+$ , no caso em que  $\tau_s$  não é a cadeia de caracteres vazia, uma transição remove o primeiro símbolo  $c$  da cadeia de caracteres para obter  $\tau_{s+1}$ , toma um membro  $\hat{q}$  do conjunto  $\Delta(q, c)$  como próximo estado da máquina e depois assume uma configuração subsequente  $C_{s+1} = \langle \hat{q}, \tau_{s+1} \rangle$ . Denote que duas configurações estão conectadas por uma transição por  $C_s \vdash C_{s+1}$ .

O outro caso é quando  $\tau_s$  é a string vazia. Esta é uma **configuração de parada**,  $C_h$ . Depois de  $C_h$ , não se seguem transições.

Uma **computação não determinística de uma máquina de estados finitos** é uma sequência de transições que termina numa configuração de parada,  $C_0 = \langle q_0, \tau_0 \rangle \vdash C_1 \vdash C_2 \vdash \dots C_h = \langle q, \varepsilon \rangle$ . A partir de uma configuração inicial pode haver muitas dessas sequências. Se pelo menos uma terminar em um estado de parada com  $q \in F$ , então a máquina **aceita** a entrada  $\tau_0$ , caso contrário ela **rejeita**  $\tau_0$ .

### Exemplo 12.2.3

Para a máquina não determinística do Exemplo 12.2.1, a figura Figura 12.7 mostra a seguinte computação:

$$\langle q_0, 00001 \rangle \vdash \langle q_0, 0001 \rangle \vdash \langle q_0, 001 \rangle \vdash \langle q_1, 01 \rangle \vdash \langle q_2, 1 \rangle \vdash \langle q_3, \varepsilon \rangle$$

Como termina num estado de aceitação, a máquina aceita a cadeia inicial, 00001.

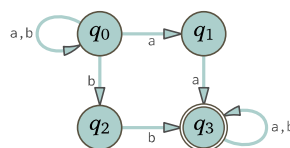
### Definição

Para uma máquina de estados finitos não determinística  $\mathcal{M}$ , o conjunto de cadeias aceitas é a **linguagem da máquina**  $\mathcal{L}(\mathcal{M})$ , ou a linguagem **reconhecida**, (ou **aceita**), por essa máquina.<sup>6</sup>

Também vamos adaptar a definição da **função de transição estendida**  $\hat{\Delta} : \Sigma^* \rightarrow Q$ . Seja  $\mathcal{M}$  uma máquina não-determinística com função de transição  $\Delta : Q \times \Sigma \rightarrow Q$ . Comece com  $\hat{\Delta}(\varepsilon) = \{q_0\}$ . Onde  $\hat{\Delta}(\tau) = \{q_{i0}, q_{i1}, \dots, q_{ik}\}$  para  $\tau \in \Sigma^*$ , defina  $\hat{\Delta}(\tau \hat{\ } t) = \Delta(q_{i0}, t) \cup \Delta(q_{i1}, t) \cup \dots \cup \Delta(q_{ik}, t)$  para todo  $t \in \Sigma$ . Então a máquina aceita  $\sigma \in \Sigma^*$  se e somente se qualquer elemento de  $\hat{\Delta}(\sigma)$  for um estado final.

### Exemplo 12.2.4

A linguagem reconhecida por esta máquina não determinística



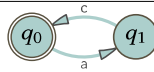
é o conjunto de cadeias que contém a substring aa ou bb. Por exemplo, a máquina aceita abaaba porque existe uma sequência de transições permitidas terminando num estado de aceitação, a saber, este:

$$\langle q_0, abaaba \rangle \vdash \langle q_0, baaba \rangle \vdash \langle q_0, aaba \rangle \vdash \langle q_1, aba \rangle \vdash \langle q_2, ba \rangle \vdash \langle q_2, a \rangle \vdash \langle q_2, \varepsilon \rangle$$

### Exemplo 12.2.5

Com  $\Sigma = \{a, b, c\}$  esta máquina não determinística.

<sup>6</sup> Abaixo definiremos algo chamado  $\varepsilon$  transições que tornam ‘reconhecida’ a ideia adequada aqui, ao invés de ‘decidida’.

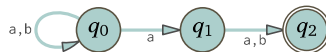


reconhece a linguagem  $\{ (ac)^n \mid n \in \mathbb{N} \} = \{ \varepsilon, ac, acac, acac, \dots \}$ . O símbolo  $b$  não aparece em nenhuma seta, portanto não desempenhará um papel em nenhuma cadeia aceitável.

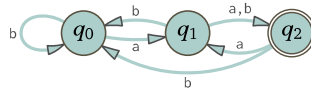
Muitas vezes uma máquina não determinística de estados finitos é mais fácil de escrever do que uma máquina determinística que faz o mesmo trabalho.

### Exemplo 12.2.6

Esta é uma máquina não-determinística que aceita qualquer cadeia de caracteres cujo penúltimo caractere é  $a$

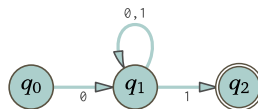


Ela é mais simples do que a máquina determinística:



### Exemplo 12.2.7

Esta máquina aceita  $\{ \sigma \in \mathbb{B}^* \mid \sigma = 0 \wedge \tau \wedge 1 \text{ onde } \tau \in \mathbb{B}^* \}$ .



### Exemplo 12.2.8

Este é um receptor de abertura de porta de garagem que espera ouvir o controle remoto enviar o sinal 0101110. Ou seja, reconhece a linguagem  $\{ \sigma \wedge 0101110 \mid \sigma \in \mathbb{B}^* \}$ .



**Observação.** Tendo visto alguns exemplos, paramos para reconhecer novamente, como fizemos quando discutimos o anjo e o demônio, que algo sobre o não-determinismo é inquietante. Se alimentarmos  $\tau = 010101110$  ao receptor do exemplo anterior, então ele a aceita.

$$\langle q_0, 010101110 \rangle \vdash \langle q_0, 10101110 \rangle \vdash \langle q_0, 0101110 \rangle \vdash \langle q_1, 101110 \rangle \vdash \langle q_2, 01110 \rangle \vdash \langle q_3, 1110 \rangle \vdash \langle q_4, 110 \rangle \vdash \langle q_5, 10 \rangle \vdash \langle q_6, 0 \rangle \vdash \langle q_7, \varepsilon \rangle$$

Mas a sequência de estados da máquina é concebida para uma string, 0101110, que começa com dois conjuntos de 01's, enquanto que  $\tau$  começa com três. Como ela adivinha que deve ignorar o primeiro 01 mas agir sobre o segundo? Claro, em matemática podemos considerar tudo o que pudermos definir com precisão. No entanto, estudamos até agora o que pode ser feito por dispositivos que, em princípio, são fisicamente realizáveis, de modo que isto pode parecer ser uma mudança de rumo.

No entanto, a seguir mostraremos como converter qualquer máquina de estados finitos não determinística em determinística que faz o mesmo trabalho. Assim, podemos pensar numa máquina de estados finitos não determinística como uma abreviatura, uma conveniência. Isto evita pelo menos uma parte do paradoxo de adivinhar, pelo menos para as máquinas de estados finitos.

**Transições  $\varepsilon$ .** Outra extensão, para além do não-determinismo, é permitir transições  $\varepsilon$ , ou movimentos  $\varepsilon$ . Alteramos a definição de uma máquina de estados finitos não determinística, de modo a que em vez de  $\Delta : Q \times \Sigma \rightarrow \mathbb{P}(Q)$ , a assinatura da função de transição é  $\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathbb{P}(Q)$ .<sup>7</sup> O comportamento associado é que a máquina pode transitar espontaneamente, sem consumir qualquer entrada.<sup>8</sup>

### Exemplo 12.2.9

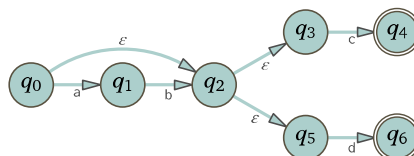
Esta máquina reconhece representações inteiras válidas. Note o  $\varepsilon$  entre  $q_0$  e  $q_1$ .



Por causa do  $\varepsilon$  ela pode aceitar cadeias que não comecem com um sinal  $+$  ou  $-$ . Por exemplo, com a entrada 123 a máquina pode começar seguindo a transição  $\varepsilon$  para o estado  $q_1$ , depois ler o 1 e fazer a transição para  $q_2$ , e ficar lá enquanto processa o 2 e 3. Este é um ramo da árvore da computação que aceita a entrada, e por isso a cadeia 123 está na linguagem da máquina.

### Exemplo 12.2.10

Uma máquina pode seguir duas ou mais transições  $\varepsilon$ . A partir de  $q_0$  esta máquina pode permanecer nesse estado, ou fazer a transição para  $q_2$ , ou  $q_3$ , ou  $q_5$ , tudo sem consumir qualquer entrada.



Ou seja, a linguagem desta máquina é o conjunto de quatro elementos  $L = \{abc, abd, c, d\}$ .

Podemos fornecer uma definição precisa da ação de uma máquina de estados finitos não-determinística com transições  $\varepsilon$ .

Primeiro definimos a coleção de estados alcançáveis por movimentos  $\varepsilon$  a partir de um determinado estado. Para tal, utilizamos  $E : Q \times \mathbb{N} \rightarrow \mathbb{P}(Q)$  onde  $E(q, i)$  é o conjunto de estados alcançáveis a partir de  $q$  dentro de, no máximo, uma quantidade  $i$  de transições  $\varepsilon$ . Ou seja, defina  $E(q, 0) = \{q\}$  e sendo  $E(q, i) = \{q_{i0}, \dots, q_{ik}\}$ , temos que  $E(q, i + 1) = E(q, i) \cup \Delta(q_{i0}, \varepsilon) \cup \dots \cup \Delta(q_{ik}, \varepsilon)$ . Observe que estes conjuntos são aninhados,  $E(q, 0) \subseteq E(q, 1) \subseteq \dots$  e que cada um deles é um subconjunto de  $Q$ .

<sup>7</sup> Assuma que  $\varepsilon \notin \Sigma$ .

<sup>8</sup> Ou, pense nisso como fazer uma transição ao consumir a cadeia de caracteres vazia,  $\varepsilon$ .

Mas  $Q$  tem apenas um número finito de estados, então deve haver um  $\hat{i} \in \mathbb{N}$  onde a sequência de conjuntos pára de crescer,  $E(q, \hat{i}) = E(q, \hat{i} + 1) = \dots$ . Defina a função de fechamento  $\varepsilon$ ,  $\hat{E} : Q \rightarrow \mathcal{P}(Q)$  por  $\hat{E}(q) = E(q, \hat{i})$ .

Com isso, estamos prontos para descrever a ação da máquina. Tal como antes, uma **configuração** é um par  $C = \langle q, \tau \rangle$ , onde  $q \in Q$  e  $\tau \in \Sigma^*$ . Uma máquina começa com alguma **configuração inicial**  $C_0 = \langle q_0, \tau_0 \rangle$ , em que a string  $\tau_0$  é a entrada.

A descrição chave é a de uma **transição**. Considere uma configuração  $C_s = \langle q, \tau_s \rangle$  para  $s \in \mathbb{N}$  e suponha que  $\tau_s$  não é a cadeia de caracteres vazia. Vamos descrever uma configuração  $C_{s+1} = \langle \hat{q}, \tau_{s+1} \rangle$  que está relacionada com a configuração dada por  $C_s \vdash C_{s+1}$  (tal como na descrição anterior de máquinas não determinísticas sem transições  $\varepsilon$ , pode haver mais do que uma configuração relacionada desta forma com  $C_s$ ). A cadeia de caracteres é fácil; basta retirar o primeiro caractere para obter  $\tau_s = t \wedge \tau_{s+1}$  onde  $t \in \Sigma$ . Para obter um estado legal  $\hat{q}$ : (i) encontre o fecho  $\varepsilon \hat{E}(q) = \{q_{s0}, \dots, q_{sk}\}$  (ii) faça  $\bar{q}$  um elemento do conjunto  $\Delta(q_{s0}, t) \cup \Delta(q_{s1}, t) \cup \dots \cup \Delta(q_{sk}, t)$ , e (iii) tome  $\hat{q}$  como um elemento do fecho  $\varepsilon \hat{E}(\bar{q})$ .

Se  $\tau_s$  é a string vazia, então esta é uma **configuração de parada**,  $C_h$ . Nenhuma transição segue  $C_h$ .

Uma **computação de uma máquina de estados finitos não determinística** é uma sequência de transições que termina numa configuração de parada,  $C_0 = \langle q_0, \tau_0 \rangle \vdash C_1 \vdash C_2 \vdash \dots \vdash C_h = \langle q, \varepsilon \rangle$ . A partir de um dado  $C_0$  pode haver muitas dessas sequências. Se pelo menos uma terminar com um estado de parada, tendo  $q \in F$ , então a máquina **aceita** a entrada  $\tau_0$ , caso contrário **rejeita**  $\tau_0$ .

Com isso, modificaremos a definição da função de transição estendida  $\hat{\Delta} : \Sigma^* \rightarrow Q$  que fornecemos anteriormente. Comece definindo  $\hat{\Delta}(\varepsilon) = \hat{E}(q_0)$ . Depois a regra para passar de uma string para a sua extensão é a de  $\tau \in \Sigma^*$  e onde  $\hat{\Delta}(\tau) = \{q_{i0}, q_{i1}, \dots, q_{ik}\}$ .  $\Delta(\tau \wedge t) = \hat{E}(\Delta(q_{i0}, t)) \cup \dots \cup \hat{E}(\Delta(q_{ik}, t))$  para  $t \in \Sigma$ .

Observe que esta máquina não determinística com transições  $\varepsilon$  aceita uma cadeia  $\sigma \in \Sigma^*$  se qualquer um dos estados em  $\hat{\Delta}(\sigma)$  for um estado final.

**Nota.** É certo que estas constituem um conjunto intrincado de definições, mas elas estão aqui para demonstrar algo. Nos exemplos usamos frequentemente termos informais tais como “adivinhar” e “demônio”. No entanto, não tome esta linguagem evocativa e informal como uma incapacidade de seguir a ortodoxia matemática. Podemos perfeitamente dar definições e resultados com total precisão.

### Exemplo 12.2.11

Para a máquina do Exemplo 12.2.10, esta sequência mostra que ela aceita  $abc$ :

$$\langle q_0, abc \rangle \vdash \langle q_1, bc \rangle \vdash \langle q_2, c \rangle \vdash \langle q_3, c \rangle \vdash \langle q_4, \varepsilon \rangle$$

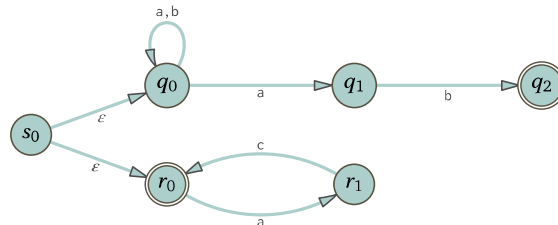
(note a transição  $\varepsilon$  entre  $q_2$  e  $q_3$ ). Esta sequência mostra que ela também aceita a cadeia de entrada  $d$ :

$$\langle q_0, d \rangle \vdash \langle q_5, d \rangle \vdash \langle q_6, \varepsilon \rangle.$$

Tal como com as máquinas não-determinísticas, uma das razões que nos leva a utilizar transições  $\epsilon$  é que elas podem tornar a resolução de um trabalho complexo muito mais fácil.

### Exemplo 12.2.12

Uma transição  $\epsilon$  pode juntar duas máquinas com uma conexão paralela. Vejamos uma máquina cujos estados são nomeados com  $q$ 's combinados com uma cujos estados são nomeados com  $r$ 's

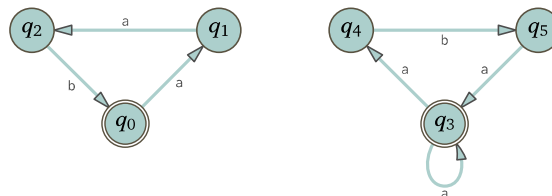


A linguagem da máquina não determinista superior é  $\{\sigma \in \Sigma^* \mid \sigma \text{ termina em } ab\}$  e a linguagem da máquina inferior é  $\{\sigma \in \Sigma^* \mid \sigma = (ac)^n \text{ para algum } n \in \mathbb{N}\}$ , onde  $\Sigma = \{a, b, c\}$ . A linguagem para a máquina inteira é a união:

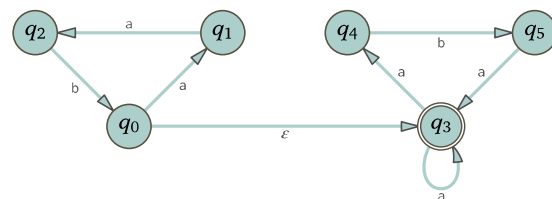
$$\{\sigma \in \Sigma^* \mid \sigma \text{ termina em } ab \text{ ou } \sigma = (ac)^n \text{ para } n \in \mathbb{N}\}$$

### Exemplo 12.2.13

Uma transição  $\epsilon$  pode também fazer uma ligação em série entre máquinas. A máquina da esquerda abaixo reconhece a linguagem  $\{(aab)^m \mid m \in \mathbb{N}\}$  e a máquina da direita reconhece  $\{(a|aba)^n \mid n \in \mathbb{N}\}$ .



Se inserirmos uma ponte  $\epsilon$  a partir de cada um dos estados finais do lado esquerdo (neste exemplo existe apenas um desses estados) para o estado inicial do lado direito



então a máquina combinada aceita cadeias na concatenação dessas linguagens.

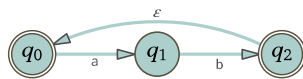
$$\mathcal{L}(\mathcal{M}) = \{\sigma \in \{a, b\}^* \mid \sigma = (aab)^m (a|aba)^n \text{ para } m, n \in \mathbb{N}\}$$

Por exemplo, ela aceita aabaababa e aabaabaaaa.



### Exemplo 12.2.14

Uma aresta de transição  $\varepsilon$  pode também produzir o fecho de Kleene de uma máquina não-determinística. Por exemplo, sem a aresta  $\varepsilon$  a linguagem desta máquina é  $\{\varepsilon, ab\}$ , enquanto que com ela a linguagem é  $\{(ab)^n \mid n \in \mathbb{N}\}$ .



A seguir veremos que o não-determinismo não muda o que podemos fazer com as máquinas de estados finitos.

### Teorema: Equivalência dos tipos de máquinas

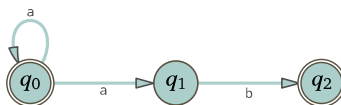
A classe de linguagens reconhecidas por máquinas de estados finitos não determinísticas é igual à classe de linguagens reconhecidas por máquinas de estados finitos determinísticas. Isto permanece verdadeiro se permitirmos que as máquinas não determinísticas tenham transições  $\varepsilon$ .

Podemos mostrar que as duas classes são iguais, mostrando que são subconjuntos uma da outra. Uma direção é fácil; qualquer máquina determinística é, basicamente, uma máquina não determinística. Ou seja, numa máquina determinística, a função do próximo estado produz estados únicos e para transformá-la numa máquina não determinística basta converter esses estados em conjuntos unitários. Assim, o conjunto de linguagens reconhecidas pelas máquinas determinísticas é um subconjunto do conjunto reconhecido pelas máquinas não determinísticas.

Vamos demonstrar a inclusão na outra direção de forma construtiva. Mostraremos como começar com uma máquina não determinística com transições  $\varepsilon$  e construir uma máquina determinística que reconhece a mesma linguagem. Não apresentaremos uma prova completa (embora certamente uma seja possível) simplesmente porque uma prova é relativamente complexa e os exemplos abaixo são inteiramente convincentes.

### Exemplo 12.2.15

Considere esta máquina não determinística,  $\mathcal{M}_N$ , sem transições  $\varepsilon$ .



O que diferencia uma máquina não determinística é que ela pode estar em múltiplos estados ao mesmo tempo. Assim, para a máquina determinística associada  $\mathcal{M}_D$ , cada linha da tabela da função de transição abaixo é um conjunto  $s_i = \{q_{i1}, \dots, q_{ik}\}$  dos estados  $\mathcal{M}_N$ 's.

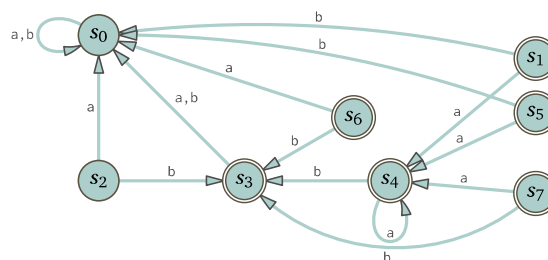
Como ilustração do processo de construção da tabela, suponha que a máquina acima está em  $s_5 = \{q_0, q_2\}$  e está lendo a. Combine os estados seguintes devido a  $q_0$ , o conjunto  $\Delta_N(q_0, a) = \{q_0, q_1\}$ , com os estados seguintes devido a  $q_2$ , o conjunto  $\Delta_N(q_2, a) = \{\}$ . A união desses dois conjuntos dá  $\Delta_D(s_5, a) = \{q_0, q_1\}$ , que abaixo é o estado  $s_4$ .

O estado inicial da  $\mathcal{M}_D$  é  $s_1 = \{q_0\}$ . Um estado da máquina determinística  $\mathcal{M}_D$  é de aceitação se algum dos seus elementos  $q$ 's é um estado de aceitação em  $\mathcal{M}_N$ .

Em geral, compute a função de transição da máquina determinística com  $\Delta_D(s_i, x) = \Delta_N(q_{i0}, x) \cup \dots \cup \Delta_N(q_{ik}, x)$ , onde  $s_i = \{q_{i0}, \dots, q_{ik}\}$  e  $x \in \Sigma$ . Um exemplo é  $\Delta_D(s_5, a) = \Delta_N(q_0, a) \cup \Delta_N(q_2, a)$ , que é igual a  $\{q_0, q_1\} \cup \{ \} = \{q_0, q_1\} = s_4$ .

$\Delta_D$	a	b
$s_0 = \{ \}$	$s_0$	$s_0$
+ $s_1 = \{q_0\}$	$s_4$	$s_0$
$s_2 = \{q_1\}$	$s_0$	$s_3$
+ $s_3 = \{q_2\}$	$s_0$	$s_0$
+ $s_4 = \{q_0, q_1\}$	$s_4$	$s_3$
+ $s_5 = \{q_0, q_2\}$	$s_4$	$s_0$
+ $s_6 = \{q_1, q_2\}$	$s_0$	$s_3$
+ $s_7 = \{q_0, q_1, q_2\}$	$s_4$	$s_3$

Além da conveniência da notação, nomear os conjuntos de estados como  $s_i$ 's torna claro que  $\mathcal{M}_D$  é uma máquina de estados finitos determinística. Tal como o seu grafo de transição.

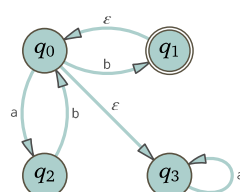


Se a máquina não determinística tem  $k$  estados, então sob esta construção a máquina determinista tem  $2^k$  estados. Tipicamente, muitos deles podem ser eliminados. Por exemplo, na máquina acima referida, o estado  $s_6$  é claramente inalcançável, uma vez que não há setas chegando nele.

A seguir expandimos essa construção para abranger transições  $\epsilon$ . Basicamente, seguimos essas transições. Por exemplo, o estado inicial da máquina determinística é o fecho  $\epsilon$  de  $\{q_0\}$ , o conjunto dos estados de  $\mathcal{M}_N$  que são alcançáveis por uma sequência de transições  $\epsilon$  a partir de  $q_0$ . Além disso, suponha que temos uma máquina não determinística e que estamos construindo a função do próximo estado da máquina determinística associada  $\Delta_D$ , que a configuração atual é  $s_i = \{q_{i1}, q_{i2}, \dots\}$  e que a máquina está lendo  $a$ . Se houver uma transição  $\epsilon$  de  $q_{ij}$  para algum  $q$ , então para o conjunto dos estados seguintes adicione  $\Delta_N(\{q\}, a)$ , e de fato adicione todo o fecho  $\epsilon$ .

### Exemplo 12.2.16

Considere esta máquina não determinística:



Para encontrar o conjunto dos próximos estados, siga as transições  $\varepsilon$ . Por exemplo, suponha que esta máquina está em  $q_0$  e que o próximo caractere da fita é  $a$ . A seta à esquerda leva a máquina de  $q_0$  para  $q_2$ . Alternativamente, seguindo a transição  $\varepsilon$  de  $q_0$  para  $q_3$  e depois lendo o  $a$  nos dá  $q_3$ . Portanto, a máquina estará em seguida nos dois estados  $q_0$  e  $q_3$ . Estes são os fechos  $\varepsilon$ .

Estado $q$	$q_0$	$q_1$	$q_2$	$q_3$
Fecho $\varepsilon \hat{E}(q)$	$\{q_0, q_3\}$	$\{q_0, q_1\}$	$\{q_2\}$	$\{q_3\}$

A máquina determinística completa está a seguir. O estado inicial é o fecho  $\varepsilon$  de  $\{q_0\}$ , o estado  $s_7 = \{q_0, q_3\}$ . Um estado é de aceitação se contiver algum elemento do fecho  $\varepsilon$  de  $q_1$ .

Em geral, para um estado  $s$  e caractere de fita  $x$ , para computar  $\Delta_D(s, x)$ : (i) encontre o fecho  $\varepsilon$  de todos os  $q$ 's do estado  $s$ , dando  $\cup_{q \in s} \hat{E}(q) = \{q_{i0}, \dots, q_{ik}\}$ , depois (ii) pegue a união dos próximos estados do  $q_{ij}$  para obter  $T = \Delta_N(q_{i0}, x) \cup \dots \cup \Delta_N(q_{ik}, x)$  e finalmente (iii) encontre o fecho  $\varepsilon$  de cada elemento de  $T$ , e tome a união de todos eles,  $\cup_{t \in T} \hat{E}(t)$ .

Como exemplo, tome  $s = s_5 = \{q_0, q_1\}$  e  $x = a$ . Então (i) nos dá  $\{q_0, q_3\} \cup \{q_0, q_1\} = \{q_0, q_1, q_3\}$ . Os estados seguintes para (ii) são  $\Delta_N(q_0, a) = \{q_2, q_3\}$ ,  $\Delta_N(q_1, a) = \{q_2, q_3\}$  e  $\Delta_N(q_3, a) = \{q_3\}$ , portanto  $T = \{q_2, q_3\}$ . Finalmente, para (iii) a união dos fechos  $\varepsilon$  nos dá  $\{q_2\} \cup \{q_3\} = \{q_2, q_3\} = s_{10}$ .

$\Delta_D$	a	b
$s_0 = \{ \}$	$s_0$	$s_0$
$+ s_1 = \{q_0\}$	$s_{10}$	$s_2$
$+ s_2 = \{q_1\}$	$s_{10}$	$s_2$
$s_3 = \{q_2\}$	$s_0$	$s_7$
$s_4 = \{q_3\}$	$s_4$	$s_0$
$+ s_5 = \{q_0, q_1\}$	$s_{10}$	$s_{12}$
$+ s_6 = \{q_0, q_2\}$	$s_{10}$	$s_{12}$
$+ s_7 = \{q_0, q_3\}$	$s_{10}$	$s_{12}$
$+ s_8 = \{q_1, q_2\}$	$s_{10}$	$s_{12}$
$+ s_9 = \{q_1, q_3\}$	$s_{10}$	$s_2$
$s_{10} = \{q_2, q_3\}$	$s_4$	$s_7$
$+ s_{11} = \{q_0, q_1, q_2\}$	$s_{10}$	$s_{12}$
$+ s_{12} = \{q_0, q_1, q_3\}$	$s_{10}$	$s_{12}$
$+ s_{13} = \{q_0, q_2, q_3\}$	$s_{10}$	$s_{12}$
$+ s_{14} = \{q_1, q_2, q_3\}$	$s_{10}$	$s_{12}$
$+ s_{15} = \{q_0, q_1, q_2, q_3\}$	$s_{10}$	$s_{12}$

## 12.3. Expressões regulares

FIXME TODO A ser atualizado em breve. Baixe novamente quando for atualizado.

## 12.4. Créditos

Todas as seções, foram adaptadas (traduzidas e modificadas) de [1], que está disponível sob a licença Creative Commons Attribution-ShareAlike 4.0 (CC BY-SA 4.0).

## 12.5. Referências

1. Hefferon, Jim. *Theory of Computation: Making Connections*. Disponível em <http://hefferon.net/computation/>

## 12.6. Licença

É concedida permissão para copiar, distribuir, transmitir e adaptar esta obra sob a Licença Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0), disponível em <http://creativecommons.org/licenses/by-sa/4.0/> .