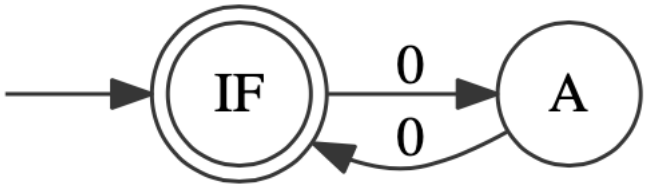
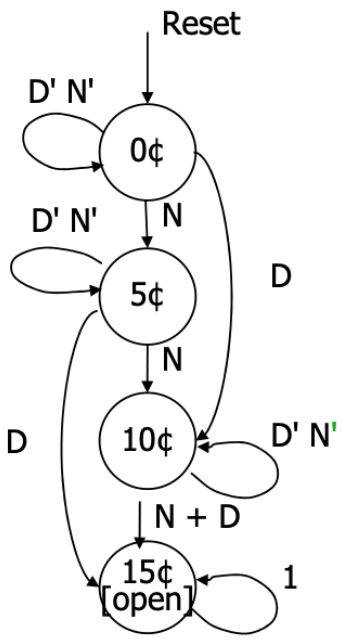
2. Autómatos Determinísticos

Copiar link



Exemplo de DFS com 2 estados



present state	inputs D	N	next state	output open
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	–	–
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	–	–
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	–	–
15¢	–	–	15¢	1

symbolic state table

Exemplo de autómato determinístico com 4 estados, que especifica o comportamento tipico de uma "vending machine" (máquina de bebidas)

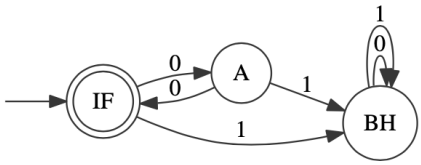
Um **Autómato Determinístico de Estados Finitos** (DFA) é um tuplo $(Q, \Sigma, \delta, q_0, F)$ em que:

- Q é um conjunto *finito* e *não vazio* de estados
- Σ é um alfabeto (igualmente *finito* e *não vazio*)
- $\delta : Q \times \Sigma \to Q$ é uma função de transição (total, ou seja, está definida para todos os *inputs*)
- $q_0 \in Q$ é o estado inicial do autómato
- $F \subseteq Q$ é um conjunto *finito* e *possivelmente vazio* de estados finais, também conhecidos por *estados de aceitação*

O significado desta definição é o seguinte:

- Existe uma noção de *execução* de um autómato:
  - em cada instante ele encontra-se num (e um só) dos seus estados
  - cada passo de execução corresponde a uma mudança de estado em resposta a um determinado estímulo (um símbolo $c \in \Sigma$)
  - o autómato *transita* do estado q para o estado q', lendo (ou consumindo) o símbolo c, se $\delta(q, c) = q'$
  - os símbolos podem ser vistos como *inputs* da execução
- a execução inicia-se sempre com o autómato no estado q_0
- e termina quando é alcançado qualquer estado final $q_f \in F$

Note-se que a necessidade de a função de transição ser total pode obrigar a incluir no autómato transições que não parece importante descrever.
Imaginemos que o autómato dado como primeiro exemplo acima tem como alfabeto $\{0,1\}$ e não $\{0\}$. Então na verdade o que se representa acima não corresponde a um DFA porque não é especificado o que acontece quando se recebe o símbolo 1 em cada estado!
O autómato pode ser totalizado criando-se um novo estado (um "buraco negro") que será o resultado de todas as transições com o símbolo 1.



Um autómato pode pois ser visto como um programa informático muito simples, em que todo o fluxo de controlo é dado por instruções *goto:* saltos directos para determinados pontos do programa, identificados por etiquetas correspondentes a estados do autómato.

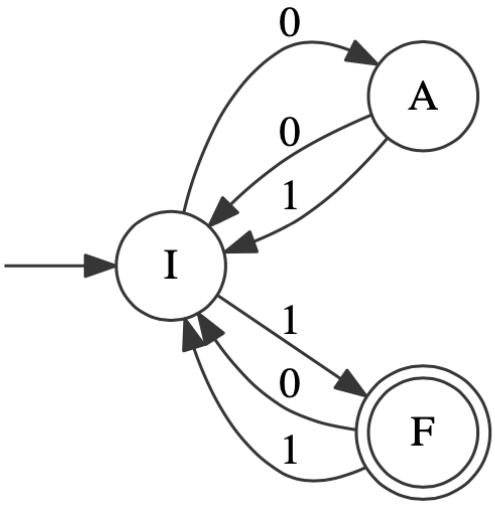
> *Este estilo de programação, dito não-estruturado, é desencorajado nas linguagens de programação de alto nível modernas, estando muito próximo do que se utiliza nas linguagens de baixo nível (assembly).*
> *No entanto ele é absolutamente universal, podendo captar o fluxo de controlo de qualquer programa, estruturado ou não.*

Note-se que cada execução do autómato corresponde a uma *palavra* $s \in \Sigma^*$, dada pela sequência de símbolos lidos desde o estado inicial ao estado final da execução. Diz-se que a palavra foi *aceite* pelo autómato.

É usual definir-se autómatos através de uma *tabela de transições,* que especifica a função δ, ou alternativamente de forma gráfica, através de um grafo orientado que tem os estados como vértices e as transições possíveis como arestas, com peso correspondente ao símbolo lido na transição.

Exemplo

State	Input	Next State
I	0	A
I	1	F
A	0	I
A	1	I
F	0	I
F	1	I



As máquinas de estados finitos, e os DFAs em particular, têm inúmeras aplicações quer em Engenharia de Software (estão por exemplo na base da linguagem de modelação UML), quer no projecto de sistemas de controlo industrial.

Do ponto de vista do estudo da computabilidade, é muito relevante estudar a relação entre autómatos e *linguagens*.

Linguagem de um Autómato Determinístico

Dado um alfabeto Σ e a sua linguagem universal Σ^*, um DFA sobre este alfabeto *aceitará* algumas palavras de Σ^*, e *rejeitará* outras. Informalmente, um DFA aceita uma palavra se os símbolos da palavra correspondem a uma sequência de transições do autómato desde o estado inicial até um qualquer estado final.

Sendo assim o autómato pode ser visto como um *classificador de palavras*. A **linguagem do autómato** é definida como o conjunto de palavras aceites por ele. Diz-se também que o autómato *reconhece* esta linguagem.

Veremos mais tarde que a classe de linguagens reconhecidas pelos autómatos de estados finitos (em particular pelos DFAs) coincide com a das linguagens regulares (um resultado fundamental devido a Kleene).

Exercícios

1. Descreva as linguagens dos dois DFAs apresentados acima como exemplos.
2. Desenhe um DFA que reconheça todas e apenas as palavras sobre o alfabeto $\{0,1\}$ que tenham um número par de 0 (podendo haver 1s pelo meio).
3. Desenhe um DFA que reconheça todas e apenas as palavras sobre o alfabeto $\{0,1\}$ que tenham um número de 0 que seja múltiplo de 3 (podendo haver 1s pelo meio).
4. Desenhe um autómato que reconheça a seguinte linguagem: $\{a^i b^j \mid i,j \geq 0\}$
5. Tente desenhar um autómato que reconheça a linguagem $\{a^j b^k c^{j+k} \mid j,k \geq 0\}$

Ao desenhar autómatos, é boa ideia começar por escrever algumas palavras aceites. Deve também notar que, caso a palavras vazia deva ser reconhecida, então o estado inicial deve ser um dos estados finais do autómato.

Definição Formal

Por forma a definir a linguagem reconhecida por um DFA $D = (Q, \Sigma, \delta, q_0, F)$ começamos por definir recursivamente uma função $\hat{\delta} : Q \times \Sigma^* \to Q$ que calcula o estado $q' = \hat{\delta}(q, s)$ em que D se encontra quando processa todos os símbolos de uma palavra s, a partir do estado q:

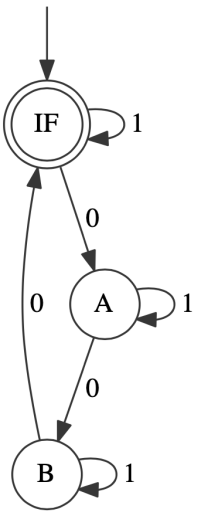
$$\begin{aligned}\hat{\delta}(q, \varepsilon) &= q \\ \hat{\delta}(q, xs) &= \hat{\delta}(\delta(q, x), s)\end{aligned}$$

O DFA D aceita a palavra $s \in \Sigma^*$ sse $\hat{\delta}(q_0, s) \in F$. A linguagem L_D reconhecida por D é definida como

$$L_D = \{s \mid s \in \Sigma^* \wedge \hat{\delta}(q_0, s) \in F\}$$

Pumping Lemma

Este lema fundamental afirma que, sendo uma linguagem L reconhecida por um qualquer um autómato finito (ou equivalentemente uma qualquer linguagem regular), todas as palavras *suficientemente longas* de L contêm uma sub-palavra que pode ser repetida um número arbitrário de vezes, resultando ainda numa palavra de L.



Neste autómato qualquer palavra de comprimento superior ou igual a 3 contém um ciclo (ou *pump*) que pode ser repetido o número desejado de vezes. Por exemplo:

- 0100 contém 2 ciclos, podendo ser vista como $0(1)00$, o que significa que $0(1)^2 00$ ou $0(1)^3 00$ são também palavras aceites. E também $0(0)^0 00 = 000$!

  Quando a execução do autómato segue um caminho que atravessa repetidamente o ciclo diz-se que se faz "pumping up". Quando o ciclo não faz parte da palavra reconhecida ($i = 0$) faz-se "pumping down".
  No primeiro exemplo 01100 é aceite por *pumping up*, e 000 por *pumping down*.

F**ormalmente:**
Seja D um DFA e $L(D)$ a linguagem reconhecida por ele. **Então**, existe um número $n \geq 0$ tal que, para qualquer palavra $s \in L(D)$ de comprimento não inferior a n, s pode ser escrita na forma xyz, em que
  - $y \neq \varepsilon$, e
  - xy corresponde a um número de símbolos lidos (transições do autómato) não superior a n, e
  - $xy^i z \in L(D)$ para qualquer $i \geq 0$.

Este lema é muito útil quando se deseja mostrar que uma determinada linguagem **NÃO é reconhecida por nenhum DFA.** Dada uma linguagem L, basta mostrar que qualquer que seja o valor de $n \geq 0$, existe uma palavra $s \in L$ de comprimento não inferior a n tal que:

- qualquer que seja a decomposição dessa palavra na forma xyz com $y \neq \varepsilon$ e em que xy tem comprimento não superior a n,
- existe um $i \geq 0$ tal que $xy^i z \notin L$,

Exemplo
Demonstremos, usando o *pumping lemma*, que a linguagem $\{a^i b^i \mid i \geq 0\}$ não é reconhecida por nenhum DFA.

Seja s a palavra $a^n b^n$ para um dado n. Note-se então que ao decompor $a^n b^n = xyz$, com a restrição de xy ter comprimento $\leq n$, necessariamente não poderá ocorrer o símbolo b em xy. Ora, como $y \neq \varepsilon$, na palavra $xy^0z = xz$ teremos menos símbolos a do que b, logo $xy^0z \notin L$. Mas poderíamos também escolher qualquer valor de $i \geq 2$: por exemplo na palavra xy^2z teremos necessariamente mais as do que bs, logo $xy^2z \notin L$.

Uma palavra de cautela: a versão do *pumping lemma* enunciada acima é simples, mas não é a mais forte que é possível enunciar. Por esta razão, nem sempre é possível provar, recorrendo a ele, que não existe um DFA para uma determinada linguagem, quando de facto não existe. Mas as linguagens para as quais se observa esta impossibilidade são bastante "rebuscadas" (consultar exemplos em Gopalakrishnan, *Automata and Computability — A Programmer's Perspective*).

Exercício
Mostre que a linguagem $\{a^j b^k c^{j+k} \mid j, k \geq 0\}$ não é regular.

Desenho de DFAs para o reconhecimento de linguagens concretas

Desenhe autómatos (DFA) que reconheçam as seguintes linguagens:

1. $L_3 = \{s \mid s \in \{0,1\}^* \wedge \text{todas as sub-palavras contíguas de 3 bits contêm dois 1s}\}$
   Note que a restrição se aplica a todos os blocos de 3 bits em "janela deslizante"! Assim, a palavra 01101 pertence à linguagem porque 011, 110, e 101 têm dois 1s, mas a palavra 01100 já não pertence.
2. $\{0,1\}^* - L_3$ (linguagem complementar)
3. $\{s \mid s \in \{0,1\}^* \wedge s \text{ termina com a sequência } 0101\}$
4. $\{s \mid s \in \{0,1\}^* \wedge \text{ o seu } \textit{penúltimo símbolo} \text{ é um } 1\}$
5. $\{s \mid s \in \{0,1\}^* \wedge \text{ o seu } \textit{antepenúltimo símbolo} \text{ é um } 1\}$

   Para esta última linguagem limite-se a calcular o número de estados do autómato. O que se pode concluir sobre o crescimento do número de estados necessários?

Minimização de Autómatos Determinísticos

Dois autómatos dizem-se *equivalentes* se reconhecem exactamente a mesma linguagem. Se além de equivalente tiverem o mesmo número de estados, dizem-se *isomorfos.*

Naturalmente, é útil em alguns contextos trabalhar com o autómato mais pequeno (i.e. com o menor número de estados) capaz de reconhecer uma dada linguagem. Estes **autómatos mínimos** têm a seguinte propriedade:

Teorema de Myhill-Nerode
> *Sejam A1 e A2 dois autómatos mínimos que reconhecem a mesma linguagem. Então A1 e A2 são isomorfos.*

O processo que permite obter um autómato mínimo a partida de um qualquer DFA é conhecido por *minimização.*
Existem vários algoritmos diferentes para a minimização de um DFA. Veremos aqui um algoritmo que utiliza a estratégia conhecida por *programação dinâmica.*

O algoritmo baseia-se na noção de *estados indistinguíveis.*

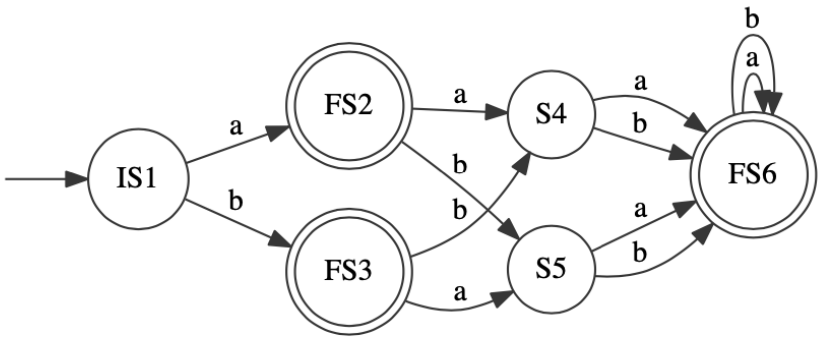
Dado o DFA $(Q, \Sigma, \delta, q_0, F)$, designaremos por $L(q)$, com $q \in Q$, o conjunto $\{s \mid \hat{\delta}(q)(s) \in F\}$, i.e. o conjunto de palavras que levam o autómato desde o estado q até um estado final.

Dois estados q_1, q_2 de um DFA dizem-se

- *k-distinguíveis* se existe alguma palavra s de comprimento k tal que
  - $\hat{\delta}(q_1)(s) \in F$ e $\hat{\delta}(q_2)(s) \notin F$,
    ou
  - $\hat{\delta}(q_1)(s) \notin F$ e $\hat{\delta}(q_2)(s) \in F$
- *indistinguíveis,* se $L(q_1) = L(q_2)$, i.e. os conjuntos de palavras que os levam até um estado final são iguais.

Num autómato mínimo não existem estados diferentes indistinguíveis, ou seja, se $L(q_1) = L(q_2)$ então necessariamente $q_1 = q_2$.

No seguinte autómato:



- IS1 e FS2 são 0-distinguíveis, uma vez que um deles é final e o outro não, logo a string ε é aceite a partir de um deles, e não do outro
- FS2 e FS6 são 1-distinguíveis, uma vez que a palavra "a" leva o primeiro para o estado não-final S4, mas leva o segundo para um estado final (o próprio FS6)
- IS1 e S5 são 2-distinguíveis, uma vez que a palavra "aa" leva o primeiro para o estado não-final S4, mas leva o segundo para o estado final FS6
- Não existe qualquer par de estados 3-distinguíveis
- Os estados S4 e S5 são indistinguíveis, sendo $L(S4) = L(S5)$

O processo de minimização identifica incrementalmente os pares de estados *distinguíveis em k passos,* começando com $k = 0$, e continuando até encontrar um valor de k tal que o conjunto de estados distinguíveis em k passos seja vazio.

O passo básico do algoritmo aplica o seguinte princípio:

> *Seja $\delta(p, c) = p'$ e $\delta(q, c) = q'$, sendo p' e q' distinguíveis em k passos. Então p e q são distinguíveis em $k + 1$ passos.*

No final do algoritmo consideraremos indistinguíveis todos os pares de estados que não tenham sido considerados distinguíveis num máximo de k passos. Esses pares podem então ser "fundidos" para construir o autómato mínimo.

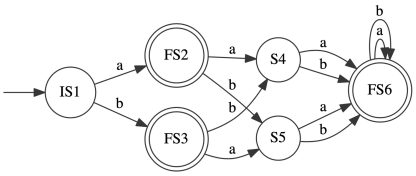
EXEMPLO

Passo 0:
Marcamos com -1 todos os pares de estados: inicialmente não sabemos se são ou não distinguíveis.

	IS1	FS2	FS3	S4	S5	FS6
IS1						
FS2	-1					

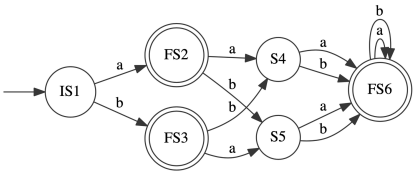
FS3	-1	-1				
S4	-1	-1	-1			
S5	-1	-1	-1	-1		
FS6	-1	-1	-1	-1	-1	

Passo 1 (estados 0-distinguíveis)



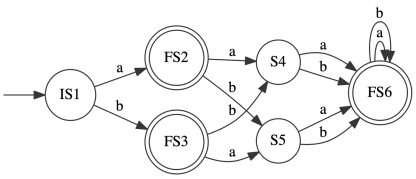
	IS1	FS2	FS3	S4	S5	FS6
IS1						
FS2	**0**					
FS3	**0**	-1				
S4	-1	**0**	**0**			
S5	-1	**0**	**0**	-1		
FS6	**0**	-1	-1	**0**	**0**	

Passo 2 (estados 1-distinguíveis)



	IS1	FS2	FS3	S4	S5	FS6
IS1						
FS2	0					
FS3	0	-1				
S4	-1	0	0			
S5	-1	0	0	-1		
FS6	0	**1**	**1**	0	0	

Passo 3 (estados 2-distinguíveis)



	IS1	FS2	FS3	S4	S5	FS6
IS1						

FS2	0					
FS3	0	-1				
S4	**2**	0	0			
S5	**2**	0	0	-1		
FS6	0	1	1	0	0	

É fácil ver que não existem estados 3-distinguíveis, por isso o algoritmo pára neste ponto.

Para obter o autómato mínimo basta agora fundir os pares de estados que permanecem indistinguíveis: