CAPÍTULO 2

AUTÓMATOS FINITOS

2.1. Introdução

No Cap.1 estudámos as noções básicas de linguagens, gramáticas e autómatos. Podem-se definir muitas linguagens a partir de um mesmo alfabeto, conforma as regras particulares de combinação dos caracteres do alfabeto.

Por exemplo a partir do alfabeto $\Sigma = \{a, b\}$ e usando a notação de conjuntos poderemos definir as linguagens $L_1 = \{ a^n b^m \mid n,m >= 0 \}$a que pertencem as cadeias : *λ,aabbbbb, aaaa, bbbb*, …ou a linguagem $L_2 = \{ a^n b^n \mid n >= 0 \}$ a que pertencem as cadeias: *λ,ab,aabb, aaabbb, aaaabbbb*, ou ainda $L_3 = \{a,b\}^*$ composta por qualquer cadeia de *a*'s e *b*'s ,incluindo *λ*.. Quantas linguagens se podem definir com este alfabeto ?

As linguagens podem ser definidas por uma gramática, que indica como se formam as cadeias de caracteres. Conhecidas as suas produções é fácil derivar cadeias da linguagem.

Pode-se agora pôr o problema ao contrário: dada uma cadeia de caracteres, como saber se pertence a uma dada linguagem ? Se a cadeia for pequena pode ser relativamente simples, por inspecção visual, decidir se pertence ou não. Mas para uma cadeia grande de um gramática mais elaborada, não é fácil decidir.

É aqui que entram os autómatos finitos. Vimos no Cap. 1 um autómato aceitador da cadeia *pai*. Se fosse possível construir um autómato que aceitasse todas as cadeias de uma dada linguagem (e só essas), então para se saber se uma cadeia pertence a uma linguagem bastaria dá-la a ler ao autómato. Se ele parasse no estado aceitador depois de concluir a sua leitura, a cadeia pertenceria à linguagem. Caso contrário não pertenceria.

Infelizmente não é possível desenhar um autómato para uma qualquer linguagem. Há linguagens para as quais ainda hoje não é possível decidir se uma cadeia lhe pertence ou não.

Num mesmo alfabeto pode-se definir um número infinito de linguagens, cada uma delas com características próprias. Felizmente para algumas classes de linguagens é possível construir autómatos finitos aceitadores: é o caso por exemplo das linguagens regulares, cujas propriedades veremos mais à frente. Por agora basta-nos saber que é possível construir um autómato finito aceitador para uma linguagem regular.

Teremos assim a Figura 2.1.1.

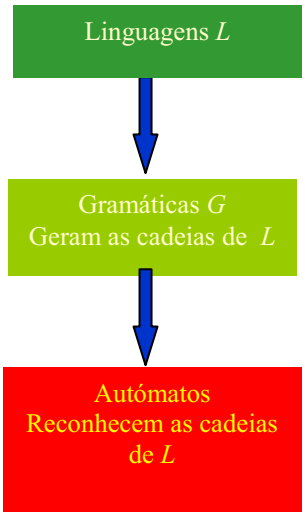


Figura 2.1.1. Relação entre linguagens, gramáticas e autómatos.

Existem outras classes de linguagens, que não regulares, para as quais é possível também construir autómatos aceitadores, naturalmente distintos dos das linguagens regulares. Vemos assim que há diferentes classes de linguagens e diferentes classes correspondentes de autómatos.

Para as linguagens regulares, as mais simples, constroem-se **autómatos finitos determinísticos**, os autómatos também mais simples. Quando um autómato é usado para reconhecer uma linguagem, é mais exacto chamar-lhe **aceitador.** No entanto usaremos com frequência o termo autómato, distinguindo-se a sua função de aceitador dentro do contexto em que é usado. Como veremos existem autómatos que não são aceitadores, isto é, não são construídos para aceitar ou não uma linguagem, mas antes para executarem sobre as cadeias de caracteres uma dada operação (como aliás já vimos no Cap. 1).

2.2. Aceitadores determinísticos

Um aceitador determinístico define-se como uma estrutura matemática e desenha-se como um grafo.

Exemplo 2.2.1.

Por exemplo o grafo seguinte representa um aceitador determinístico. O seu alfabeto é $\Sigma=\{0,1\}$.

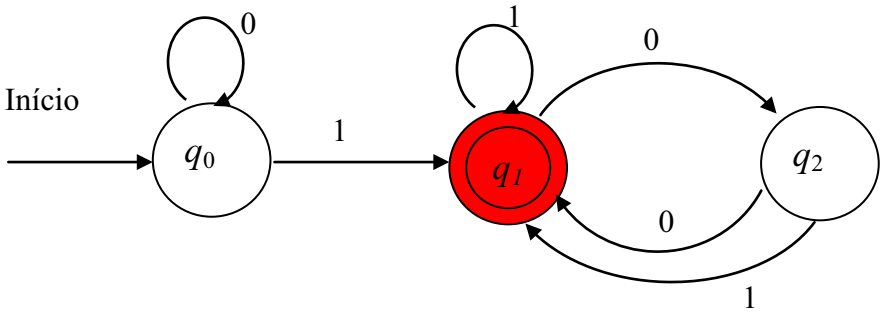


Figura 2.2.1 Grafo de um aceitador determinístico.

Vejamos como funciona.

O autómato está no estado inicial, q_0, e apresenta-se-lhe à entrada uma cadeia de 0's e 1's, com por exemplo 1100.

Ele vai ler os caracteres da esquerda para a direita, isto é 1 >1 > 0 >0. As arestas representam as transições de estado quando o autómato lê o carácter etiqueta da aresta. Estando em q_0 e lendo 1, transita para q_1. Estando em q_2 e lendo 1, transita para q_2, isto é, fica onde está. Lendo agora 0, estando em q_1, transita para q_2. Lendo depois 0 transita para q_3 e aí fica, porque não há mais nada para ler.

O estado q_1 tem uma forma especial. Ele é o estado aceitador. Se o autómato, partindo do estado inicial, leu a cadeia 1100 e terminou no estado aceitador, então **aceita** essa cadeia. Poderemos ver outras que também aceita. Por exemplo 001, 0100, 1110001, etc. Se estiver em q_0 e aparece um 1, vai para o aceitador; se estiver em q_1 e aparece um 1, mantém-se no aceitador. Se estiver em q_2 e aparece um 1, vai para o aceitador.

Então conclui-se que qualquer cadeia que termine num 1 é aceite pelo autómato: de facto qualquer que seja o seu penúltimo estado, o seu último será sempre o aceitador de ele ler apenas mais um 1. Mas há cadeias de outro tipo, como por exemplo 0100, que também, são aceites. Se depois do último 1 aparecem dois zeros seguidos (e mais nada) ele termina também no estado aceitador. Portanto o autómato aceita as cadeias que terminam num 1 ou em 00 depois do último 1.

Note-se que em cada estado o aceitador determinístico sabe exactamente o que deve fazer, porque tido lhe é dito: de cada estado há duas arestas definindo as duas transições possíveis, uma para cada carácter do alfabeto. Essas duas transições podem ser iguais, com acontece no estado q_2. Pode-se simplificar o grafo colocando apenas uma aresta com duas etiquetas, com o na figura seguinte.

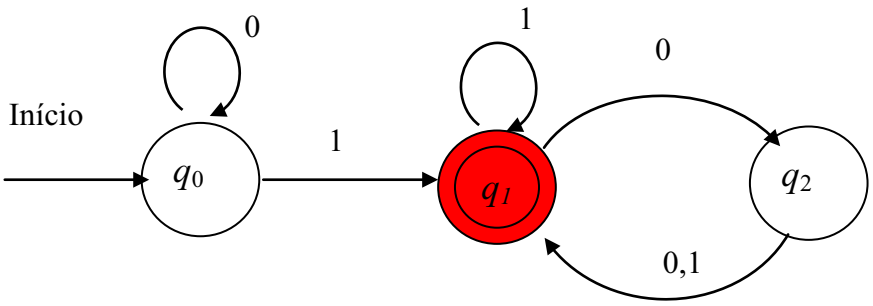


Figura 2.2.2 Grafo do mesmo aceitador da Fig. 2.2.1

Se tivéssemos uma alfabeto com três caracteres (por exemplo $\{a,b,c\}$, então de cada estado teriam que partir sempre três arestas explicitamente definidas (ainda que pudessem ser iguais). Um aceitador determinístico não tem qualquer capacidade de decidir em qualquer circunstância, daí o nome de determinístico. Consequentemente todas as transições possíveis têm que estar explicitamente definidas. A função de transição é por isso uma função total, isto é, está explicitamente definida para todos os valores do seu domínio.

Note-se que esta definição que estamos a adoptar não é seguida por todos os autores. Alguns admitem que possam existir transições não definidas. Nestes, se aparecer à entrada, num dado estado, um carácter para o qual não esteja explicitamente definida uma transição, o DFA morre, isto é, passa a um estado não aceitador e não sai mais de lá, quaisquer que sejam os caracteres seguintes na cadeia lida (mais tarde chamaremos ratoeira a este estado).

Vimos que um autómato finito determinístico é facilmente definido e descrito por um grafo. Tem cinco partes: um conjunto de estados, um conjunto de regras para transitar entre eles, um alfabeto de entrada que define os caracteres aceitáveis à entrada, um estado inicial, um estado final. Veremos casos que têm vários estados finais. Para definir formalmente o autómato, usaremos todas essas cinco partes que compõem um quinteto.

Definição 2.2.1 Aceitador determinístico

Um aceitador determinístico (usaremos o acrónimo **dfa-**, de *deterministic finite accepter*) é definido pelo quinteto

$$M=(Q,\Sigma,\ \delta,\ q_0,\ F\ )$$

em que : Q: é o conjunto finito de **estados** internos

Σ: **alfabeto** de entrada (conjunto finito de caracteres)

δ : $Qx\Sigma \rightarrow Q$ é a função **total** chamada **função de transição**

$q_0 \in Q$ é o **estado inicial**

$F \subseteq Q$ é o conjunto de **estados finais** ou **aceitadores**

Uma função é total quando é definida para todos os valores possíveis dos seus argumentos; caso contrário diz-se parcial. Ao dizer-se que a função de transição é total, atendendo á sua definição, isso quer dizer que ela tem que estar definida para todas as combinações possíveis de um estado e um carácter do alfabeto, isto é, para todos os elementos do produto cartesiano $Qx\Sigma$.

Exemplo 2.2.2. O interruptor do Cap. 1

Retomemos aqui o exemplo do interruptor do Cap. 1. Se definirmos a linguagem das sequências de *Press(P)* tais que o interruptor fica ligado após a sua aplicação (partindo do estado inicial *F*), teremos *P, PPP, PPPPP,* etc., ou seja, um número ímpar de accionamentos do interruptor. Agora poderemos definir o autómato como um aceitador dessa linguagem, colocando a dupla circunferência no estado *A*.

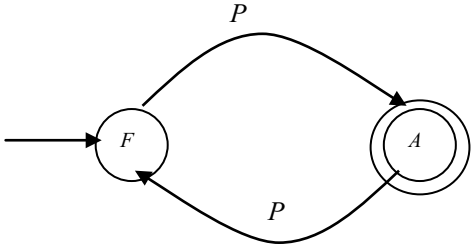


Figura 2.2.3 Grafo do interruptor do Cap. 1.

Aplicando-lhe a definição 2.1 vê-se que

Q, conjunto de estados internos: $\{F, A\}$

Σ, alfabeto de entrada: $\{P\}$

δ, função de transição: $F\times P \rightarrow A$; $A \times P \rightarrow F$

$\boldsymbol{q_0}$, é o estado inicial: F

F, o estado final: $\{A\}$

Exemplo 2.2.3.

Seja o autómato da Fig. 2.2.4.

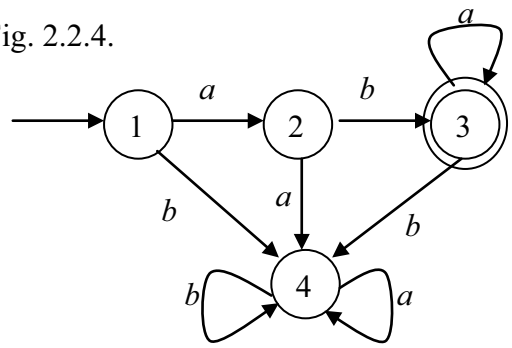


Figura 2.2.4. Autómato do exemplo 2.2.3

Aplicando-lhe de igual modo a definição 2.1 vê-se que:

Q, conjunto de estados internos: {1,2,3,4}

Σ, alfabeto de entrada: $\{a,b\}$

δ, função de transição: $1\times a \rightarrow 2$; $1 \times b \rightarrow 4$, $2 \times b \rightarrow 3$, etc.

q_0, é o estado inicial: 1

F, o estado final: {4}

Neste caso a função de transição tem muitos elementos. Usando uma tabela especifica-se mais facilmente.

Note-se que pelo facto de a função de transição der total, a tabela tem que ter **todas** as células preenchidas. Neste caso para cada estado existem duas arestas possíveis, uma para a e outra para b.

Tabela 2.2.1. Tabela de transições do DFA da Fig. 2.2.3.

Se estado actual é	e a entrada actual é	o estado seguinte será
1	*a*	2
1	*b*	4
2	*a*	4
2	*b*	3
3	*a*	3
3	*b*	4
4	*a*	4
4	*b*	4

Qual será a linguagem aceite por este autómato ?

O estado 4 tem uma característica: se o autómato lá chegar, nunca mais de lá sai, e por isso chama-se estado ratoeira, ou armadilha (*trap*) ou poço.

A figura seguinte reduz o esquema geral de um autómato que vimos no Cap. 1 ao caso particular de um dfa.

Note-se que um dfa não tem nem dispositivo de memória nem cadeia de saída. Tem apenas dispositivo de leitura e unidade de controlo. Conforme o conteúdo actual da célula lida, dá-se ou não uma transição de estado dentro da unidade de controlo.

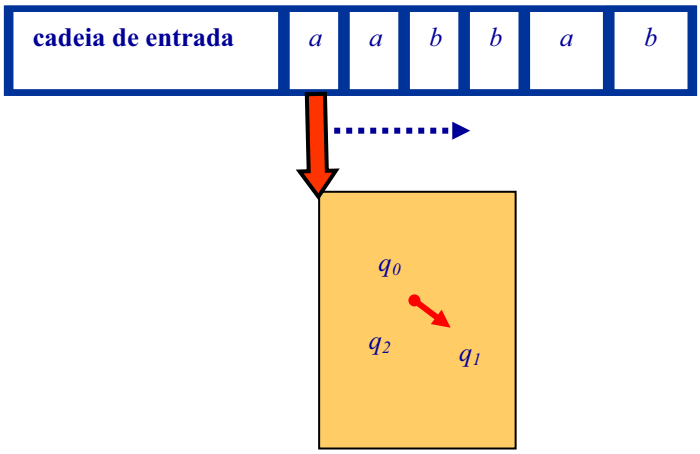


Figura 2.2.5. O esquema de um DFA. Note-se a ausência de qualquer dispositivo de memória.

A transição de um estado para outro depende do estado actual (antes da transição) e do carácter lido à entrada no instante actual.

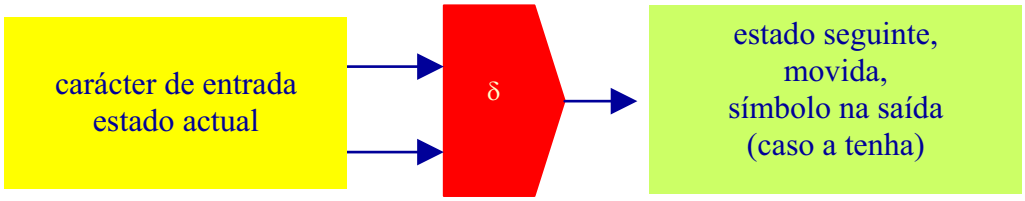
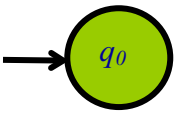


Figura 2.2.6. A função de transição de um DFA.

Um autómato representa-se por um grafo em que os vértices (nós) representam os estados e as arestas orientadas têm como etiquetas os caracteres lidos na cadeia de entrada. Há três tipos de vértices, indicados na Tabela 2.2.2., ao lado

Tabela 2.2.2

Símbolo	Significado
	Estado normal
	Estado inicial
	Estado aceitador (ou final)
[*] carácter 	Aresta

Exemplo 2.2.4

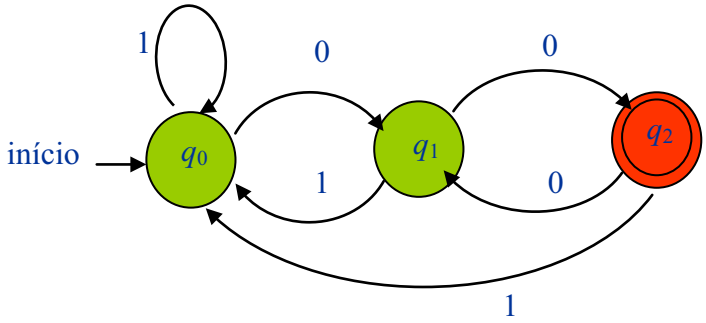


Figura 2.2.7 DFA do exemplo 2.2.4.

Temos um autómato com três estados, um de cada tipo. O alfabeto do autómato é Σ= {0,1}. Lendo com atenção as etiquetas das arestas pode-se escrever a seguinte tabela de transições. O DFA pode-se assim definir formalmente como

M= (Q, S, Σ, ,q_0,F) com

Q = {q_0,q_1,q_2}, Σ= {0,1}, F = {q_2} e δ é definida pela Tabela 2.2.4.

Tabela 2.2.3. Função de transição do exemplo 2.2.4.

Estados	Entradas	
	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_1	q_0

Qual será a linguagem aceite pelo autómato? Um bom desafio para o leitor ...

No Capítulo 3 estudaremos as expressões regulares, uma técnica de especificação de linguagens que tem uma álgebra própria muito adequada para deduzir a definição da linguagem a partir de um DFA qualquer. Neste momento poderemos fazer o seguinte procedimento heurístico:

-colocamo-nos no estado aceitador e vemos como lá poderemos chegar,

-depois andamos para trás e em cada estado vemos o mesmo. Até que se consiga visualizar mentalmente a linguagem do autómato.

Chega-se ao estado final a partir de q_1 com um 0, ou seja, com (****)0.

Chega-se a q_1 com um 0 depois de 1: (***10)0. De modo que sempre que uma cadeia termine em 100, termina no estado aceitador. Se terminar em 1000 não aceita a cadeia.

Mas aceita se terminar em 10000, 1000000, ou qualquer número par de zeros precedido de 1.

Poderemos definir formalmente linguagem de um autómato finito.

Definição 2.2.2a. Linguagem de um DFA

Dado um DFA qualquer, *M,* a linguagem *L* **aceite** (ou **reconhecida**) por *M* é o conjunto de todas as cadeias que, começando a ser lidas no estado inicial, fazem com que o autómato alcance um dos estados finais depois de toda a cadeia ter sido lida. Escreve-se $L(M)$ para dizer que *L* é a linguagem aceite ou reconhecida por *M.*

Que linguagens aceitam os seguintes autómatos ?

Exmplo 2.2.5

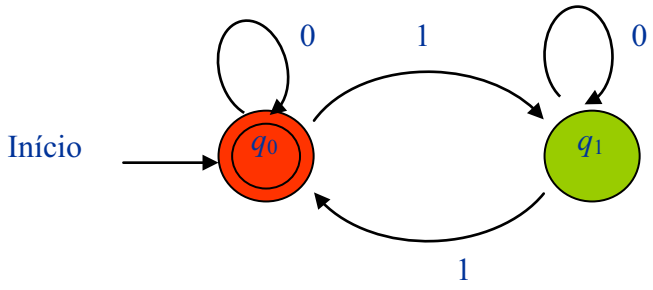


Fig. 2.2.8.

Exemplo 2.2.6

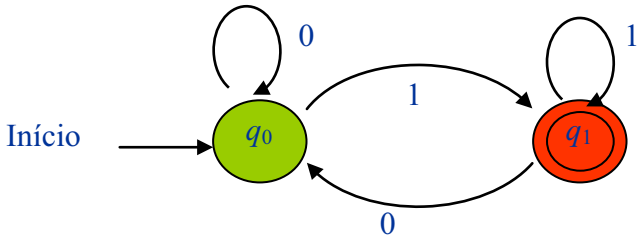


Fig. 2.2.9.

Exemplo 2.2.7

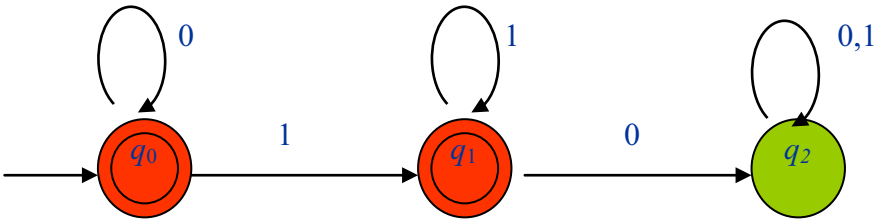


Fig. 2.2.10

Um DFA pode ser definido pela explicitação da função de transição e, a partir dela, facialmente se desenha o seu grafo.

Exemplo 2.2.8

Seja o DFA $M = (\{q_0, q_1, q_2\}, \{a,b\}, \delta, q_0, \{q_2\})$ cuja função de transição é a seguinte:

$\delta(q_0, a) = q_1$

$\delta(q^0, b) = q_2$

$\delta(q_1, a) = q_0$

$\delta(q_1, b) = q_2$

$\delta(q_2, a) = q_2$

$\delta(q_2, b) = q_2$

Tabela de transições

	a	b
q_0	q_1	q_2
q_1	q_0	q_2
q_2	q_2	q_2

Nas transições aparecem três estados q_1, q_2 e q_3. O grafo do autómato desenha-se "graficando" as transições, depois de se desenharem os seus três estados. Obtém-se assim a Fig. 2.2.11.

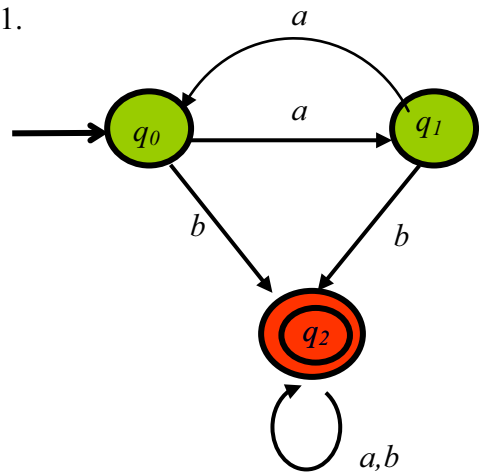
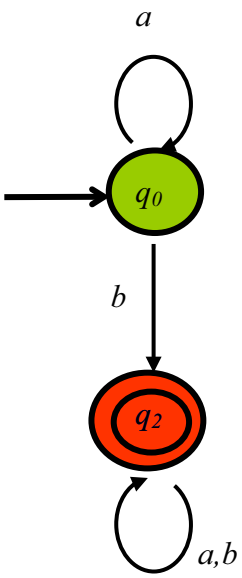


Fig. 2 .2.11

Será possível simplificar este DFA, isto é, encontrar um outro com menos estados que aceite a mesma linguagem (e só a mesma) ?

Veremos em capítulos posteriores algoritmos para simplificar autómatos, reduzindo ao mínimo possível o número de estados. Neste momento poderemos apenas olhar com atenção para o DFA e verificar o que ele faz. Se aparecer um *b*, no estado inicial, vai para o aceitador q_2. Se estiver em q_1 e aparece um *b,* transita para q_2 ; e se estiver em q_2 e aparecer um *b*, mantém-se aí. Portanto qualquer que seja o seu estado se parecer um *b* na cadeia de entrada, ele aceita-a: a sua linguagem é portanto o conjunto de todas as cadeias em $\{a,b\}^*$ que contenham pelo menos um *b.* O autómato seguinte aceita essa linguagem.

Fig. 2.2.12. Grafo equivalente ao da Fig. 2.2.11.



Função de transição estendida a cadeias de caracteres, δ^*

Na definição do DFA a função de transição é definida de tal forma que ela é activada por um carácter. Se quisermos calcular a transição do DFA de uma configuração inicial para outra configuração após um conjunto de transições, seria interessante dispor de uma forma de representação sucinta, compacta, que exprimisse essa transição "salto" resultante da leitura de um conjunto de caracteres ou de uma cadeia completa.

Pode-se estender a noção de transição com esse objectivo. Em vez de δ escreve-se δ^*, em que o asterisco quer dizer "após um certo número de transições…".

A função de transição estendida é assim definida por

$$\delta^*: Q \times \Sigma^* \rightarrow Q$$

em que

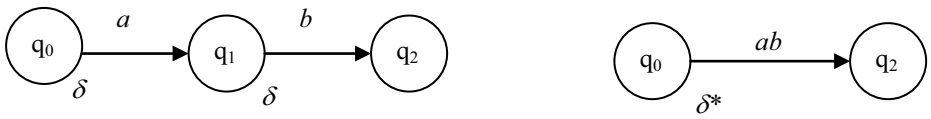
- o seu segundo argumento é uma cadeia em vez de um carácter, e

- o seu valor de saída dá o estado do autómato depois de ler a (sub)cadeia.

Por exemplo, se tivermos num autómato tal que

$$\delta(q_0, a) = q_1 \text{ e } \delta(q_1, b) = q_2 \text{ então } \delta^*(q_0, ab)=q_2$$

Representando por grafos, teremos a Fig. 2.2.13.



a) Função de transição b) Função de transição estendida

Figura 2.2.13. Ilustração da função de transição estendida. Em a) grafo normal; em b) grafo compactado usando uma transição estendida.

A função de transição estendida pode definir-se recursivamente do modo seguinte:

$$(i)\ \delta^*(q,\lambda)= q$$

$$(ii)\ \delta^*(q,wa)=\delta(\delta^*(q, w), a)$$

para todo o $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

Para o exemplo anterior teremos :

$$\delta^*(q_0, ab)=\delta(\delta^*(q_0, a), b)$$

$$\delta^*(q_0, a)=\delta^*(q_0, a\lambda)=\delta(\delta^*(q_0, \lambda), a)=\delta(q_0, a)=q_1$$

$$\delta^*(q_0, ab)=\delta(q_1, b)=q_2$$

Definição 2.2.2b.Linguagens de um DFA (definição formal)

A linguagem L aceite (ou reconhecida) por um DFA $M=(Q, \Sigma, \delta, q_0, F)$ é o conjunto de todas as cadeias em Σ aceites por M, i.e.,

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}$$

A função de transição δ e δ^* são funções totais (definidas para todos os elementos do seu domínio). Em cada passo define-se um e um só movimento, e por isso o autómato se chama determinístico, não podendo fazer escolhas. Um autómato processa todas as cadeias em Σ^*, aceitando-as ou não.

Quando não aceita uma cadeia, pára num estado não aceitador. O conjunto das cadeias em que isso se verifica constitui o complemento da linguagem $L(M)$, que se pode definir formalmente por

$$Complem(L(M)) = \{w \in \Sigma^* : \delta^*(q_0, w) \notin F\}$$

Consideremos os autómatos da Fig. 2.2 14, diferentes apenas na identificação do estado final.

O primeiro reconhece, como vimos, a linguagem das cadeias em $\{a,b\}^*$ que tenham pelo menos um b. O segundo aceita, no mesmo alfabeto, as cadeias que tenham apenas a's. Isto é, a linguagem do segundo é o complemento da linguagem do primeiro: se uma cadeia só tem a's não tem nenhum b, e se tem pelo menos um b não contém só a's.

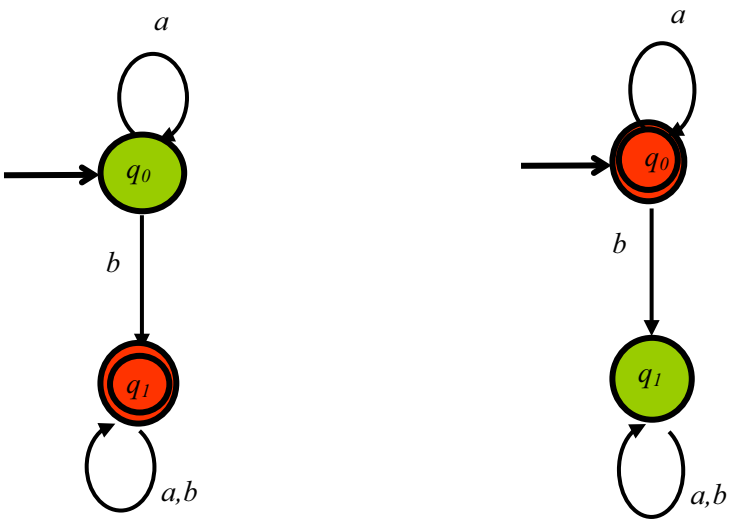


Figura. 2.2.14. Autómatos complementares no alfabeto $\{a,b\}$.

Repare-se que o estado não aceitador do primeiro autómato é o estado aceitador do segundo; e o estado aceitador do primeiro é o estado não aceitador do segundo. De facto,:

- se uma cadeia é aceite pelo primeiro não o pode ser pelo segundo: uma cadeia que termine no estado aceitador do primeiro tem que terminar no estado não aceitador do segundo;

- e se uma cadeia é recusada pelo primeiro tem que ser aceite pelo segundo: se uma cadeia termina no estado não aceitador do primeiro tem que terminar no estado aceitador do segundo.

Para este autómato com dois estados, é fácil de ver a relação entre os autómatos de linguagens complementares.

Será assim no geral ?

De facto é. Se um autómato M aceita uma linguagem $L(M)$, então o complemento de L será reconhecida pelo autómato que se obtém de M invertendo neste a função dos estados: os não aceitadores passam a aceitadores e os aceitadores passam a não aceitadores.

2.3. A arte de construir DFA's

Os exemplos de autómatos que vimos até aqui parecem simples, e a sua construção (em grafo) é relativamente expedita.

Casos há em que é bem mais difícil obter uma solução simples (na medida do possível) e apropriada para uma dada função.

A *expertise* para desenhar DFAs depende mais da arte aprendida e treinada do que de uma teoria sofisticada.

Vejamos um exemplo:

Exemplo 2.2.9 Paridade individual

Desenhar um autómato que, no alfabeto $\Sigma=\{0,1\}$, aceite todas as cadeias com um número ***ímpar*** *de* 1*'s*.

Podemos começar a desenhar estados e arestas, por tentativa e erro, até que consigamos, com mais ou menos remendos, uma solução. Se o conseguirmos, muito provavelmente obteremos um autómato confuso, com mais estados do que o necessário, difícil de interpretar por outrem.

Em vez disso, pensemos numa abordagem mais sistemática, partindo de um algoritmo mental para a folha de papel.

Em primeiro lugar quantos estados terá que ter o nosso autómato ? Lembremo-nos do significado de um estado: o autómato chega lá depois de um certo número de transições resultantes da leitura de uma cadeia. O estado contém por isso a história até a um certo ponto da cadeia. Ora num problema qualquer, o que deve memorizar o autómato? Apenas a informação mínima para ser capaz de responder ao que se lhe pede: neste caso lembrar-se da paridade do número de 1s entrados até ao momento. Se for par deve estar num certo estado, se não for deve estar noutro. O estado que corresponde a ímpar deve ser aceitador porque se não há mais nada para ler, a cadeia deve ser aceite porque entrou até ao momento um número ímpar de 1s. Temos então (pelo menos) dois estados: Par (P) e Ímpar (I), que podemos desde já desenhar.

Figura 2.3.1. Os estados do autómato de paridade individual

Se o DFA está em *P* e lê 1 na entrada, o que acontece ? Se está em *P*, é porque até ao momento entrou um número par de 1s. Se agora entra mais um, passa a ímpar, naturalmente. E então desenhe-se a transição correspondente: $\delta(P,1)=I$. E se lê um zero estando em *P* ? Como nada lhe é pedido em relação aos zeros, é-lhe perfeitamente indiferente e por isso deve manter-se em *P*. Logo $\delta(P,0)=P$. Temos então, para já,

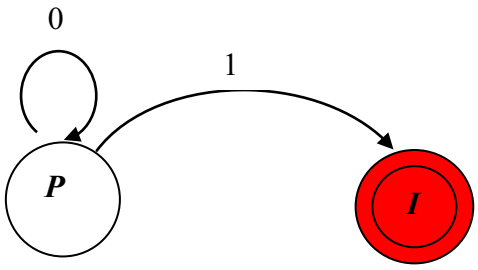


Figura 2.3.2. Transições a partir de **P**

Faça-se de seguida o mesmo raciocínio par ao estado *I*. Ele está lá porque entrou um número ímpar de 1s. Entrando mais um, fica um número par, e por isso $\delta(I,1)=P$. Pela mesma razão de cima, $\delta(I,0)=I$. Todas as transições possíveis estão agora identificadas. O estado aceitador também. Falta apenas identificar o estado inicial, onde o DFA deve estar **antes** de iniciar a leitura da cadeia, ou seja, quando entraram zero 0s e zero 1s. Ora zero é número par, e por isso *P* é o estado inicial, pelo que se lhe coloca a setinha.

Concluímos assim o desenho do DFA requerido na Fig. 2.3.3

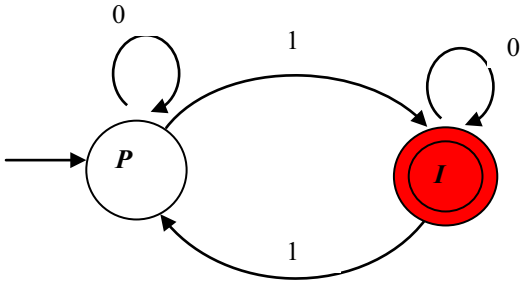
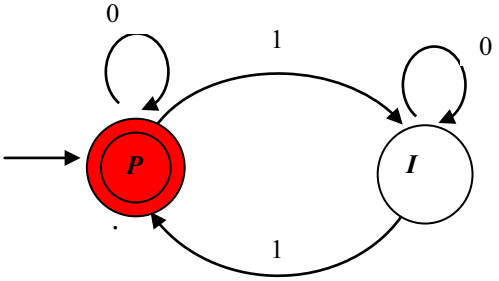


Figura 2.3.3 Autómato de paridade de 1's completo.

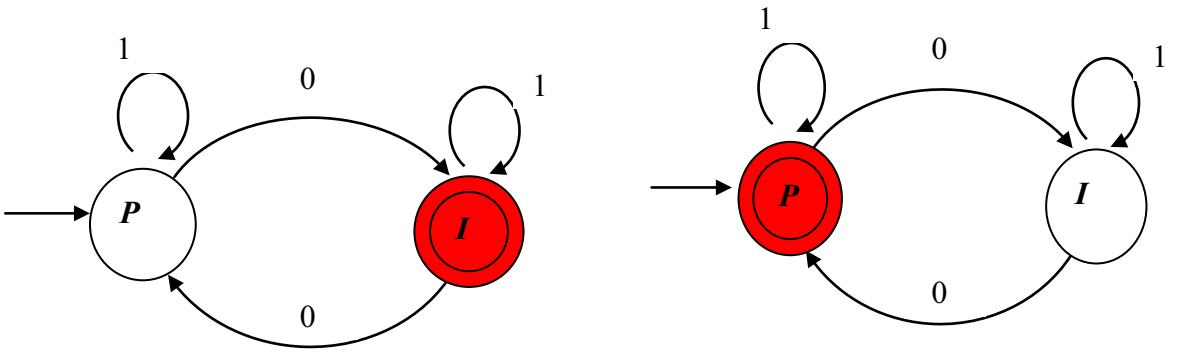
Simulando o seu funcionamento verifica-se que cumpre a sua missão.

E se quiséssemos um DFA que aceitasse, no mesmo alfabeto, todas as cadeias com um número par de 1s ? Facilmente se obtém, seguindo o mesmo algoritmo mental, o autómato da Fig. 2.3.4.

Figura 2.3.4. Autómato de um número par de 1's. Notem-se as semelhanças e diferenças relativamente ao anterior



Fica o leitor desafiado a verificar que os autómatos seguintes aceitam, ainda no alfabeto $\Sigma = \{0,1\}$, as cadeias com um número ímpar de zeros e um número par de zeros.



a) b)

Figura 2.3.5. Autómatos de paridade de zeros. a) ímpar, b) par

Exemplo 2.3.2 Paridade de grupo

Desenhar um DFA que no alfabeto Σ={0,1} aceite todas as cadeias com um número ímpar de 0*s e um número ímpar de* 1*s.*

Temos aqui uma situação mais complicada porque a imparidade é exigida simultaneamente aos dois caracteres do alfabeto.

A primeira questão, e a decisiva: quantos estados deve ter o autómato ? Ele tem que saber identificar a situação de paridade entre todas as situações de paridade possíveis. E quais são ? Tendo em conta que temos que medir simultaneamente a paridade de 0s e de 1s, elas são:

- número par de 0s e número par de 1s;

- número par de 0s e número ímpar de 1s;

- número ímpar de 0s e número par de 1s;

- número ímpar de 0s e número ímpar de 1s.

Não havendo mais nenhuma situação relevante para o caso, concluímos que quatro estados serão suficientes; chamem-se *PP, PI, IP, II*, sendo aceitador o *II*.

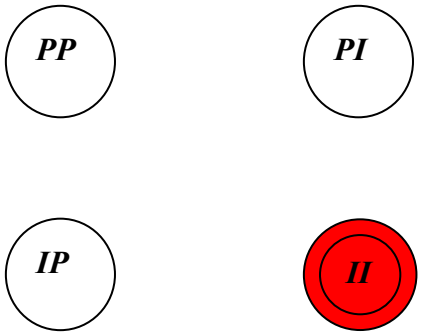


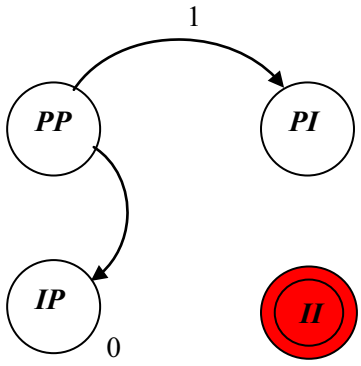
Figura 2.3.6. Os estados necessários para o exemplo 2.3.6.

Se o autómato está em *PP* é porque leu até ao momento um número par de 0s e um número par de 1s, se estão em PI, leu um número par de 0s e um número ímpar de 1s; etc.

Calculemos agora as suas transições.

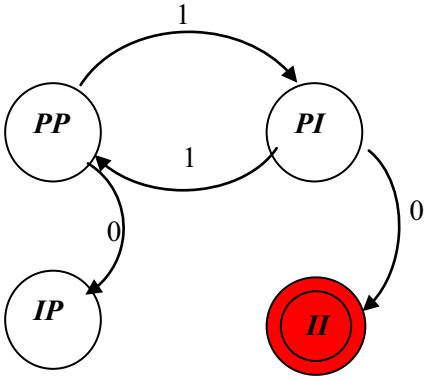
Estando em *PP*, até aí leu um número par de 0s e um número par de 1s. Se aparece agora um 0, fica um número ímpar de zeros e um número par de 1s e por isso tem que transitar para *IP.* Se pelo contrário aparece um 1, fica um número ímpar se 1s e um número par de zeros e vai assim para *PI.* Podemos então desenhar as arestas que partem de *PP*.

Figura 2.3.7. Transições a partir de *PP*



Se o DFA está em *PI* (par 0s ímpar 1s) , se lê 1 passa a *PP* e se lê 0 passa a *II*.

Figura 2.3.8. Transições a partir de *PI*



É agora fácil de ver que o autómato completo é representado pelo grafo seguinte. O estado inicial (zero 0s e zero 1s) é PP porque zero é um número par.

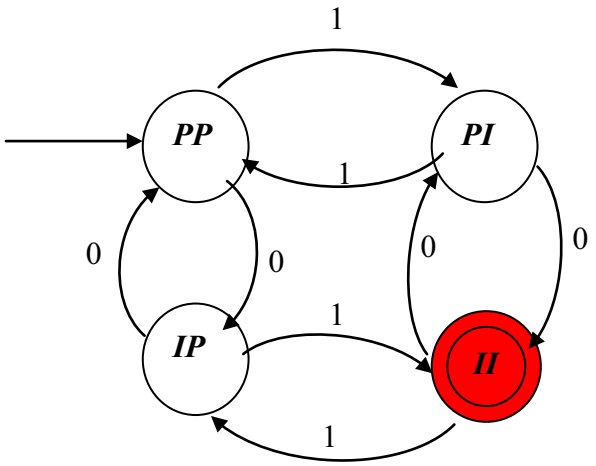


Figura 2.3.9. Grafo completo do exemplo 2.3.2.

Exemplo 2.3.3. Desenhar o autómato aceitador da linguagem composta por todas as cadeias com um número ímpar de 0s e um número ímpar de 1s.

Se quisermos o DFA que aceita as cadeias com um número impar de 0s e um número par de 1s (*IP*), basta mudar o estado aceitado no autómato anterior, obtendo-se

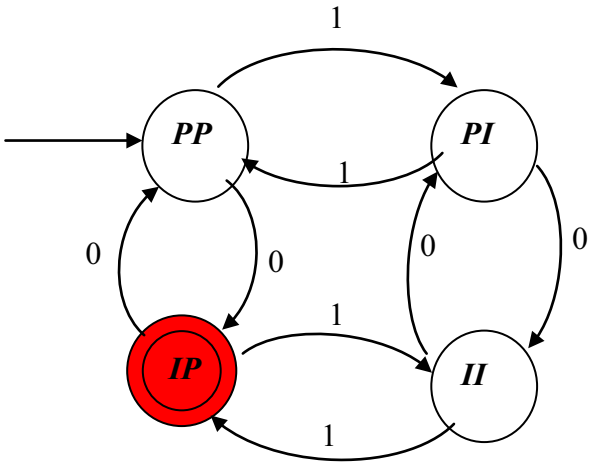


Figura 2.3.10. Grafo completo do exemplo 2.3.3.

E juntando as duas condições, ímpar-ímpar **ou** ímpar-par, obtém-se o autómato com dois estados finais:

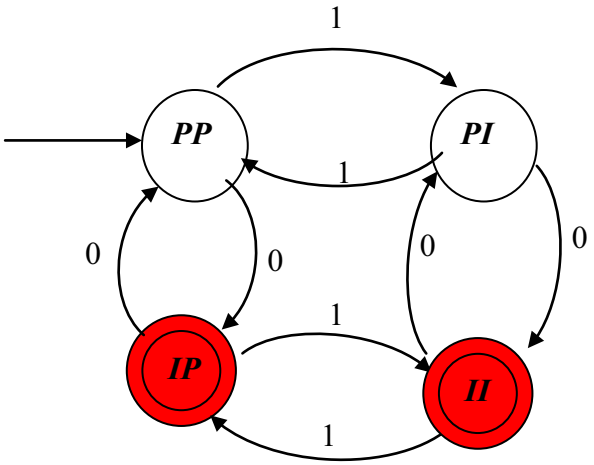


Figura 2.3.11. Grafo do DFA aceitador de *II* **ou** de *IP*

Exemplo 2.2.3. Contagem de sequências (corridas) de caracteres.

Desenhe-se um DFA que em $\Sigma=\{a,b\}$ aceite todas as cadeias que tenham corridas de três ou mais *a*'s. Uma corrida de três é *aaa*. No caso *aaaa* temos uma corrida de 4.Portanto *baabbaaababab*, *bbbbbbababababaaaaabab*, pertencem à linguagem, mas *baabaababbabbbb, aabbbbbaabbabbbbaab,* não pertencem.

A questão aqui é mais complicada do que a simples paridade. O autómato tem que ser capaz de contar uma corrida de três *a*'s : zero, um, dois, três. Como pode contar ? Tem que ter memória para isso (contar tem memória implícita). A memória de um DFA só existe nos estados, e por isso para contar até três são precisos quatro estados a que chamaremos 0, 1, 2 e 3. O número do estado é o número de a's seguidos contados até aí.

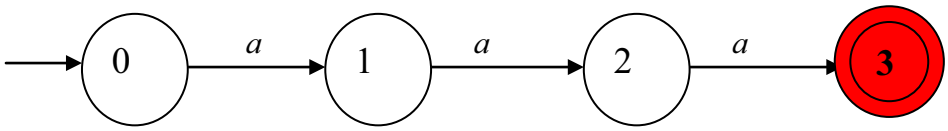


Figura 2.3.12. Como contar 3 *a*'s.

Serão estes quatro estados suficientes? É o que vamos ver.

Se depois de 3 *a*'s vier um quarto, a cadeia continua aceite. De facto depois da corrida de três *a*'s a cadeia é aceite independentemente do que acontecer a partir daí, venham mais *a*'s ou mais *b*'s. Por isso pode-se colocar uma aresta do estado 3 para si mesmo com *a* ou com *b*.

Por outro lado, se a cadeia começa por *b*, *bb*, *bbb*, o autómato conta sempre zero *a*'s e por isso existe uma aresta de 0 para 0 com *b*. Temos para já a Fig. 2.3.13.

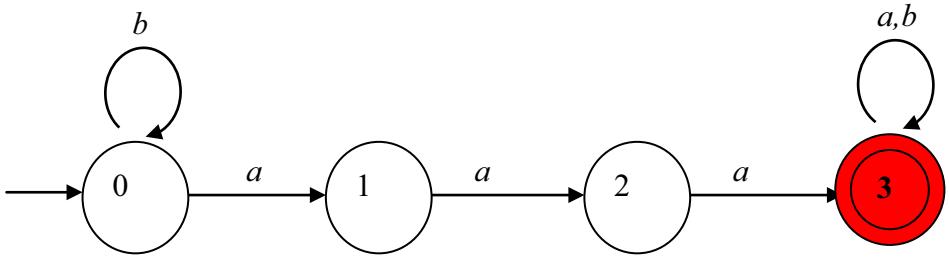


Figura 2.3.13.

E o autómato já vai tomando forma.

E se aparece *bab*?

Interessam-nos as corridas de *a*'s. Se aparece um *a* temos uma corrida de um *a*, mas se de seguida aparece um *b* ele anula a corrida, obrigando ao reinício da contagem. Quer isso dizer que do estado 1 se volta ao estado 0 se aparecer um *b* na entrada.

Se aparece *baab* este último *b* obriga de igual modo ao reinício da contagem. Teremos portanto completo o grafo do autómato na Fig. 2.3.14.

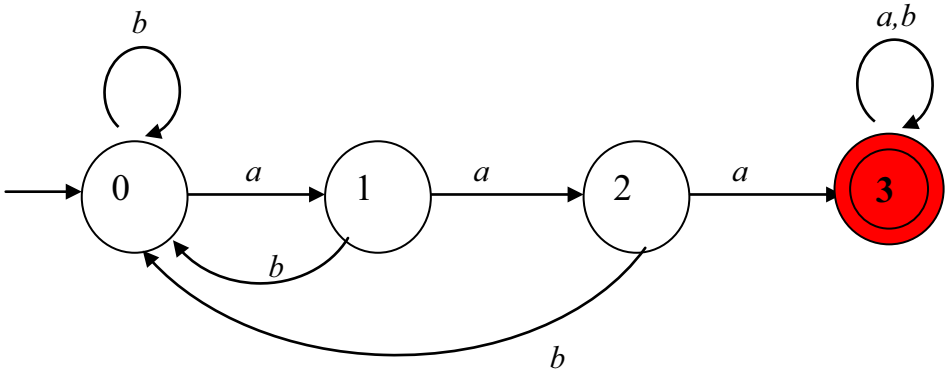


Figura 2.3.14. DFA que conta corridas de 3 ou mais *a*'s.

Fica o leitor desafiado a encontrar uma qualquer cadeia com uma corrida de três *a*'s que não seja aceite por este autómato.

Se em vez de uma corrida de três tivéssemos uma corrida de quatro ou cinco, o autómato teria uma forma semelhante, com cinco ou seis estados, respectivamente.

Se em vez de corridas de *a*'s quiséssemos corridas de *b,* bastaria trocar os *a*'s pelos *b*'s e os *b*'s pelos a's nas arestas do grafo.

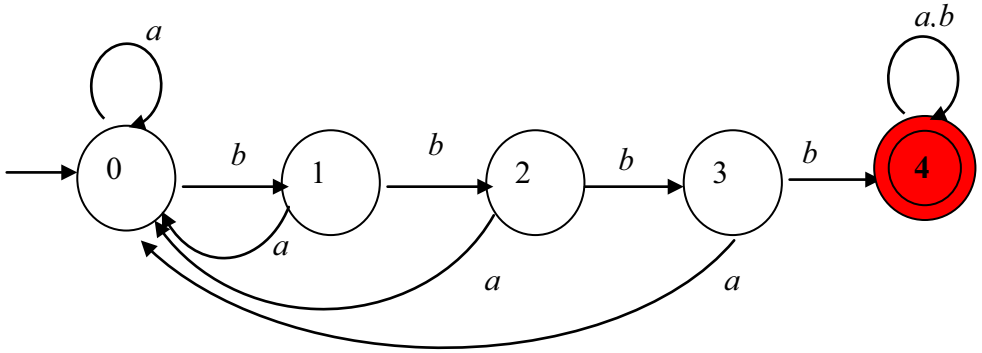


Figura 2.3.15. DFA aceitador das cadeias com corridas de quatro ou mais *b*'s.

Exemplo 2.3.4.

Desenhar o autómato que aceita no mesmo alfabeto {a, b} qualquer cadeia que tenha uma corrida de três a's ***ou*** *uma corrida de três b's.*

A situação aqui complicou-se um pouco. Vejamos o problema por partes.

Para corridas de três *a*'s, sabemos como fazer. No autómato respectivo, quando surge um *b*, reinicia-se a contagem com retorno ao estado 0. Mas agora quando aparece um *b* duas coisas têm que acontecer: por um lado a anulação da contagem de *a*'s e por outro lado o início da contagem de *b*'s.

Para corridas de três *b*'s também sabemos como fazer. E de modo análogo, quando aparece um *a* por um lado anula-se a contagem de *b*'s e por outro lado inicia-se a contagem de *a*'s.

Para iniciar, cria-se um estado inicial 0. Se aparece um *a*, inicia-se a contagem de *a*'s, se for um *b*, inicia-se a contagem de *b*'s, como na Fig.2.3.16. Os nomes dos estados foram alterados para os relacionar com a sua função: chega a 2a depois de uma corrida de dois *a*'s e chega a 2b depois de uma corrida de dois *b*'s.

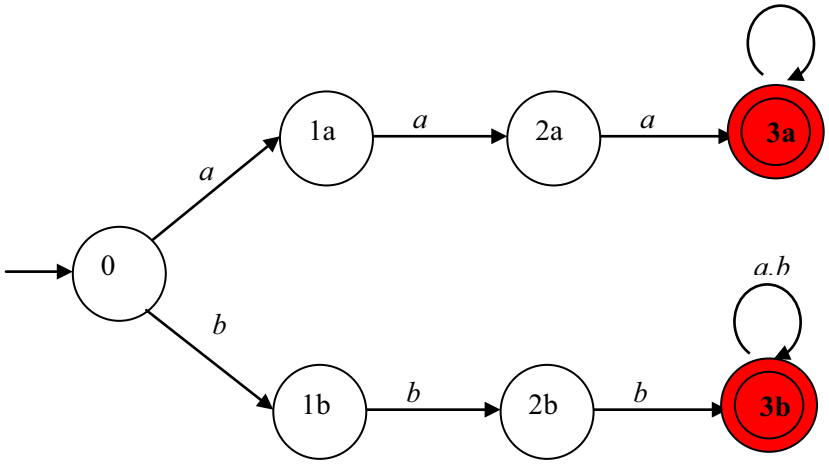


Figura 2.3.16. A caminho da solução do exemplo 2.3.4

Agora vamos à corrida de *a*'s e quando aparece lá um *b* transita-se para o estado correspondente a um *b*, ou seja, 1b.

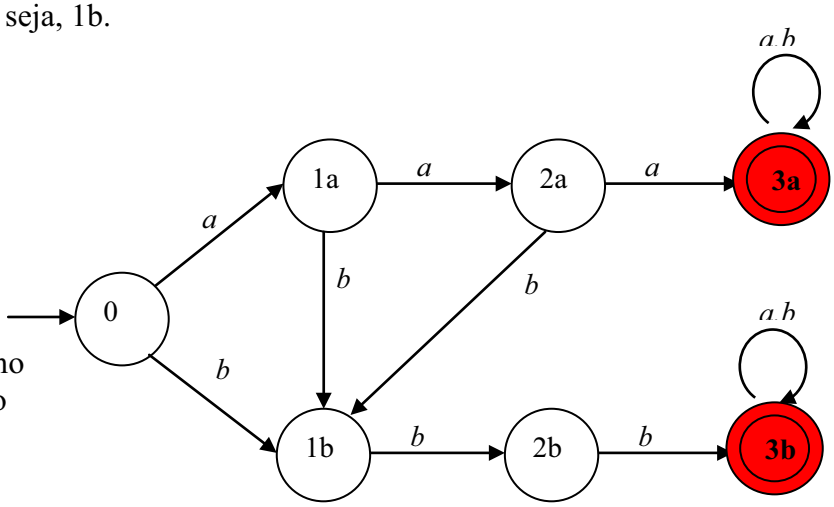


Figura 2.3.17. A caminho da solução do exemplo 2.3.4

Finalmente, na corrida de *b*'s procede-se de modo semelhante quando aparece um *a* e obtém-se o autómato da Fig. 2.3.18 que completa a resolução do exemplo.

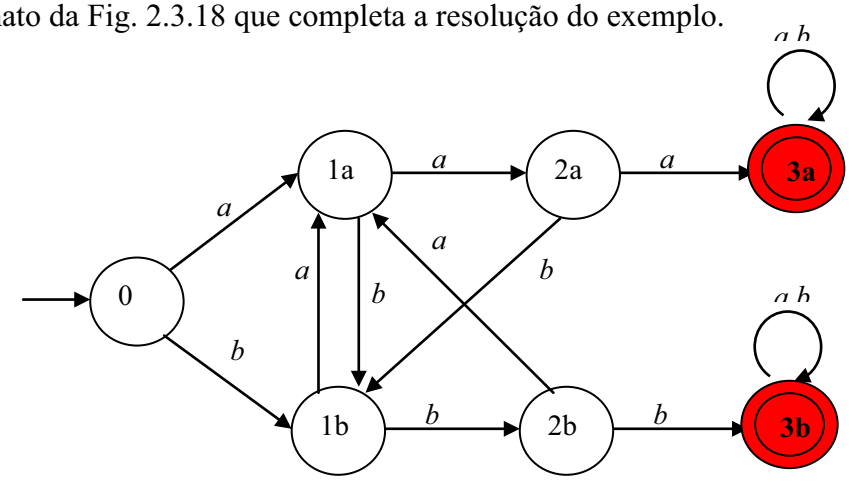


Figura 2.3. 18. Autómato completo do exemplo 2.3.4: aceita corridas de três ou mais *a*´s **ou** corridas de três ou mais b´s

Exemplo 2.2.4 Detecção de uma sub cadeia de caracteres

Pretende-se um autómato que aceite todas as cadeias no alfabeto {0,1} que contenham a sub cadeia **001**.

Por exemplo 1**001**0, 01**00**10, 1010101010**00**1111 pertencem a esta linguagem; mas 11010100 e 0101010 não pertencem.

A primeira tarefa é determinar quais os estados que o autómato deve ter. Ele tem que reconhecer 001. Pode ainda não ter encontrado nenhum dos caracteres da sequência, ou pode já ter encontrado um, dois ou os três. São por isso necessários pelo menos quatro estados. Chamem-se "Nada", "0", "00", "001". O "Nada" será o estado inicial.

Iniciado, se lê um 1 isso não lhe interessa e deixa-se ficar. Se lê um 0, então pode ser o primeiro da sequência procurada e por isso transita para o estado de já ter lido um 0; o estado "0". Se de seguida lhe aparece um outro 0, está no bom caminho e transita para o estado "00". Mas se em vez do 0 lhe aparece um 1, então não leu um número suficiente de zeros e tem que reiniciar o escrutínio, voltando assim ao estado inicial. Estamos na Fig. 2.3.19

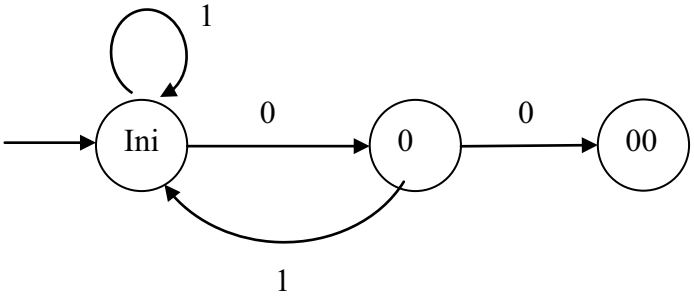


Figura 2.3.19.Início da resolução do exemplo 2.2.4.

Estando em "00" se aparece 1 então apareceu 001 e a cadeia deve ser desde já aceite, independentemente do que acontecer depois, transitando por isso para o estado aceitador "001" e não saindo de lá mais. Este último requisito alcança-se colocando uma aresta de "001" para si mesmo, seja com 0 seja com 1.

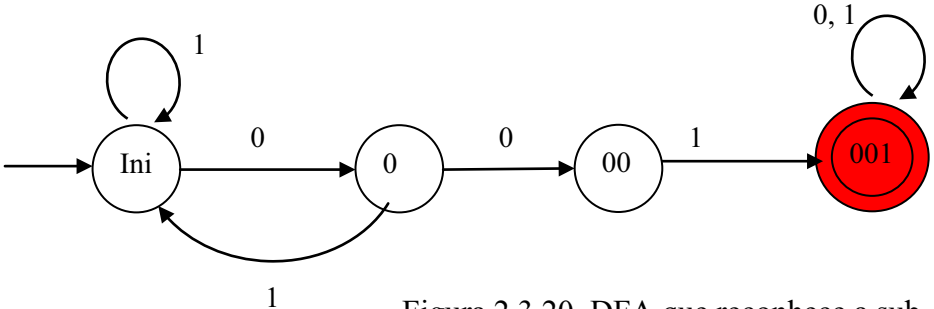


Figura 2.3.20. DFA que reconhece a sub-cadeia 001.

Contagem em módulo *n*.

A contagem em módulo *n* é uma contagem de números inteiros cíclica que se inicia em zero e volta a zero *n* passos à frente.

Por exemplo a contagem módulo 3 é 0, 1, 2, 0, 1, 2, …. A tabela seguinte ilustra a contagem em módulo para números inteiros positivos e negativos.

Tabela 2.3.1 . Contagem em módulo.

N	Contagem dos números inteiros positivos e negativos																			
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	**0**	1	2	3	4	5	6	7	8	9	10

Módulo																					
	2	0	1	0	1	0	1	0	1	0	1	**0**	1	0	1	0	1	0	1	0	1	0
	5	0	1	2	3	4	0	1	2	3	4	**0**	1	2	3	4	0	1	2	3	4	0
	6	2	3	4	5	0	1	2	3	4	5	**0**	1	2	3	4	5	0	1	2	3	4
	7	4	5	6	0	1	2	3	4	5	6	**0**	1	2	3	4	5	6	0	1	2	3
	10	0	1	2	3	4	5	6	7	8	9	**0**	1	2	3	4	5	6	7	8	9	0
	12	2	3	4	5	6	7	8	9	10	11	**0**	1	2	3	4	5	6	7	8	9	10

Exemplo 2.2.5. Desenhar um autómato que seja capaz de contar em módulo 2.

Sem perda de generalidade consideraremos apenas os números positivos. Os números a contar são codificados em binário, $\Sigma=\{0, 1\}$, ou seja, teremos a Tabela 2.3.2.

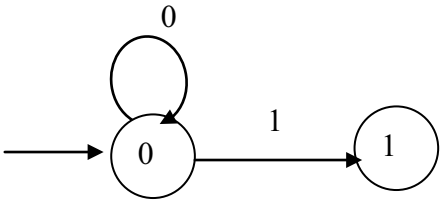
Tabela 2.3.2 Contagem em módulo 2

N	0	1	2	3	4	5	6	7	8	9	10
Código	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	10010
Mod2	0	1	0	1	0	1	0	1	0	1	0

A contagem em módulo dois tem dois resultados possíveis: 0 ou 1. Por isso o autómato só necessita de dois estados, o 0 (para contar 0) e o 1 (para contar 1). O número a contar está codificado em binário, iniciando-se a sua leitura da esquerda para a direita: o DFA lê, por exemplo, 10101111 pela ordem 1-0-1-0-1-1-1-1. Isto quer dizer que o LSB (*Least Significant Bit*) é o lido mais à direita, e só é conhecido no fim da leitura. O DFA conta com qualquer número de bits: só quando a cadeia acaba se conhece o seu tamanho.

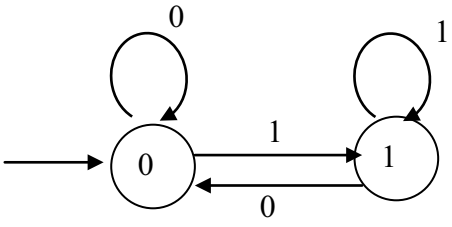
Inicialmente está no estado 0. Se aparece um 0, mantém-se no estado 0; se aparece um 1, conta 1 e por isso vai para o estado 1.

Figura 2.3.21. Início da construção do DFA contador em módulo 2



Estando no estado 1, se aparece um 0 (leu até agora 10, modulo 2 dá 0), transita para o estado 0. Se aparece um 1 (leu até agora 11=3, modulo 2 dá 1) mantém-se no estado 1.

Figura 2.3.22. Continuação da construção do DFA contador em módulo 2



Estando no estado 0 depois de ler 10: se aparece 1 (101=5, mod2(5)=1) vai para o estado 1; se aparece 0 (100=4, mod2(4)=0), mantém-se no estado 0.

Estando no estado 1 depois de ler 11: se aparece 0 (110) vai para o estado 0), se aparece 1 (111) mantém-se no estado 1.

Para que o DFA esteja completo, falta definir o estado aceitador, que depende da linguagem especificada:

- se forem as cadeias que em módulo 2 dêem zero, o estado final será o estado 0

- se forem as cadeias que em módulo 2 dêem um, o estado final será o estado 1. Neste caso teremos a Fig. 2.3.23.

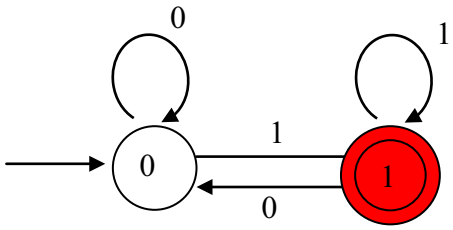


Figura 2.3.23. DFA da linguagem $L(M)=\{w \in \{0,1\}^* : \text{mod2}(w)=1\}$

Exemplo 2.2.6. Desenhar o DFA capaz de contar os números inteiros (não negativos) correctamente em **módulo 4**.

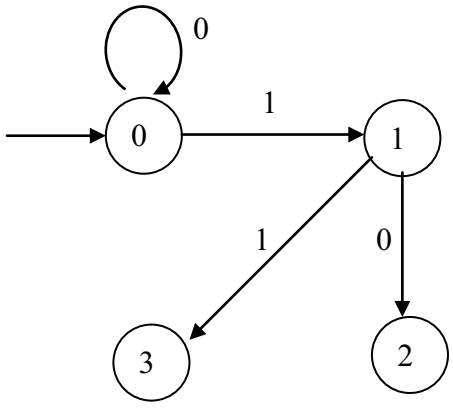
A contagem módulo 4 tem quatro valores possíveis: 0, 1, 2, 3. Quatro estados serão em princípio suficientes. O nome do estado é o valor da contagem que lhe corresponde. A leitura da cadeia é feita da esquerda para a direita.

No início temos o estado inicial 0.

Aparece 0, mantém-se no estado 0; aparece 1, vai para o estado 1.

Estando no estado 1 (depois de ler 1): aparece 0 (10) vai para o estado 2, aparece 1 (11) vai para o estado 3.

Figura 2.3.24. Construção do DFA contador em módulo 4.



Estando no estado 2 (depois de ler 10): aparece 0 (100) vai para o estado 0, aparece 1 (101=5, mod4(5)=1) vai para o estado 1.

Estando no estado 3 (depois de ler 11): aparece 0 (110=6, mod4(6)=2) vai para o estado 2, aparece 1 (111=7, mod4(7)=3) vai para o estado 3.

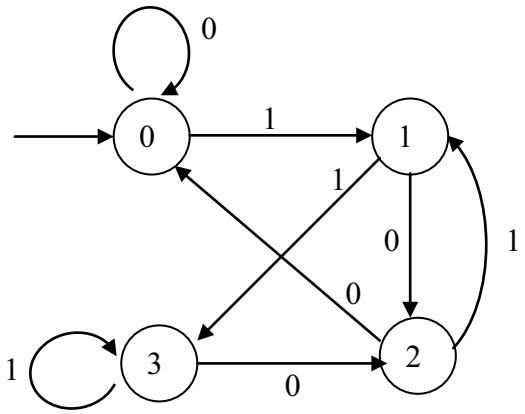


Figura 2.3.25. DFA contador em módulo 4.

Se continuarmos veremos que o autómato não se engana. O estado final será determinado pela linguagem requerida. Se for por exemplo 2, ela será composta pelas cadeias (binárias, em que o LSB é o primeiro à direita) que valem 2 em módulo 4.

É curioso verificarmos esta periodicidade, quando a cadeia binária é lida da direita para a esquerda. E se fosse lida da esquerda para a direita ?

Exemplo 2.2.7. **Contagem módulo** 7

Neste caso precisaremos de um autómato com 7 estados 0,1,2,3,4,5,6,7, sendo 0 o estado inicial.

Estando em 0: se aparece 0 mantém-se em zero, se aparece 1 vai para 1.

Estando em 1 (lido 1): com 0 (10) vai para 2, com 1 (11) vai para 3.

Estando em 2 (lido 10): com 0 (100) vai para 4, com 1 (101) vai para 5.

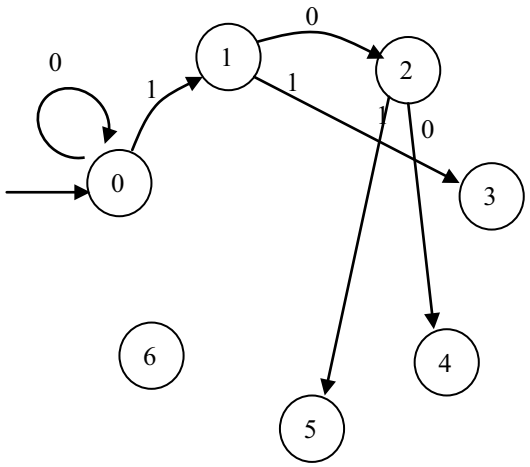


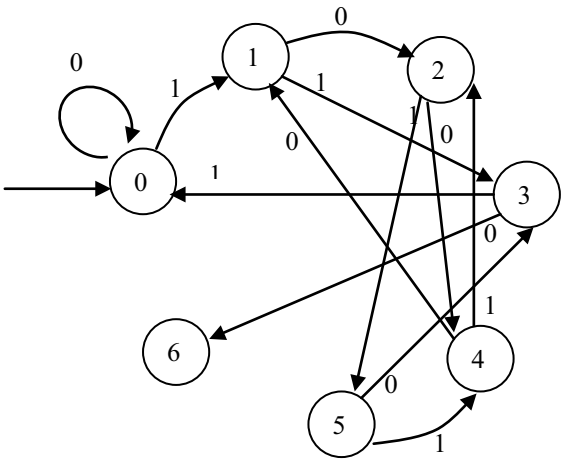
Figura 2.3.26. Construção do DFA contador em módulo 7.

Continuando,

Estando em 3 (lido 11): com 0 (110) vai para 6, com 1 (111) vai para 0.

Estando em 4 (lido 100): com 0 (1000=8, 8mod7=1) vai para 1, com 1 (1001=9, 9mod7=2) vai para 2.

Estando em 5 (lido 101): com 0 (1010=10, 10mod7=3) vai para 3, com 1 (1011=11, 11mod7=4) vai para 4.



Falta apenas definir o que acontece no estado 6.

Estando em 6 (lido 110): com 0 (1100=12, 12mod7=5) vai para 5, com 1 (1101=13, 13mod7=6) vai para 6.

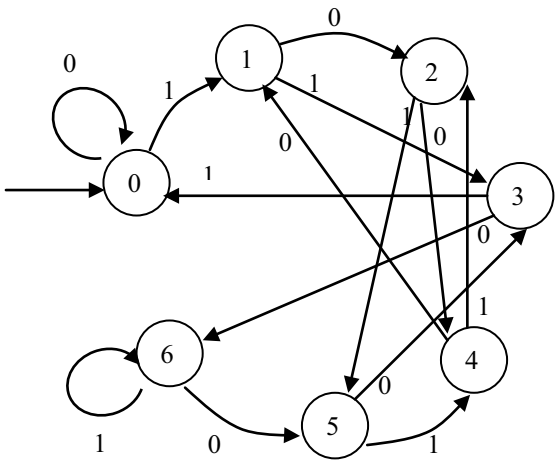


Figura 2.3.27. Continuação da construção do DFA contador em módulo 7.

Testemos o DFA para :

1110101 =64+32+16+4+1=117 117mod7=5; dá certo.

1010101011=512+128+32+8+2+1=683, 683mod7=4; confere.

Em resumo, numa boa estratégia de projecto de um DFA:

- A primeira e decisiva operação é determinar quanto estados terá o DFA e qual a função de cada um.

- A etiqueta que atribuímos a cada estado deve exprimir a função do estado.

- Depois colocamo-nos dentro de cada estado e imaginamo-nos a ler a cadeia de entrada e a imitar o DFA.

2.4. Linguagens regulares

Qualquer DFA reconhece uma linguagem. Tal afirmação é fácil de provar: uma linguagem é um conjunto de cadeias. Um DFA aceita sempre alguma cadeia, mesmo que seja a cadeia nula, como é o caso do autómato seguinte, em $\Sigma=\{a,b\}$:

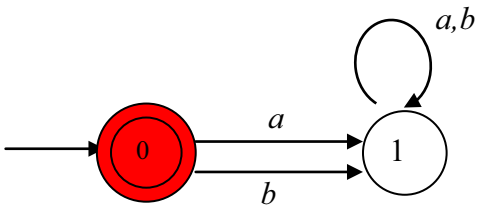


Figura 2.4.1. DFA aceitador da linguagem $L=\{\lambda\}$, só com a cadeia vazia.

E dada uma linguagem qualquer, definida por um conjunto de cadeias, existirá sempre um DFA, autómato finito determinístico, que a reconheça ? Nem sempre.

Há uma família de linguagens em que tal acontece sempre: as linguagens **regulares**.

Uma linguagem L diz-se regular **se e só se** existir um DFA M que a reconheça, ou seja,

$$L=L(M)$$

A família das linguagens regulares é composta por todas as linguagens que são aceites por algum DFA.

E como se pode provar que uma linguagem é regular? Por prova construtiva: se formos capazes de construir um autómato que a reconheça, então ela é regular. E se não formos capazes? Aqui é que a porca torce o rabo, pois pode ser por falta de perícia nossa. Mas também pode ser porque é impossível (o que acontece se a linguagem for não regular).

Exercícios de construção de DFAs.

1. Construir um DFA que aceite a linguagem L em $\Sigma = \{0, 1\}$ tal que

(a) L contém apenas "010" e "1".

(b) L é o conjunto de todas as cadeias que terminam com "00".

(c) L é o conjunto de todas as cadeias sem "1"s consecutivos nem "0"s consecutivos.

2. Desenhar o DFA que aceita todas as cadeias em {0,1}, com excepção das que contêm 010:

0111101100 , 0000, 1011000 $\in$ L(M)

1<u>0101</u>000, 111100<u>010</u> $\notin$ L(M)

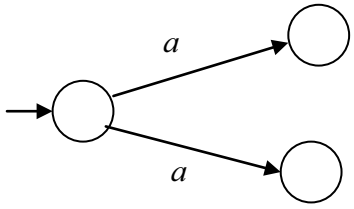
3. Desenhar o grafo do DFA que aceite as cadeias binárias que valem 5 em módulo 6.

2.5. Autómatos finitos não-determinísticos

Um DFA sabe sempre o que fazer: o seu caminho está completamente definido no seu grafo, nunca há qualquer situação em que o DFA exite, por poder escolher entre dois caminhos alternativos, ou por não saber o que fazer.

Nem todos os autómatos assim são. Poderemos imaginar uma situação em que a partir de um estado sejam possíveis zero, uma ou mais transições com o mesmo símbolo do alfabeto, isto é, em que há caminhos alternativos (para a mesma situação na entrada). Quando isso acontece o autómato deixa de ser determinístico. Diz-se autómato não-determinístico ou indeterminístico.

Figura 2.5.1 Parte de um autómato finito com dois caminhos alternativos para a mesma leitura na entrada.



Havendo caminhos alternativos pode haver, entre o estado inicial e o estado aceitador, zero, um ou vários caminhos. Pode aceitar uma cadeia seguindo um dado caminho e não aceitar a mesma cadeia seguindo um outro caminho alternativo.

Como definir neste caso a condição de aceitação de uma cadeia?

Do modo seguinte: uma cadeia de entrada é aceite se houver para ela (pelo menos) um caminho que leve a um estado aceitador quando ela é toda lida.

Definição 2.5.1: **Um aceitador não-determinístico** (NFA–*Nondeterministic Finite Accepter*) é definido pelo quinteto

$$M = (Q, \Sigma, \delta, q_0, F)$$

em que a função de transição δ é definida por

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$$

Os restantes membros do quinteto definem-se do mesmo modo que no caso do DFA. Lendo com atenção a definição constatam-se as seguintes diferenças em relação ao DFA (definição 2.2.1):

1- O contradomínio de δ é a potência 2^Q do conjunto Q, e não Q, precisamente porque o resultado de δ pode ser no limite um qualquer subconjunto de Q. A potência 2^Q de um conjunto é composta por todos os seus subconjuntos possíveis, incluindo o vazio.

2- O conjunto $\delta(q_i, a)$ pode ser vazio, significando que não há qualquer transição nesta situação. No grafo não há uma aresta partindo de q_i com o símbolo a. Não é obrigatório que exista uma aresta a partir de q_i para cada um dos símbolos do alfabeto.

3- λ também pode ser argumento de δ, ou seja, pode dar-se uma transição sem consumir um símbolo de entrada (o mecanismo de leitura pode ficar parado em alguns movimentos). No grafo do NFA pode existir uma ou mais arestas com o símbolo λ.

Os NFAs podem representar-se por grafos tal como o DFA. Os nós são os estados e as arestas as transições. Por exemplo, a transição $\delta(q_1, a) = \{q_2, q_3\}$ só pode existir num NFA, dado que há dois caminhos alternativos: de q_1 lendo a ou se transita para q_2 ou se transita para q_3. Com um grafo será

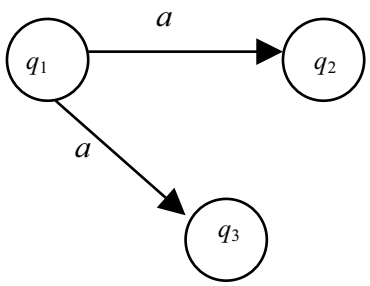


Figura 2.5.2. Grafo da transição $\delta(q_1, a) = \{q_2, q_3\}$

Num NFA pode dar-se uma transição com o carácter λ. Este facto parece estranho na medida em que aceitar o nada numa primeira análise parece até contrariar a própria noção de aceitação. Mas veremos a utilidade deste facto.

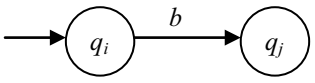
Consequências da possibilidade de transição com λ

Há algumas consequências da possibilidade de transição com o λ que convém aqui referir.

1º - de um qualquer estado para ele mesmo pode-se sempre transitar com λ.

Suponhamos que estamos no estado q_i depois de ler o prefixo *aba* da cadeia *ababb*.

Figura 2.5.3. Grafo (NFA) da transição $\delta(q_i, b) = q_j$



Lendo agora o carácter *b* o autómato transita par o estado q_j.

Mas repare-se que a cadeia *ababb* é a mesma que *abaλbb* ou que *abaλλbb* ou que *abaλλλbb* e por aí além. Como representar esse facto no grafo? Simplesmente colocando uma transição λ do estado q_i para ele mesmo.

O mesmo irá depois acontecer em q_j . Por isso o grafo completo terá a forma da Fig. 2.5.4.

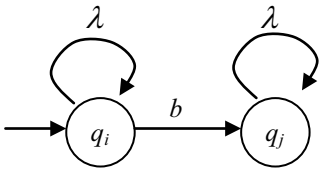


Figura 2.5.4. Grafo (NFA) da transição $\delta(q_i, b) = q_j$, com explicitação (em relação ao da Fig. 2.5.3) das transições com λ.

Em qualquer NFA existe essa característica: há uma transição com λ de um qualquer estado para ele mesmo. Por ser sempre verdade, geralmente não se desenha, considerando-se que a transição está lá implicitamente.

Nunca esquecer no entanto que no caso de um DFA não há transições-λ.

2º Quando a transição-λ for uma alternativa a uma transição com um outro carácter, o autómato escolhe "comer" um carácter ou não. Suponhamos, por exemplo, que um NFA chega a um estado q_1, da figura seguinte, e se lhe apresenta de seguida o carácter *b*. O NFA tem dois caminhos alternativos: ou "consome" *b* e vai para q_2, ou não consome *b* (consome λ que está implícito na cadeia *aabab*, *aaλbab*) e vai para q_3.

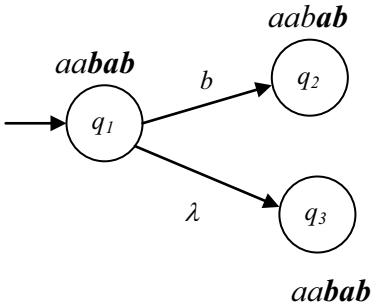


Figura 2.5.5.

Havendo (por vezes tantas) escolhas, como se pode saber se uma cadeia é aceite pelo DFA? Se ele escolhe um caminho e não consegue chegar ao estado final conclui-se que não aceita a cadeia? E se ele tivesse escolhido outro, aceitaria ou não? Teria que tentar todos os caminhos alternativos possíveis. Se encontrasse um que o levasse a um estado aceitador então aceitaria a cadeia.

A diferença entre uma computação determinística (podendo terminar ou não num estado aceitador) e uma computação não determinística em que existe um caminho aceitador é ilustrada pela Figura 2.5.6 (de Sipser).

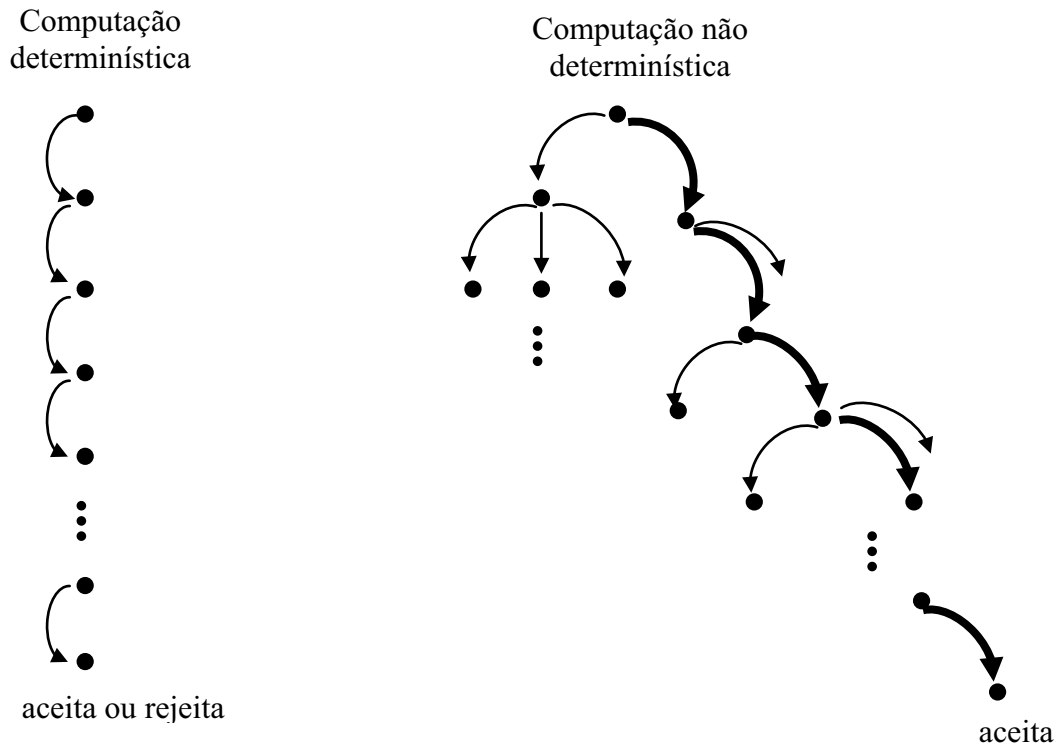


Figura 2.5.6. Uma computação determinística cria um caminho, enquanto que uma computação não-determinística cria uma árvore.

Na computação determinística nunca há escolhas. A cadeia lida ou é aceite ou não, conforme o último estado seja aceitador ou não. Na computação não determinística cria-se uma árvore, com sucessivas ramificações sempre que haja uma escolha possível. Se um dos caminhos de uma computação terminar num estado aceitador, o NFA aceita a cadeia. Caso **nenhum** caminho termine num estado aceitador, a cadeia é rejeitada.

Uma cadeia é aceite pela NFA se houver **alguma** sequência de movimentos possíveis que coloquem o autómato num estado final (aceitador) no fim da cadeia. Caso contrário é rejeitado.

Há uma outra forma de concebermos uma computação não determinística. Sempre que há uma escolha possível, o NFA replica-se uma vez por cada escolha, de modo que o número de configurações vai aumentado em cada ramificação da árvore supra. Cada configuração está

no estado consequente da escolha feita e na sua entrada estão aos caracteres ainda não lidos. Se um dos autómatos replicados está numa configuração tal que não pode transitar para qualquer estado (seja com λ seja com um outro carácter), e se a leitura ainda não chegou ao fim, então esse autómato morre. Se no fim da computação houver algum autómato sobrevivente e se pelo menos um deles estiver num estado aceitador, a cadeia é aceite.

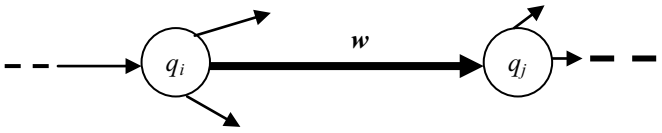
Função de transição estendida δ^*

Tal como no caso do DFA, também num NFA se pode introduzir o conceito de função de transição estendida δ^*. Ela resulta da leitura de uma cadeia w que é o seu segundo argumento (em vez de um carácter),

$$\delta^* (q_i, w) = Q_j$$

sendo Q_j o **conjunto** de todos os estados possíveis do autómato que podem ser alcançados a partir de q_i e pela leitura de w. Isto é $\delta^* (q_i, w)$ contém o estado q_j se e só se existir um caminho no grafo de transições desde q_i até q_j com etiqueta w, para todos os $q_i, q_j \in Q$ e $w \in \Sigma^*$, como na Fig. 2.5.7.

Figura 2.5.7.



Definição 2.5.2. Linguagem aceite por um NFA

A linguagem L aceite pelo NFA $M = (Q, \Sigma, \delta, q_0, F)$ é definida como o conjunto de todas as cadeias w que levem o autómato a um estado final aceitador, ou seja,

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \cap F \neq \varnothing\}$$

i.e., é composta por todas as cadeias w para as quais existe **pelo menos um** caminho etiquetado w desde o vértice inicial até **algum** vértice final.

Exemplo 2.5.1 (Linz)

O autómato da Figura 2.5.8 é não determinístico porque:

- de q_1 partem duas arestas etiquetadas por 0
- de q_0 parte uma aresta etiquetada por λ
- de q_2 não é definida qualquer transição, $\delta(q_2, 0) = \varnothing$
- existe sempre a transição $\delta(q_i, \lambda) = \{ q_i\}$, para todo o i, mesmo se não desenhada.

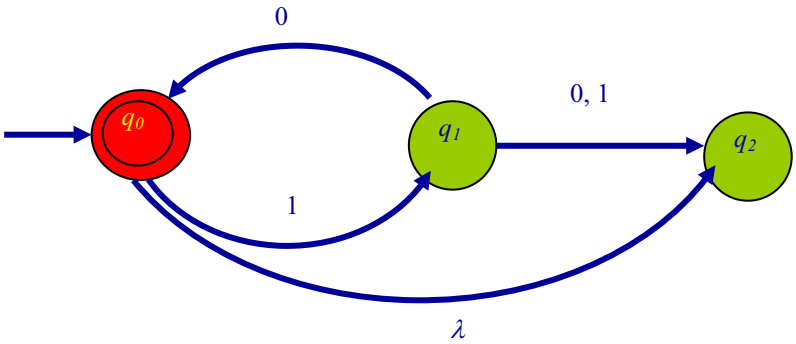


Figura 2.5.8. Um aceitador não-determinístico

Se escrevermos todas as transições possíveis, teremos:

$\delta(q_0, 1) = \{q_1\}$

$\delta(q_0, 0) = \varnothing$

$\delta(q_1, 0) = \{q_0, q_2\}$

$\delta(q_1, 1) = \{q_2\}$

$\delta(q_1, \lambda) = \{q_1\}$

$\delta(q_0, \lambda) = \{ q_0, q_2\}$

$\delta(q_2, 0) = \varnothing$

$\delta(q_2, 1) = \varnothing$

$\delta(q_2, \lambda) = \{q_2\}$

O autómato aceita:

10, porque dos dois estados alcançados um é final.

1010, 101010, …

Não aceita : 1, 11, 01, …

A sua linguagem é composta palas cadeias de potências de 10, ou seja,

$$L(M) = \{ (10)^n : n \geq 0 \ \}$$

E se aparece a cadeia 110 ?

Depois de ler 11 chega a q_2. Como não está definida a transição $\delta(q_2, 0)$, obtém-se uma **configuração morta** (*dead configuration*), em que se transita para o estado vazio, $\varnothing$. Mais formalmente

$$\delta^*(q_0, 110)=\varnothing.$$

Como não se pode atingir um estado final processando 110, esta cadeia não é aceite pelo NFA.

Exemplo 2.5.2.: (Linz).

Considere-se o autómato da Fig. 2.5.9, com o alfabeto $\Sigma=\{a\}$. Ele é não-determinístico porque (i) no estado inicial há uma escolha possível com o carácter a e (ii) não estão definidas todas as transições possíveis.

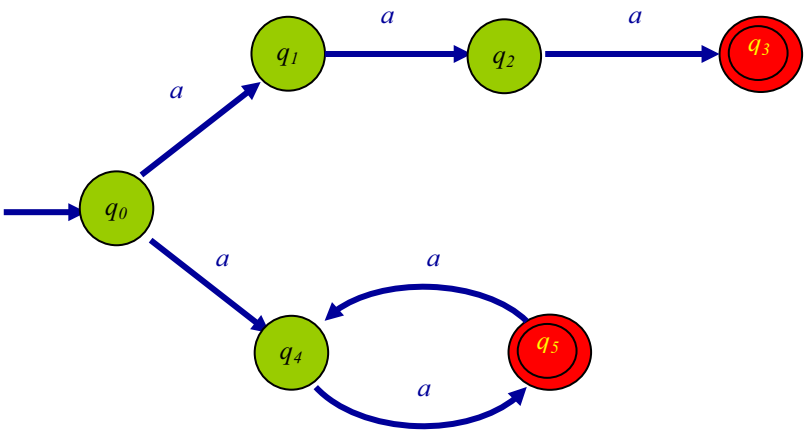


Figura 2.5.9. Exemplo 2.5.2

Qual é a linguagem aceite pelo autómato?

Pela parte de cima aceita a cadeia *aaa* e só essa.

Pela parte de baixo aceita *aa, aaaa, aaaaaa, ...,* a^{2n}, $n \geq 1$

O conjunto das duas, ou seja a união das duas, dá a linguagem aceite pelo autómato

$$L(M) = \{a^3\} \cup \{a^{2n} : n \geq 1\}$$

Se o NFA pode escolher, na prática com o se constrói? Para que servem?

Os NFA são construções matemáticas que não passam disso: não se pode construir um NFA para uma aplicação prática. Pense-se por exemplo o que aconteceria a um autómato que controlasse uma porta automática se ele não fosse determinístico …

No entanto têm uma grande utilidade em Teoria da Computação. Usam-se como um ferramenta matemática muito útil para

- modelizar algoritmos em que tem que se fazer escolhas (*search-and-backtracking*, por exemplo)

- definir linguagens compostas por conjuntos bastante diferentes (como no exemplo anterior $\{a^3\} \cup \{a^{2n} : n \geq 1\}$.

- descrever de forma simples e concisa linguagens complicadas. Na definição de uma gramática pode existir um elemento de não-determinismo, como na produção

$$S \rightarrow aSb \mid \lambda$$

em qualquer ponto pode-se escolher a primeira ou a segunda produção. Podemos assim especificar um grande número de cadeias usando apenas duas regras. Se quisermos construir um autómato equivalente (estudaremos como num capítulo posterior) será mais fácil através de um NFA.

- construir DFAs complicados. Como veremos de seguida não há nenhuma diferença essencial entre NFA's e DFA's: dado um NFA é possível encontrar um DFA equivalente. Se quisermos construir um DFA, poderemos numa primeira etapa construir um NFA, mais fácil de desenhar, e depois obter, pela técnica que estudaremos, um DFA equivalente.

2.6. Equivalência entre autómatos (aceitadores) finitos determinísticos e não-determinísticos

Como vamos ver, a partir de um qualquer NFA com um dado alfabeto pode-se obter um DFA que aceita as mesmas cadeias e rejeita as mesmas cadeias no mesmo alfabeto. Diz-se que o DFA é equivalente ao NFA. Vejamos mais formalmente a definição de equivalência entre autómatos.

Definição 2.6.1. Aceitadores equivalentes : dois aceitadores M_1 e M_2 são equivalentes se

$$L(M_1) = L(M_2)$$

isto é, se ambos aceitam a mesma linguagem.

Existem geralmente muitos aceitadores para uma dada linguagem, e por isso qualquer DFA ou NFA tem muitos aceitadores equivalentes.

Os DFA podem-se considerar casos especiais de NFA, em que os caminhos possíveis se reduzem sempre a um. Por isso se uma linguagem é aceite por um DFA então existe um NFA que também a aceita.

E o contrário? Se uma linguagem é aceite por um NFA, existirá um DFA que a aceite?

Será que o não-determinismo eliminou esta possibilidade ?

De facto não eliminou: se uma linguagem é aceite por um NFA então existe um DFA que também a aceita.

Como encontrá-lo ? Como converter um NFA num DFA ? Como transformar uma situação de multi-escolha numa escolha única ?

Um NFA pode escolher vários caminhos. Depois de um NFA ler uma cadeia w, não se sabe precisamente em que estado está, mas sabe-se apenas que está num estado entre um conjunto Q_w de estados possíveis

$$Q_w = \{q_i, q_j, \ldots, q_k\}.$$

Em contrapartida depois de um DFA ler a mesma cadeia tem que estar num estado bem definido, q_w.

Como encontrar uma correspondência entre estas duas situações?

Pode-se conceber um truque simples: cria-se um estado no DFA equivalente ao conjunto Q_w, isto é, etiqueta-se, no DFA, um estado por Q_w. Se isso for possível temos o problema resolvido.

Mas resultará isso num autómato finito? Qual o número máximo de estados do DFA que se podem obter por este processo?

Se no NFA existem no total Q estados, o número máximo de estados que se podem obter no DFA é igual à potência de conjuntos de Q, isto é $2^{|Q|}$, e portanto finito. No caso extremo de total indeterminismo no NFA, o número de estados no DFA será $2^{|Q|}$.Se tivermos 6 estados no NFA teremos 2^6=64 estados no DFA ! Como geralmente o indeterminismo é apenas parcial, o DFA equivalente tem um número de estados substancialmente inferior.

Vejamos um exemplo.

Exemplo 2.6.1.

Construir um DFA que aceite a linguagem L composta pelas cadeias em Σ={0,1} que contenham três 0's seguidos ou três 1's seguidos , como por exemplo

000, 111, 10100010, 00111101000.

A procura directa deste DFA pode tornar-se numa operação bastante complicada, mesmo para um projectista experiente. É relativamente fácil desenhar um DFA para o caso de 000, ou para o caso de 111, mas não é assim para ambos os casos simultaneamente. No entanto usando o não determinismo a solução surge de forma bastante simples.

Primeiro, poderemos desenhar o NFA que aceita as cadeias com 000. O NFA, vendo um zero, decide uma de duas coisas: ele não é o início de 000 (e neste caso deixa-se ficar onde está), ele é o início de 000 (e então transita para o estado seguinte na corrida de três zeros). Será então:

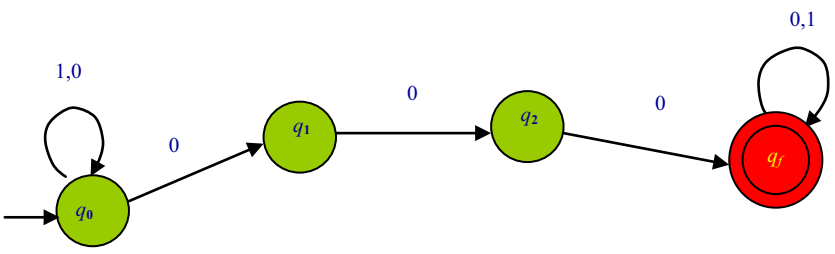


Figura 2.6.1. Primeira parte do exemplo 2.6.1

Este NFA aceita todas as cadeias que contenham 000, e só essas. De facto:

- se uma cadeia contém 000, há sempre um caminho aceitador para ela. Por exemplo no caso da cadeia 1001110110001101010 deixa-se ficar em q_0 até ao nono carácter e depois transita sucessivamente para q_1, q_2 e q_f, ficando aí até ao fim da cadeia.

- se uma cadeia não contém 000 não é possível chegar ao estado aceitador. Por exemplo para a cadeia 1100101001, se o NFA transita com algum dos zeros para o estado q_1, ele acaba por morrer porque lhe aparece um 1 em q_1 ou em q_2 e não está aí definida qualquer transição com 1.

Se o autómato não tivesse a possibilidade de ficar em q_0 com 0, avançaria sempre para q_1 quando lesse um 0 (por exemplo em 1<u>0</u>10001). Mas se a seguir aparece 1, ele entra numa configuração morta, dado que não há saída de q_1 com 1, e por isso terminaria a computação não aceitando a cadeia 10<u>1</u>0001). Ora acontece que a sequência 000 aparece mais à frente e a cadeia deveria ser aceite. O facto de poder esperar em q_0, palpitando-lhe que aquele 0 não é ainda da sequência pretendida, evita esta situação.

Segundo, desenhamos o NFA que aceita as cadeias com 111 (e só essas). Por razões análogas ele será

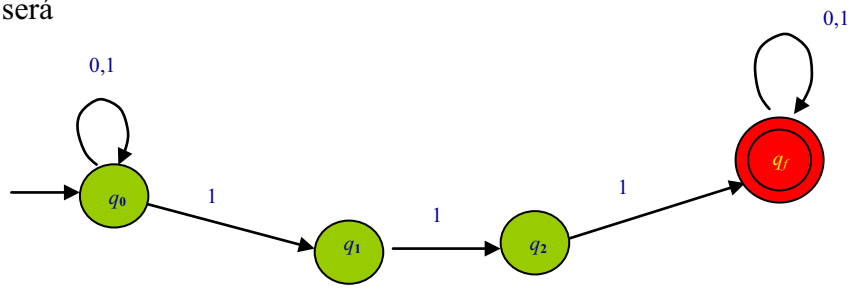


Figura 2.6.2. Segunda parte do exemplo 2.6.1

Agora, para termos a possibilidade das duas situações, basta-nos juntar os dois autómatos, a partir do estado inicial comum, obtendo-se a Fig 2.6.3. O estado final aceitador pode também ser comum (embora este facto não seja obrigatório).

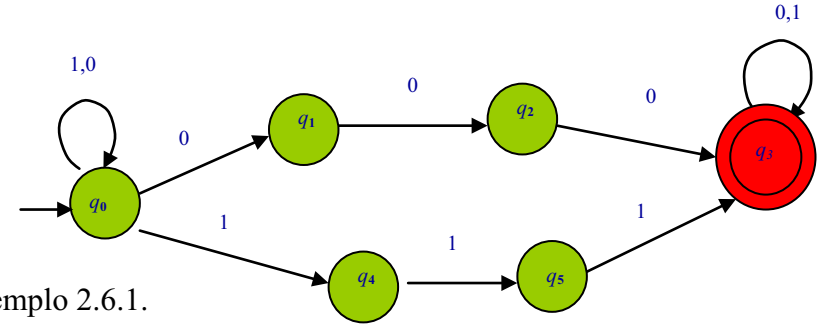


Figura 2.6.3. Exemplo 2.6.1. completo

E temos o NFA procurado.

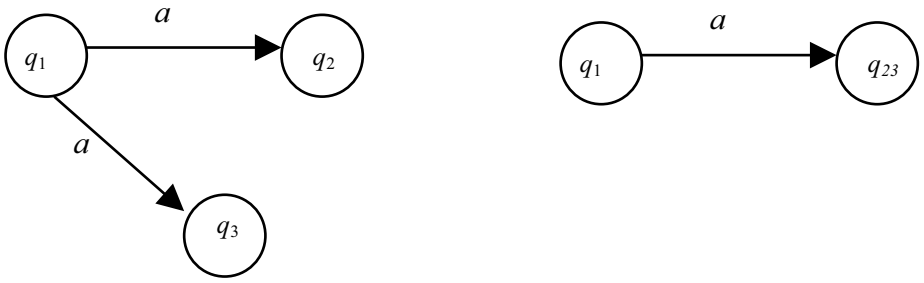
Vamos agora procurar um DFA equivalente.

Para isso construa-se a tabela de transições:

Tabela 2.6.1. Transições da Fig. 2.6.3

δ_N	0	1
→ q_0	q_0, q_1	q_0, q_4
q_1	q_2	-
q_2	q_3	-
q_3	q_3	q_3
q_4	-	q_5
q_5	-	q_3

A primeira dificuldade deriva do facto de algumas transições terem vários estados alternativos de chegada, que são por isso co-alcançáveis. O significado dos estados é definido pelo projectista. Assim sendo, nada nos impede de criarmos estados no DFA que correspondam a conjuntos de estado do NFA. E será lógico que todos os estados do NFA co-alcançáveis pela mesma transição componham um estado do DFA alcançável por essa mesma transição, como na Fig. 2.6.4.



a) no NFA b) no DFA equivalente

Figura 2.6.4. Pormenor do cálculo do DFA equivalente: dois estados q_2 e q_3 do NFA co-alcançáveis pela transição *a* produzem um só estado q_{23} no DFA equivalente.

Para definir as transições no DFA, etiquetem-se os seus estados com índices agrupando os índices dos estados do NFA co-alcançáveis pelas mesmas transições.

Na Fig. 2.6.3 de q_0 com 0, pode-se ir ou para q_0 ou para q_1, ou seja para o conjunto $\{q_0, q_1\}$ Crie-se um estado equivalente a este conjunto. Vamos chamar-lhe q_{01} para sabermos a que corresponde.

De q_0 com 1, pode-se ir ou para q_0 ou para q_4, ou seja para o conjunto $\{q_0, q_4\}$ Crie-se de igual modo um estado equivalente a este conjunto, q_{04}.

Estes estados q_{01} e q_{04} vão integrar o DFA. Temos que saber o que se passa neles quando aparece 1 ou 0. Para isso colocamo-los na primeira coluna da tabela de transições, como indicado na Tabela 2.6.2..

Tabela 2.6.2. Transições do DFA equivalente.

δ	0	1
→ q_0	$\{q_0, q_1\} = q_{01}$	$\{q_0, q_4\} = q_{04}$
q_{01}		
q_{04}		

Agora temos que imaginar o seguinte: se o DFA está em q_{01} e aparece 0, o que se passará?

Sabemos que q_{01} corresponde do conjunto $\{q_0, q_1\}$ do NFA. E portanto aquela pergunta é equivalente a: se o NFA está ou em q_0 ou em q_1 e aparece um 0, o que se passará?

Se está em q_0 pode ir para q_0 ou para q_1. Se está em q_1 pode ir para q_2. Portanto estando em $\{q_0, q_1\}$ pode ir para $\{q_0, q_1, q_2\}$. Crie-se por isso no DFA o estado q_{012}.

E se o DFA está em q_{01} e aparece 1, o que se passará? Fazendo uma análise semelhante conclui-se que estará em q_{04}. Neste caso há uma nuance que convém realçar: se está em q_1, com 1 vai para o vazio, $\varnothing$. Mas $\{\varnothing, q_0, q_4\}=\{q_0, q_4\}$.

Nesta segunda etapa surgiu o novo estado do DFA, q_{012}, que deve ser colocado na primeira coluna. q_{04} já lá estava.

Tabela 2.6.3. Mais transições do DFA equivalente

δ	0	1
→ q_0	$\{q_0, q_1\} = q_{01}$	$\{q_0, q_4\} = q_{04}$
q_{01}	$\{q_0, q_1, q_2\} = q_{012}$	$\{\varnothing, q_0, q_4\} =$
q_{04}		
q_{012}		

E agora continua-se a análise. De q_{04} com 0 para onde se vai? E com 1? E de q_{012}?

Sempre que aparece um novo estado do DFA (vê-se palas etiquetas) tem que se colocar na primeira coluna na linha seguinte ainda não ocupada.

Termina-se quando só se obtêm estados do DFA que já existem na 1ª coluna.

Tabela 2.6.4. Todas as transições do DFA equivalente

δ	0	1
→ q_0	$\{q_0, q_1\} = q_{01}$	$\{q_0, q_4\} = q_{04}$
q_{01}	$\{q_0, q_1, q_2\} = q_{012}$	$\{\varnothing, q_0, q_4\} = q_{04}$
q_{04}	q_{01}	q_{045}
q_{012}	q_{0123}	q_{04}
q_{045}	q_{01}	q_{0345}
q_{0123}	q_{0123}	q_{034}
q_{0345}	q_{013}	q_{0345}

q_{013}	q_{0123}	q_{034}
q_{034}	q_{013}	q_{0345}

Resta-nos identificar o(s) estado(s) aceitador(es) para termos o DFA completamente definido. O NFA aceita uma cadeia quando existe pelo menos uma computação que leve a um estado final.

Um autómato, seja DFA, seja NFA, pode ter vários estados finais aceitadores.

Os estados do DFA, deduzido do NFA, correspondem a conjuntos de estados do NFA. Se num desses conjuntos existir um estado aceitador (no NFA) então quer dizer que o estado correspondente do DFA também será aceitador. De facto para se chegar a esse conjunto de estados (no NFA), leu-se uma certa cadeia. Se um dos estados do conjunto é aceitador, quer dizer que depois de lida essa cadeia se chegou a um aceitador, e portanto a cadeia é aceite pelo NFA. Para que o DFA seja equivalente, tem também que a aceitar, e em consequência o novo estado criado no DFA correspondente ao conjunto de estados do DFA tem que ser aceitador.

Portanto, neste exemplo, serão aceitadores no DFA todos os estado que "contenham" o estado q_3 do NFA, ou seja, q_{0123}, q_{0345}, q_{013}, q_{034}.

A partir da tabela de transições desenha-se o grafo do DFA:

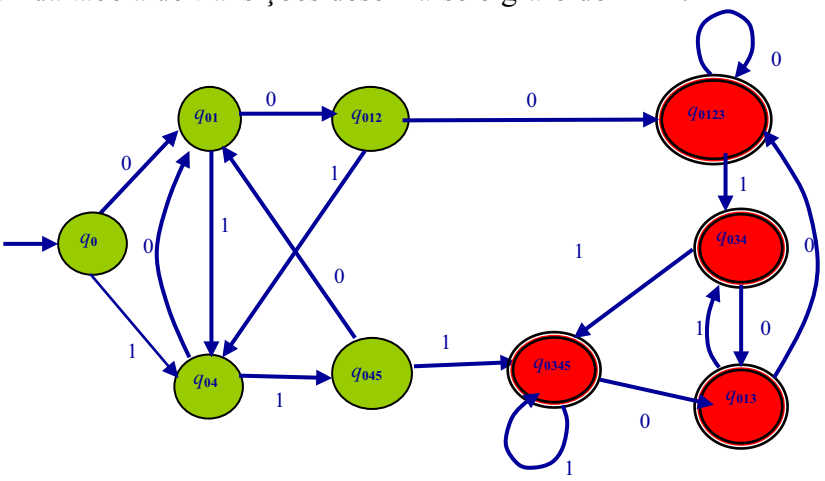


Figura 2.6.4. Autómato determinístico equivalente com vários estados finais

Depois de o autómato entrar num dos estados aceitadores, apenas pode transitar entre estados aceitadores e por isso todos os estados aceitadores se podem fundir num só, como no grafo seguinte. As etiquetas dos estados foram escolhidas de modo sugestivo, indicando como se chega ao estado (qual ou quais os últimos caracteres lidos). Excepto para o estado final, que poderíamos etiquetar por 000 ou por 111, ou por ambas.

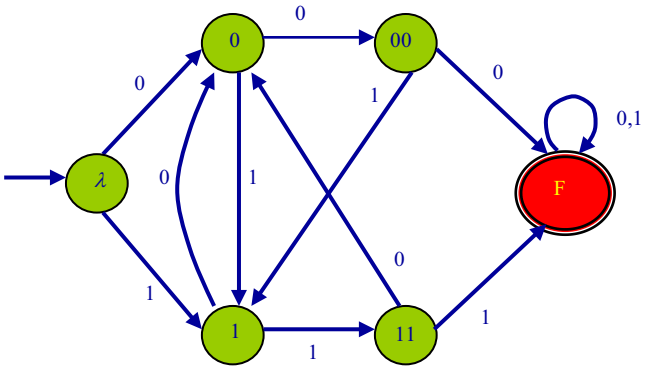


Figura 2.6.5. Simplificação do grafo da Fig. 2.6.4, com um só estado final

Olhando agora com atenção para o grafo do DFA poderemos ver que ele está de acordo com as orientações que vimos para desenhar DFAs no parágrafo 2.3. As etiquetas dos estados na Fig. 2.6.5 reflectem esse facto.

Exercício 2.6.1. Calcular o DFA equivalente ao NFA da Figura 2.6.6.

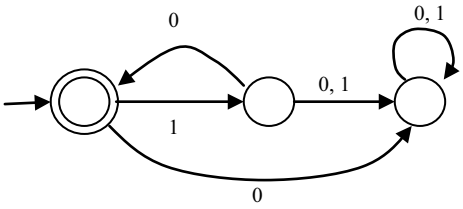


Figura 2.6.6. Exercício 2.6.1

Exemplo 2.6.2.

Desenhar o DFA que aceita em Σ={0,1} todas as cadeias que terminam em 10.

Em primeiro lugar desenha-se um NFA, por ser mais simples. Depois procura-se o DFA equivalente.

O NFA está no estado inicial até que lhe palpita que vêm aí os dois últimos caracteres e que eles serão 10. Logo,

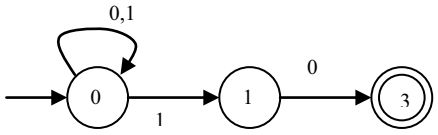


Figura 2.6.7. Exemplo 2.6.2

Vamos agora calcular a tabela de transições do DFA equivalente:

Tabela 2.6.5.Transiçoes da Fig. 2.6.7

Estado actual	0	1
0	0	01
01	03	01
03	0∅=0	0∅=0

O DFA terá 3 estados, os que se obtêm na primeira coluna da Tabela 2.6.5.

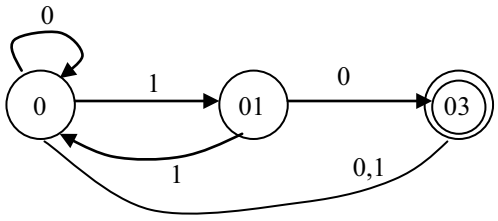


Figura 2.6.8. O DFA equivalente ao NFA da Fig. 2.6.7.

Exemplo 2.6.3.

Desenhar o DFA que aceita em Σ={0,1} todas as cadeias em que o 3º símbolo a contar do fim é 1. Por exemplo1000101 pertence, mas não 010101011.

Por muito que tenhamos treinado a nossa arte de desenhar DFAs não é fácil encontrar de imediato uma solução. No entanto no contexto não determinístico é simples: imaginemos o

NFA que está a ler a cadeia, não se move até que lhe palpita que o 1 que aí vem é o antepenúltimo carácter da cadeia e então avança para o estado aceitador. Teremos então o seguinte NFA com três estados, Fig. 2.6.9.

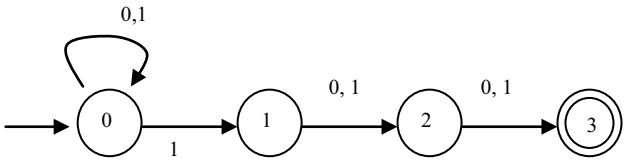


Figura 2.6.9. NFA do exemplo 2.6.3

Vamos agora calcular o DFA equivalente. Começamos pela tabela de transições, usando a técnica descrita acima, nomeadamente quanto à etiquetagem dos estados.

Tabela 2.6.6. Transições do NFA da Fig. 2.6.9.

	0	1
0	0	01
01	02	012
02	03	013
012	023	0123
03	0∅	0∅
013	02∅	012∅
023	03∅	013∅
0123	023∅	0123∅

O DFA terá 9 estados e está desenhado na Fig.2.6.10.

Todos os estados que contenham 3 na sua etiqueta serão aceitadores.

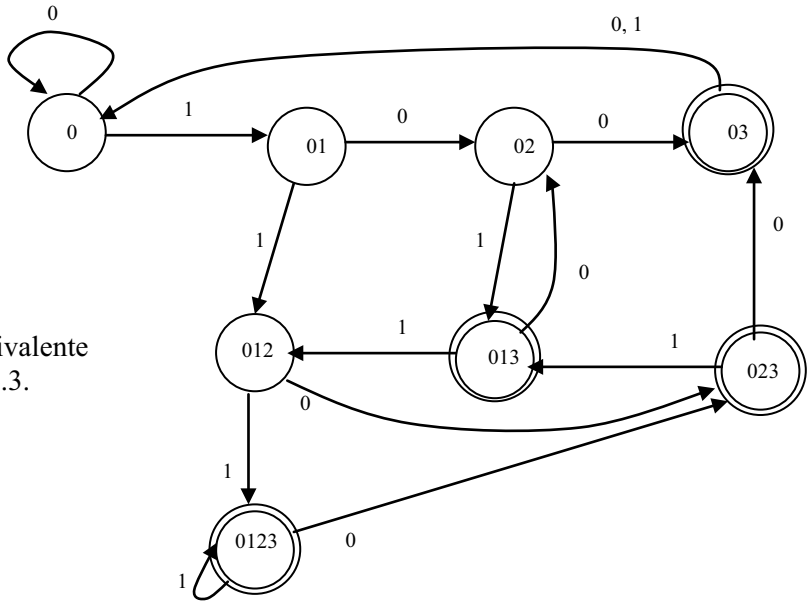


Figura 2.6.10. DFA equivalente ao NFA da Fig. 2.6.3.

Não seria fácil desenhar directamente no contexto determinístico … Compare-se a simplicidade do NFA com a do DFA. Este caso evidencia de forma clara a utilidade dos NFAs.

Será possível reduzir o número de estados aceitadores na Fig. 2.6.10 ?

Caso em que o NFA tem transições-λ

Se um NFA tem transições-λ, quando calculamos o DFA equivalente temos que ter muito cuidado para não cometermos erros.

A figura seguinte representa parte de um NFA. A partir do estado 1 com *a*, onde é possível chegar ? Vê-se imediatamente que ao estado 2. Mas existe também uma transição λ de 1 para 3, e depois com *a* transita-se para 4. Portanto partindo de 1 com um simples *a,* que é o mesmo que λa, pode-se chegar a 4. Mais ainda, de 4 para 5 transita-se com λ. Portanto de 1 até 5 também se pode ir com $\lambda a \lambda = a$. Por isso a partir de 1 com *a* chega-se a um elemento do conjunto {2,3,4,5} e portanto na tabela de transição do DFA equivalente temos que criar o estado q_{2345}.

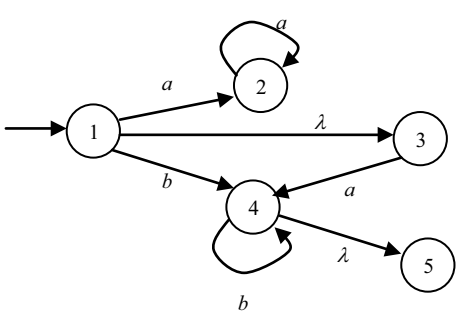


Figura 2.6.11. NFA com transições-λ.

Ao conjunto {2,3,4,5} chama-se o fecho-*a (a-closure*) do estado 1. Ao conjunto {1,3} chama-se fecho-λ (λ-*closure*), sendo composto por todos os estados que se podem alcançar de 1 com λ (inclui naturalmente o próprio); pode ter um número qualquer de estados, dependendo do caso, usando transiçõesλ, até ao número total de estados do autómato. Na tabela de transições, quando calcularmos o DFA equivalente, tem que se ter muita atenção para com o fecho-λ de um estado quando estamos a calcular as transições do estado com os caracteres do alfabeto. Se houver uma transição-λ a partir do estado inicial no NFA, então o estado inicial do DFA equivalente será o estado composto pelo inicial do NFA mais o fecho-λ do estado inicial do NFA.

Exemplo 2.6.4. Seja o autómato anterior com um estado 6 aceitador e uma transição-λ do estado inicial para o estado 6, Fig. 2.6.12 . Calcule-se o DFA equivalente. A primeira tarefa será vermos qual o estado inicial do DFA. Como se vai de 1 para 6 aceitador com λ , quer isto dizer que o NFA aceita λ. Ora a única forma de o DFA também aceitar λ é fazendo o seu estado inicial aceitador. O estado 3 do NFA pode ser alcançado sem consumir caracteres, ele faz parte do fecho-λ de 1. Portanto o estado inicial do DFA será 136 e será por ele que iniciaremos a construção da sua tabela de transições.

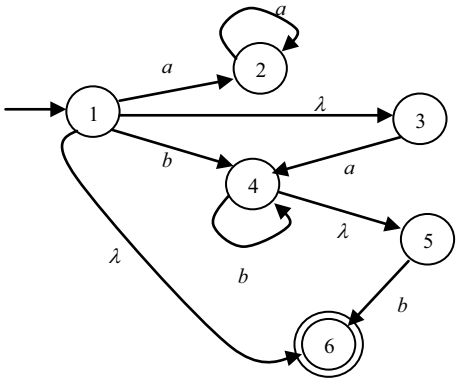


Figura 2.6.12. Exemplo 2.6.4.

Tabela 2.6.6. Transições da Fig. 2.6.12.

Estado	*a*	*b*
136	245∅	4∅
245	2∅	45∅
4	∅	46
2∅	2∅	∅
45	∅	46∅
46	∅	46∅
∅	∅	∅

O DFA equivalente terá 7 estados, dos quais dois serão aceitadores: 136 e 46. Fica ao cuidado do leitor o seu desenho e a identificação da linguagem que ele aceita.

2.7. Redução do número de estados em Autómatos Finitos

Qualquer DFA define uma única linguagem. Mas uma linguagem não define um único DFA. Para uma linguagem existem muitos possíveis DFA's, isto é, existem muitos DFA's equivalentes, cujo número de estados pode ser bem diferente. Por razões de simplicidade, importa obter o de menor número de estados. Por exemplo se quisermos programar um DFA, o espaço necessário para computar é proporcional ao número de estados. Por razões de eficiência deve-se reduzir o número de estados ao mínimo possível.

Vejamos alguns exemplos simples.

Exemplo 2.7.1

O autómato da Fig. 2.7.1 aceita todas as cadeias no seu alfabeto, porque todos os estados são aceitadores.

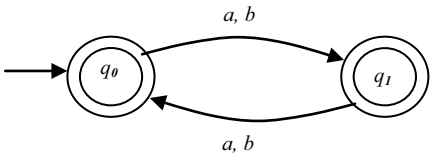


Figura 2.7.1.

Ele tem um equivalente com um só estado:

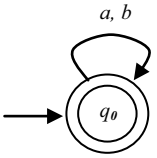


Figura 2.7.2. Autómato equivalente ao da Fig. 2.7.1

Exemplo 2.7.2.

No autómato DFA seguinte há estados a mais.

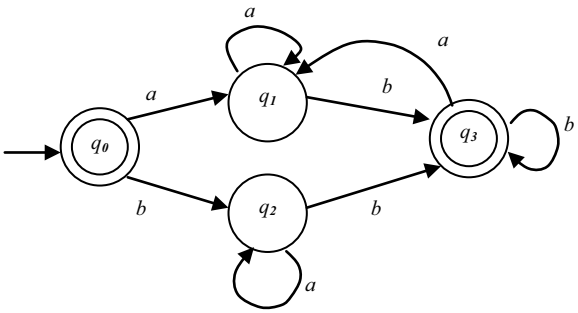


Figura 2.7.3. Exemplo 2.7.2.

Se se chega a q_3 com b quer seja a partir de q_1 quer seja a partir de q_2, coloca-se a possibilidade de fundir q_1 com q_2 no estado q_{12}.

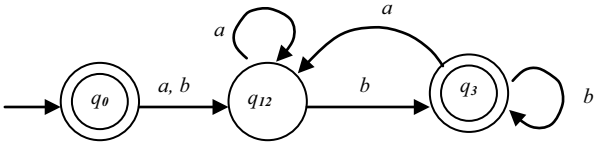


Figura 2.7.4. DFA equivalente ao da Fig. 2.7.3.

Têm que se introduzir as transições que mantenham a equivalência com o autómato original.

No autómato seguinte, Fig. 2.7.2 também se podem eliminar alguns estados.

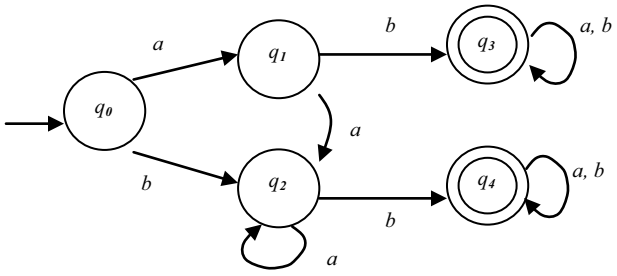


Figura 2.7.5

Em primeiro lugar os dois estados finais podem-se fundir num só, dado que as transições a partir deles são reflexivas (voltam a eles): seja ele q_{34}.

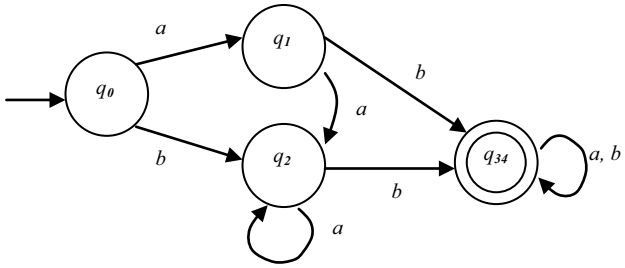


Figura 2.7.6. Simpificação do DFA da Fig. 2.7.5.

Em segundo lugar chega-se a q_{34} com b, quer de q_1, quer de q_2. Por outro lado de q_1 com a vai-se para q_2 e de q_2 com a vai-se para q_2. Se fundirmos q_1 e q_2 em q_{12}, mantendo essas transições, o DFA resultante, Fig. 2.7.7 com apenas 3 estados, é equivalente ao inicial.

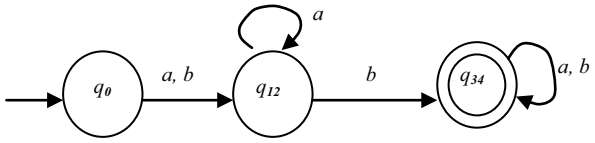


Figura 2.7.7. DFA final no exemplo 2.7.2

Este procedimento heurístico, baseado na nossa habilidade, pode formalizar-se num algoritmo de minimização do número de estados de um DFA. Para o seu desenvolvimento vejamos algumas definições prévias.

Estados indistinguíveis

Dois estados p e q dizem-se indistinguíveis se:

- se a partir de um deles se chega a um estado aceitador com uma cadeia w , também se chega a um estado aceitador a partir do outro com a mesma cadeia w, formalmente,

$\delta^*(p,w) \in F \Rightarrow \delta^*(q,w) \in F$, para todas as cadeias $w \in \Sigma^*$.

e

- se a partir de um deles se chega a um estado não aceitador com uma cadeia x, também se chega a um estado não aceitador a partir do outro com a mesma cadeia x:

$\delta^*(p,x) \notin F \Rightarrow \delta^*(q,x) \notin F$, para todas as cadeias $x \in \Sigma^*$.

Num grafo teremos a seguinte situação:

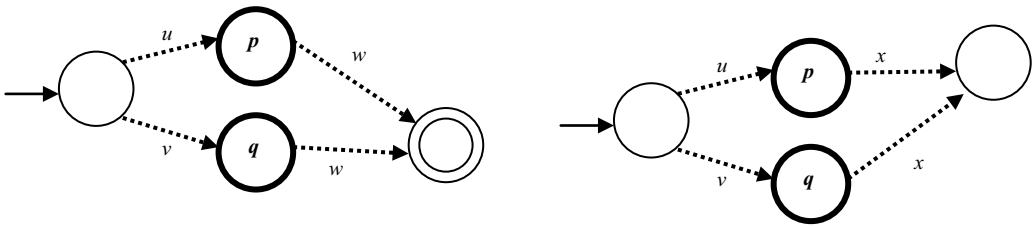


Figura 2.7.8. Os estados *p* e q são indistinguíveis.

Estados distinguíveis por uma cadeia

Se existir uma cadeia *w* tal que de *p* se chegue a um estado aceitador e de *q* a um não aceitador,

$$\delta^*(p,w) \in F \Rightarrow \delta^*(q,w) \notin F$$

ou tal que de *p* se chegue a um não aceitador e de *q* a um aceitador,

$$\delta^*(p,w) \notin F \Rightarrow \delta^*(q,w) \in F$$

então os estados *p* e *q* são **distinguíveis pela cadeia** *w*.

Dois estados ou são distinguíveis ou são indistinguíveis. A indistinção tem as propriedades de uma relação de equivalência R:

1. p R p (reflexiva)

2. p R $q \Rightarrow q$ R p (simétrica)

3. p R q e q R $r \Rightarrow p$ R r (transitiva)

Para reduzir a dimensão de um DFA, procuram-se os estados indistinguíveis e combinam-se entre si. A racionalidade desta operação é evidente: se são indistinguíveis para que se hão-de ter lá ambos?

O algoritmo divide-se em duas etapas:

- primeiro marcam-se os estados distinguíveis,

- segundo, fundem-se os estados indistinguíveis.

Procedimento de marcação dos estados distinguíveis

1º- Removem-se todos os estados inacessíveis (limpeza do DFA). Pode fazer-se enumerando todos os caminhos simples no grafo (caminhos sem ciclos fechados) que partem do estado inicial. Um estado acessível tem que estar num desses caminhos. Se um estado não está em qualquer caminho, quer dizer que não se pode lá chegar a partir do estado inicial, e portanto é inacessível.

2º Consideram-se todos os pares de estados (p, q). Se p é aceitador, $p \in F$, e se q é não aceitador, $q \notin F$, então o par (p, q) é distinguível. De facto que maior distinção poderia haver entre dois estados? Marcam-se todos esses pares como distinguíveis. Marcam-se os pares, não os estados individualmente. Um estado individual pode pertencer a um par distinguível e ao mesmo tempo a um par indistinguível.

3º Considere-se um par de estados (p, q). Calculem-se as transições a partir deles para **todos** os caracteres do alfabeto, tomados um de cada vez.

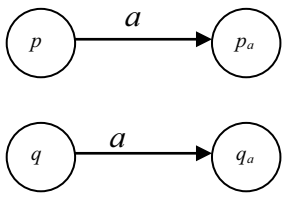


Figura 2.7.9. Transições de p e de q com a.

Se houver pelo menos um carácter $a \in \Sigma$ tal que o par (p_a, q_a) resultante dessas transições seja distinguível, então o par (p, q) é distinguível.

Este procedimento faz-se para todos os pares de estados (p, q) em Q.

No fim todos os pares distinguíveis estão marcados como tal. Ver a demonstração por exemplo em Linz, Teorema 2.3 (pág. 64). Note-se que a marcação é para os pares, não para os estados individualmente.

Depois de se fazer a marcação dos estados distinguíveis, particiona-se o conjunto Q dos estados do DFA num certo número de subconjuntos disjuntos $Q_1=\{q_i, q_j, \ldots, q_k\}$, $Q_2=\{q_l, q_m, \ldots, q_n\}$, …, $Q_x=\{q_o, q_p, \ldots, qr\}$ tais que

- qualquer estado q pertence a um e só a um deles,

- os estados do mesmo subconjunto são indistinguíveis entre si,

- qualquer par de estados, tal que um dos elementos pertence a um dos subconjuntos e o outro pertence a outro, é distinguível.

Esta partição é sempre possível.

A partir desta partição aplica-se o procedimento seguinte.

Dado o DFA $M=(Q,\Sigma,\delta,q_0,F)$ calcula-se o equivalente $\hat{M} = (\hat{Q},\Sigma,\hat{\delta},\hat{q}_0,\hat{F})$ por:

1º Para cada um dos conjuntos indistinguíveis Q_1, Q_2, … Q_x cria-se um estado equivalente etiquetado apropriadamente (a sua etiqueta é o conjunto das etiquetas dos estados do subconjunto, por exemplo para Q_1 será o estado equivalente $q_{ij\ldots k}$.)

2º No autómato original M considera-se a transição $\delta(q_r, a)=q_p$ do estado q_r para o estado q_p. Procuram-se os dois subconjuntos a que pertencem q_r e q_p. Traça-se a transição a entre os estados equivalentes a esses subconjuntos. Isto é, se $q_r \in Q_1=\{q_i,q_j,\ldots, q_k\}$, e $q_p \in Q_2=\{q_l,q_m,\ldots, q_n\}$ então desenhar no grafo a transição

$$\hat{\delta}(q_{ij\ldots k},a) = q_{lm\ldots n}$$

3º O estado inicial de $\hat{M}$, $\hat{q}_0$, é o que contém a etiqueta 0 do autómato original.

4º O conjunto dos estados aceitadores de $\hat{M}$, $\hat{F}$, é o conjunto de todos os estados em cuja etiqueta entra um estado aceitador do DFA original.

Exemplo 2.7.3 .Minimizar o conjunto de estados do DFA da Figura 2.7.10.

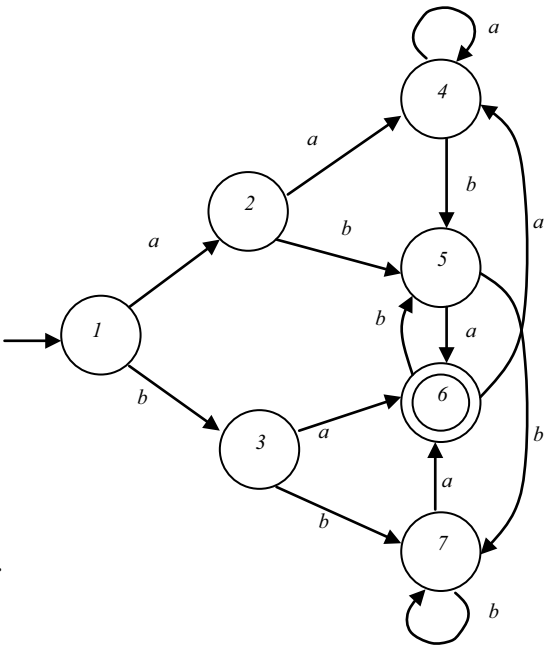


Figura 2.7.10. DFA a minimizar.

1º Marcação dos estados distinguíveis

Havendo um só estado final, ele é distinguível de todos os outros. Portanto qualquer par que o contenha é distinguível.

Vamos agora considerar a segunda etapa: ver as transições de cada para com um *a* e com um *b*. Obteremos as duas tabelas seguintes.

Tabela 2.7.1. Transições com *a*

.	1	2	3	4	5	6	7
1	2,2	2,4	2,6	2,4	2,6	2,4	2,6
2	4,2	4,4	4,6	4,4	4,6	4,4	4,6
3	6,2	6,4	6,6	6,4	6,6	6,4	6,6
4	4,2	4,4	4,6	4,4	4,6	4,4	4,6
5	6,2	6,4	6,6	6,4	6,6	6,4	6,6
6	4,2	4,4	4,6	4,4	4,6	4,4	4,6
7	6,2	6,4	6,6	6,4	6,6	6,4	6,6

Tabela 2.7.2 Transições com *b*

b	1	2	3	4	5	6	7
1	3,3	3,5	3,7	3,5	3,7	3,5	3,7
2	5,3	5,5	5,7	5,5	5,7	5,5	5,7
3	7,3	7,5	7,7	7,5	7,7	7,5	7,7
4	5,3	5,5	5,7	5,5	5,7	5,5	5,7
5	7,3	7,5	7,7	7,5	7,7	7,5	7,7
6	5,3	5,5	5,7	5,5	5,7	5,5	5,7
7	7,3	7,5	7,7	7,5	7,7	7,5	7,7

Da 1ª etapa, nestas duas tabelas todos os pares que contenham o estado 6 devem ser marcados como distinguíveis (colorido), dado que 6 é o único aceitador, excepto o par (6,6) que é indistinguível.

Os pares que correspondem às células sombreadas serão agora marcados distinguíveis.

Na tabela de transições com *b* marcam-se todos os distinguíveis com *a*. Basta fazê-lo na metade superior (ou inferior) da tabela. Neste caso não há nenhum..

Será o par (1,2) distinguível ? (1,2) com *a* dá (2,4), (2, 4) com *a* dá (4, 4), nenhum distinguível; portanto não se marca. Vai-se ver à tabela de *b* se o par (2,4) é distinguível.

Para o par (3,4): (3,4) com *a* dá (6,4), já marcado e portanto (3,4) marca-se como distinguível.

O mesmo para todos os outros pares. Obtém-se a tabela seguinte, em que 1 identifica os distinguíveis.

Tabela 2.7.3. Estados distinguíveis (1)e indistinguíveis (0)

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	1	0	1	0	1
2	0	0	1	0	1	0	1
3	1	1	0	1	0	1	0
4	0	0	1	0	1	0	1
5	1	1	0	1	0	1	0
6	0	0	1	0	1	0	1
7	1	1	0	1	0	1	0

Agora particiona-se o conjunto dos estados em subconjuntos que obedeçam às três condições acima enunciadas. Não é uma tarefa fácil.

O estado 6 só é indistinguível de si próprio, portanto tem que ficar isolado num subconjunto.

Para os restantes desenhe-se o grafo da propriedade **indistinguível.** Os vértices do grafo serão os estados. Entre dois estados existirá uma aresta se eles pertencerem a um par indistinguível, isto é, se na última tabela a respectiva célula tiver 0. Não é necessário desenhar a aresta reflexiva (do estado para ele mesmo).

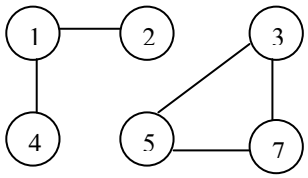


Figura 2.7.11. Partição dos estados

É bem visível a formação de dois sub-grafos conexos : {1,2,4}, {3,5,7}.

A partição {1, 2, 4} {3, 5, 7} {6} verifica as condições. O DFA mínimo terá 3 estados.

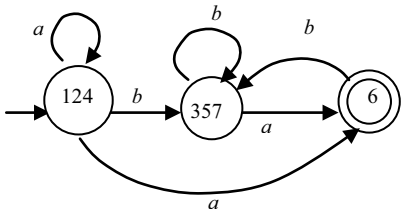


Figura 2.7.12. o DFA mínimo equivalente ao da Fig. 2.7.10.

As linguagens dos dois autómatos, o original e o reduzido, são equivalentes. Para um autómato com um elevado número de estados, este procedimento manual de inspecção para se identificarem os pares distinguíveis não é fácil. No entanto este algoritmo pode programar-se. Ele também é conhecido pelo algoritmo de enchimento da tabela (*table- filling algorithm*, por exemplo em Hopcroft p.156).

2.8 Um aplicação dos DFA: busca de texto (*text search*, Hopcroft, 68)

Os autómatos finitos encontram na busca de texto uma aplicação interessante. Como sabemos a pesquisa de cadeias de caracteres é uma das operações mais executadas na *web*. Dado um

conjunto de palavras, que poderemos chamar palavras chave, queremos encontrar todas as ocorrências delas num certo texto. Os autómatos finitos não determinísticos permitem fazê-lo de um modo expedito e simples. Para uma palavra chave, por exemplo *chave*, desenha-se um autómato com:

- um estado inicial que transita para si próprio com todo e qualquer carácter do alfabeto,

- uma corrida de estados aos quais se chega depois de ler sucessivamente um carácter da palavra *chave*

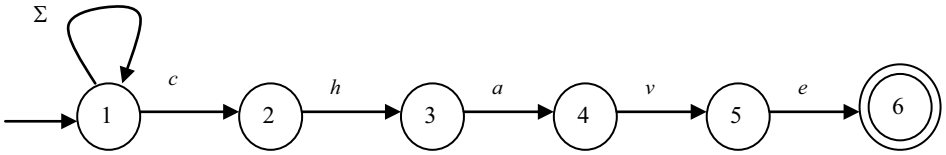


Figura 2.8.1. NFA para busca da palavra *chave*.

Para se implementar o NFA, pode-se escrever um programa que simula o seu funcionamento, calculando os estados em que está após ler um carácter. Sempre que há uma escolha, faz-se uma ou várias réplicas do NFA.

É no entanto mais prático calcular o DFA equivalente e então simulá-lo. Para isso aplica-se a técnica padrão: constrói-se a tabela de transições e desenha-se o grafo depois.

Se o alfabeto for composto pelos 26 caracteres do português (*a b c d e f g h i j k l m n o p q r s t u v w x y z*), teremos muitas etiquetas para as arestas do autómato. Para simplificar a sua escrita poderemos usar a notação *Σ-a-e* querendo dizer todos os caracteres do alfabeto menos *a* e *e*. Esta notação é aceite pelo DEM (*Deus Ex-Máquina*).

Assim sendo, o DFA equivalente ao NFA da Fig. 2.8.1 tem o aspecto da Fig. 2.8.2. Fica ao cuidado do leitor verificar, construtivamente, que assim é.

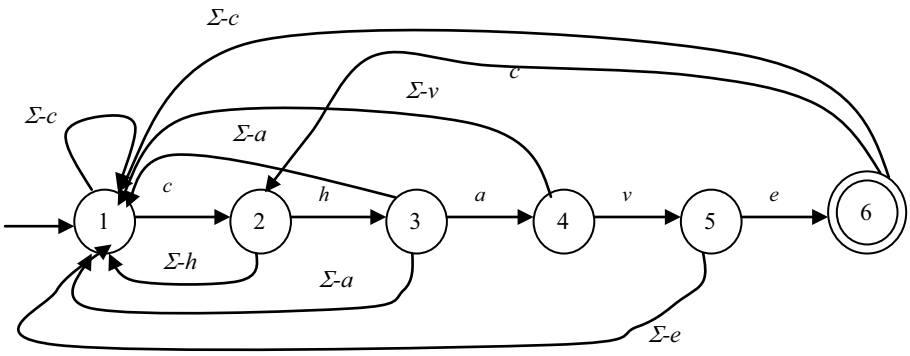


Figura 2.8.2. DFA equivalente ao NFA da Fig. 2.8.1.

Exercício 2.8.1

O seguinte NFA encontra num texto as palavras *doce* e *mel*. Desenhar o o DFA equivalente. Σ = { *a, b, c, ..., z*}.

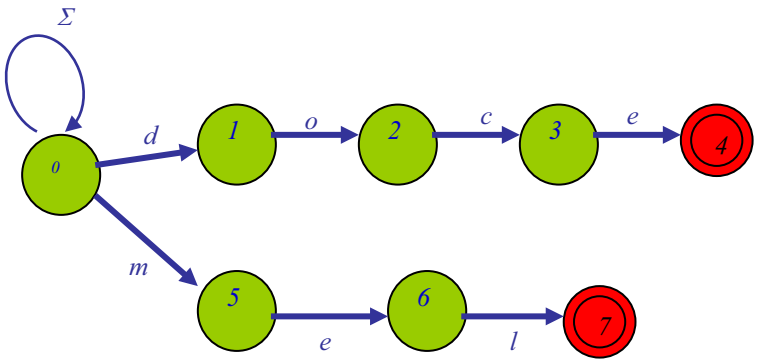


Figura 2.8.3 NFA para o exemplo 2.8.1.

Desenhar o DFA equivalente.

Exercício 2.8.2.

Desenhar o DFA que aceita as palavras *amarga* e *carta*. Σ= { *a, b, c, ..., z*}.

2.9 Autómatos finitos transdutores

Os DFA e NFA que estudámos até aqui são aceitadores e não têm saída. Apenas lêem cadeias, reconhecendo-as ou não como pertencentes a uma dada linguagem.

Há outros tipos de autómatos, como vimos já no Cap. 1, que têm um registo de saída, cujo conteúdo pode variar conforme a evolução da configuração do autómato. São os transdutores, de que se conhecem dois tipos: as máquinas de Mealy e de More. Ambos se podem enquadrar no esquema geral que vimos no Cap. 1 e aqui se reproduz na Fig. 2.9.1

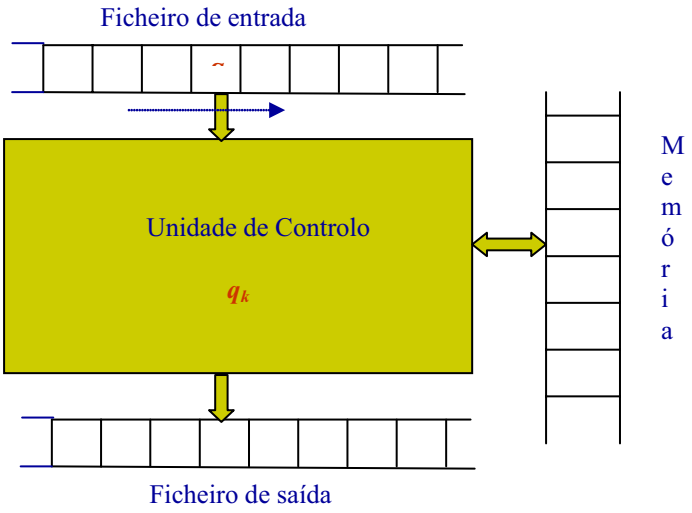


Figura 2.9.1. Esquema geral da um autómato finito

2.9.1 Máquinas de Mealy

Numa máquina de Mealy a saída depende do estado interno do autómato actual e da entrada actual

$$y_k = f(s_k, q_k)$$

Definição 2.9.1 Máquina de Mealy.

Uma Máquina de Mealy é definida pelo sexteto (agora há mais um músico …)

$$M = (Q, \Sigma, \Lambda, \delta, \gamma, q_0)$$

em que

Q é o conjunto finito de estados internos,

Σ é o alfabeto de entrada (conjunto finito de caracteres),

Λ é o alfabeto de saída (conjunto finito de caracteres),

$\delta: Q x \Sigma \rightarrow Q$ é a função de transição de estado,

$\gamma: Q x \Sigma \rightarrow \Lambda$ é a função de saída,

$q_0 \in Q$ é o estado inicial

Comparando com a definição de autómato aceitador verificamos que não há aqui estados aceitadores (F), mas há em contrapartida um alfabeto de saída e uma função de saída.

Tal como nos aceitadores, também um transdutor é representado por um grafo. A etiquetagem dos estados e das arestas é no entanto diferente. Numa máquina de Mealy usa-se a notação da figura seguinte:

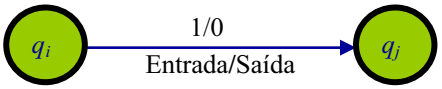


Figura 2.9.2 Notação da Máquina de Mealy

Que tem a seguinte interpretação: estando a máquina no estado q_i e lendo 1 na entrada,

- transita para o estado q_j

- envia 0 para a saída.

Exemplo 2.9.1

Considere-se a máquina de Mealy da Fig.2.9.3

Se lhe dermos a ler a cadeia 10111 ela faz:
- lê 1 e escreve na saída λ transitando para o estado q_1,
- lê 0, escreve 1, passando a q_2
- lê 1, escreve 0, passa a q_1,
- lê 1, escreve 1, mantém-se em q_1,
- lê 1, escreve 1, mantém-se em q_1

No fim da leitura está em q_1 e escreveu na saída λ1011.

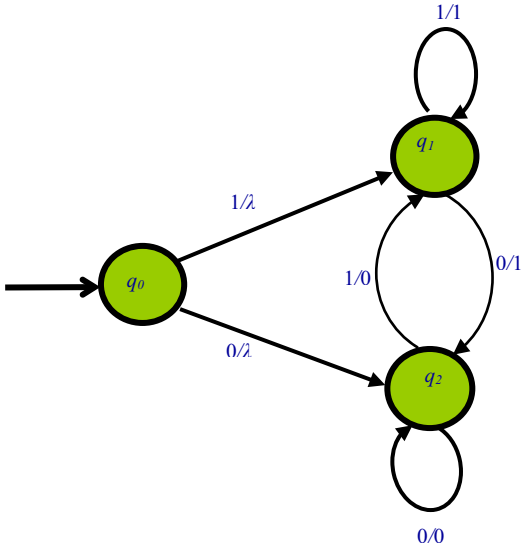


Figura 2.9.3. Máquina de Mealy do exemplo 2.9.1.

Isto é, deslocou a cadeia para a direita uma casa (*shift right*).

As máquinas de Mealy podem implementar-se por circuitos lógicos, e funcionam em modo assíncrono: logo que a entrada variar pode variar a saída.

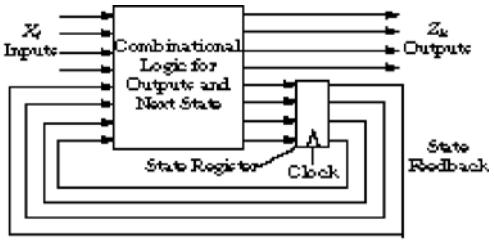


Figura 2..9.4. Esquema geral de uma máquina de Mealy.
(em http://www2.ele.ufes.br/~ailson/digital2/cld/chapter8/chapter08.doc4.html)

2.9.2. Máquinas de Moore

As máquinas de Moore distinguem-se das de Mealy pelo facto de a sua saída depender apenas do estado actual

$$y_k = f(q_k)$$

isto é, não dependem da entrada actual.

Definição 2.9.2. Máquina de Moore.

Uma Máquina de Moore é definida pelo sexteto

$$M = (Q, \Sigma, \Lambda, \delta, \gamma, q_0)$$

Sendo

Q o conjunto finito de estados internos,

Σ o alfabeto de entrada (conjunto finito de caracteres),

Λ o alfabeto de saída (conjunto finito de caracteres),

$\delta: Q x \Sigma \rightarrow Q$ é a função de transição de estado,

$\gamma: Q \rightarrow \Lambda$ é a função de saída,

$q_0 \in Q$ é o estado inicial.

No seu grafo usa-se a notação

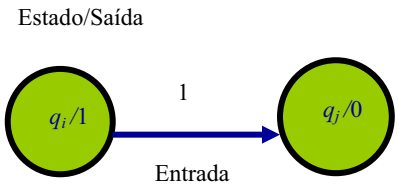


Figura 2.9.5.Notação das Máquinas de Moore.

significando que quando entra no estado q_i escreve 1 na saída, depois com 1 na entrada transita para q_j e escreva 0 na saída.

Exemplo 2.9.2

A Máquina de Moore da Fig. 2.9.6 faz, tal como a anterior, o deslocamento à direita da entrada que se lhe fornece.

Dá-se-lhe 1110101, estando no estado 0

- lê 1, passa a 1 escreve λ
- lê 1, passa a 3, escreve 1
- lê 1, mantém 3, escreve 1
- lê 0, passa a 4, escreve 1
- lê 1, passa a 5, escreve 0
- lê 0, passa a 4, escreve 1
- lê 1, passa a 5, escreve 0

Portanto escreveu λ111010.

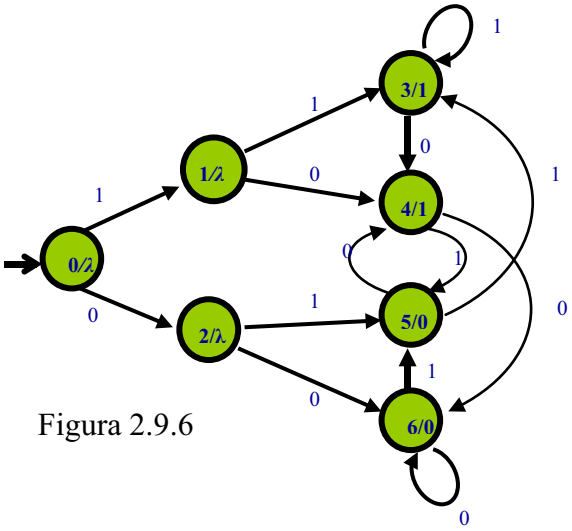


Figura 2.9.6

A máquina de Moore escreve na saída logo após transitar de estado. Por isso a sua implementação por hardware faz-se com ilustrado na figura seguinte. Trata-se de uma máquina síncrona (as mudanças na saída são sincronizadas com as mudanças de estado).

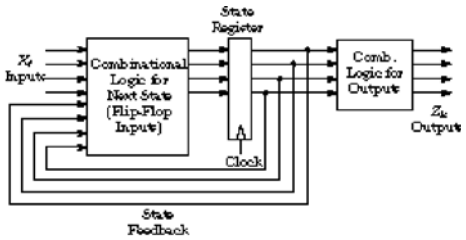


Figura 2.9.7. Esquema geral da Máquina de Moore (de http://www2.ele.ufes.br/~ailson/digital2/cld/chapter8/chapter08.doc4.html).

Estes dois exemplos indiciam que existe equivalência entre as máquinas de Mealy e de Moore: dada uma máquina de Mealy é possível encontrar uma máquina de Moore que, para as mesmas entradas, dá as mesmas saídas. E vice-versa. (Ignorando a saída do estado inicial na máquina de Moore).

A de Moore equivalente tem em geral um maior número de estados.

As máquinas de Mealy e de Moore são os cavalos de batalha no projecto de circuitos lógicos complexos. Existe até software que simula estas máquinas e gera o circuito lógico a partir do autómato desenhado com um grafo (ver por ex. http://www.seas.upenn.edu/~ee201/foundation/foundation_impl4.html ou http://www.xilinx.com/univ/xse42.html).

Bibliografia.

An Introduction to Formal Languages and Automata, Peter Linz, 3rd Ed., Jones and Bartelett Computer Science, 2001

Models of Computation and Formal Languages, R. Gregory Taylor, Oxford University Press, 1998.

Introduction to Automata Theory, Languages and Computation, 2nd Ed., John Hopcroft, Rajeev Motwani, Jeffrey Ullman, Addison Wesley, 2001.

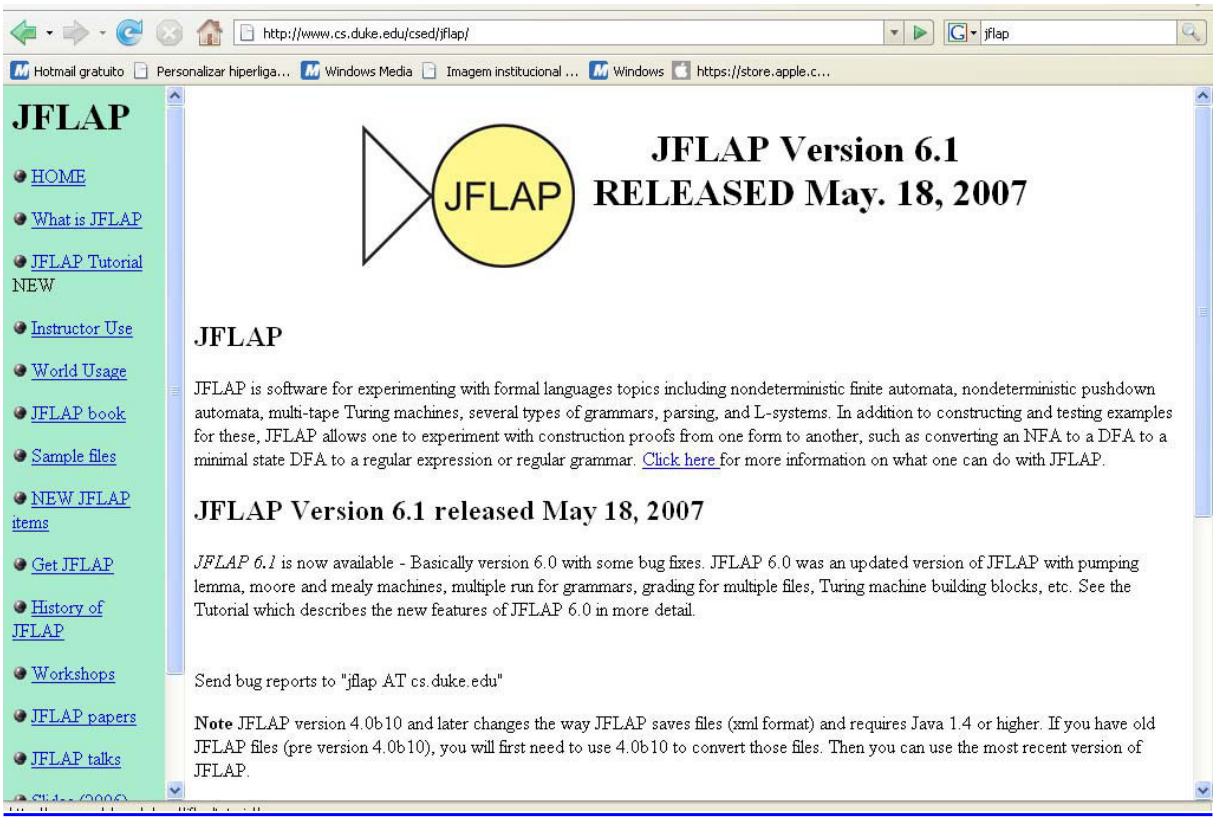
Elements for the Theory of Computation, Harry Lewis and Christos Papadimitriou, 2nd Ed., Prentice Hall, 1998.

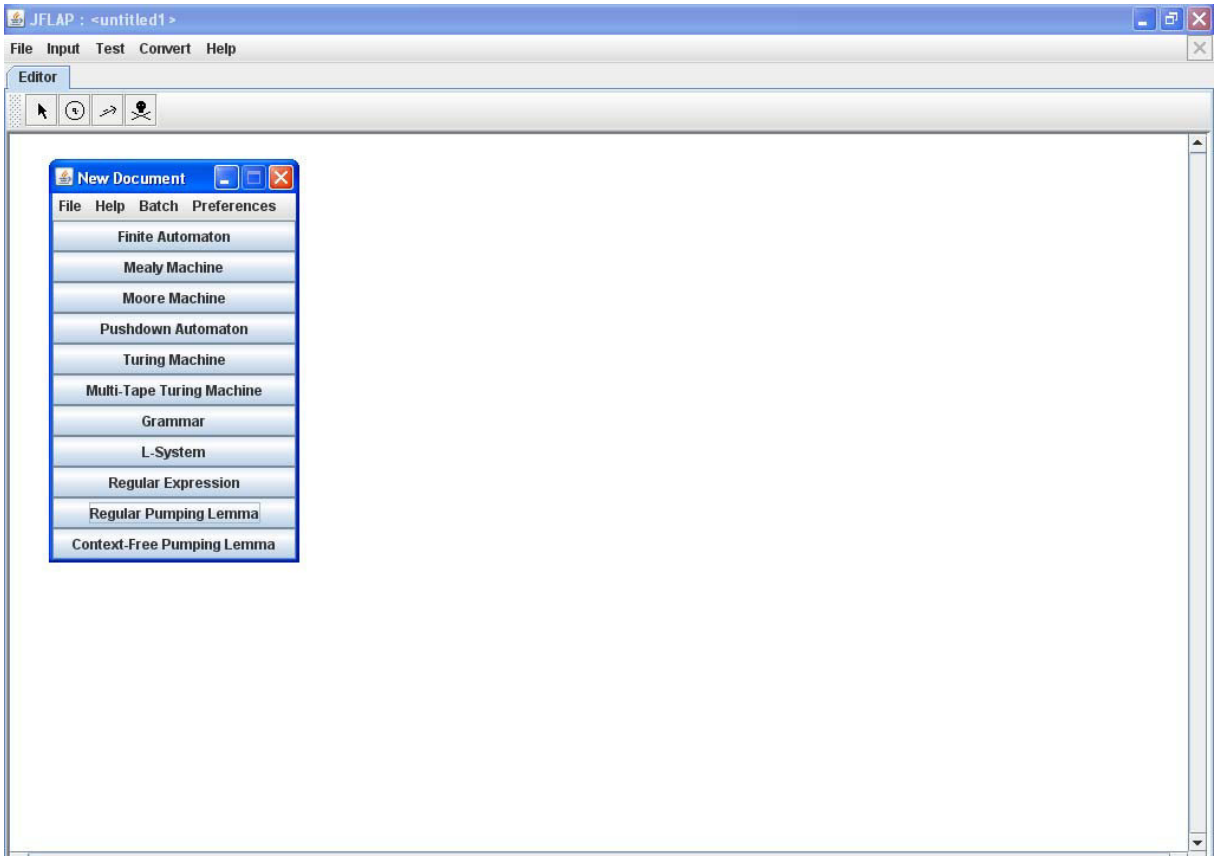
Introduction to the Theory of Computation, Michael Sipser, PWS Publishing Co, 1997.

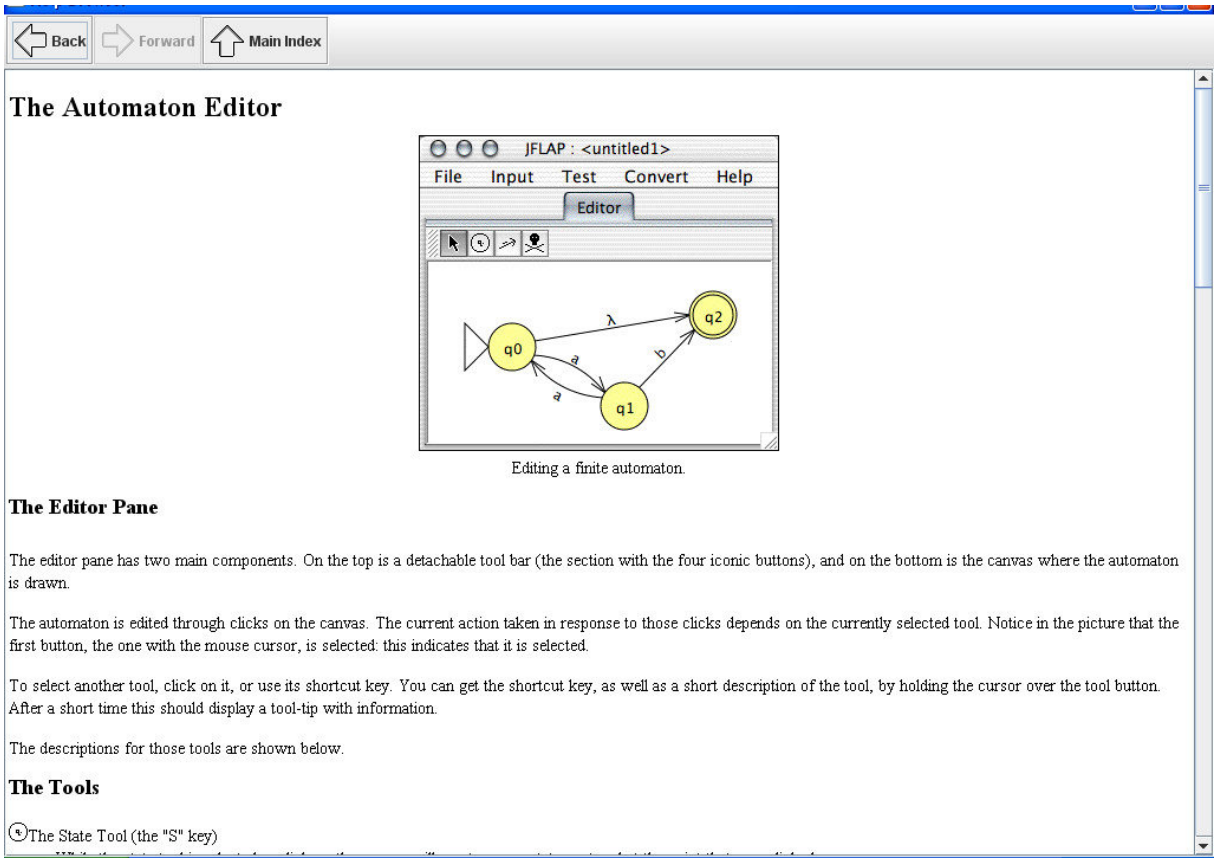
G. H. Mealy, *A Method for Synthesizing Sequential Circuits*, Bell System Tech. J. vol 34, pp. 1045–1079, Sept 1955.

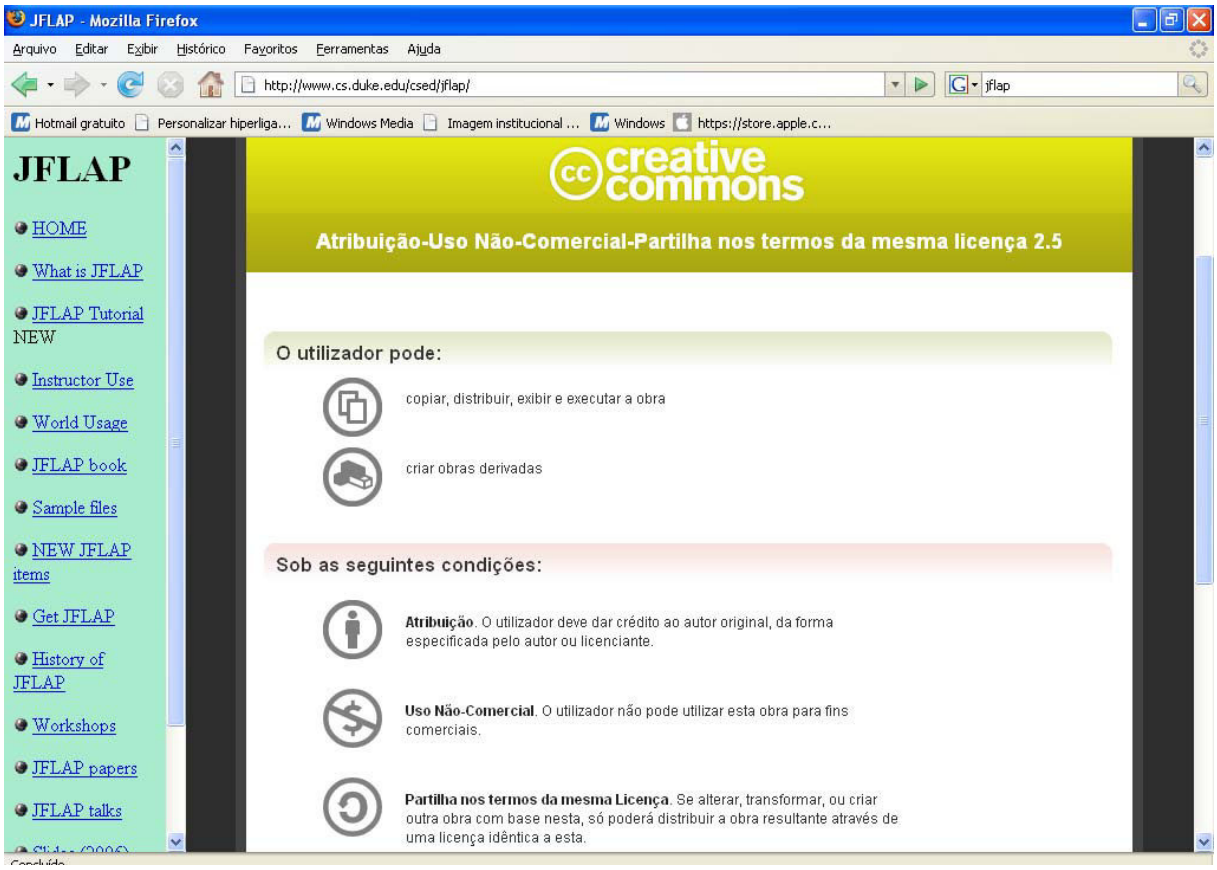
Apêndice . O software JFLAP (livre) para desenho e simulação de autómatos.

http://www.cs.duke.edu/csed/jflap




JFLAP : <untitled1 >
File Input Test Convert Help
Editor
New Document
File Help Batch Preferences
Finite Automaton
Mealy Machine
Moore Machine
Pushdown Automaton
Turing Machine
Multi-Tape Turing Machine
Grammar
L-System
Regular Expression
Regular Pumping Lemma
Context-Free Pumping Lemma


Back
Forward
Main Index
The Automaton Editor
JFLAP : <untitled1>
File Input Test Convert Help
Editor
q0
q1
q2
λ
a
b
a
Editing a finite automaton.
The Editor Pane
The editor pane has two main components. On the top is a detachable tool bar (the section with the four iconic buttons), and on the bottom is the canvas where the automaton is drawn.
The automaton is edited through clicks on the canvas. The current action taken in response to those clicks depends on the currently selected tool. Notice in the picture that the first button, the one with the mouse cursor, is selected: this indicates that it is selected.
To select another tool, click on it, or use its shortcut key. You can get the shortcut key, as well as a short description of the tool, by holding the cursor over the tool button. After a short time this should display a tool-tip with information.
The descriptions for those tools are shown below.
The Tools
The State Tool (the "S" key)



Tem várias outras funcionalidades muito úteis em capítulos posteriores (equivalência de autómatos, gramáticas, etc.).

O software DEM- *Deus Ex-Maquina* (também para a simulação de autómatos) está disponível na página da cadeira.