CAPÍTULO 9

MÁQUINAS DE TURING

9.1. Introdução

Uma pesquisa no Google em 6 de Dezembro de 2007 com "*turing machine*" deu 1.820.000 resultados. Este número mostra a importância do tema.

As Máquinas de Turing (MT) estiveram no centro do desenvolvimento dos computadores e da computação durante os últimos 70 anos. Alain Turing (1912-1954) foi um brilhante matemático, em Cambridge, Inglaterra, numa época efervescente de desenvolvimento da lógica e da matemática que haveria de resultar no computador digital, os anos 30 e 40 do século passado. É geralmente considerado como o fundador das ciências da computação. Outros matemáticos famosos, como Gödel, Bertrand Russel na Europa, Church nos EUA, foram contemporâneos de Alain Turing. Existe em Portugal um blog como seu nome, http://turing-machine.weblog.com.pt, que merece uma visita.

Em 1940 Alan Turing procura formalizar a noção de algoritmo, identificando as operações fundamentais e primitivas que possam servir de base ao cálculo matemático. Depois, definiu uma máquina abstracta capaz de executar essas operações segundo regras bem definidas. A MT foi assim concebida para ser um modelo de computação, formalizando um conjunto de operações básicas às quais se pode reduzir qualquer computação.

Os autómatos finitos são para as linguagens regulares, os autómatos de pilha para as linguagens livres de contexto. E as MT? Com que linguagens se relacionam?

Já encontrámos linguagens que não são livres de contexto, como por exemplo $a^n b^n c^n$. Por isso não é possível construir um autómato de pilha que as aceite.

Será que uma MT é suficientemente poderosa para aceitar linguagens dependentes do contexto? Todas elas? Qual é o autómato mais poderoso? Quais os limites da computação? São questões às quais as MT respondem.

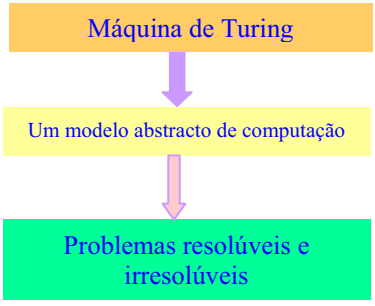


Figura 9.1.1. Importância da Máquina de Turing. Até hoje não foi ainda inventado um computador capaz de resolver um problema que a MT não possa resolver. Este facto mantém ainda válida a tese de Church-Turing

9.2. A máquina de Turing padrão

A máquina de Turing é um autómato, com uma unidade de controlo e com um dispositivo especial que funciona simultaneamente como entrada (onde lê), armazenamento, e saída (onde escreve). Esse dispositivo é uma **fita** unidimensional que contém um número ilimitado de células cada uma das quais pode conter um único símbolo. Essa é a sua característica distintiva em relação aos autómatos finitos (que não têm dispositivo de armazenamento) e aos autómatos de pilha (que armazenam numa pilha).

A fita da MT prolonga-se indefinidamente em ambos os sentidos e por isso pode conter uma quantidade infinita de informação. Esta informação pode ser lida e alterada em qualquer ordem e daí o potencial da MT.

Associada à fita está uma **cabeça de leitura-escrita** que se pode mover sobre a fita para a esquerda ou para a direita, podendo escrever ou ler um único símbolo em cada movida.

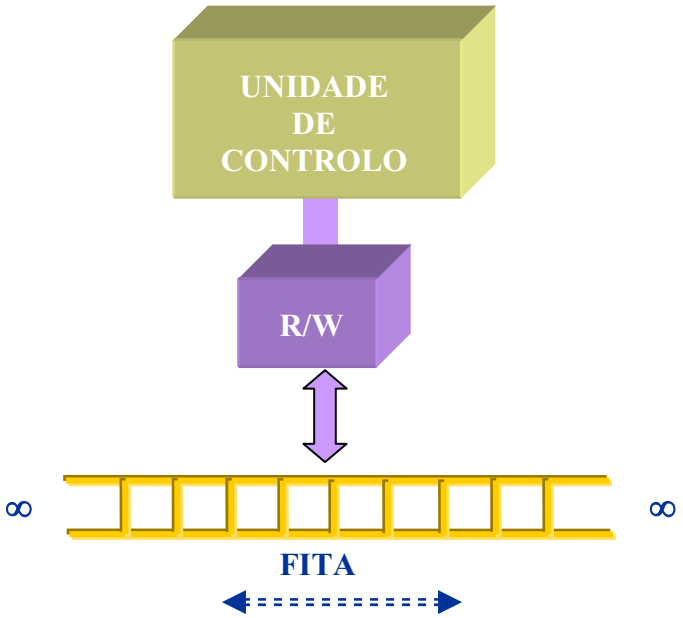


Figura 9.2.1 Componentes da Máquina de Turing

Definição 9.2.1.

Uma máquina de Turing M é definida pelo septeto

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

em que

Q é o conjunto de estados internos da unidade de controlo

Σ é o alfabeto de entrada

Γ é o conjunto finito **alfabeto da fita**

δ é a função de transição

q_0 é o estado inicial

$\square \in \Gamma$ é o carácter **branco**, símbolo especial de Γ

$F \subseteq Q$ é o conjunto de estados finais (aceitadores)

O alfabeto de entrada é igual ao alfabeto da fita, com excepção do símbolo $\square$ (branco, um carácter novo, não confundir com λ) que não existe no alfabeto de entrada. Portanto

$$\Sigma = \Gamma - \{\square\}$$

A função de transição é em geral uma função parcial em $Q \times \Gamma$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

em que *R* significa movida para a direita e *L* movida para a esquerda. Os dois argumentos de entrada da função de transição e os três de saída são os seguintes:

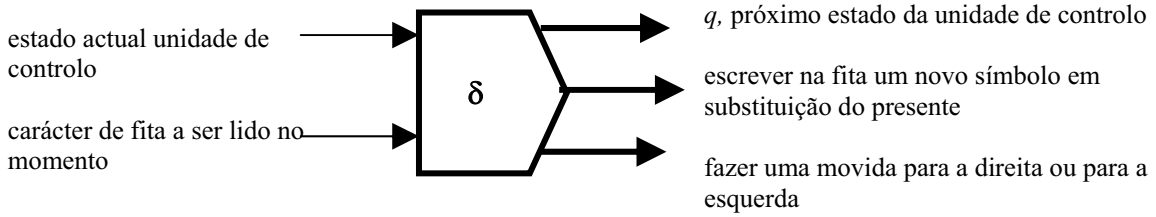


Figura 9.2.2 Função de transição de uma Máquina de Turing

A movida da cabeça faz-se depois da escrita do novo símbolo na fita.

Exemplo 9.2.2.1

Sejam

$$\Sigma = \{ a, b \}$$

$$\Gamma = \{ a, b, \square \}$$

$$\delta: (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{ L, R \})$$

e a transição

$$\delta(q_0, a) = (q_1, b, R)$$

Esta transição é ilustrada na figura seguinte.

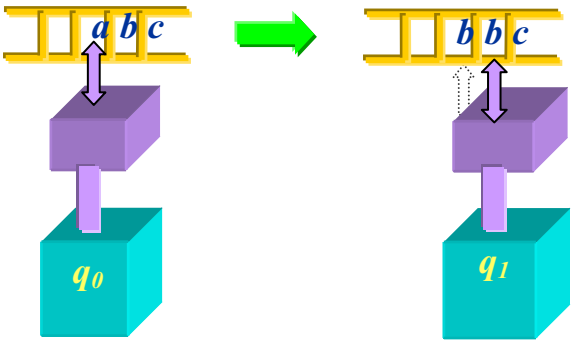


Figura 9.2.3. Movida da MT do exemplo 9.2.1.

Lendo *a*,

-apaga-o e escreve *b* no seu lugar

-move a cabeça de leitura uma célula para a direita, ficando a apontar para *b*.

Uma máquina se Turing pode ser encarada como um computador muito simples. De facto:

- tem uma unidade de processamento com memória finita (número finito de estados);
- tem uma segunda unidade de armazenamento de capacidade infinita;
- a função de transição é o "programa" do computador. É uma função parcial.

O autómato é inicializado (no estado inicial q_0) com alguma coisa já escrita na fita.

Executa depois uma sequência de operações, uma computação, definidas pela função δ.

Como δ é uma função parcial, pode chegar a uma configuração para a qual não está definita nenhuma transição. O autómato fica então no estado parado (**halt**).

Nunca se definem transições a partir dos estados finais e por isso uma máquina de Turing pára sempre que atinge um estado final.

Exemplo 9.2.2.
Seja a máquina de Turing definida por

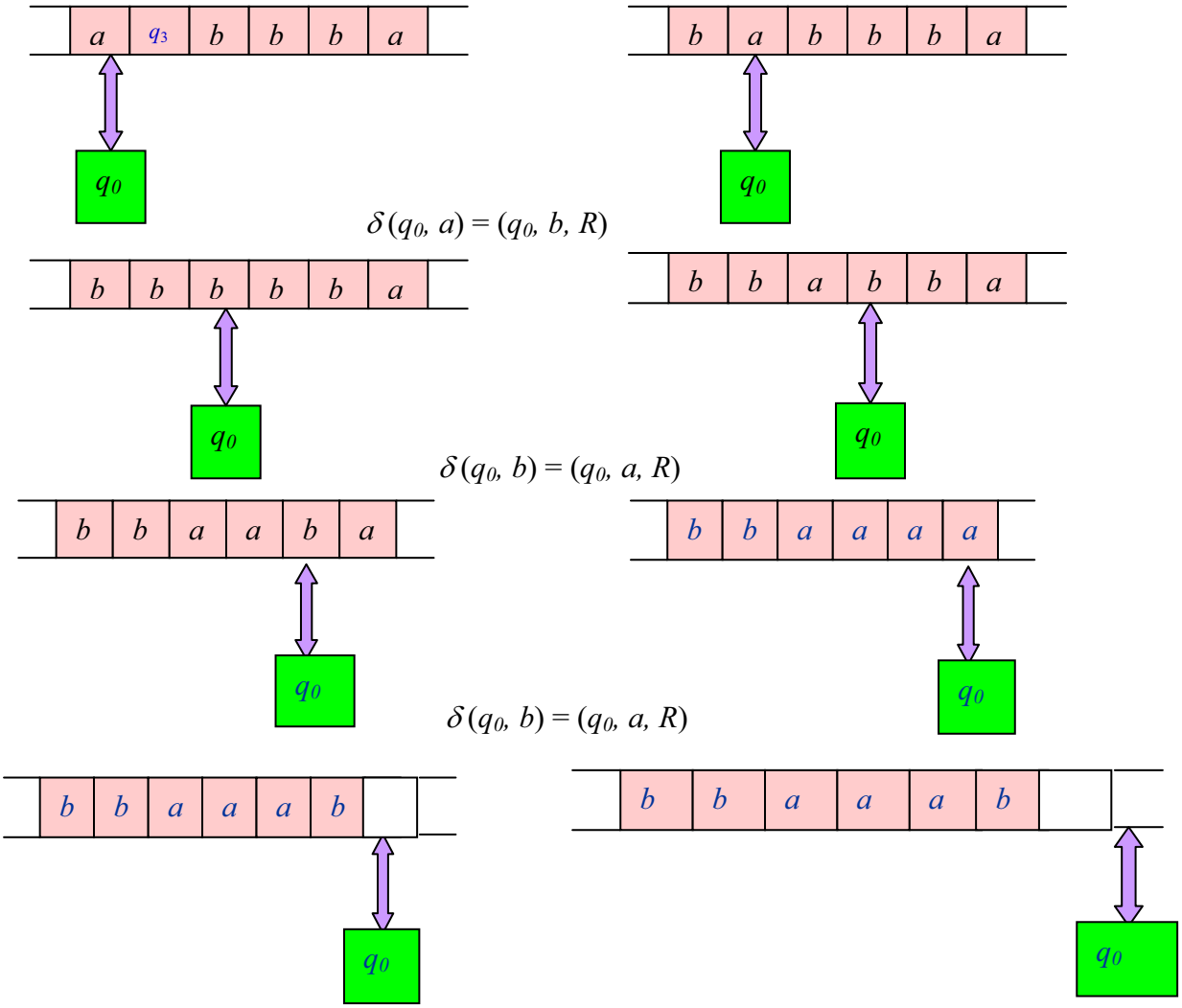
$$Q = \{q_0, q_1\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{a, b, \square\}$$
$$F = \{q_1\}$$

e

$$\delta(q_0, a) = (q_0, b, R)$$

$$\delta(q_0, b) = (q_0, a, R)$$

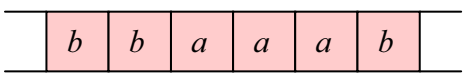
$$\delta(q_0, \square) = (q_0, \square, R)$$



$$\delta(q_0, \square) = (q_0, \square, R)$$

Figura 9.2.4. Movidas do exemplo 9.2.2

No fim obtém-se o seguinte conteúdo da fita



Se para a mesma máquina a função de transição for

$$\delta(q_0, a) = (q_0, a, R)$$

$$\delta(q_0, b) = (q_1, b, R)$$
$$\delta(q_0, \square) = (q_0, \square, R)$$
$$\delta(q_1, a) = (q_0, b, L)$$
$$\delta(q_1, b) = (q_0, b, L)$$
$$\delta(q_1, \square) = (q_0, \square, L)$$

obtém-se um funcionamento em ciclo infinito, como se pode ver para a cadeia *aabbba*:

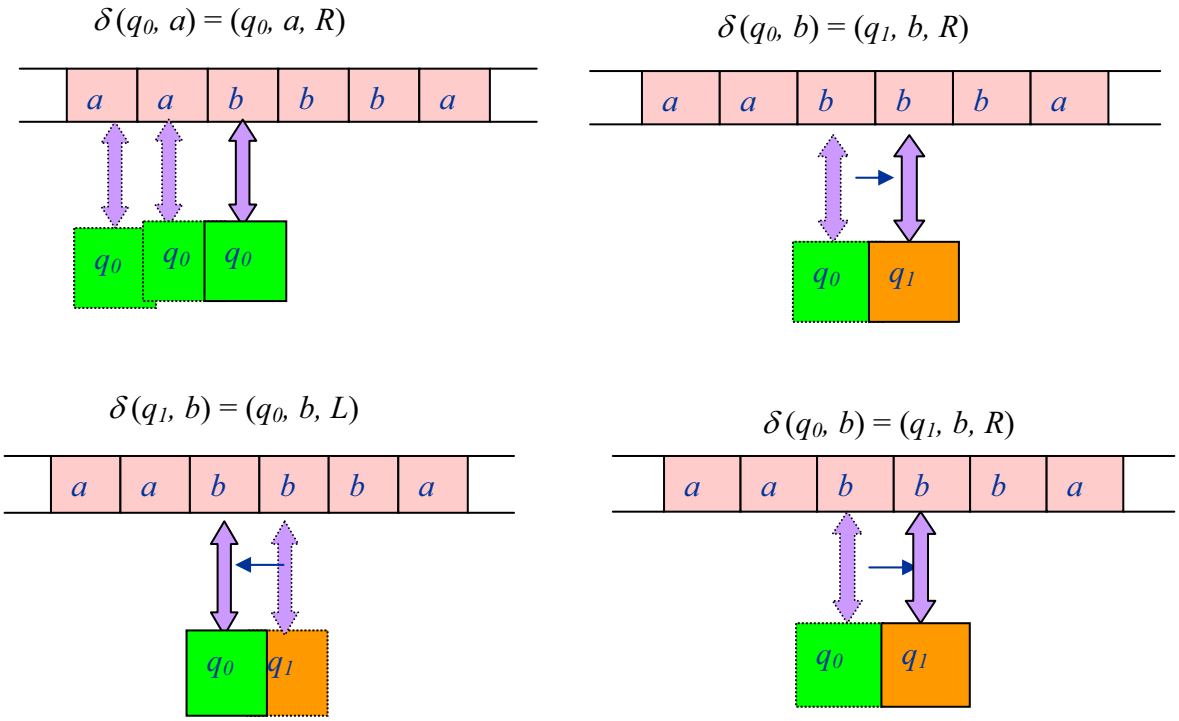


Figura 9.2.5 Movidas para o exemplo 9.2.2, segundo caso.

A máquina de Turing padrão tem as seguintes características:

1- Uma fita ilimitada em ambas as direcções, permitindo um número arbitrário de movidas à esquerda e à direita.

2- É determinística na medida em que δ define no máximo uma movida para cada configuração, não havendo escolhas possíveis.

3- Não existe qualquer ficheiro de entrada especial. Pressupõe-se que no instante inicial a fita tem algum conteúdo especificado aí introduzido pelo programador. Parte deste

pode ser considerado a entrada. Sempre que a máquina pare (**halt**), algum do conteúdo da fita pode ser considerado como saída.

Como veremos posteriormente, existem outras versões de máquinas de Turing mas que se baseiam neste modelo base, estendendo algumas características (mas não aumentando o seu potencial limite de computação).

A configuração da máquina de Turing é definida pela sua **descrição instantânea** composta por três partes:

- estado actual da unidade de controlo
- o conteúdo da fita
- a posição da cabeça de leitura/escrita.

Utiliza-se a notação seguinte para a descrever uma configuração

$$x_1 q x_2$$

ou

$$a_1 a_2 \ldots a_{k-1} q a_k a_{k+1} \ldots a_n$$

correspondente à figura seguinte. A cabeça aponta para a célula que contém o símbolo que segue q, i.e., a_k.

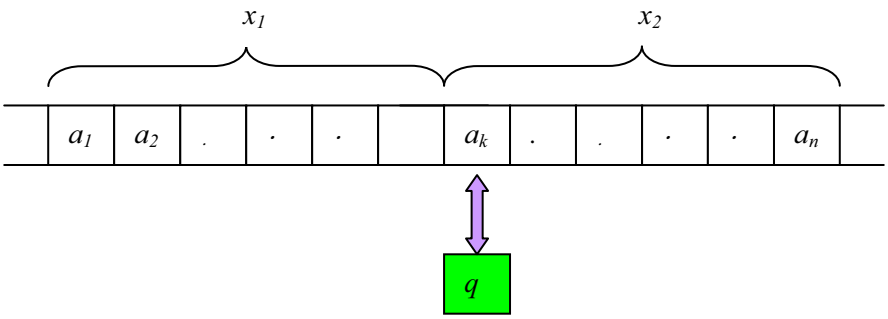


Figura 9.2.6 Configuração de uma MT

A descrição instantânea só mostra uma parte da fita. Assume-se que a restante parte contém apenas símbolos brancos, que são considerados irrelevantes e por isso não é necessário mostrá-los. No entanto se a posição dos brancos for relevante, deve ser especificada na descrição instantânea.

Para representar as movidas de uma MT utiliza-se a notação de configuração em configuração

configuração (k) $\vdash$ configuração (k+1)

Por exemplo para a transição

$$\delta(q_1, c) = (q_2, e, R)$$

em termos de configurações escreve-se, admitindo que na fita temos por exemplo *abcd* e que a cabeça aponta para *c*,

$$abq_1cd \vdash abeq_2d$$

configuração (k) $\vdash$ configuração (k+1)

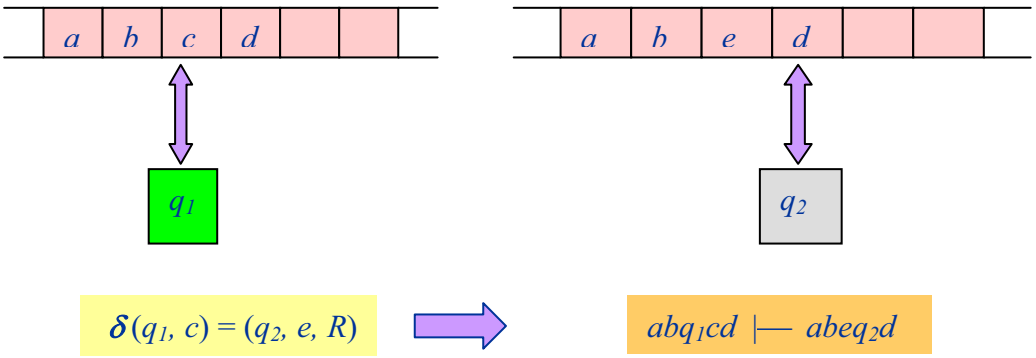


Figura 9.2.7. Uma movida entre duas configurações instantâneas

O símbolo $\overset{*}{\vdash}$ indica um número arbitrário de movidas. Se houver necessidade de distinguir entre diversas MT's, usa-se $\overset{*}{\vdash}_M$.

Definição 9.2.2 Movida

Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ uma MT.

Então qualquer cadeia $a_1a_2.....a_{k-1}q_1a_ka_{k+1}...a_n$ com $a_i \in \Gamma$ e $q_1 \in Q$, é uma descrição instantânea de *M*.

É possível uma movida

$$a_1a_2.....a_{k-1}q_1a_ka_{k+1}...a_n \vdash a_1a_2.....a_{k-1}bq_2\ a_{k+1}...a_n$$

se e só se existir a transição

$$\delta(q_1, a_k) = (q_2, b, R)$$

É possível uma movida

$$a_1a_2.....a_{k-1}q_1a_ka_{k+1}...a_n \vdash a_1a_2.....q_2a_{k-1}ba_{k+1}...a_n$$

se e só se existir a transição

$$\delta(q_1, a_k) = (q_2, b, L)$$

Diz-se que M parou partindo de alguma configuração inicial $x_1q_ix_2$ se

$$x_1q_ix_2 \overset{*}{\vdash} y_1q_ja y_2$$

para algum q_j e algum a, para os quais $\delta(q_j, a)$ seja indefinido.

Chama-se **computação** à sequência das configurações que levam a MT do estado inicial ao estado de paragem (halt).

A situação em que a MT entra num ciclo infinito, nunca parando, nem de lá podendo sair, denota-se por

$$x_1qx_2 \overset{*}{\vdash} \infty$$

Esta situação, de ciclo infinito, em que a máquina nunca mais pára, é muito importante na aplicação da teoria das MT.

9.3. Diagrama de estados ou de transição da MT

A notação da função de transição da MT é diferente no JFLAP e no DEM:

JFLAP e Linz $\delta: (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$ i.e. $\delta(q_0, b) = (q_1, b, R)$

DEM e Taylor $\delta: (Q \times \Gamma) \rightarrow (\Gamma \cup \{L, R\}) \times Q$ i.e. $\delta(q_0, \square) = (R, q)$

São necessárias, no DEM, mais instruções para a mesma funcionalidade : primeiro escreve, e depois desloca-se. Usaremos preferencialmente a do JFAP.

Os diagramas de estados são uma forma alternativa à escrita da função de transição, tal como nos outros autómatos estudados.

Exemplo 9.3.1.

Consideremos a MT, com uma fita completamente em branco, que faz o seguinte:

- inicializada lendo uma casa em branco,
- escreve os caracteres *ab* e

- pára lendo o carácter *a*.

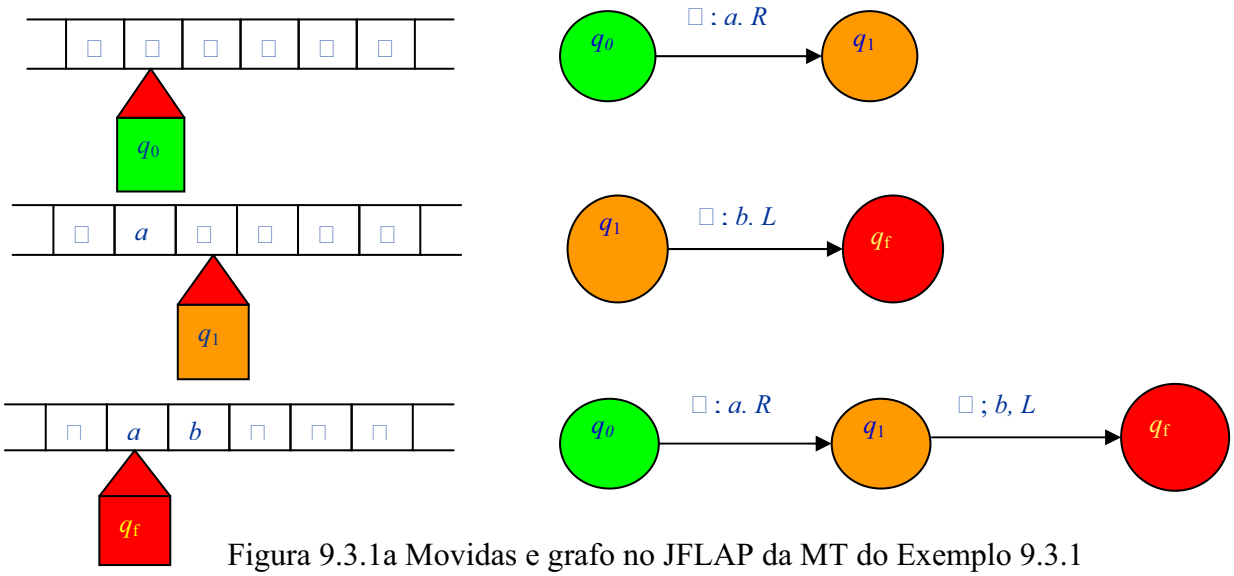


Figura 9.3.1a Movidas e grafo no JFLAP da MT do Exemplo 9.3.1

No DEM teremos um grafo diferente:

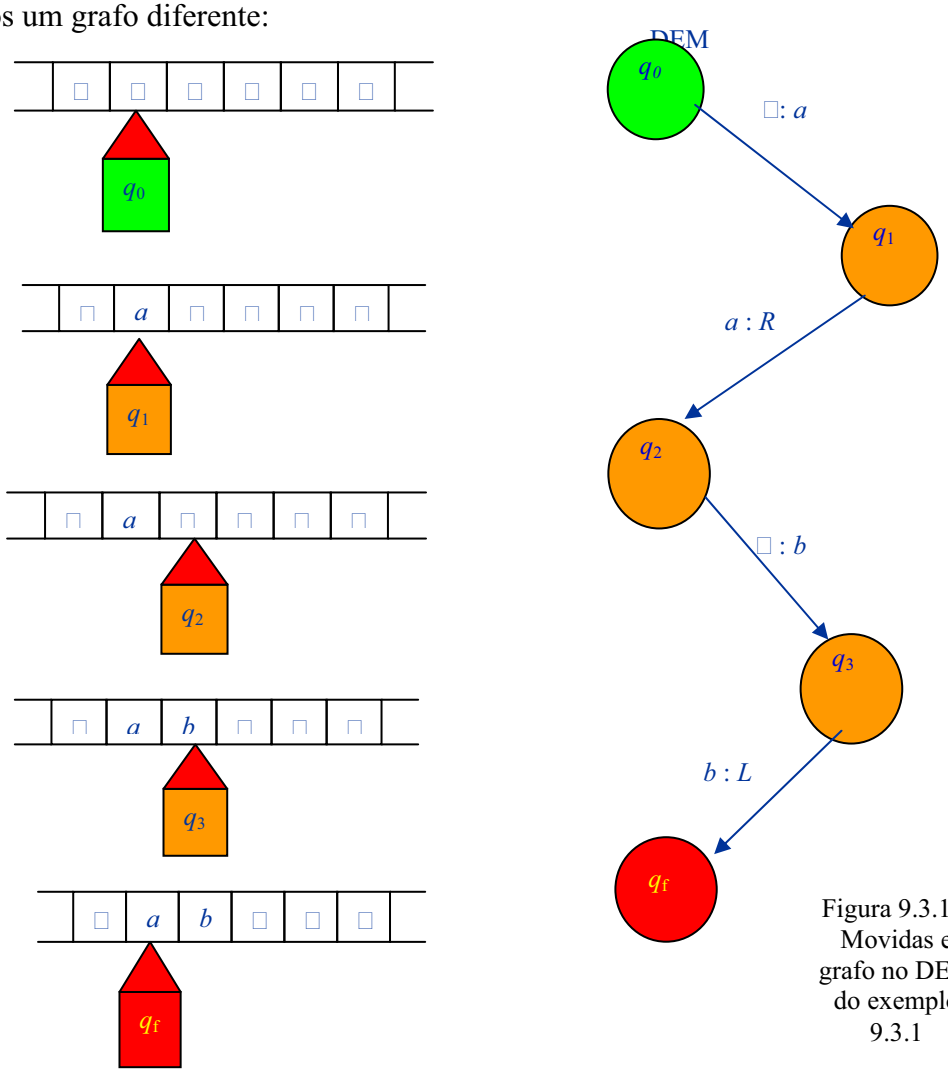
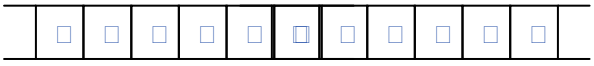


Figura 9.3.1b. Movidas e grafo no DEM do exemplo 9.3.1

Exemplo 9.3.2.

Para uma MT com a fita inicial vazia



que faz a MT da Figura seguinte ?

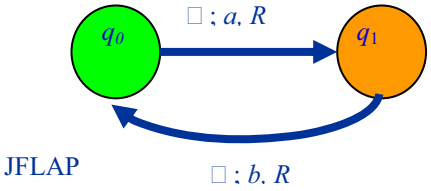


Figura 9.3 2a. MT do exemplo 9.3.2, no JFLAP.

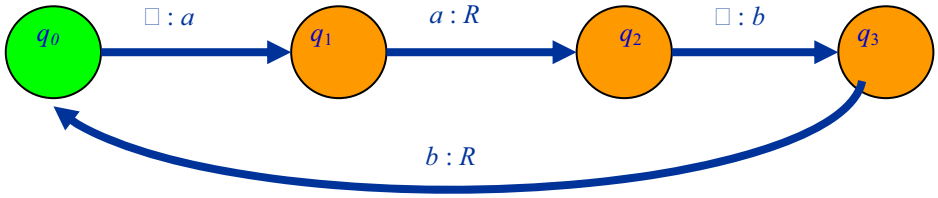


Figura 9.3.2b. MT do exemplo 9.3.2 no DEM

Estando no estado inicial q_0 lendo branco, escreve a, passa ao estado q_1 e vai para a direita, ficando assim a apontar para branco. Estando em q_2 apontando para branco, escreve b , passa a q_0 desloca-se para a esquerda. Agora está em q_0 lendo a na fita; não estando nada previsto para este caso, faz *halt* e aí fica para sempre.

9.4. MT como aceitador de linguagens.

Tal como nos autómatos anteriores, também as MT podem funcionar como aceitadores de linguagens.

O processo de aceitação de uma linguagem pela MT é o seguinte:

- dada uma cadeia *w* de caracteres do alfabeto de entrada (note-se que este não inclui λ, o carácter vazio), escrita na fita. A cadeia *w* não contém brancos,
- antes de *w* estão brancos e depois de *w* brancos estão,
- a MT inicializa-se no estado q_0 com a cabeça de leitura-escrita posicionada no carácter mais à esquerda de *w,*
- dá-se uma sequência de movidas,
- se a MT entrar num estado final e parar (halt), *w* é aceite,
- se a MT parar (halt) num estado não final, *w* não é aceite,
- se a MT não parar (entrando num círculo infinito), *w* não é aceite.

Note-se que há estados que têm características especiais, são os estados finais aceitadores.

Definição 9.4.1. Linguagem aceite pela MT.

Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, \square , F)$ uma MT. Então a linguagem aceite por M é

$$L(M) = \{ w \in \Sigma^+ : q_0 w \overset{*}{\vdash} x_1 q_f x_2 \text{ para algum } q_f \in F \text{ e } x_1, x_2 \in \Gamma^* \}$$

Se a cadeia tivesse brancos, a MT não saberia quando ela terminaria e teria que continuar a percorrer a fita até ao infinito.

Segundo Taylor (def. 1.3. p. 76), MT aceita uma linguagem se aceita todas as palavras de *L* e só essas (all and only the words in *L*).

Nota: a definição de Linz e de Hopcroft é equívoca quanto à necessidade de a MT ler toda a cadeia de entrada antes de fazer *halt*. A definição de Taylor (tem que parar apontando para um 1, com o resto da fita em branco, e 1 não pertence ao alfabeto de entrada) implica que toda a cadeia tenha que ser lida para ser, eventualmente, aceite.

Exemplo 9.4.1

Seja o alfabeto $\Sigma = \{0, 1\}$ e nele a linguagem $L = L(00^*)$, composta por todas as cadeias com um ou mais zeros (sem 1's).

Construir uma MT que aceite esta linguagem.

Trata-se de uma linguagem regular (porquê, caro leitor ?) e por isso não seria necessária uma MT para a aceitar. No entanto o exemplo tem objectivos didácticos.

Solução:

A primeira operação a fazer no projecto da MT (de qualquer MT) é definir um algoritmo, ou seja, uma estratégia de operações elementares que levem ao fim pretendido.

Admitamos que uma cadeia de zeros está escrita na fita, com brancos e só brancos à esquerda e à direita. Admitamos também que A MT se inicializa com a cabeça de leitura apontando para o primeiro carácter da cadeia. Estas hipóteses são as normalmente aceites.

Nós queremos que a máquina "varra" a cadeia, da esquerda para a direita, e a aceite se só tiver zeros e a rejeite de tiver algures algum 1. Então pode ser assim:

1º Lê o primeiro zero, deixa estar e desloca-se para a direita

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

Havendo vários zeros, ela ficou a apontar para o segundo zero. Então esta transição dá-se novamente, porque ela se mantém no estado q_0 , indo por aí além, até acabarem os zeros. Se existisse um 1 algures no meio dos zeros a MT pararia (não está definida uma transição para essa eventualidade).

2º Depois de ler todos os zeros há-de chegar ao primeiro branco à direita da cadeia. Aí pode seguir para a direita, deixando o branco tal como está. Como correu tudo bem até aqui, já leu a cadeia toda, deve passar a um estado aceitador, q_f:

$$\delta(q_0, \square) = (q_f, \square, R).$$

Para desenharmos um diagrama de estados para esta função de transição, no JFLAP ou no DEM, temos que ter em atenção a notação a usar . A notação de Taylor para a função de transição não é igual à de Linz:

Linz: $\delta: (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$ i.e. $\delta(q_0, b) = (q_1, b, R)$

Taylor: $\delta: (Q \times \Gamma) \rightarrow (\Gamma \cup \{L, R\}) \times Q$ i.e. $\delta(q_0, \square) = (R, q)$

O que obriga à criação de um estado adicional no DEM.

Por isso o diagrama de estados seguindo Linz e o JFLAP será

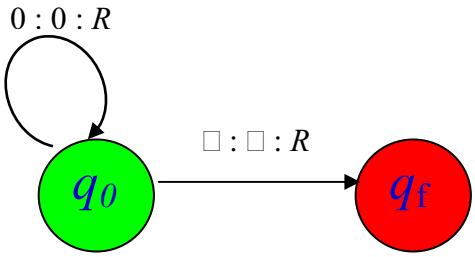


Figura 9.4.1. Grafo JFLAP da MT do exemplo 9.4.1.

Seguindo Taylor e o DEM teremos que criar mais estados:

$\delta(q_0, 0) = (q_1, 0)$

$\delta(q_1, 0) = (q_1, R)$

$\delta(q_1, \square) = (q_f, \square)$

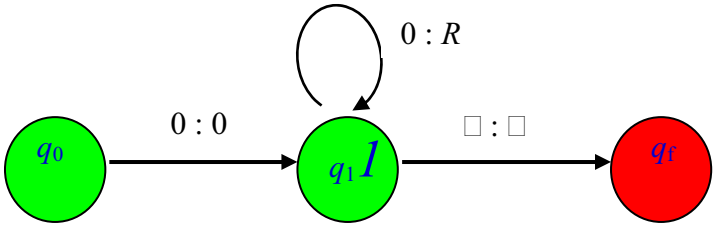


Figura 9.4.2. Grafo DEM da MT do exemplo 9.4.1

Note-se que δ é uma função parcial. Não é definida por exemplo $\delta(q_0, 1)$. Se aparece um 1, no estado q_0, a MT pára (halt) para sempre, indicando que w não é aceite, a menos que q_0 fosse um estado final. Mas a definição de estado final deve ser feita previamente à resolução do problema: um estado é final se antes dele só apareceram 0's e se já se chegou ao fim da cadeia, i.e., se a cabeça apontar para um branco. Não pode ser q_0 e por isso cria-se um outro estado q_f para estado final. Chega-se a ele apenas quando se leu a cadeia toda sem encontrar qualquer 1.

As MT podem reconhecer algumas linguagens que não são livres de contexto. São máquinas mais poderosas do que os autómatos estudados anteriormente. Voltaremos a este assunto mais tarde.

Exemplo 9.4.2. MT aceitadora de Linguagem $n_a=n_b$

Seja o alfabeto $\Sigma=\{a,b\}$ e a linguagem composta pelas cadeias com um número de a's igual ao número de b's, em qualquer ordem., por exemplo

ab, ba, abaaabbabbbb, *bbaababababbaaa*, etc.

Desenhar o grafo de uma MT capaz de reconhecer esta linguagem

No projecto de uma MT a questão primeira e decisiva é a da concepção do algoritmo a programar: como é que uma MT pode contar *a*'s e *b*'s ?

Uma solução possível: a máquina apaga um *a* e um *b* de cada vez; se no fim não sobrar nenhum carácter, é porque o número de *a*'s era igual ao número de *b*'s. Se sobrar um ou mais *a*'s (ou *b*'s) a cadeia não pertence à linguagem.

Podemos imaginar varrimentos sucessivos da cadeia , da direita para a esquerda) de tal modo que em cada varrimento substitui um *a* e um *b* por um* (uma forma de apagar). Chegando ao fim da cadeia (encontra um branco) volta ao princípio, deslocando-se para a esquerda até encontrar um branco.

Por exemplo para *ababab* teremos sucessivamente na fita

aaababbb **aa*abbb* ***a*a*bb* *****a**b* ********

bbabaa **b*baa* ****b*a* ******

aabbbabb **a*bbabb* *****babb* *****b**b*

Programa da MT

- Carrega-se a cadeia na fita
- Desloca-se a cabeça até ao primeiro carácter, que será *a* ou *b.* Suponhamos que é *a.*
- Apaga-se o *a* e substitui-se por *. Segue para a direita. Aí encontra *a*, *b* (ou *).

O grafo está na página seguinte.

Exemplo 9.4.3.

Seja a MT com $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \square\}$, $F = \{q_f\}$, $Q = \{q_0, q_1, q_f\}$

$\delta(q_0, \square) = (q_0, \square, R)$,
$\delta(q_0, b) = (q_1, b, R)$,
$\delta(q_1, a) = (q_1, a, R)$,
$\delta(q_1, b) = (q_1, b, R)$,
$\delta(q_1, \square) = (q_f, \square, R)$.

Que linguagem aceita ?

Analisando as transições, verifica-se que em q_0 com *b* passa a q_1 (e em q_0 com *a* pára). Em q_1 lê *a*'s e *b*'s por qualquer ordem e em qualquer número, avançando até ao fim da cadeia. Por

isso a linguagem aceita pela MT á L= L[b(a+b)*], todas as cadeias que se iniciam por b em $\{a,b\}$.

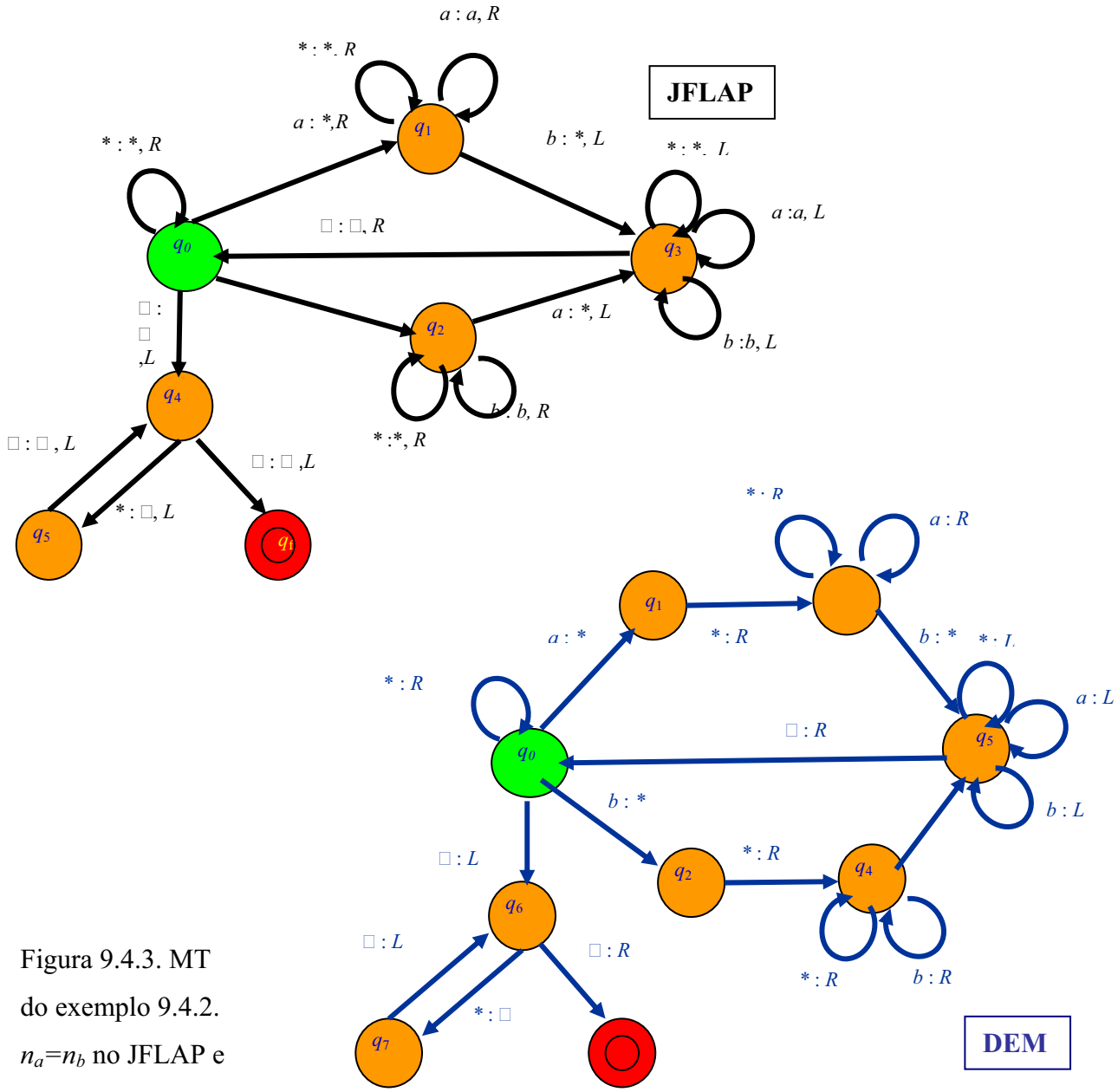


Figura 9.4.3. MT do exemplo 9.4.2. n_a=n_b no JFLAP e no DEM.

Exemplo 9.4.4

Construir no JFLAP uma MT que aceite a linguagem $L = \{ a^n b^m : n \geq 1, m \neq n \}$

Um algoritmo possível: depois de escrita a cadeia na fita

- em cada viagem da esquerda para a direita apaga um a e um b, marcando-os com *. Vai até ao fim e volta ao princípio.

- recomeça o ciclo até que ou acabem os *a*'s ou os *b*'s. Se sobrarem *a*'s ou *b*'s (ou exclusivo) a cadeia é aceite, terminando a computação no estado SIM. Caso contrário termina no estado NÃO.

A Figura 9.4.5 apresenta um grafo possível.

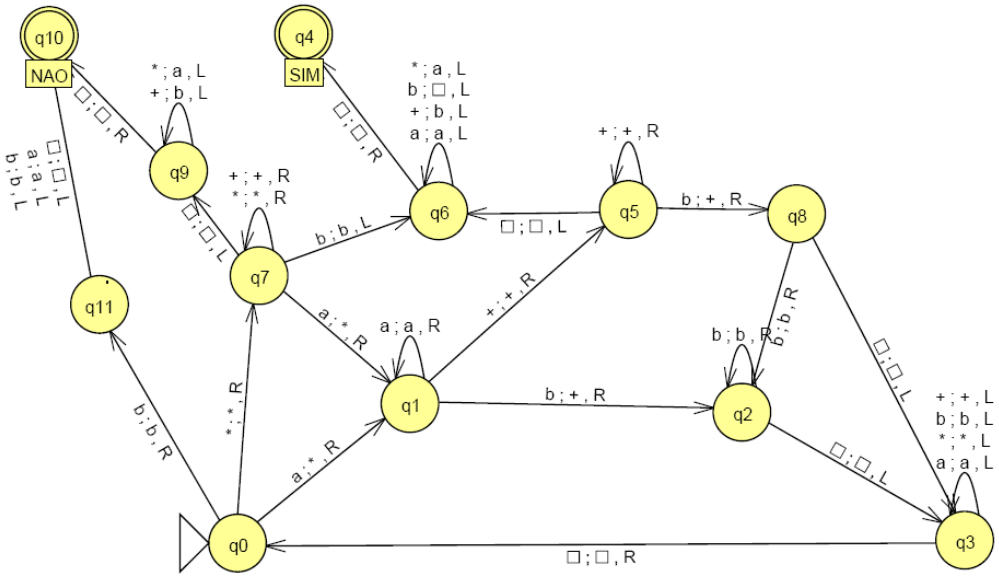


Figura 9.4.4. Esta MT decide se uma cadeia pertence à linguagem (termina em SIM) ou se não pertence à linguagem (termina em NÃO).

Exemplo 9.4.5. Desenhar uma MT que aceite a linguagem $L = \{ a^n b^n a^n b^n : n \geq 0 \}$

- procura o próximo *a* , marca-o, muda de estado e vai para a direita
- encontra o primeiro *b*, marca-o e muda de estado
- continua até ao próximo *a*, marca-o e muda de estado
- procura o próximo *b*, marca-o, muda de estado e regressa ao início
- continua até que acabem os a's e os b's
- aceita quando marcar todos e não sobrar nenhum
- não aceita de ou os *a*'s ou ao *b's* acabarem num dos segmentos da cadeia, sobrando nos outros.

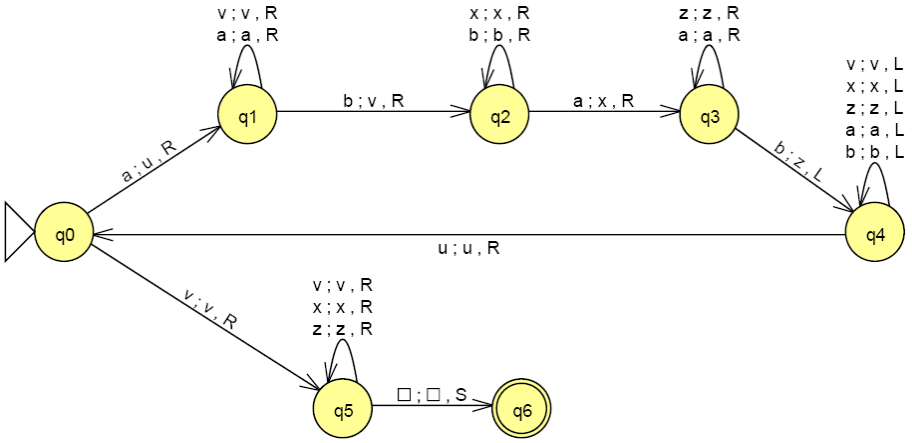


Figura 9.4.5. Implementação FLAPP do exemplo 9.4.5 pelo aluno António Damasceno em 2004-5.

Exemplo 9.4.6 O mesmo para o caso $L = \{ a^n b^{2n} : n \geq 1 \}$

- por cada *a* apaga dois *b*'s, usando para isso dois estados (a forma de contar dois *b*'s)

- repete até que acabem todos os *a*'s e *b*'s

-se sobrar algum *a* ou algum *b* não aceita

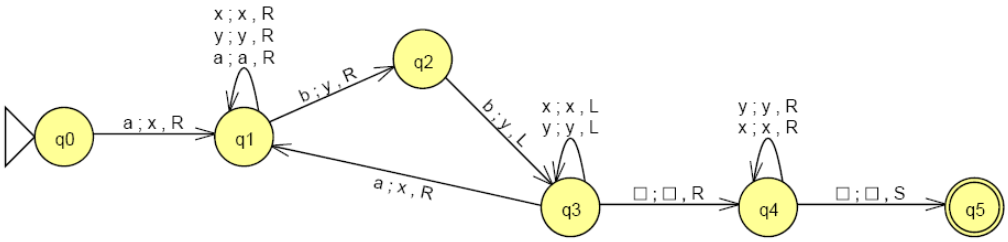
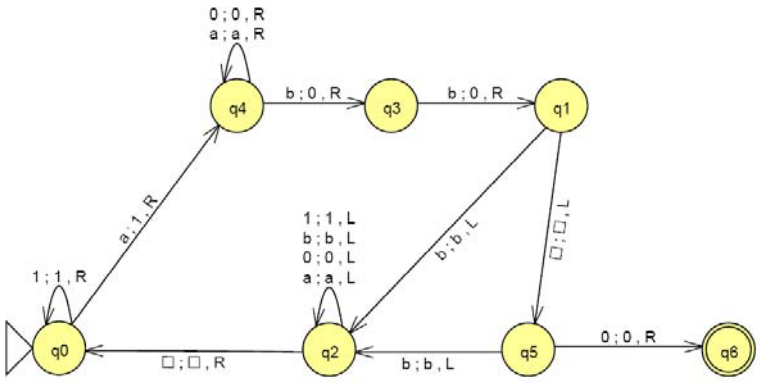


Figura 9.4.6a. JFLAP do exemplo 9.4.6 pelo aluno António Damasceno em 2004-5.

Uma outra implementação possível

Figura 9.4.6b. Implementação JFLAP pelo aluno Cristóvão Pires em 2004-5 (exemplo 9.4.6).



Exemplo 9.4.7 Para $L = \{ a^n b^m a^{n+m} : n \geq 0, m \geq 1 \}$ temos o algoritmo

- 1º apaga *n a*'s à esquerda e *n c*'s à direita; se os c's não chegarem não aceita;

- 2º apaga m b's à esquerda e m c's à direita; se os c's não chegarem ou se sobrarem, não aceita.

Para apagar os *n a*'s tem que percorrer *n* vezes um ciclo de apagar um; por isso cria-se um ciclo "apaga um *a* e vai apagar um *c* regressa". Se os *c*'s acabarem antes dos *a*'s não aceita.

Analogamente para apagar os *b*'s.

Para distinguir as diversas situações criam-se estados afectos a cada situação.

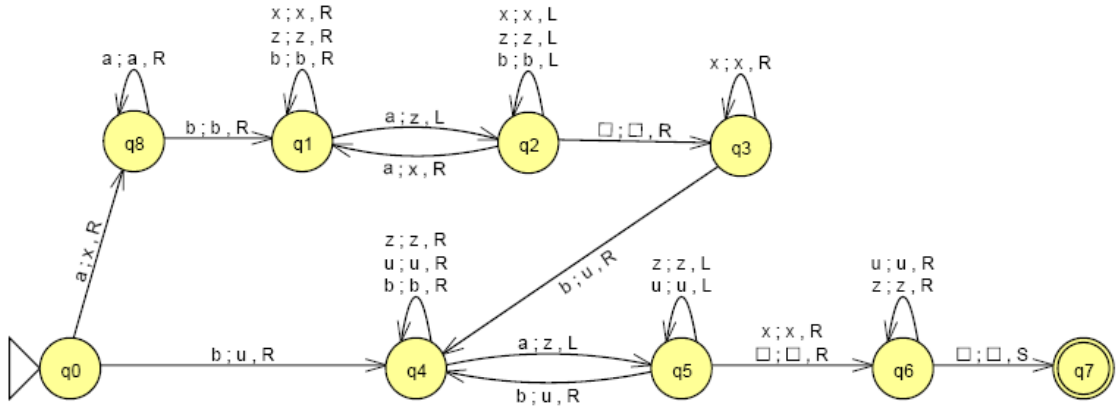


Figura 9.4.7. JFLAP do ex. 9.4.7 pelo aluno António Damasceno no ano lectivo 2004-5.

9.5 TM como transdutores

As MT são um modelo abstracto de computadores digitais em geral. Ora um computador é sobretudo um transdutor: transforma uma entrada numa saída, por exemplo fazendo um cálculo. As MT podem funcionar também como transdutores.

A entrada para uma computação é composta por todos os símbolos não brancos escritos na fita no instante inicial.

A saída da computação é o que resultar escrito na fita, depois de todas as movidas da MT.

Com estes conceitos pode-se encarar a MT *M*, enquanto transdutor, como implementando uma função f definida por

$$\widehat{w} = f(w),$$

e portanto com saída $\widehat{w}$, desde que se verifique que partindo de *w* no estado inicial após a computação de *f* chega a um estado final q_f

$$q_0 w \overset{*}{\vdash\!\!\!-}_M q_f$$

As especificações do estado final são feitas pelo programador.

Será que qualquer função é computável por uma MT ?

Definição 9.5.1. Função computável.

Uma função *f* com o domínio *D* diz-se **Turing – computável** ou simplesmente **computável** se existir alguma máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ tal que

$$q_0 w \overset{*}{\vdash\!\!\!-}_M q_f\, f(w), \qquad q_f \in F$$

para toda a $w \in D$.

Esta definição pressupõe que a MT pára, depois de calcular *f*(*w*), apontando para o primeiro carácter do resultado do cálculo.

Todas as funções matemáticas usuais, por mais complicadas que sejam, são Turing-computáveis. Esta afirmação não se pode provar, mas acredita-se que assim seja porque até hoje ninguém foi capaz de propor uma função matemática que não fosse computável. Vejamos alguns exemplos.

Exemplo 9.5.1. Adição de inteiros (representação unária) (Linz, 233

Para adicionar números inteiros com uma MT, primeiro faz-se a sua representação unária, só com 1's. Depois temos que especificar como se escrevem as duas parcelas na fita da MT. Admitamos que as separamos por um 0, ficando brancos à esquerda e à direita:

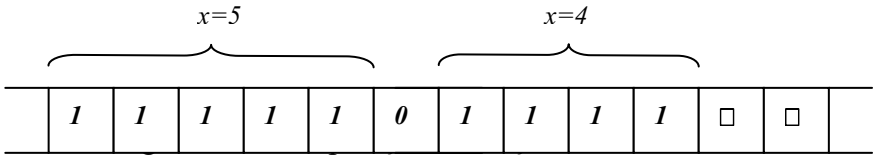


Figura 9.5.1. Preparação da fita para a adição

e que no fim da computação o resultado deve ser apresentado em unário (como uma sequência de 1s)

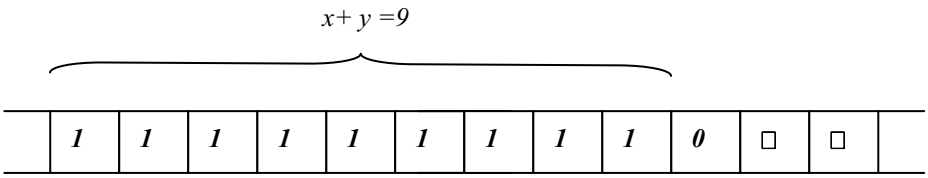


Figura 9.5.2. Resultado final da adição

Definido o início e o fim, temos agora que escrever o programa da MT.

Outros exemplos:

Exemplo 9.5.2. A máquina copiadora (Linz, 234).

Vamos projectar uma MT que copie um número escrito em binário na fita, escrevento a cópia à direita, deixando um branco entre os dois.

A MT inicia-se no primeiro 1 de uma cadeia de *n* 1's e pára no primeiro 1 de uma cadeia ininterrupta de *n* 1's ,seguida por um branco, seguido por uma cadeia ininterrupta de n 1's, com o indicado na fig. 9.5.4.

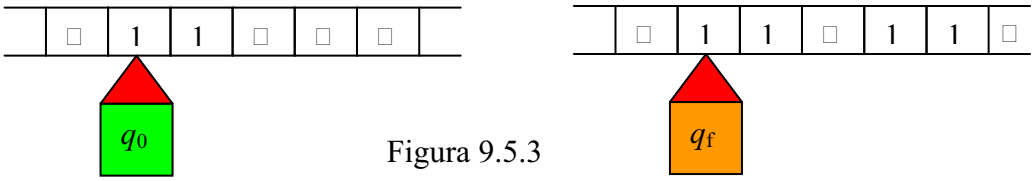


Figura 9.5.3

Este é também o Ex. 1.3.2 de Taylor, Copying Machine.tm implementada em *DEM*, fita 8.tt

Exemplo 9.5.3.

Comparação de dois números positivos, *x* e *y*, em representação unária, Linz 235.

Escrevem-se os números na fita separados por um 0, por exemplo, comparar 4 e 6,

11110111111

Um algoritmo possível

- apagar um 1 da primeira parte e outro 1 da segunda, até os 1s acabarem numa das partes

Caso 4&6

$$11110111111 \Rightarrow *1110*11111 \Rightarrow **110**1111 \Rightarrow ***10***111 \Rightarrow ****0****11$$

Agora quando procura mais 1 do lado esquerdo não encontra, passa pelo zero e continua até ao último 1, o que provoca a transição para o estado $x<y$.

Caso 6&4

$$11111101111 \Rightarrow *111110*111 \Rightarrow **11110**11 \Rightarrow ***11101*** \Rightarrow ****110****$$

Agora ao apagar mais um 1 do lado esquerdo e procurar o outro 1 do lado direito, não o encontra, chega ao branco e passa ao estado $x>y$.

No caso de igualdade, quando acabam os 1's em ambos os lados, passa ao estado $x=y$.

Fica o leitor desafiado a desenhar o respectivo grafo, usando o JFLAP.

Exemplo 9.5.4

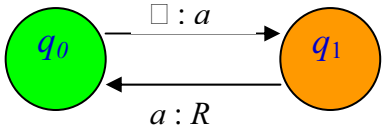


Figura 9.5.4

Que faz ?

1º Inicializando-se numa fita vazia, lê um branco, substitui-o por um a, desloca-se para a direita e repete estas operações indefinidamente.

2º Percorre a fita até ao infinito sem escrever nada.

Exemplo 9.5.5. E esta ?

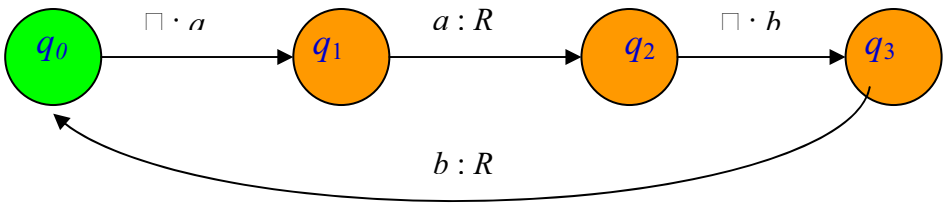


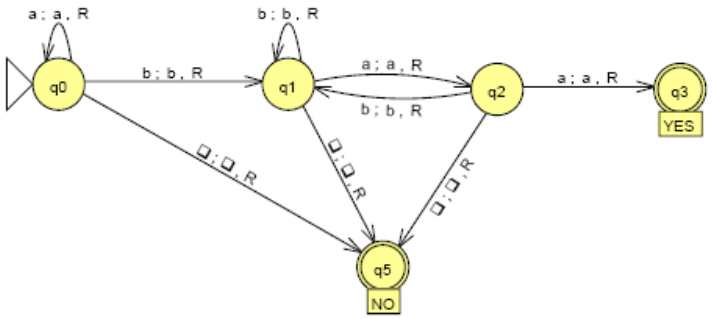
Figura 9.5.5

Escreve sucessivamente *abababababab*.....

Exemplo 9.5.6.

Construa uma MT que decida a linguagem em $\Sigma=\{a,b\}$ composta por todas as cadeias que contenham a subcadeia *baa* (isto é, que diga se sim ou não uma cadeia pertence à linguagem).

Figura 9.5.6



Note-se que esta linguagem é regular, e por isso também se pode desenhar um DFA que faz o mesmo.

Exemplo 9.5.7

Desenhe uma MT que calcula a função $f(x)=2x$, estando x em unário.

Primeiro devemos conceber um algoritmo que resolva o problema. Não existe uma única solução. Uma delas poderá ser

1º- substituir cada 1 por um x e de seguida adicionar um y no fim da cadeia. Por exemplo para $f(111)$= 111111 a fita terá sucessivamente

$$111 \Rightarrow x11y \Rightarrow xx1yy \Rightarrow xxxyyy$$

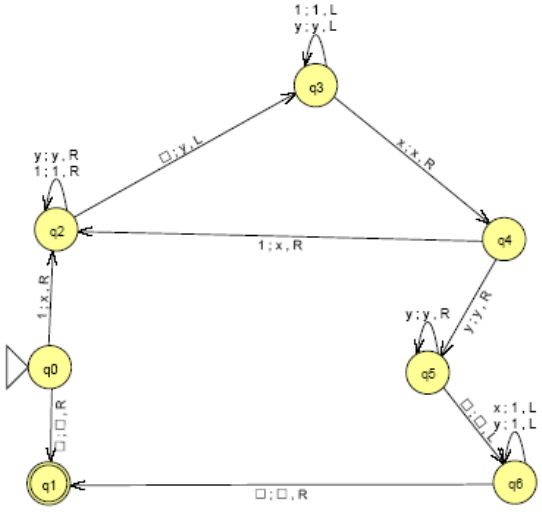
2º - substituir cada x por 1 e cada y por 1

$xxxyyy \Rightarrow\Rightarrow xxxyy1 \Rightarrow xxxy11 \Rightarrow xxx111$

$\Rightarrow xx1111 \Rightarrow x11111 \Rightarrow 111111$

O grafo da MT será o da Figura.

Figura 9.5.7



Exemplo 9.5.8 Desenhar uma MT que conte o número de *a*'a em cadeias no alfabeto {*m, a, r, g, a, r, i, d, a*}.

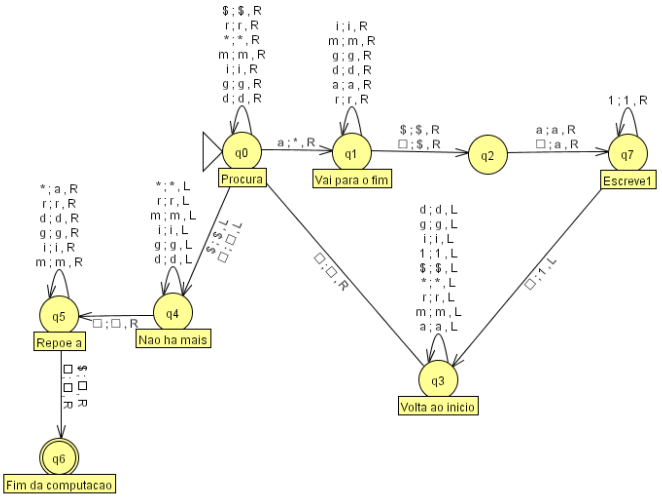


Figura 9.5.8. MT que conta os *a*'s em cadeias de {*m,a,r,g,a,r,i,d,a*}

9.6. Combinações de máquinas de Turing para tarefas complicadas

Muitas funções complicadas podem-se decompor em sequências de funções simples computáveis (toda a análise numérica visa esse fim). Tal conceito também se pode aplicar ao nível da Turing-computabilidade.

Pode-se projectar uma máquina de Turing composta por combinações de máquinas de Turing elementares a fim de realizar operações mais complexas.

Exemplo 9.6.1 (Linz)

Seja a função *f* definida por $f(x, y) = \begin{cases} x + y, & \text{se } x \geq y \\ 0, & \text{se } x < y \end{cases}$

Podemos decompô-la em três operações: uma comparação, uma soma, a um apagamento, que se podem implementar pelos três blocos da Figura 9.6.1.

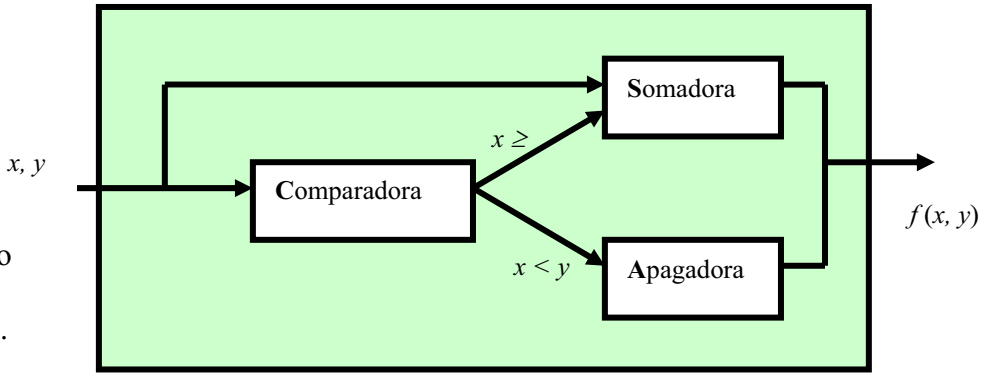


Figura 9.6.1. Implementação da função do exemplo 9.6.1.

Agora cada bloco pode ser executado por uma MT. A função total resulta da combinação das três MTs.

9.7. A tese de Turing

Nos anos 30 do séc. XX Turing conjecturou que qualquer computação que possa ser implementada por processos mecânicos (i.e., por uma máquina) pode também ser implementada por uma máquina de Turing. Pode ser interpretado ao contrário: uma computação é mecânica se e só se puder ser executada por uma máquina de Turing.

Alguns argumentos a favor da tese de Turing:

1- Qualquer computação que possa ser feita por qualquer computador digital existente também pode ser feita por uma máquina de Turing.

2- Ninguém conseguiu ainda encontrar um problema, resolúvel por um qualquer algoritmo, para o qual não possa ser escrito um programa para uma máquina de Turing.

3- Foram propostos modelos alternativos para a computação mecânica, mas nenhum deles é mais poderoso do que a máquina de Turing.

A tese de Turing é (ainda) uma lei básica das ciências da computação. Nesses tempos o desenvolvimento de um algoritmo consistia fundamentalmente na escrita de um programa de uma MT que o resolvesse. Mesmo actualmente um algoritmo pode-se definir como uma MT,

Se considerarmos uma função qualquer, *f*, com um domínio D e o contradomínio *R*, para um elemento *d* do domínio escrito na fita de uma MT, escreve-se um programa tal que iniciando-se a MT apontando para o primeiro símbolo de *d*, termina uma computação depois de escrever na fita o valor de *f*(*d*). Teremos assim a seguinte definição de algoritmo.

Definição 9.7.1. Algoritmo.

Um algoritmo para uma função $f : D \rightarrow R$ é uma máquina de Turing *M* que,

- dada como entrada um qualquer $d \in D$ na sua fita
- pára (*halt*) com uma resposta correcta para $f(d) \in R$ na sua fita. Requer-se que

$$q_0 d \overset{*}{\vdash\!\!-}_M q_f\, f(d), \qquad q_f \in F$$

para todo o $d \in D$.

No DEM existe uma livraria de MTs que executam diversos algoritmos. A tabela seguinte enumera-os. Alguns deles recorrem a uma MT especial, com várias fitas (tapes). No Capítulo 10 abordaremos essas arquitecturas especiais da MT, que no entanto não alteram as suas capacidades computacionais, apenas tornam mais fácil a programação de algoritmos.

Tabela 9.7.1 Lista de exemplos do *DEM*. Os exemplos estão em Taylor.

balpatent	balanced parenthesis
beaver	busy beaver function w/ 5 arguments
c-interp	C- language Compiler/Interpreter Prototype
Cnfsat	CNF- Sat (Example 8.5.1) (5 tapes)
Convert	Convert to unary
Copying	Copia cadeia de caracteres
Cvp	Circuit Value Problem (example 8.8.1)
div2mult	f(n)=n div 2 (multitape machine)
div2single	f(n)=n div 2 (single tape)
Ex12-3-1	Language Emulator (Example 12.3.1)
Ex2-2-3	Example 2.2.3 $w=w^R$, odd palindromes
Ex2-6-2	nondeterm accepter a**(2n)+a**(3n), n≥0
Ex2-6-3	nondeterm accepter a**(n**2) n≥0
Ex2-6-5	nondeterm f(n) = cubicroot(n), se cubo perfeito, indefinida caso contrário
Ex4.5.1	Markov algorithm simulator 4 tapes
Factoria	f(n)= n!
Fspacon	f(n) = n**2 is fully space-constructible
God	Exodus 3.13-14 I am who I am …
Multiply	f(n,m)=nm the multiplication machine

N^n	n^n, the n to n power
Nntravsa	Nearest-Neighboor Traveling Salesman Machnine/Exxample 8.10.2
Palindro	Palindromes
Recdesct	Recursive-descent Parser
Recognzr	Reconhecimento da cadeias (pares, impares, caráter do meio, palindromes)
Reverse	reverse word
Reverse2	reverse word multitape
Samenumb	same numbare of a's as b's single tape
Samenumb4t	same number of a's and b's 4 tapes
Selctstr	select a string
Sqrt	square root (perfect square)

Exemplo 9.7.1 Desenvolver um algoritmo para calcular n^3

Uma solução possível:

1º - Escreve-se na fita *n*0*n*0*n*.

2º - Copia *n* à direita um número *n* de vezes, obtendo-se assim *n*x*n*=*m*.

3º - Copia depois *m* um número *n* de vezes, obtendo-se *m*x*n*=*n*x*n*x*n*.

Exemplo 9.7.2. Desenvolver as linhas gerais de um algoritmo para calcular *n*!

Com sabemos *n*!=*n*x*(n*-1*)*x…..x2x1. Para isso na MT

- escreve-se *n* na fita, em unário
- copia-se *n* à direita
- decrementa-se *n*, (subração de 1)
- multiplica-se (*n*-1)*n* = *p*
- decrementa-se *n*-1, vindo *n*-2
- multiplica-se (*n*-2)*p* obtendo-se um novo valor de *p*.
-…
- até se atingir *n*-*i*=1, e nesta altura o valor de *p* é o factorial.

Um programa para isso terá uma extensão considerável …

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

http://www.turing.org.uk/turing/Turing.html (Alan Turing Home Page).

http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Turing.htm.