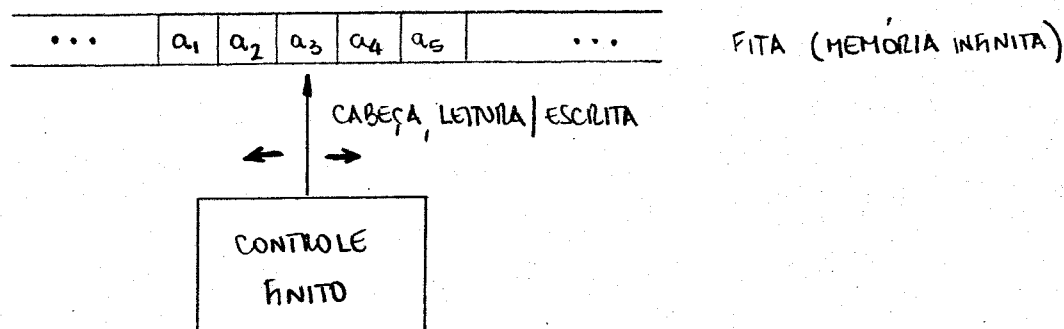


CAPÍTULO 4 - MÁQUINAS DE TURING



Nem movimento, dependendo do símbolo lido e do estado do controle finito, a máquina de Turing:

- troca de estado
- escreve um símbolo na fita
- move sua cabeça de leitura/escrita uma posição para a direita ou uma posição para a esquerda ou a mantém na mesma posição.

definição 4.1. Uma MÁQUINA DE TURING SIMPLES é uma 6-upla $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ onde:

- Q - conjunto finito não vazio de estados
- Σ - alfabeto de entrada
- Γ - alfabeto da fita ($\Sigma \subset \Gamma$; $b \in \Gamma$; $b \notin \Sigma$)
- $q_0 \in Q$ - estado inicial
- $F \subseteq Q$ - conjunto de estados finais

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{ \leftarrow, \downarrow, \rightarrow \}$ - função de transição

($\delta(q, a)$ é indefinida se $q \in F$; $\forall a \in \Gamma$)

definição 4.2 . Uma configuração de $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ é um par $(q, \alpha \hat{a} \beta)$ onde, $q \in Q$; $\alpha, \beta \in \Gamma^*$; $a \in \Gamma$ e o símbolo " $\hat{}$ " indica a posição da cabeça leitora/escrita.

definição 4.3 . (transições da máquina de Turing)

(a) se $\delta(q, a) = (p, b, \downarrow)$ então $(q, \alpha \hat{a} \beta) \vdash (p, \alpha \hat{b} \beta)$

(b) se $\delta(q, a) = (p, b, \rightarrow)$ e $\beta = c\beta'$
então

$$(q, \alpha \hat{a} \beta) \vdash (p, \alpha b \hat{c} \beta')$$

(c) se $\delta(q, a) = (p, b, \leftarrow)$ e $\alpha = \alpha'c$
então

$$(q, \alpha \hat{a} \beta) \vdash (p, \alpha' \hat{c} b \beta)$$

onde : $q, p \in Q$; $\alpha, \beta, \alpha', \beta' \in \Gamma^*$; $a, b, c \in \Gamma$

definição 4.4 . Uma configuração $(q, \alpha \hat{a} \beta)$ é **TERMINAL** se $\delta(q, a)$ for indefinida. Se $q \in F$ a configuração terminal é **FINAL**.

definição 4.5 . O processamento da máquina $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ com entrada $w = c_1 \dots c_n$ é uma sequência de

configurações

$$(q_0, \alpha_0 \hat{a}_0 \beta_0), (q_1, \alpha_1 \hat{a}_1 \beta_1), \dots$$

tal que: (a) $(q_0, \alpha_0 \hat{a}_0 \beta_0) = (q_0, \hat{c}_1 \dots c_n)$

(b) $(q_i, \alpha_i \hat{a}_i \beta_i) \vdash (q_{i+1}, \alpha_{i+1} \hat{a}_{i+1} \beta_{i+1})$

(c) se a sequência for finita, então a última configuração é terminal

onde: $\alpha_i, \beta_i \in \Gamma^*$ ($i=0,1,\dots$); $c_1 \dots c_n \in \Sigma^*$; $a_0, a_1, \dots \in \Gamma$

definição 4.6. A linguagem aceita por M (ou processamento aceitável) é definida por:

$$L(M) = \{ w \in \Sigma^* / (q_0, \hat{w}) \vdash^* (q, \hat{\alpha}) ; q \in F, \alpha \in \Gamma^+ \}$$

onde \hat{w} e $\hat{\alpha}$ indicam que a posição da cabeça é sobre o símbolo mais a esquerda de w e α , respectivamente.

EXEMPLO. Máquina de Turing que aceita $L = \{ 0^n 1 0^n / n \geq 0 \}$

$$M = (\{ q_0, q_1, q_2, q_3, q_4, q_5, q_6 \}, \{ 0, 1 \}, \{ 0, 1, b \}, \delta, q_0, \{ q_6 \})$$

com δ dada por:

$\delta(q_0, 0) = (q_1, b, \rightarrow)$	apaga o <u>0</u> mais à esquerda
$\delta(q_0, 1) = (q_5, 1, \rightarrow)$	não existe <u>0</u> à esquerda
$\delta(q_1, 0) = (q_1, 0, \rightarrow)$	avança na esperança de encontrar <u>1</u>
$\delta(q_1, 1) = (q_2, 1, \rightarrow)$	fui encontrado <u>1</u> ; deve ser apagado o <u>0</u> mais à direita
$\delta(q_2, 0) = (q_2, 0, \rightarrow)$ $\delta(q_2, b) = (q_3, b, \leftarrow)$	avança para o símbolo mais à direita
$\delta(q_3, 0) = (q_4, b, \leftarrow)$	apaga o <u>0</u> mais à direita
$\delta(q_4, 0) = (q_4, 0, \leftarrow)$ $\delta(q_4, 1) = (q_4, 1, \leftarrow)$ $\delta(q_4, b) = (q_0, b, \rightarrow)$	volta para o símbolo mais à esquerda
$\delta(q_5, b) = (q_6, b, \downarrow)$	aceita a cadeia

definição 4.7 . Seja L uma linguagem . L é LINGUAGEM RECURSIVAMENTE ENUMERÁVEL se $L = L(M)$, para alguma máquina de Turing M .

TÉCNICAS PARA CONSTRUÇÃO DE MÁQUINAS DE TURING SIMPLES

1. FITA COM VÁRIAS TRILHAS

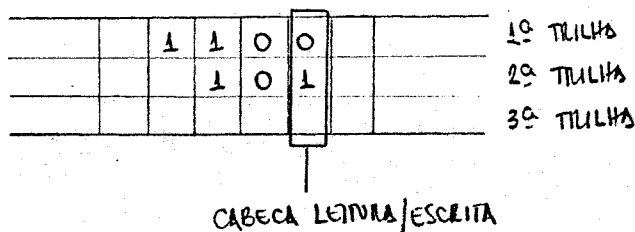
Pode-se imaginar que a fita da MTS é construída em k trilhas,

para algum inteiro k . Isto é equivalente a imaginar que os símbolos que aparecem na fita são k -uplas, com um componente colocado em cada trilha.

EXEMPLO: Máquina de Turing que calcula $n-m$, onde n e m são inteiros escritos em binário e $n \geq m$.

Pode-se imaginar uma MT com fita de 3 trilhas, com n colocado na primeira trilha e m na segunda. A terceira trilha é inicialmente em branco. Ao final, o resultado de $n-m$ estará na 3ª trilha, e as outras duas trilhas estarão em branco. Por exemplo, seja $n = 12$ (1100) e $m = 5$ (101):

CONFIGURAÇÃO
INICIAL

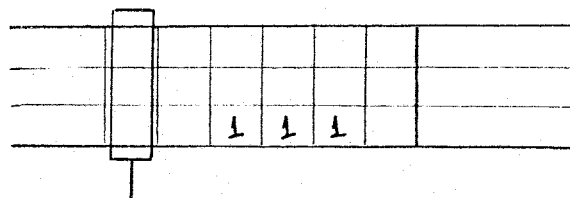


	(00b)	(01b)	(10b)	(11b)	(0bb)	(1bb)
$\Rightarrow (q_0)$	$q_0, bb0, \leftarrow$	$q_1, bb1, \leftarrow$	$q_0, bb1, \leftarrow$	$q_0, bb0, \leftarrow$	$q_0, bb0, \leftarrow$	$q_0, bb1, \leftarrow$
q_1	$q_1, bb1, \leftarrow$	$q_1, bb0, \leftarrow$	$q_0, bb0, \leftarrow$	$q_1, bb1, \leftarrow$	$q_1, bb1, \leftarrow$	$q_0, bb0, \leftarrow$

Para a situação acima tem-se:

$$\begin{aligned}
 & (q_0, (1bb)(11b)(00b)(\hat{01b})) \vdash (q_1, (1bb)(11b)(00b)(bb1)) \vdash \\
 & \vdash (q_1, (1bb)(\hat{11b})(bb1)(bb1)) \vdash (q_1, (\hat{1bb})(bb1)(bb1)(bb1)) \\
 & \vdash (q_0, (bb0)(bb1)(bb1)(bb1))
 \end{aligned}$$

ou seja:

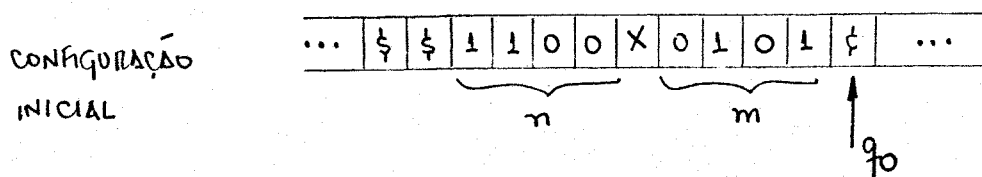


CONFIGURAÇÃO
FINAL

2. SUBROTINAS

Pode-se considerar, sob determinadas condições, que uma MT M_1 é uma subrotina de outra MT M_2 . Para "chamar" M_1 a máquina M_2 entra no estado inicial de M_1 . Quando M_1 entra num estado final (estado de "retorno"), M_2 "recupera o controle".

EXEMPLO: Máquina de Turing para calcular $n+m$, onde n e m são inteiros escritos em binário. Sejam n e m do mesmo tamanho (se os tamanhos forem diferentes é sempre possível acrescentar zeros à esquerda do menor). Por exemplo, seja $n = 1100$ (12) e $m = 0101$ (5).



Considere o algoritmo:

(a) se $m = 0$, então n é a soma

(b) $m = m - 1$; $n = n + 1$; voltar para (a)

SUBROTINA SUBTRAI-UM :

$$\delta(t, 0) = (t, 1, \leftarrow)$$

$$\delta(t, 1) = (r_1, 0, \leftarrow)$$

$$\delta(t, x) = \text{fim}$$

SUBROTINA SOMA-UM :

$$\delta(\lambda, 0) = (r_2, 1, \rightarrow)$$

$$\delta(\lambda, 1) = (\lambda, 0, \leftarrow)$$

$$\delta(\lambda, \phi) = (r_2, 1, \rightarrow)$$

onde t, λ são estados iniciais e r_1, r_2 são estados de retorno.
Logo, o programa da máquina de Turing será :

$$\delta(q, \phi) = (t, \phi, \leftarrow)$$

chama SUBTRAI-UM

$$\delta(r_1, a) = \begin{cases} (r_1, a, \leftarrow) & \text{se } a \neq x \\ (\lambda, a, \leftarrow) & \text{se } a = x \end{cases}$$

prepara-se para chamar SOMA-UM

chama SOMA-UM

$$\delta(r_2, a) = \begin{cases} (r_2, a, \rightarrow) & \text{se } a \neq \phi \\ (q, a, \downarrow) & \text{se } a = \phi \end{cases}$$

prepara-se para voltar

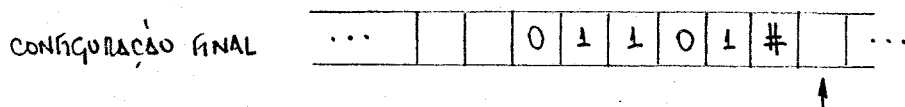
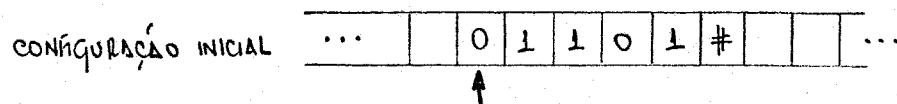
volta ao início

3. MEMÓRIA EM ESTADOS

Pode-se imaginar que os estados do controle finito armazenam uma quantidade finita de informações. Isto pode ajudar a compreender o programa da MT.

EXEMPLO : Máquina de Turing para fazer deslocamento da cadeia de entrada 1 posição para a direita.

Seja, por exemplo, $n = 01101$

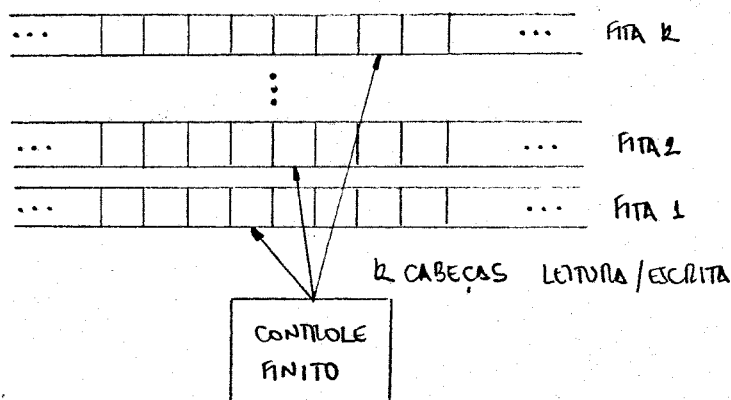


O programa para essa máquina pode ser :

$$\delta([q_0], x) = ([q_0 x], b, \rightarrow) \quad x \in \{0, 1\}$$

$$\delta([q_0 x], y) = ([q_0 y], x, \rightarrow) \quad y \in \{0, 1, \#\}$$

MÁQUINA DE TURING COM VÁRIAS FITAS



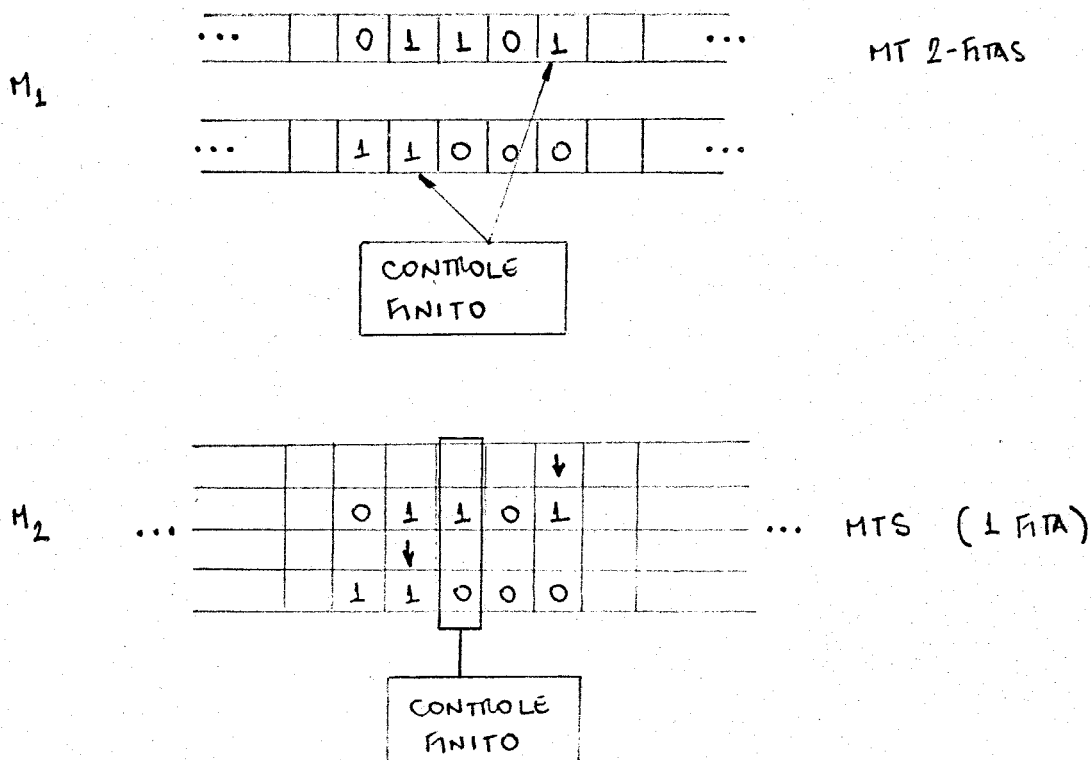
Num único movimento, a MT k -FITAS, dependendo do estado do controle finito e dos k símbolos lidos pelas cabeças de leitura/escrita :

- troca de estado
- escreve um novo símbolo em cada uma das posições das cabeças
- move cada uma das cabeças, independentemente, uma posição para a direita ou para a esquerda ou a mantém estacionária

Teorema 4.1 . Seja L uma linguagem. Se L é aceita por uma MT k -FITAS, então L é aceita por uma MTS.

Prova. Seja M_1 a MT k -FITAS que aceita L .
Construir MTS, M_2 , com uma fita de $2k$ trilhas. Cada fita de M_1 irá corresponder a um par de trilhas em M_2 : uma das trilhas armazena o conteúdo da fita de M_1 e a outra trilha é toda branca exceto por um marcador que armazena a posição da cabeça correspondente de M_1 .

Seja, por exemplo:



Um movimento de M_1 é simulado por M_2 da seguinte maneira (considere que a cabeça de leitura/escrita de M_2 ao iniciar a simulação de um movimento de M_1 encontra-se sobre o símbolo mais a esquerda de sua fita)

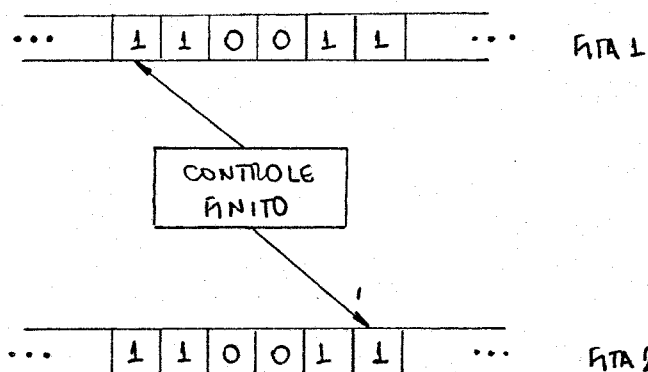
- (1) M_2 desloca sua cabeça para a direita até encontrar os k marcadores \downarrow . Deste modo, M_2 descobre os k símbolos lidos por M_1 neste passo e os armazena em estado
- (2) uma vez descobertos esses símbolos, M_2 desloca a cabeça para a esquerda escrevendo os símbolos que M_1 escreveria neste passo, nas posições correspondentes.
- (3) deslocando sua cabeça para a direita, M_2 atualiza os marcadores \downarrow que correspondem às cabeças de M_1 que se deslocam para a direita neste passo.
- (4) finalmente, deslocando sua cabeça para a esquerda, M_2 atualiza os marcadores \downarrow que correspondem às cabeças de M_1 que se deslocam para a esquerda neste passo, posicionando também sua cabeça para simular o próximo passo de M_1 e refletindo no estado do controle finito a mudança de estado de M_1 .

EXEMPLO: MT que aceita a linguagem $L = \{ ww^R / w \in (0+1)^* \}$

Pode-se construir facilmente um reconhecedor para L , considerando-se uma MT 2-FITAS:

- a entrada é colocada em ambas as fitas
- as duas fitas são lidas em sentidos opostos
- o comprimento da entrada é verificado ser par.

Seja, por exemplo : $w = 110011 \in L$



Esta máquina pode ser definida por :

$$\delta : Q \times \Gamma \times \Gamma \rightarrow Q \times \Gamma \times \{ \leftarrow, \downarrow, \rightarrow \} \times \Gamma \times \{ \leftarrow, \downarrow, \rightarrow \}$$

onde :

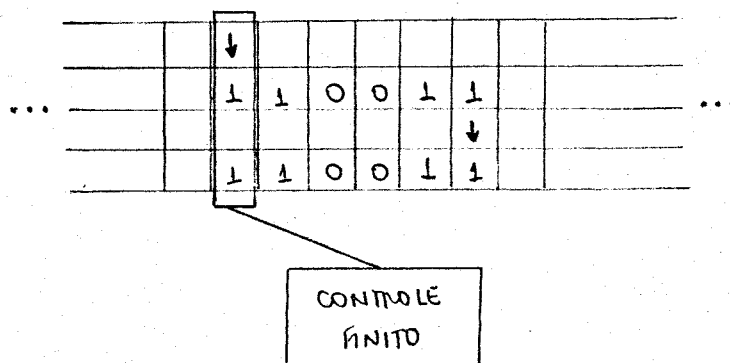
$$\delta(q_0, a, a) = (q_1, a, \rightarrow, a, \leftarrow)$$

$$\delta(q_1, a, a) = (q_0, a, \rightarrow, a, \leftarrow)$$

verifica os
símbolos e o
comprimento

$$\delta(q_0, b, b) = \text{ACEITA!}$$

Para simular essa MT, pode-se construir uma MTS com uma fila de 4 trilhas



Seja a MTS definida por $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ onde

$$Q = \{q_0, q_1, q_2, q_3, [0], [1], [2], [3]\} ; \quad \Sigma = \{0, 1\} ; \quad F = \{q_0\}$$

$$\Gamma = \text{conjunto de quádruplas do tipo } (m_1, a_1, m_2, a_2) ; \quad m_1, m_2 \in \{\square, \downarrow\} \\ a_1, a_2 \in \Sigma$$

(o símbolo \square representa uma célula em branco)

e δ dada por :

(1)	$\begin{aligned} \delta(q_0, (\square x \square y)) &= (q_0, (\square x \square y), \rightarrow) \\ \delta(q_0, (\downarrow x \square y)) &= ([x], (\downarrow x \square y), \rightarrow) \\ \delta([x], (\square x \square y)) &= ([x], (\square x \square y), \rightarrow) \\ \delta([x], (\square y \downarrow x)) &= (q_1, (\square y \downarrow x), \downarrow) \end{aligned}$	onde: $x, y \in \Sigma$
(2)	$\begin{aligned} \delta(q_1, \alpha) &= (q_1, \alpha, \leftarrow) \quad \alpha \neq (\square \square \square \square) \\ \delta(q_1, (\square \square \square \square)) &= (q_2, (\square \square \square \square), \rightarrow) \end{aligned}$	
(3)	$\begin{aligned} \delta(q_2, (\square x \beta y)) &= (q_2, (\square x \beta y), \rightarrow) \\ \delta(q_2, (\downarrow x \square y)) &= ([2], (\downarrow x \square y), \rightarrow) \\ \delta([2], (\square x \beta y)) &= (q_2, (\downarrow x \beta y), \rightarrow) \\ \delta(q_2, (\square \square \square \square)) &= (q_3, (\square \square \square \square), \leftarrow) \end{aligned}$	$\beta \in \{\square, \downarrow\}$
(4)	$\begin{aligned} \delta(q_3, (\square x \square y)) &= (q_3, (\square x \square y), \leftarrow) \\ \delta(q_3, (\beta x \downarrow y)) &= ([3], (\beta x \downarrow y), \leftarrow) \\ \delta([3], (\square x \square y)) &= (q_3, (\square x \downarrow y), \leftarrow) \\ \delta(q_3, (\square \square \square \square)) &= (q_0, (\square \square \square \square), \rightarrow) \end{aligned}$	

Seja $w = 11$

$$\begin{aligned} & (q_0, (\uparrow \hat{1} \square 1)(\square 1 \downarrow 1)) \vdash ([1], (\downarrow \square 1)(\square 1 \downarrow 1)) \vdash (q_1, (\downarrow \square 1)(\square 1 \downarrow 1)) \\ & \vdash (q_2, (\downarrow \hat{1} \square 1)(\square 1 \downarrow 1)) \vdash ([2], (\square 1 \square 1)(\square 1 \downarrow 1)) \vdash (q_2, (\square 1 \square 1)(\downarrow 1 \downarrow 1)^\wedge) \\ & \vdash (q_3, (\square 1 \square 1)(\downarrow 1 \downarrow 1)) \vdash ([3], (\square 1 \square 1)(\downarrow \square 1)) \vdash (q_3, ^\wedge(\square 1 \downarrow 1)(\downarrow \square 1)) \\ & \vdash (q_0, (\square 1 \downarrow 1)(\downarrow \square 1)) \end{aligned}$$

como não existe transição definida para esta configuração e $q_0 \in F$,
 $w = 11 \in L$.

Se $w = 101$, tem-se:

$$\begin{aligned} & (q_0, (\uparrow \hat{1} \square 1)(\square 0 \square 0)(\square 1 \downarrow 1)) \stackrel{2}{\vdash} ([1], (\downarrow \square 1)(\square 0 \square 0)(\square 1 \downarrow 1)) \vdash \\ & \vdash (q_1, (\downarrow \square 1)(\square 0 \square 0)(\square 1 \downarrow 1)) \stackrel{4}{\vdash} (q_2, (\downarrow \hat{1} \square 1)(\square 0 \square 0)(\square 1 \downarrow 1)) \vdash \\ & \vdash ([2], (\square 1 \square 1)(\square 0 \square 0)(\square 1 \downarrow 1)) \vdash (q_2, (\square 1 \square 1)(\downarrow 0 \square 0)(\square 1 \downarrow 1)) \vdash \\ & \stackrel{2}{\vdash} (q_3, (\square 1 \square 1)(\downarrow 0 \square 0)(\square 1 \downarrow 1)) \vdash ([3], (\square 1 \square 1)(\downarrow 0 \square 0)(\square 1 \downarrow 1)) \end{aligned}$$

como não existe transição definida para esta configuração e $[3] \notin F$,
 $w = 101 \notin L$.

TESE DE CHURCH-TURING : "Qualquer procedimento efetivo pode ser realizado por uma máquina de Turing".

A noção de procedimento efetivo é informal e assim, a tese de

Church-Turing afirma a equivalência entre um conceito informal e um conceito formal, não sendo portanto passível de demonstração. Existe entre tanto, considerável evidência que sustenta essa proposição. Por exemplo: diversas tentativas já foram feitas para definir e caracterizar a classe dos processos efetivos, incluindo, a máquina de Turing, as funções recursivas gerais, as funções μ -recursivas de Gödel, Herbrand e Kleene, o cálculo- λ de Church, os sistemas de produção de Post, o algoritmo de Markov e os predicados, de Smullyan. Sem exceção, mostrou-se que cada uma dessas formulações - apesar de pouco similares - é equivalente a qualquer uma outra. O fato de tal variedade de formulações definir a mesma classe de funções computáveis é uma evidência muito forte para a generalidade de qualquer uma delas.

Para mais detalhes acerca da tese de Church-Turing e dos vários formalismos propostos para caracterizar a classe das funções computáveis, ver:

N. D. JONES - "Computability theory - an introduction"

MÁQUINAS DE TURING NÃO DETERMINÍSTICAS

definição 4.8. Uma MTND é uma 6-upla $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ onde Q, Σ, Γ, q_0 e F são definidos como no caso de MTS e

$$\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\})$$

tal que:

$$(q, \alpha \hat{a} \beta) \vdash (p, \alpha \hat{b} \beta) \Leftrightarrow (p, b, \downarrow) \in \delta(q, a)$$

$$(q, \alpha \hat{a} c \beta) \vdash (p, \alpha b \hat{c} \beta) \Leftrightarrow (p, b, \rightarrow) \in \delta(q, a)$$

$$(q, \alpha c \hat{a} \beta) \vdash (p, \alpha \hat{c} b \beta) \Leftrightarrow (p, b, \leftarrow) \in \delta(q, a)$$

onde: $q, p \in Q$; $a, b, c \in \Gamma$; $\alpha, \beta \in \Gamma^*$

Alem disso, se $\delta(q, a) = \emptyset$, então $(q, \alpha \hat{a} \beta)$ é configuração terminal. Se $q \in F$, então a configuração terminal é final.

Teorema 4.2. Se L é uma linguagem reconhecida por uma MTND, M , então existe MTS, M' , tal que $L = L(M')$.

Prova.

Seja C uma lista de configurações da máquina de Turing não determinística M . Para uma entrada w qualquer, a máquina M' simula M da seguinte maneira:

(1) fazer $C = ((q_0, \hat{w}))$, onde q_0 é o estado inicial de M .

(2) enquanto C não contém uma configuração final, fazer:

a) para toda configuração c em C , substituí-la por todas as configurações c' tais que $c \vdash_M c'$.

b) se não existe c' tal que $c \vdash_M c'$, então retirar c de C

c) se $C = \emptyset$, então parar rejeitando w .

(3) parar, aceitando w .

Obviamente, se $w \in L(M)$ então C irá conter uma configuração final e portanto $w \in L(M')$. Contudo, se $w \notin L(M)$, M pode não parar, o mesmo acontecendo com M' .

MAQUINA DE TURING UNIVERSAL

Seja uma máquina de Turing $M_1 = (Q, \Sigma, \Gamma, \delta, \sqcup, F)$ tal que $Q = \{1, \dots, n\}$, $\Sigma = \{0, 1\}$, e $\Gamma = \{b, 0, 1\}$. M_1 pode ser completamente especificada por uma tabela.

EXEMPLO:

	b	0	1
1	-	-	(2, 0, \rightarrow)
2	(3, 1, \leftarrow)	(3, 1, \leftarrow)	(2, 1, \rightarrow)
3	(4, 0, \rightarrow)	(4, 0, \rightarrow)	(3, 1, \leftarrow)
4	-	-	-

Uma tabela como essa entretanto, pode ser codificada da seguinte maneira:

- Os estados podem ser codificados como $1, 11, \dots, 11\dots 1$
- para cada estado pode-se construir um bloco (correspondente a uma linha da tabela) dividido em 3 sub-blocos (correspondente às colunas da tabela, ou seja, aos símbolos $b, 0$ e 1).
- os sub-blocos podem ser separados uns dos outros por $\$$; os blocos podem ser separados por $\$ \$$ e a codificação toda pode ser delimitada por $\$ \$ \$$.

CODIFICAÇÃO DE UM SUB-BLOCO:

se $\delta(i, a) = (j, x, d)$ onde $d \in \{\leftarrow, \rightarrow\}$, então o sub-bloco será codificado como:

$$\phi \underbrace{1 \dots 1}_{j \text{ vezes}} d x \phi$$

se $\delta(i, a) = \phi$, então sua codificação será $\phi 0 \phi$.

EXEMPLO: a codificação da máquina de Turing especificada pela tabela do exemplo anterior, será:

$$\phi \phi \phi 0 \phi 0 \phi 11 \rightarrow 0 \phi \phi 111 \leftarrow 1 \phi 111 \leftarrow 1 \phi 11 \rightarrow 1 \phi \phi 1111 \rightarrow 0 \phi 1111 \rightarrow 0 \phi 111 \leftarrow 1 \phi \phi 0 \phi 0 \phi 0 \phi \phi \phi$$

Uma MÁQUINA DE TURING UNIVERSAL é uma MT que quando apresentada a uma codificação de uma máquina de Turing M e a uma sequência α , simula o comportamento de M com entrada α .

Institivamente (em termos de linguagem de programação) pode-se imaginar uma codificação para MT como:

- existem 3 matrizes: NOVO ESTADO, ESCRITA e MOVIMENTO para armazenar as informações da tabela que especifica uma MT qualquer
- se $\delta(i, a) = (j, x, d)$ onde $d = \{-1, 0, 1\}$, então

$$\text{NOVO ESTADO}(i, a) = j$$

$$\text{ESCRITA}(i, a) = x$$

$$\text{MOVIMENTO}(i, a) = d$$

- com essa codificação, o programa para a MTU pode ser:

```

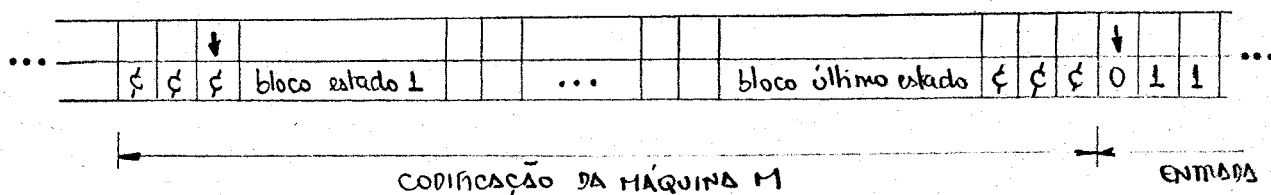
ESTADO := 1;
CABECA := 1;
SÍMBOLO := FITA (CABECA);
WHILE TRANSICAO DEFINIDA DO BEGIN
    ESTADO := NOVO ESTADO (ESTADO, SÍMBOLO);
    FITA (CABECA) := ESCRIBE (ESTADO, SÍMBOLO);
    CABECA := CABECA + MOVIMENTO (ESTADO, SÍMBOLO)
END;

```

Mais formalmente, usando a codificação com símbolos $\langle, 0, 1, \leftarrow, \rightarrow$ e \vdash , pode-se definir a MTU como:

- (1) a MTU tem uma fita com 2 trilhas
- (2) a trilha de baixo irá usar os símbolos $\emptyset, 0, 1, \leftarrow, \rightarrow, b$ e será usada para armazenar a codificação de uma MT qualquer, M , e também para armazenar a codificação de uma entrada qualquer de $(0+1)^*$.
- (3) a trilha de cima irá usar os símbolos \downarrow e b . O símbolo \downarrow será usado para armazenar o estado atual de M e para armazenar a posição da cabeça de M .

Inicialmente:



FUNIONAMENTO DO MTU :

- (1) a MTU movimenta sua cabeça para a direita até encontrar o símbolo \downarrow sobre um símbolo x da entrada, armazenando-o em seu controle finito. A MTU movimenta então sua cabeça para a esquerda até encontrar o símbolo \downarrow que armazena o estado de M . A MTU então, apaga esse símbolo e movimenta sua cabeça para a direita para o sub-bloco correspondente a x e escreve \downarrow sobre o primeiro símbolo desse sub-bloco (que deve ser 1 ; do contrário a MTU pára, pois não existe próximo movimento de M). Seja m_1 esse último marcador.
- (2) a MTU movimenta agora sua cabeça para a esquerda até encontrar $\phi\phi\phi$, marcando com \downarrow o ϕ mais à direita. Seja m_2 esse marcador.
- (3) a MTU movimenta então sua cabeça para a direita até encontrar o marcador m_1 e chama uma subrotina que move alternadamente, m_1 uma posição para a direita e m_2 , um bloco para a direita. Quando m_1 apontar para um símbolo diferente de 1 , m_2 estará localizado sobre o ϕ imediatamente anterior ao bloco correspondente ao próximo estado de M . Neste ponto, a MTU apaga m_1 e armazena em seu controle finito o símbolo que M irá escrever e o sentido do movimento da cabeça de M . Em seguida, a MTU movimenta sua cabeça para a direita até encontrar o marcador \downarrow sobre a entrada. O símbolo sob esse marcador é então trocado e \downarrow é movimentado segundo o sentido armazenado. Neste ponto, a MTU simula um movimento de M . Em seguida, o processo é repetido.

Se M pára com uma entrada α qualquer, a MTU irá parar também.

e a parte da fita que inicialmente contém α , ao final vai estar como a fita de M . Quando M pára, a MTU pode descobrir se M está num estado final ou não, indo para seu estado final ou não, em firme o caso. Se M não parar, então a MTU também não vai parar. Portanto, a MTU simula M .

O PROBLEMA DA PARADA DE MÁQUINAS DE TURING:

"Seja uma máquina de Turing M e uma entrada α qualquer.
 M pára com entrada α ?"

Será mostrado, a seguir, que esse problema é INDECIDÍVEL, ou seja, não existe um algoritmo que responda sempre "SIM" ou "NÃO" para todo par (M, α) (isso não significa que não se pode determinar se uma máquina de Turing específica sob uma entrada específica irá parar ou não).

ENUMERAÇÃO DE MÁQUINAS DE TURING

Como se mostrou anteriormente, uma MT pode ser codificada como uma cadeia de $\{ \dot{c}, 0, 1, \rightarrow, \leftarrow \}^*$. Pode-se escolher uma ordenação dos símbolos $\dot{c}, 0, 1, \rightarrow$ e \leftarrow e então enumerar as cadeias de $\{ \dot{c}, 0, 1, \rightarrow, \leftarrow \}^*$. Considerando que cada uma dessas cadeias é a codificação de uma MT (eventualmente existirão cadeias mal formadas e que podem ser consideradas como codificações de MT com zero movimentos), faz sentido falar da i -ésima MT ($i > 0$).

Da mesma maneira, as cadeias de $\{0, 1\}^*$ podem ser ordenadas (1, 0, 1, 00, 01, 10, 11, ..., por exemplo) e portanto também faz sentido falar da j -ésima cadeia de $\{0, 1\}^*$.

Seja então, a linguagem:

$$L = \{ x_i \in (0+1)^* \mid x_i \text{ não é aceita por } T_i \} \quad (i > 0)$$

Seja T_j ($j > 0$) uma MT que aceita L . Então

$$x_j \in L \Leftrightarrow x_j \text{ não é aceita por } T_j$$

Mas, como T_j aceita L :

$$x_j \in L \Leftrightarrow x_j \text{ é aceita por } T_j \quad \text{CONTINGÊNCIA!}$$

Portanto, não existe uma MT que aceita L .

Teorema 4.3. Não existe um algoritmo (ou seja, uma MT que sempre pára) que resolva o problema da parada de máquinas de Turing.

Prova (informal)

Considere (por absurdo) que existe um algoritmo A que determina se uma MT qualquer, com entrada x vai parar ou não. Seja M uma máquina de Turing. Tem-se então:

- (1) dada uma sentença x , M enumera as sentenças de $\{0,1\}^*$ até encontrar um inteiro i tal que $x_i = x$.
- (2) em seguida, M gera a codificação da máquina M_i por enumeração das cadeias de $\{0,1,\rightarrow,\leftarrow\}^*$.
- (3) aplicar o algoritmo A para determinar se M_i com entrada

x_i pára ou não.

- a) se A determina que M_i não pára com entrada x_i , então M pára e aceita x_i ;
- b) se A determina que M_i pára com entrada x_i , então transferir o controle para uma MTU que simula M_i com entrada x_i . Como M_i pára com entrada x_i , a MTU pára e determina se M_i aceita x_i ou não. Se M_i aceita x_i , então M pára e não aceita x_i . Se M_i não aceita x_i , então M pára e aceita x_i .

Portanto, M aceita L. ABSURDO! Logo, o algoritmo A não existe.

LINGUAGENS RECURSIVAS E LINGUAGENS RECURSIVAMENTE ENUMERÁVEIS.

Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ uma MT. Define-se o conjunto de cadeias aceitas por M ($A(M)$) e o conjunto de cadeias rejeitadas por M ($R(M)$) como:

$$A(M) = \{ x \in \Sigma^* / (q_0, \hat{x}) \vdash^* (q, \hat{\alpha}) ; q \in F \}$$

$$R(M) = \{ x \in \Sigma^* / (q_0, \hat{x}) \vdash^* (q, \alpha \hat{\alpha} \beta) ; q \notin F \}$$

$$\delta(q, a) = \emptyset \}$$

Pode-se definir também

$$C(M) = \Sigma^* - (A(M) \cup R(M))$$

ou seja, o conjunto de cadeias de Σ^* que levam M a entrar em "loop".

definição 4.9. L é recursiva \Leftrightarrow existe máquina de Turing M tal que $A(M) = L$ e $R(M) = \Sigma^* - L$ (ou seja, M sempre pára)

definição 4.10. L é recursivamente enumerável \Leftrightarrow existe máquina de Turing M tal que $A(M) = L$ (ou seja, se $x \notin L$ a máquina M com entrada x pode não parar).

Lema 4.1. Se L é uma linguagem recursiva, então \bar{L} é recursiva.

Prova.

L é recursiva. Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ tal que $A(M) = L$ e $R(M) = \Sigma^* - L$.

Construir $M' = (Q \cup \{q\}, \Sigma, \Gamma, \delta', q_0, \{q\})$ onde q é um novo estado e δ' é dada por:

1) se $\delta(p, a) = p'$ então $\delta'(p, a) = p'$; $p, p' \in Q$; $a \in \Gamma$

2) se $\delta(p, a) = \phi$; $p \notin F$ então $\delta'(p, a) = q$

3) $\delta'(q, a) = \phi$, $\forall a \in \Gamma$

Logo:

$$A(M') = R(M) = \Sigma^* - L = \bar{L}$$

$$R(M') = A(M) = L = \Sigma^* - \bar{L}$$

Portanto, \bar{L} é recursiva.

Lema 4.2 . Seja x_1, x_2, \dots uma enumeração das cadeias de Σ^* e T_1, T_2, \dots uma enumeração das máquinas de Turing sobre Σ .
Seja $L = \{x_i \in \Sigma^* \mid x_i \in L(T_i)\}$. Então:

- (a) L é recursivamente enumerável
- (b) \bar{L} não é recursivamente enumerável

Prova . Exercício!

Teorema 4.4 : A classe das linguagens recursivas é um subconjunto próprio da classe das linguagens recursivamente enumeráveis.

Prova . Consequência dos lemas 4.1 e 4.2.

MAQUINAS DE TURING E GRAMÁTICAS TIPO 0

Proposição 4.1 : A linguagem L é reconhecida por uma MT $\Leftrightarrow L$ é gerada por uma gramática tipo 0.

O teorema a seguir estabelece metade da prova da proposição 4.1.

Teorema 4.5 : Se L é gerada por uma gramática tipo 0, então L é reconhecida por uma MT.

Prova . Seja $G = (N, \Sigma, P, S)$ uma gramática tipo 0 tal que $L = L(G)$.

Construir uma MT $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ tal que

$$\Gamma = N \cup \Sigma \cup \{b, \#, x\} \quad \text{onde} \quad b, \#, x \notin N \cup \Sigma.$$

Inicialmente M tem uma entrada $w \in \Sigma^*$ na fita. M então, insere $\#$ antes de w e $\# S \#$ depois. Logo:

	#	w	#	S	#	
--	---	---	---	---	---	--

é o conteúdo da fita após esse passo. Em seguida M irá (não deterministicamente) simular derivações de G começando com S , substituindo símbolos não terminais que apareçam entre os dois últimos $\#$ usando todas as possíveis produções de P .

Seja $\# w \# A_1 A_2 \dots A_k \#$ o conteúdo da fita de M num dado passo. M movimentará sua cabeça por $A_1 A_2 \dots A_k$, escolhendo não deterministicamente todas as possíveis subcadeias $A_i \dots A_{i+l}$ que são lados esquerdos de produções de P , substituindo essas subcadeias pelos correspondentes lados direitos das produções (nessa substituição M pode deslocar $A_{i+l+1} \dots A_k \#$ para a esquerda ou para a direita a fim de preencher ou conseguir espaço, caso o lado direito da produção usada tiver um comprimento diferente do lado esquerdo).

Com essa simulação de derivações de G , M irá escrever uma cadeia $\# w \# \alpha \#$ na sua fita, exatamente se $S \xRightarrow{*} \alpha$. Caso $\alpha = w$ (M pode comparar as cadeias w e α a cada passo para testar essa condição), M aceita w .

EXEMPLO. Seja $G = (\{A, B\}, \{0, 1\}, P, A)$ com P dado por:

$$A \rightarrow B0$$

$$A0 \rightarrow B1$$

$$B \rightarrow 1A$$

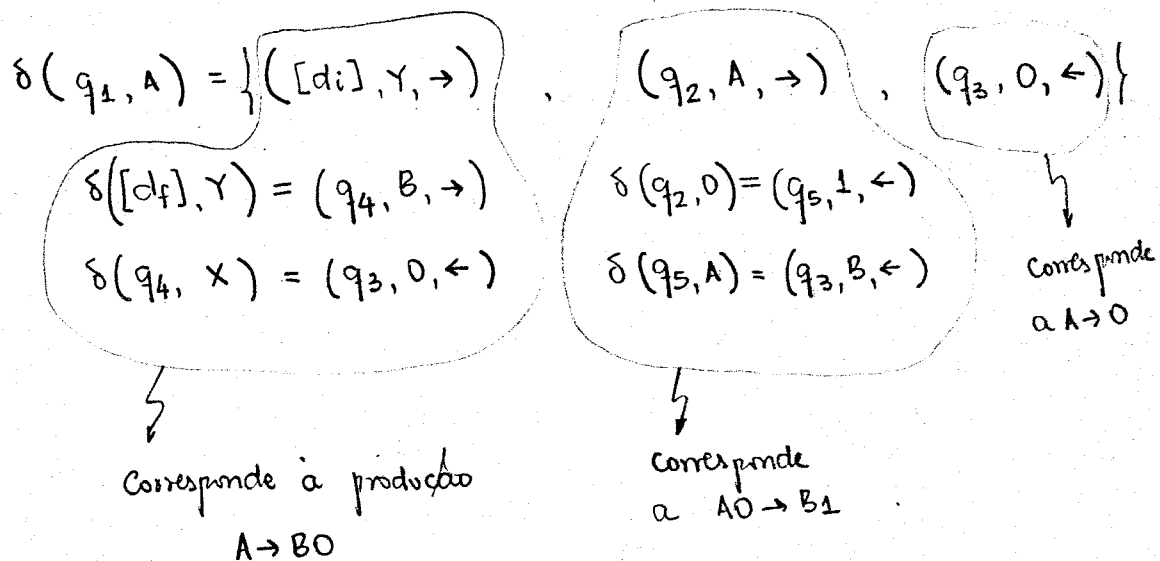
$$B \rightarrow BB$$

$$B \rightarrow 1$$

$$A \rightarrow 0$$

Considere duas subrotinas para deslocamento da cadeia da fita uma posição para a direita e uma posição para a esquerda, com estados inicial e de retorno, respectivamente, $[di]$, $[df]$ e $[ei]$, $[ef]$.

Um trecho do programa da MT que corresponde às ações que devem ser tomadas quando é encontrada uma subcadeia A (isto é, quando se tem $\# w \# \dots A \dots \#$ na fita) é dado por:



onde é utilizado um símbolo adicional, Y, para indicar chamadas a subrotinas (a subrotina DESLOCA-PARA-DIREITA utiliza um símbolo adicional, X, para marcar o espaço criado)