Linguagens Formais

Capítulo 7: Máquinas de Turing

José Lucas Rangel

7.1 - Introdução

Neste capítulo vamos estudar a máquina de Turing (mT), introduzida por Alan Turing como um modelo matemático do processo de computação. Sua estrutura simples é proposital, uma vez que Turing pretendia, com sua definição, chegar a um modelo que fosse universalmente aceito. A máquina que vamos apresentar aqui é apenas uma das diversas variantes encontradas na literatura, escolhida por ser suficientemente próxima de um sistema de computação real, e por não oferecer maiores dificuldades de definição.

A mT é o principal modelo usado para o estudo do que é ou não computável. De acordo com a tese de Church, todos os modelos razoáveis de procedimento são equivalentes, e a mT se revelou simples e flexível o suficiente para permitir todas as demonstrações dos resultados principais. Pode-se usar uma mT como aceitador ou reconhecedor, ou ainda como a implementação de um procedimento mais geral, que transforma uma cadeia de entrada em uma cadeia de saída. Neste capítulo vamos definir a mT e apresentá-la como o reconhecedor das linguagens tipo 0, e procurar mostrar, através de exemplos, sua universalidade. Demonstrar a tese de Church, naturalmente, está fora de cogitação, de maneira que a universalidade da máquina de Turing só pode ser *confirmada* pelo fato de que todos os procedimentos encontrados na prática podem ser implementados através de máquinas de Turing.

7.2 - A máquina de Turing

Vamos definir a máquina de Turing com um controle finito, como todas as máquinas até aqui, e uma fita, que pode servir como dispositivo de entrada (inicialmente a cadeia de entrada está escrita na fita), como área de trabalho (memória, ou rascunho) , ou como dispositivo de saída (possíveis resultados estarão escritos na fita ao final da computação).

O modelo aqui apresentado tem uma fita “semi-infinita”: considera-se a fita arbitrariamente extensível apenas para o lado direito. Por convenção, a fita é finita, mas pode ser aumentada sempre que houver necessidade de mais espaço, com mais uma célula em branco. A justificativa para isso é que o que é computável não pode depender do tamanho do bloco usado como rascunho pelo matemático: sempre deve ser possível arranjar outro bloco e continuar escrevendo. O novo bloco do matemático estará em branco, como sempre estará em branco a parte da fita em que ainda nada foi escrito.

Por convenção, um trecho da fita em branco será representado por um ou mais símbolos “branco”. Além disso, vamos convencionar um branco não pode ser escrito, ou seja um trecho de fita em que ocorre o símbolo branco é um trecho onde nada foi escrito até agora. Assim, a qualquer momento, valem as seguintes propriedades:

- apenas um número finito de posições da fita pode conter um símbolo distinto de branco;

- após o primeiro branco da fita, todos os símbolos são brancos.

Isto acontece porque a máquina avança ou recua na fita uma célula de cada vez e, em um número finito de passos, apenas um número finito de posições da fita pode ter sido alcançado.

Alguns autores preferem falar de uma fita infinita, cujo conteúdo é de símbolos "branco", exceto nas posições que já foram visitadas; outros preferem dizer que a fita é potencialmente infinita e que é estendida a cada vez que se tenta andar para a direita a partir da última posição. O fato importante a ser considerado é que, a qualquer momento, a quantidade de informação contida na fita é finita, embora não haja um limite para a quantidade de informação que nela pode ser armazenada.

Essa informação pode ser representada por uma cadeia de caracteres composta pelos símbolos gravados na fita, da primeira posição até o primeiro branco (exclusive). Note que, ao contrário do que ocorre em gravadores de fita, consideramos que é a cabeça de leitura/gravação que se move, e não a fita.

Outros características poderiam ser acrescentadas ao nosso modelo, tais como:

- várias fitas;
- fitas extensíveis para ambos os lados;
- fitas com finalidade específica, por exemplo fitas para entrada e fitas para saída
- a máquina pode ter mais de uma cabeça de leitura por fita;
- em vez de fita, a máquina pode usar como memória um reticulado bi-dimensional;
- a máquina pode ser não determinística.

De acordo com a tese de Church, entretanto, detalhes como estes não devem influenciar a capacidade de computação da mT. O modelo que vamos utilizar é um modelo determinístico bastante simples, próximo do mínimo necessário para uma máquina razoável, de acordo com a tese de Church. Discutiremos mais tarde a equivalência entre este modelo e suas variantes que possuem características como as mencionadas.

Definição. Uma máquina de Turing M é uma tupla $M = < K, \Sigma, \Gamma, \delta, i, F >$, onde K é um conjunto (finito, não vazio) de estados, Γ é o alfabeto (finito) de símbolos da fita, $\Sigma \subseteq \Gamma$ é o alfabeto de símbolos de entrada, δ é a função de transição, $i \in K$ é o estado inicial, e $F \subseteq K$ é o conjunto de estados finais.

O símbolo branco será representado aqui por $\Diamond$. Este símbolo é um símbolo de Γ que não pode ser escrito, nem pode fazer parte da entrada. Assim, temos $\Diamond \in \Gamma - \Sigma$.

A função de transição δ é um mapeamento $\delta: K \times \Gamma \rightarrow K \times \Gamma \times \{L, R\}$. Quando tivermos $\delta(q, a) = (p, b, R)$, a mT M, quando está no estado q, e lê o símbolo a na fita, escreve o símbolo b na mesma posição em que a estava escrito, move-se para a direita uma célula, e passa para o estado p. (Idem, para a esquerda no caso de $\delta(q, a) = (p, b, L)$). Por simplicidade, podemos deixar alguns valores de δ indefinidos, de maneira que δ deve ser entendida como uma função parcial.

Atingida uma situação em que o estado é q, o símbolo lido é a, e $\delta(q, a)$ é indefinido, dizemos que a máquina pára. Poderíamos, se desejado, definir máquinas de Turing que sempre param quando atingem um estado final, e que nunca param em um estado não final. Mas sempre que esta propriedade for desejada, podemos alterar

uma mT introduzindo um estado adicional, não final, do qual a máquina nunca mais sai.

Configuração. Para representar uma *configuração* (ou às vezes uma *descrição instantânea*) de uma mT usaremos uma cadeia $xqy \in \Gamma^* \bullet K \bullet \Gamma^*$, em que $xy \in \Gamma^*$ é o conteúdo da fita, sem incluir nenhum dos brancos que existem à direita, e $q \in K$ é o estado corrente. Para que esta notação seja usada, é necessário que os símbolos de Γ sejam distintos dos símbolos usados para representar os estados de K, tornando impossível a confusão entre as partes constituintes da configuração, que são o estado q e o conteúdo xy da fita.

Esta notação tem a vantagem de dispensar a necessidade de indicar separadamente a posição da cabeça da máquina, uma vez que o próprio estado é usado para indicar essa posição. Assim, em uma configuração xqy, a posição onde a leitura e a escrita se realizam é o primeiro símbolo de y. Se y é a cadeia vazia, ou seja, se a configuração é da forma xq, a máquina lê um branco, à direita da parte escrita x da fita.

Configurações inicial e final. Para iniciar a computação, a configuração inicial para a entrada x é ix, composta pelo estado inicial i e pela cadeia de entrada x. O símbolo lido na configuração inicial é o primeiro símbolo de x, exceto no caso $x=\varepsilon$, em que o símbolo lido é um branco. Uma configuração xfy é final se o estado f é final. Como veremos a seguir, a mT pode, se for conveniente, aceitar sua entrada sem lê-la até o final.

Mudança de configuração. A definição da relação "mudança de configuração", $\vdash\!\!\!-$, não oferece surpresas:

- movimento para a direita:
  se $\delta(q, a) = (p, b, R)$, então $xqay \vdash\!\!\!- xbpy$
  (no estado q, lendo a, escreve b, anda para a direita e muda para o estado p)
- movimento para a direita, lendo um branco:
  se $\delta(q, \Diamond) = (p, b, R)$, então $xq \vdash\!\!\!- xbp$
  (caso particular do movimento para a direita: no estado q, lendo um branco, escreve b, anda para a direita e muda para o estado p)
- movimento para a esquerda:
  se $\delta(q, a) = (p, b, L)$, então $xcqay \vdash\!\!\!- xpcby$
  (no estado q, lendo a, escreve b, anda para a esquerda e muda para o estado p)
- movimento para a esquerda, lendo um branco:
  se $\delta(q, \Diamond) = (p, b, L)$, então $xcq \vdash\!\!\!- xpcb$
  (caso particular do movimento para a esquerda: no estado q, lendo um branco, escreve b, anda para a esquerda e muda para o estado p)

Naturalmente, se o valor de $\delta(q, a)$ é indefinido, nenhuma configuração pode ser alcançada a partir de uma configuração xqay, e a máquina pára. No caso particular $a=\Diamond$, se a configuração for xq, a máquina pára quando $\delta(q, \Diamond)$ é indefinido.

As assimetrias que podem ser notadas na definição da relação mudança de configuração $\vdash\!\!\!-$ são devidas às assimetrias na definição de configuração e na definição da própria máquina de Turing: *(1)* na configuração xqy o símbolo lido é o

primeiro de y, e *(2)* a fita é semi-infinita para a esquerda. Note que esta definição impede a passagem da cabeça para a esquerda do primeiro símbolo.

Notamos, para uso futuro, que a mudança de configuração pode ser descrita através da substituição de cadeias que envolvem no máximo três símbolos:

$qa \rightarrow bp$, $q \rightarrow bp$, $cqa \rightarrow pcb$ ou $cq \rightarrow pcb$,

O restante da configuração não se altera.

Linguagem reconhecida por uma mT. A linguagem aceita, ou reconhecida por uma mT M é definida por

$$L(M) = \{ x \in \Sigma^* \mid ix \vdash\!\!\!\!-^* yfz, \text{ onde } f \in F \}$$

ou seja, L(M) é o conjunto das cadeias x, compostas de símbolos do alfabeto de entrada, que levam M da configuração inicial ix (correspondente à entrada x) para alguma configuração final yfz (em que o estado f é final).

A aceitação se dá quando o estado final é atingido, não interessando em que ponto da fita está a cabeça, ou se há possibilidade de continuar a partir desse ponto. Se é possível atingir uma configuração cujo estado é final, a cadeia de entrada já está aceita. Caso contrário, a cadeia não será aceita. Não é sequer necessário que a entrada seja inteiramente lida.

Para não aceitar uma cadeia x (x não pertence à linguagem), a máquina pode parar em algum estado não final, ou pode simplesmente <u>não parar</u>, e continuar se movendo indefinidamente sem nunca aceitar.

<u>Exemplo 7.1</u>: Considere a mT $M = (K, \Gamma, \Sigma, \delta, i, F)$, onde

$K = \{ q_0, q_1, q_2, q_3, q_4 \}$

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

$\Gamma = \{ a, b, X, Y, \Diamond \}$

$i = q_0$

$F = \{ q_4 \}$,

e δ é dada pela tabela abaixo, onde a notação foi simplificada, e cada valor da função δ em algum ponto é representado por uma quíntupla. Assim, se $\delta(q, a)=(p, b, D)$, representaremos este fato pela quíntupla qapbD. Quando nenhuma quíntupla iniciada por qa aparece, o valor de $\delta(q, a)$ é indefinido. (Como estamos considerando apenas o caso determinístico, no máximo pode ocorrer uma quíntupla para cada par (q, a).

q_0aq_1XR	q_0Yq_3YR	$q_0\Diamond q_4XR$
q_1aq_1aR	q_1bq_2YL	q_1Yq_1YR
q_2aq_2aL	q_2Xq_0XR	q_2Yq_2YL
q_3Yq_3YR	q_3Yq_4XR	$q_3\Diamond q_4XR$

Considere a cadeia de entrada x = aaabbb. A seqüência de configurações assumida por M com entrada x é:

$q_0aaabbb \vdash\!\!\!\!- Xq_1aabbb \vdash\!\!\!\!- Xaq_1abbb \vdash\!\!\!\!- Xaaq_1bbb \vdash\!\!\!\!- Xaq_2aYbb$
$\vdash\!\!\!\!- Xq_2aaYbb \vdash\!\!\!\!- q_2XaaYbb \vdash\!\!\!\!- Xq_0aaYbb \vdash\!\!\!\!- XXq_1aYbb \vdash\!\!\!\!- XXaq_1Ybb$
$\vdash\!\!\!\!- XXaYq_1bb \vdash\!\!\!\!- XXaq_2YYb \vdash\!\!\!\!- XXq_2aYYb \vdash\!\!\!\!- Xq_2XaYYb \vdash\!\!\!\!- XXq_0aYYb$
$\vdash\!\!\!\!- XXXq_1YYb \vdash\!\!\!\!- XXXYq_1Yb \vdash\!\!\!\!- XXXYYq_1b \vdash\!\!\!\!- XXXYq_2YY$
$\vdash\!\!\!\!- XXXq_2YYY \vdash\!\!\!\!- XXq_2XYY \vdash\!\!\!\!- XXXq_0YYY \vdash\!\!\!\!- XXXYq_3YY$

$\vdash XXXYYq_3Y \vdash XXXYYYq_3 \vdash XXXYYYXq_4$

A mT M acima aceita a linguagem $\{ a^n b^n \mid n \geq 0 \}$. Podemos observar as ações correspondentes a cada um dos estados:

q_0	o primeiro a é marcado com um X, passa para q_1; se não houver mais nenhum a, passa para q_3;
q_1	procura o primeiro b, que é marcado com Y, passa para q_2;
q_2	volta até o X, e passa para q_0 ;
q_3	verifica que todos os símbolos já estão marcados: vai para a direita até encontrar um ◊, e passa para q_4;
q_4	aceita.

Outros exemplos de computações:

$q_0 \vdash Xq_4$ (aceita)

$q_0ab \vdash Xq_1b \vdash q_2XY \vdash Xq_0Y \vdash XYq_3 \vdash XYXq_4$ (aceita)

q_0ba (para sem aceitar)

$q_0aab \vdash Xq_1ab \vdash Xaq_1b \vdash Xq_2aY \vdash q_2XaY \vdash Xq_0aY \vdash XXq_1Y \vdash XXYq_1$ (para sem aceitar)

mostrando que ε e ab pertencem à linguagem, mas isso não acontece com ba e aab.

Exercício 7.1: Construa uma MT que aceite a linguagem $L = \{ a^n b^n c^n \mid n \geq 0 \}$.

7.3 - A máquina de Turing como modelo geral de procedimento.

A máquina de Turing é, como já foi observado anteriormente, o modelo formal mais usado de procedimento. Pela tese de Church, todos os modelos razoáveis do processo de computação, definidos e por definir, são equivalentes. Por essa razão podemos usar a máquina de Turing como sendo nosso modelo formal de procedimento. Lembramos que, por sua natureza, a tese de Church não admite prova formal.

A definição da linguagem aceita por uma mT caracteriza o uso de uma mT como procedimento aceitador, de forma que (via tese de Church) a classe das linguagens aceitas por máquinas de Turing e a classe dos conjuntos recursivamente enumeráveis se identificam. Semelhantemente, identificamos algoritmo com "mT que sempre pára" - ou seja, para qualquer entrada sempre atinge uma configuração a partir da qual não há nenhuma outra configuração acessível. Um conjunto recursivo é, então, a linguagem aceita por uma mT que sempre pára. Fazemos assim a correspondência entre dizer "Sim" e estar num estado final, e dizer "Não" e estar em um estado não final.

Exemplo 7.2. Seja construir uma máquina de Turing para converter um inteiro dado em notação unária, para a notação binária. (Em notação unária, o natural i é representado pela cadeia 1^i, obtida pela concatenação de i instâncias do símbolo i.) Suporemos que entrada e saída da máquina estarão delimitadas entre dois $. Por exemplo, se a entrada é 1111, a saída deve ser 100, sendo permitida a presença adicional de algum "lixo" na fita.

A idéia usada para a construção da mT é a seguinte: se um número é ímpar, o último bit" de sua representação binária é 1; se é par, 0. Para obter os demais bits, o número é dividido por 2. Na fita da mT, dividir por 2 é cancelar um símbolo sim, um não. Isto combina com a determinação da paridade do número: se não sobrar nenhum, o número é par.

Podemos usar a seguinte máquina, descrita através de quíntuplas:

ABR		A1A1L	AxAxL	
BHR		B1CxR	BxBxR	
CER		C1D1R	CxCxR	
DFR		D1CxR		
	E0E0R	E1E1R		E◊1GL
	F0F0R	F1F1R		F◊0GL
GAL	G0G0L	G1G1L		
	H0H0R	H1H1R		H◊I$L
IMR	I0JxR	I1KxR	IxIxL	
JJR	J0J0R	J1J1R	JxJxR	J◊L0L
KKR	K0K0R	K1K1R	KxKxR	K◊L1R
LIL	L0L0L	L1L1L		
MMR	M0M0R	M1M1R	MxMxR	M◊N$R

Nesta mT, o estado inicial é A, e o final é N. A computação abaixo corresponde à entrada 1111. A última parte da computação corresponde à inversão da representação binária, que é necessária porque os bits do resultado são gerados a partir do menos significativo. No nosso caso, $4 = 1111_1 = 100_2$ aparece como 001, e precisa ser invertido.

A1111	$B1111$	$xC111$
$x1D11$	$x1xC1$	$x1x1D$
$x1x1$F	$x1x1G$0	$x1xA1$0
$x1Ax1$0	$xA1x1$0	$Ax1x1$0
A$x1x1$0	$Bx1x1$0	$xB1x1$0
$xCxx1$0	$xxCx1$0	$xxxC1$0
$xxx1D$0	$xxx1$F0	$xxx1$0F
$xxx1$G00	$xxx1G$00	$xxxA1$00
$xxAx1$00	$xAxx1$00	$Axxx1$00
A$xxx1$00	$Bxxx1$00	$xBxx1$00
$xxBx1$00	$xxxB1$00	$xxxxC$00
$xxxx$E00	$xxxx$0E0	$xxxx$00E
$xxxx$0G01	$xxxx$G001	$xxxxG$001
$xxxAx$001	$xxAxx$001	$xAxxx$001
$Axxxx$001	A$xxxx$001	$Bxxxx$001
$xBxxx$001	$xxBxx$001	$xxxBx$001
$xxxxB$001	$xxxx$H001	$xxxx$0H01
$xxxx$00H1	$xxxx$001H	$xxxx$00I1$
$xxxx$00xK$	$xxxx$00x$K	$xxxx$00xL$1
$xxxx$00Ix$1	$xxxx$0I0x$1	$xxxx$0xJx$1
$xxxx$0xxJ$1	$xxxx$0xx$J1	$xxxx$0xx$1J

$xxxx$0xx$L10	$xxxx$0xxL$10	$xxxx$0xIx$10
$xxxx$0Ixx$10	$xxxx$I0xx$10	$xxxx$xJxx$10
$xxxx$xxJx$10	$xxxx$xxxJ$10	$xxxx$xxx$J10
$xxxx$xxx$1J0	$xxxx$xxx$10J	$xxxx$xxx$1L00
$xxxx$xxx$L100	$xxxx$xxxL$100	$xxxx$xxIx$100
$xxxx$xIxx$100	$xxxx$Ixxx$100	$xxxxI$xxx$100
$xxxx$Mxxx$100	$xxxx$xMxx$100	$xxxx$xxMx$100
$xxxx$xxxM$100	$xxxx$xxx$M100	$xxxx$xxx$1M00
$xxxx$xxx$10M0	$xxxx$xxx$100M	$xxxx$xxx100N

Exercício 7.2. Construa uma máquina de Turing que faça a conversão inversa, isto é, recebendo como entrada a representação binária de um número natural, constrói sua representação unária.
Sugestão: A saída desejada pode ser construída usando o seguinte ciclo: se o bit for 1, acrescente um 1 à saída; dobre o número de 1's para todos os bits exceto o mais significativo. Por exemplo, para converter 1101 (13) podemos fazer:
para o bit 1, acrescenta 1 (1);
para o bit 1, dobra (11) e acrescenta 1 (111)
para o bit 0, dobra (111111)
para o bit 1, dobra (111111111111) e acrescenta 1 (1111111111111)

Como a máquina de Turing só trabalha com símbolos e cadeias, para que uma máquina de Turing possa trabalhar com conjuntos que não são linguagens, é necessário usar uma codificação, usando cadeias de símbolos para representar os elementos do conjunto. Na nossa definição informal de procedimento, não foi feita nenhuma restrição sobre os valores aceitáveis como entradas para procedimentos, mas esta restrição existe para a mT: apenas cadeias de símbolos podem ser usadas como entradas ou saídas. Conjuntos que não são linguagens só podem ser tratados por um procedimento através de alguma codificação de seus elementos em cadeias de alguma linguagem. Por exemplo: podemos dizer que o conjunto Nat^2 de pares de naturais é r.e., e justificar esta afirmativa apresentando o procedimento enumerador (ou gerador) abaixo.

1. Faça i=0.
2. Para j=0, ..., i, emita (i-j, j).
3. Incremente i de 1.
4. Vá para 2.

Neste procedimento, não foi especificada a forma de representação dos números naturais, ou de seus pares, de forma que qualquer representação pode ser usada.

Entretanto, usando máquinas de Turing como reconhecedores, só podemos mostrar que Nat^2 é r.e. descrevendo uma linguagem cujos elementos representam os elementos de Nat^2, e uma mT que reconhece esta linguagem. As linguagens L_1, L_2 e L_3 abaixo são exemplos de linguagens que podem ser usadas com essa finalidade.

$L_1 = \{ a^i b^j \mid i, j \in Nat \}$
$L_2 = \{ \$ 1^i \$ 1^j \$ \mid i, j \in Nat \}$
$L_3 = \{ \$ x \$ y \$ \mid x, y \in Z\}$, onde $Z = \{ 0 \} \cup \{1\} \bullet \{ 0, 1 \}^*$

Nas duas primeiras codificações, o par (2, 3) é representado respectivamente por aabbb ou por $11$111$. No terceiro caso, usamos a representação binária dos dois elementos do par, de forma que (2, 3) corresponde a $10$11$.

A única propriedade exigida de uma codificação, é que deve ser definida por uma bijeção. Tomando L_1 como exemplo, a cada elemento de Nat^2 corresponde exatamente um elemento de L_1=L[a*b*], e vice-versa. Já que existe a codificação, para mostrar que Nat^2 é r.e., basta mostrar que L_1 é r.e., o que pode ser feito neste caso observando que L_1 é regular.

Exercício 7.3: Construa máquinas de Turing que aceitem as linguagens L_1, L_2 e L_3.

Sugestão: Como as linguagens são regulares, construa afd's, que podem então ser transformados em mT's que nunca se movem para a esquerda.

Para representar um procedimento qualquer com uma mT precisamos permitir a geração de cadeias de saída, uma vez que a fita da mT funciona como memória e como dispositivo de entrada/saída. As cadeias geradas como saída de um procedimento devem ser escritas na fita. Isto vale para o caso de um procedimento gerador, que enumera uma linguagem infinita, e que portanto não pode parar (veja Exemplo 7.3), ou para um procedimento que calcula uma função que transforma cadeias de entrada em cadeias de saída (veja Exemplo 7.2).

Exemplo 7.3: Vamos descrever a mT M que enumera os elementos de Nat^2, codificados usando elementos de L_1. Note que M não recebe nenhuma entrada. As diversas cadeias geradas estarão separadas por $'s.

1. Inicialmente, M escreve $$ na fita, enumerando o par (0, 0).
2. Para cada cadeia $x=a^ib^j$ não considerada na fita,
   M acrescenta no fim da fita a cadeia $ax\$xb\$ = a^{i+1}b^j\$a^ib^{j+1}\$$,

correspondente à enumeração de (i+1, j) e de (i, j+1).

Em seguida M marca a cadeia a^ib^j como já considerada.

O conteúdo da fita de M, após alguns passos, seria

$$a$b$aaababbbaaaaab

onde $ marca a separação entre as cadeias já consideradas e as ainda não consideradas. O processo se repete indefinidamente. Observe que é necessário conhecer a codificação, para identificar quais os elementos de Nat^2 que foram enumerados.

A construção da máquina completa fica como exercício.

Exercício 7.4: Descreva mT's que enumeram Nat^2 de acordo com as outras duas codificações consideradas.

Exemplo 7.4: Vamos construir M, uma mT que soma dois números em unário, ou seja um procedimento que implementa a soma unária. M recebe como entrada uma cadeia A da forma xy$, em que x e y são cadeias de 1's, representando dois naturais i e j em unário. Ao final o resultado z (a cadeia de 1's que representa i+j) deve ficar na fita, delimitado por $'s; para "remover" quaisquer caracteres que não façam parte de z, usaremos o símbolo X.

A idéia é a seguinte:

substituímos o segundo $ por 1;
substituímos o último 1 por $;
substituímos o terceiro $ por X.

Assim, com entrada $11$111$, representando as parcelas 2 e 3, teremos

$11$111$ ⊢—* 111111 ⊢—* $11$11$$ ⊢—* 11111X

Ignorando o X, esta última cadeia é a representação do resultado 5. Outro exemplo seria 0+0=0, em que a entrada $$$ deveria ser transformada em $$X.

A máquina de Turing pode ter estados { A, … F } sendo A inicial e F final, com as seguintes transições:

ABR
B1B1R B$C1R
C1C1R CDL
D1E$R
E$FXR

de maneira que para a entrada $11$111$ teríamos a seguinte computação:

A$11$111$ ⊢— $B11$111$ ⊢— $1B1$111$ ⊢— $11B$111$ ⊢— $111C111$
⊢— $1111C11$ ⊢— $11111C1$ ⊢— $111111C$ ⊢— $11111D1$ ⊢— 11111E$
⊢— 11111XF

ou seja, 2+3=5. Semelhantemente, para a entrada $$$ teríamos

A$$$ ⊢— $B$$ ⊢— $1C$ ⊢— $D1$ ⊢— $$E$ ⊢— $$XF

ou seja, 0+0=0.

Exemplo 7.5: Seja descrever uma mT M, que soma dois números em binário, ou seja mais uma mT que implementa o procedimento soma. M recebe como entrada uma cadeia A da forma xy$, em que x e y são cadeias de 0's e 1's, representando dois naturais i e j em binário. Vamos supor que o resultado z (a cadeia que representa i+j) deve ficar no fim da parte escrita, com delimitadores especiais: $$z$$. (Antes do primeiro $$ fica o "lixo" resultante das computações.)

O plano de execução de M é o seguinte:

M “decora” um bit de x, soma com o bit correspondente de y e escreve no fim da fita o bit correspondente de z. Se houver um “carry” (“vai-um”), este deve ser considerado da próxima vez. Este processo é repetido até que os bits de x ou de y estejam todos marcados. A partir deste ponto, só os bits da outra cadeia são levados em consideração, até serem todos marcados. Note que os bits de z estão sendo gerados na ordem invertida, de forma que a sentença no fim da fita é efetivamente z^R. Para terminar, os bits de z são copiados um a um, a partir do último, para o fim da fita, onde a cadeia z é construída.

Note que os bits decorados e o carry devem ser anotados em estados especiais para cada combinação de valores. Os conteúdos da fita em alguns pontos da computação estão mostrados a seguir, para a soma 3+7=10.

$11$111$
$1x$11x$0
xx1xx$01
xxxxx$010
xxxxx$0101
xxxxx$010x$$1
xxxxx$01xx$$10

xxxxx$0xxx$$101
xxxxx$xxxx$$1010
xxxxx$xxxx$$1010$$

Outra possibilidade é a seguinte: usamos técnicas semelhantes às do Exemplo 7.2 e do Exercício 7.2, para as conversões de binário para unário e vice-versa; a soma pode ser feita em unário. Neste caso, podemos ter conteúdos de fita, em alguns instantes da computação, como os seguintes:

$11$111$	$3 = 11_2$; $7=111_2$
xx111$111$	$3= 111_1$
xxxxx$111$1111111$	$7 = 1111111_1$
xxxxx1111111111x	$10 = 1111111111_1$
xxxxx$xxxxxxxxxx$x1010	$10 = 1010_2$

A efetiva construção da máquina, por uma das duas maneiras mencionadas, fica como um exercício.

7.4 - Técnicas para a construção de máquinas de Turing

Vamos apresentar a seguir algumas técnicas de construção de mT's. Em virtude do interesse apenas teórico das mT's, apenas algumas máquinas muito pequenas são efetivamente construídas. Isso se deve ao fato de que a construção de uma mT pode ser extremamente trabalhosa. Assim, normalmente é feita apenas uma descrição da máquina, como fizemos nos exemplos 7.3 e 7.5. Vamos examinar aqui algumas técnicas, que podem ser usadas para efetivamente construir mT's, mas que são mais frequentemente usadas em descrições de mT's. Vamos apresentar também algumas modificações do modelo básico, que não alteram a capacidade de computação da mT.

1. *anotação de informações no controle finito*. A mT pode mudar de estado para guardar uma informação adicional. Por exemplo, a mT pode decorar o símbolo que encontrou em uma posição, para movê-lo para outra, ou para compará-lo com outro símbolo em outra posição.
2. *cópia e verificação de símbolos*, possívelmente em ordem inversa. Uma mT pode decorar e marcar símbolos em uma parte A (devidamente demarcada) da fita, para alguma ação em outra parte B (também devidamente demarcada) da fita. O símbolo decorado em A pode ser copiado em B, ou pode ser comparado com outro símbolo em B para verificar se duas cadeias são iguais. Se em A começamos pelo último símbolo, e em B pelo primeiro, a operação se dá na ordem inversa.
3. *Múltiplas trilhas*. Podemos incluir no alfabeto Γ símbolos que sejam tuplas de outros símbolos. Por exemplo, Γ pode conter pares de elementos de Σ. Esta construção permite simular a existência de várias trilhas na fita. Por exemplo, usando triplas como símbolos, (a_1, b_1, c_1) (a_2, b_2, c_2) ... (a_n, b_n, c_n) pode ser vista como uma cadeia de n símbolos (que são triplas) ou pode ser vista como a combinação de três cadeias, uma em cada trilha: $a_1a_2...a_n$, $b_1b_2...b_n$, e $c_1c_2...c_n$.

A situação está representada na figura abaixo, onde podemos ver a cadeia de triplas,

…	(a1, b1, c1)	(a2, b2, c2)	…	(an, bn, cn)	…

ou, alternativamente, três trilhas separadas.

	a1	a2	…	an	
…	b1	b2	…	bn	…
	c1	c2	…	cn	

Em alguns casos, queremos verificar alguma condição sobre uma cadeia x, o que usualmente destrói a cadeia, mas, para alguma finalidade, a cadeia deve ser preservada. Podemos construir uma cópia de x na "outra" trilha, simplesmente trocando cada símbolo a_i de x por (a_i, a_i). Se $x = a_1a_2...a_n$, então x pode ser substituído por (a_1, a_1) (a_2, a_2) ... (a_n, a_n). Uma das cópias pode ser destruída no processamento, e ao final x é reconstituído a partir da cópia salva na outra trilha.

4. *Subrotinas*. Podemos considerar que qualquer ação para a qual uma mT foi construída (ou descrita) pode ser usada na construção de outra mT, em que a ação é desejada como parte do processamento. Basta para isso incluir uma cópia dos estados e transições da subrotina. Se quisermos fazer várias chamadas, entretanto, será necessário dar novos nomes aos estados, para isolar uma chamada da sub-rotina das demais.

5. *Fita infinita nos dois sentidos*. Se for interessante, podemos considerar que a fita da mT se estende de forma ilimitada também para a esquerda da posição inicial. Para a simulação de uma mT com fita infinita nos dois sentidos, bastaria considerar uma fita semi-infinita com duas trilhas, cada qual representando uma das "metades". Podemos anotar no controle finito em qual das duas metades da fita a mT está no momento, e ignorar a outra metade.

Por exemplo, se o conteúdo da fita "infinita" é

…	a_{-3}	a_{-2}	a_{-1}	a_0	a_1	a_2	a_3	…

podemos usar uma fita semi-infinita para representar o mesmo conteúdo:

a_0	a_1	a_2	a_3	…
a_{-1}	a_{-2}	a_{-3}	a_{-4}	…

ou seja, por

(a_0, a_{-1})	(a_1, a_{-2})	(a_2,a_{-3})	(a_3, a_{-4})	…

6. *Várias fitas*. Podemos considerar uma mT com várias fitas, a que podem ser atribuídas funções diferentes, caso desejado. Por exemplo, podemos considerar que a máquina tem uma fita de entrada, uma fita de saída, e uma ou mais fitas de memória. A simulação de uma mT com n fitas pode ser feita em uma mT com um fita só, cujos símbolos são tuplas, que contém todos os símbolos de todas as fitas, e marcas com as posições de todas as cabeças.

7. *Máquinas de Turing não determinísticas*. Para definir uma máquina não determinística, devemos considerar uma função δ que pode oferecer mais de uma possibilidade de transição, a partir de cada configuração. Uma mT não determinística pode ser simulada por uma mt determinística. Isto será descrito a seguir.

Definição. Uma máquina de Turing não determinística M pode ser definida por uma tupla $M = < K, \Sigma, \Gamma, \delta, i, F >$, onde K é um conjunto (finito, não vazio) de estados, Γ é

o alfabeto (finito) de símbolos da fita, $\Sigma \subseteq \Gamma$ é o alfabeto de símbolos de entrada, δ é a função de transição, $i \in K$ é o estado inicial, e $F \subseteq K$ é o conjunto de estados finais.

A função de transição δ é um mapeamento $\delta: K \times \Gamma \rightarrow P(K \times \Gamma \times \{L, R\})$. Todos os valores de δ são conjunto finitos.

Configuração. Da mesma maneira que para uma mT determinística, uma *configuração* de uma mT é uma cadeia xqy, em que $xy \in \Gamma^*$ é o conteúdo da fita, sem incluir nenhum dos brancos que existem à direita, e $q \in K$ é o estado corrente. Semelhantemente, a configuração inicial para a entrada x é ix, e uma configuração xfy é final se o estado f é final.

Mudança de configuração. A definição da relação "mudança de configuração", $\vdash\!\!\!-$, não oferece surpresas:

- movimento para a direita:
  se $(p, b, R) \in \delta(q, a)$, então $xqay \vdash\!\!\!- xbpy$
  caso particular: se $(p, b, R) \in \delta(q, \Diamond)$, então $xq \vdash\!\!\!- xbp$
- movimento para a esquerda:
  se $(p, b, L) \in \delta(q, a)$, então $xcqay \vdash\!\!\!- xpcby$
  caso particular: se $(p, b, L) \in \delta(q, \Diamond)$, então $xcq \vdash\!\!\!- xpcb$

Naturalmente, se $\delta(q, a)=\varnothing$, nenhuma configuração pode ser alcançada a partir de uma configuração xqay, e a máquina pára. Caso particular: se a configuração for xq, a máquina pára quando $\delta(q, \Diamond)=\varnothing$.

Linguagem reconhecida por uma mT não determinística. A linguagem aceita, ou reconhecida por uma mT M é definida por

$$L(M) = \{ x \in \Sigma^* \mid ix \vdash\!\!\!-^* yfz, \text{ onde } f \in F \}$$

A aceitação se dá quando o estado final é atingido, não interessando em que ponto da fita está a cabeça, se há possibilidade de continuar a partir desse ponto, ou se há outras configurações não finais atingíveis com a mesma entrada. Como nos outros casos em que definimos máquinas não determinísticas, só nos interessa saber se existe ou não a possibilidade de atingir um estado final.

Exemplo 7.6. A mT a seguir é não determinística. A linguagem aceita por essa máquina é $\{ xx \mid x \in \{0,1\}^* \}$.

A0BxR	A1CxR			A◊KxR
B0B0R B0DyL	B1B1R			
C0C0R	C1C1R C1DyL			
D0D0L	D1D1L	DxExR	DyDyL	
E0FxR	E1GxR		EyJyR	
F0F0R	F1F1R		FyHyR	
G0G0R	G1G1R		GyIyR	
H0DyL			HyHyR	
	I1DyL		IyIyR	
			JyJyR	J◊KxR

A escolha não determinística é feita em dois estados, B e C. Em ambos os casos, trata-se de adivinhar qual o primeiro símbolo da segunda metade da cadeia xx. No caso do

estado B este símbolo é um 0; no caso do estado C, trata-se de um 1. Os demais símbolos são apenas conferidos nos estados H e I. Abaixo, um exemplo de computação:

A11011101	xC1011101	x1C011101	x10C11101	x101C1101
x10D1y101	x1D01y101	xD101y101	Dx101y101	xE101y101
xxG01y101	xx0G1y101	xx01Gy101	xx01yI101	xx01Dyy01
xx0D1yy01	xxD01yy01	xDx01yy01	xxE01yy01	xxxF1yy01
xxx1Fyy01	xxx1yHy01	xxx1yyH01	xxx1yDyy1	xxx1Dyyy1
xxxD1yyy1	xxDx1yyy1	xxxE1yyy1	xxxxGyyy1	xxxxyIyy1
xxxxyyIy1	xxxxyyyI1	xxxxyyDyy	xxxxyDyyy	xxxxDyyyy
xxxDxyyyy	xxxxEyyyy	xxxxyJyyy	xxxxyyJyy	xxxxyyyJy
xxxxyyyyJ	xxxxyyyyxK			

Teorema 7.1. Os modelos determinístico e não determinístico da máquina de Turing são equivalentes.

Dem. Basta mostrar como simular uma mT não determinística M em uma mT determinística M'. A fita de M' conterá diversas configurações de M. Se alguma delas for final, M' aceitará sua entrada. O símbolo $ será usado como separador de configurações.

1. Inicialmente, M' encontra na sua fita a entrada x de M. Acrescentando marcas e o estado inicial i de M, M' escreve na fita a configuração inicial de M para x, C_0=ix, de forma que a fita de M' contém $\$C_0\$$.

2. Para cada configuração C não considerada na fita, M' constrói no fim da fita, as configurações atingíveis a partir de C, separadas por $. Se M tem m opções de escolha, e as configurações atingíveis são C_1, C_2, ... C_m, neste passo a cadeia $\$C_1\$C_2\$\ldots\$C_m\$$ é acrescentada no final da fita.

3. Se alguma das configurações de M construídas por M' for final, M' pára e aceita a entrada.

Para construir as configurações, observamos que no máximo três símbolos são alterados, quando se passa de uma configuração C para uma configuração C' em um passo. Ou seja, boa parte do trabalho de construção de C' é simplesmente um trabalho de cópia.

Exercício 7.5. Construa uma máquina determinística que aceite a linguagem do Exemplo 7.6, sem usar a construção sugerida no Teorema 7.1.

7.5 - Linguagens tipo 0 e conjuntos recursivamente enumeráveis

Vamos agora caracterizar a classe de linguagens aceitas por mT's, mostrando que se trata da mesma classe de linguagens geradas por gramáticas (tipo 0), ou seja, via tese de Church, que a classe dos conjuntos recursivamente enumeráveis é idêntica à classe das linguagens tipo 0. Este resultado será objeto dos dois teoremas a seguir.

Teorema 7.2: Toda linguagem tipo 0 é recursivamente enumerável.

Demonstração. Seja L uma linguagem tipo 0. Seja G uma gramática tal que L = L(G). Vamos descrever a construção de uma mt M, não determinística, que aceita L(G).

M ignora sua entrada, e escreve S no fim da fita. Em seguida, M simula uma derivação em G escolhendo e aplicando não-deterministicamente uma regra em cada passo. Para isso, escolhida a regra $\alpha \rightarrow \beta$, M encontra (não-deterministicamente) uma ocorrência de α e substitui esta ocorrência por β. Quando esta derivação tiver como resultado a cadeia x, M terá em sua fita duas cópias de x que devem ser comparadas. Neste ponto M passa para um estado final.

Uma outra possibilidade de construção de M é através de uma mT determinística, que em vez de "adivinhar" as regras a serem usadas, testa todas, sucessivamente.

O próximo resultado é a recíproca do teorema acima.

Teorema 7.3. Toda linguagem recursivamente enumerável é tipo 0.

Demonstração. Seja L uma linguagem r.e. Portanto, L é aceita por alguma mT (determinística) M. Vamos mostrar que L é tipo 0, construindo uma gramática (tipo 0) G que aceita L. Uma derivação em G simula uma computação de M, e tem o seguinte aspecto:

$$S \Rightarrow^* ix\$x \Rightarrow^* yfz\$x \Rightarrow^* x$$

A partir do símbolo inicial S, derivamos uma cadeia da forma ix$x, onde i é o estado inicial de M, de forma que ix é a configuração inicial de M com entrada x. A simulação da computação de M será feita a partir dessa configuração inicial. A segunda cópia de x fica inalterada até o final. A cadeia x contém apenas símbolos do alfabeto de entrada de M, que são os terminais de M; o separador $ e os símbolos de estados de M são nãoterminais de G. Para esta fase as regras de G podem ser obtidas adaptando as regras de uma gramática sensível ao contexto de $\{ xx \mid x \in \Sigma^* \}$, acrescentando os símbolos nãoterminais i e $.

A segunda parte,

$$ix\$x \Rightarrow^* yfz\$x$$

simula a computação de M com entrada x. Para isto, as regras de G são construídas a partir das transições de M. Para cada valor $\delta(q, a)$, G tem uma ou mais regras que simulam a mudança de configuração. Se $\delta(q, a) = (p, b, R)$, basta uma regra é $qa \rightarrow bp$. Se $\delta(q, a) = (p, b, L)$, há necessidade de uma regra $cqa \rightarrow pcb$ para cada símbolo c do alfabeto da fita. O fim da fita de M é identificado pela presença do separador $. Portanto, quando M lê um branco, o separador $ é alcançado, e transições em que um $\Diamond$ é envolvido tem regras correspondentes das formas $q\$ \rightarrow pb\$$ e $cq\$ \rightarrow pcb\$$. Supondo que x pertence a L(M), uma configuração final yfz será eventualmente alcançada, com $ix \vdash^* yfz$. Portanto, em G, será possível a derivação $ix\$x \Rightarrow^* yfz\$x$, onde f é um estado final.

Na última fase, os restos da computação de M são removidos, restando apenas a cópia não usada da cadeia x. Para obter $yfz\$x \Rightarrow^* x$, G tem regras $f \rightarrow X$ para cada estado final f. O nãoterminal X pode ser visto como um “apagador” usando regras $cX \rightarrow X$ e $Xc \rightarrow X$ para cada símbolo c do alfabeto da fita, e $X\$ \rightarrow \varepsilon$, podemos eliminar todos os vestígios da simulação, e ficar apenas com a cadeia x.

Por construção, G gera x se e somente se M aceita x. Para a construção da primeira parte de G, veja o exercício abaixo.

Exercício 7.6: Seja Σ um alfabeto qualquer, e seja $\Sigma' = \Sigma \cup \{ i, \$ \}$. Considere a linguagem L' no alfabeto Σ', dada por $L' = \{ ix\$x \mid x \in \Sigma'^* \}$. Mostre que L' é sensível ao contexto, construindo uma gsc para L'.

Nota: na demonstração do teorema anterior, i e $ são nãoterminais.

Este capítulo foi escrito a partir de uma versão inicial escrita com a colaboração de Luiz Carlos Castro Guedes

2jul99