Computabilidade: uma introdução

Nelma Moreira
Departamento de Ciência de Computadores
Faculdade de Ciências, Universidade do Porto
email: nam@ncc.up.pt

Revisão:1996
Revisão: Maio 2001
Revisão alargada:2003

Conteúdo

1 Computação efectiva

Pretende-se responder às seguintes questões:

- Que linguagens podem ser reconhecidas por algum tipo de autómato?
- O que é ser computável?
- Que linguagens são computáveis?
- Existem linguagens que não são computáveis? Isto é, existem problemas que não se podem resolver computacionalmente, que são indecidíveis?
- Existem máquinas universais, isto é, que aceitam todas as linguagens computáveis?

Ou por outras palavras, o que é que um computador pode fazer?

1.1 Ser computável

Existir um método algorítmico de resolver, isto é, uma sequência finita de instruções que possa ser efectuada mecanicamente.

Em estudos dos fundamentos da Matemática, iniciados por David Hilbert no início do séc. XX, pretendia-se reduzir a Matemática à manipulação pura de símbolos, e encontrar um algoritmo que determinasse a veracidade ou a falsidade de qualquer proposição matemática.

Foram propostos vários formalismos que se demonstrou serem equivalentes (no sentido de aceitarem as mesmas linguagens):

- Máquinas de Turing (Alan Turing, 1936)
- Funções recursivas parciais (Kurt Gödel, 1931)
- λ-Calculus (Alonso Church, 1933)
- Lógica combinatória (Haskell Curry, 1929)
- Gramáticas não restritas ou Tipo 0 (Noam Chomsky,1956)

e podemos acrescentar: linguagens de programação

- `C`,
- `Java`,
- `Haskell` ... etc.

1.2 Tese de Church-Turing

Embora não demonstrado, um problema *efectivamente computável*, i.e tem um método algorítmico para o resolver se e só se existe uma máquina de Turing que o resolve.

As razões pelas quais a quase totalidade dos investigadores pensa que é verdadeira são de vários tipos.

1. As máquinas de Turing são tão gerais que parecem poder simular qualquer possível computação.
2. Nunca ninguém descobriu um modelo "algorítmico" mais geral que as máquinas de Turing.
3. Vários investigadores, ao estudar modelos de computação suficientemente gerais, têm sempre chegado a "algo" que nunca é mais gera que as máquinas de Turing e, muitas vezes, *é equivalente* às máquinas de Turing.

Um modelo de computação diz-se *universal* se toda o problema efectivamente computável o for nesse modelo.

1.3 Existência de problemas não computáveis (indecidíveis)

Um problema corresponde a uma linguagem num alfabeto Σ. Resolvê-lo é determinar se um palavra pertence a essa linguagem.

E o número de linguagens num dado alfabeto não é numerável:

$\#\mathcal{P}(\Sigma^\star) = 2^{\#\Sigma^\star} = 2^{\#\mathbb{N}} (\succ \#\mathbb{N})$

Uma máquina (programa) que o resolva também pode ser visto como uma palavra num dado alfabeto. Mas então só há um numerável de máquinas...

Conclusão: Há mais problemas que programas...

Donde:

- Têm de existir problemas indecidíveis
- Mas pode ser difícil demonstrar que um dado problema é indecidível

1.4 Indecidibilidade dos sistemas formais

Um sistema formal dedutivo da lógica de predicados é constituído por um conjunto de axiomas e de regras de inferência.

Uma proposição é demonstrável no sistema se existir para ela uma dedução em que cada passo ou é um axioma ou resulta da aplicação das regras de inferência.

A axiomática de Peano (PA) é um sistema formal adequado para a teoria dos números inteiros.

Gödel mostrou que existiam proposições da teoria dos números que não podiam ser demonstradas ou contraditadas em PA (Teorema da incompletitude de Gödel).

Este resultado terminou todas as tentativas de reduzir a matemática a um sistema dedutivo formal.

1.5 Máquinas Universais

Todos os formalismos referidos são suficientemente poderosos para se aceitarem a si próprios! Isto é,

- têm programas que aceitam como dados codificações de programas.
- existem máquinas de Turing que aceitam como dados palavras que são a descrição de máquinas de Turing.
- podemos escrever em `C` um programa que interprete programas em `C`

Nota que a noção de Universalidade está na base da noção de programa guardado em memória e portanto na de software.

Relacionado com a Universalidade está a noção de Auto-referência. Esta capacidade vai permitir a construção de problemas indecidíveis.

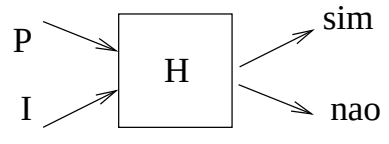
Por exemplo, (e infelizmente) pode-se demonstrar que não há algoritmos que dado um programa em `C` determinem o seu resultado ou se ele pára.

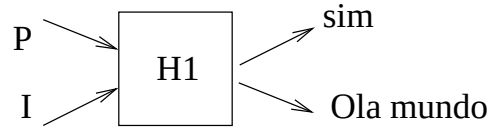
1.6 Um problema indecidível

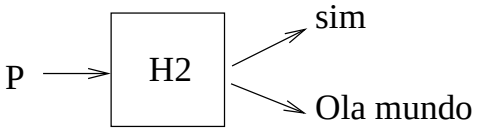
Dado um programa em `C`, determinar se os primeiros 9 caracteres que ele escreve são `Ola mundo`.

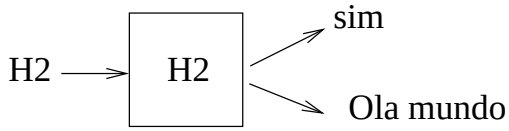
Vamos ver que não existe nenhum programa em C para resolver este problema.

- Por contradição, suponhamos que H é um programa que aceita como dados um programa P e um ficheiro I com os dados para P, e escreve `sim` se P resolver o problema, e escreve `não`, caso contrário: $H(I, P) =$ `sim` ou $H(I, P) =$ `nao`
- Modificamos H para um programa H_1 que actua como H excepto que quando H escreve `nao`, H_1 escreve `Ola mundo`.
- Modificamos H_1 para H_2. H_2 aceita apenas como dados o programa P, e actua como H_1 com P como seu programa e dados dele, isto é, $H_2(P) = H_1(P, P)$.









- H_2 não pode existir. Se existisse o que escrevia $H_2(H_2)$?
  - Se $H_2(H_2)$ escreve `sim` então H_2 com dados H_2 não escreve `Ola mundo`.
    Mas $H_2(H_2) = H_1(H_2, H_2)$ e $H_1(P, I)$ escreve `sim` se e só se P com dados I escreve `Ola mundo`.

Mas neste caso P é H_2 e I é H_2 ! Então $H_2(H_2)$ escrever `sim` implica que $H_2(H_2)$ escreve `Ola mundo`. Absurdo!

– Se $H_2(H_2)$ escreve `Ola mundo`, então $H_1(H_2, H_2)$ também o escreve o que indica que H_2 com dados H_2 não escreve `Ola mundo`.
Então $H_2(H_2)$ escrever `Ola mundo` implica que $H_2(H_2)$ não escreve `Ola mundo`. Absurdo!

2 Máquinas de Turing

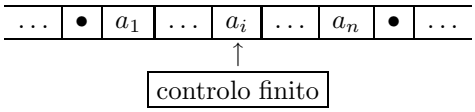
Este modelo de computação foi desenhado por Alan Turing[Tui36] para descrever o mínimo indispensável para se obter um método efectivo de computação.

Começamos por definir um tipo básico de máquina de Turing. Como veremos, existem muitos outros modelos, todos equivalentes a este.

2.1 Máquina de Turing básica

Uma **máquina de Turing** (MT) é constituída por um controlo finito (conjunto finito de estados), uma fita dividida em células, e uma cabeça de leitura/escrita (RW) que actua sobre uma célula da fita de cada vez.

Supomos que a fita é infinita para a esquerda e para a direita. No início, supomos que os dados – sequência finita de símbolos de um alfabeto – se encontram escritos na fita, um símbolo em cada célula, e as restantes células da fita contém um caracter especial da fita, designado por caracter *branco*.



Num passo de computação (movimento), dependendo do símbolo lido na fita pela cabeça e do estado do controlo finito, a máquina

1. muda de estado
2. escreve um símbolo na célula que está debaixo da cabeça
3. move a cabeça para a esquerda ou para a direita

Formalmente, uma máquina de Turing (MT) é um tuplo

$$M = (S, \Sigma, \Gamma, \delta, s_0, b, F)$$

onde

S é um conjunto finito de estados

Γ é o conjunto finito de *símbolos da fita*

Σ é um subconjunto de Γ que não incluí $\bullet$, é o conjunto dos *símbolos de entrada*

δ é a *função de transição*, função parcial de $S \times \Gamma$ em $S \times \Gamma \times \{\rightarrow, \leftarrow\}$

s_0 é o estado inicial

$\bullet$ é um símbolo de Γ, designado por *branco*

$F \subseteq S$ é o conjunto de estados finais

Uma *descrição instantânea* (ID) ou *configuração* representa o estado da fita, do controlo finito e da cabeça em cada instante. Embora a fita seja infinita, ao fim dum número finito de passos a cabeça só visitou um número finito de células. As restantes têm apenas caracteres *brancos*. Assim, basta considerar a fita entre o símbolo mais à esquerda não-branco e o mais à direita não branco.

Podemos representar uma configuração ou descrição instantânea (ID) por

$X_1 \cdots X_{i-1} s X_i \cdots X_n$ ou $(s, X_1 \cdots X_{i-1} \underline{X_i} \cdots X_n)$

onde:

1. s é o estado da MT
2. a cabeça da fita está a reconhecer o i-ésimo símbolo a partir da esquerda
3. $X_1 \ldots X_n$ é a sequência de caracteres da fita desde ou o caracter não-branco mais à esquerda ou o símbolo que está sobre a cabeça, conforme o que for mais à esquerda, e analogamente para o seu extremo direito.

A relação $\overset{M}{\vdash}$, um *movimento* ou *mudança de configuração num passo*, entre duas configurações é definida da seguinte forma. Seja

$$X_1 \cdots X_{i-1} s X_i \cdots X_n$$

uma configuração. Seja $\delta(s, X_i) = (s', Y, D)$. Se $i = 1$ e $D = \leftarrow$ então

$$s X_1 \cdots X_n \overset{M}{\vdash} s' \bullet Y X_{i+1} \cdots X_n$$

Se $i = 1$, $D = \leftarrow$ e $Y = \bullet$ então

$$s X_1 \cdots X_n \overset{M}{\vdash} s' X_{i+1} \cdots X_n$$

Se $i > 1$ e $D = \leftarrow$ então,

$$X_1 \cdots X_{i-1} s X_i \cdots X_n \overset{M}{\vdash} X_1 \cdots X_{i-2} s' X_{i-1} Y X_{i+1} \cdots X_n$$

Analogamente se $D = \rightarrow$ tem-se que

$$X_1 \cdots X_{i-1} s X_i \cdots X_n \overset{M}{\vdash} X_1 \cdots X_{i-1} Y s' X_{i+1} \cdots X_n$$

Excepto se:

1. Se $i = n$ então $X_1 \cdots X_{n-1} s X_n \overset{M}{\vdash} X_1 \cdots X_{n-1} Y s' \bullet$
2. Se $i = 1$ e $Y = \bullet$ então $s X_1 \cdots X_n \overset{M}{\vdash} s' X_2 \cdots X_{n-1}$

Diz-se que $xsy \overset{M^k}{\vdash} x's'y'$ se $x's'y'$ resulta de xsy em k movimentos e $xsy \overset{M^\star}{\vdash} x's'y'$ se existe $k > 0$ tal que $xsy \overset{M^k}{\vdash} x's'y'$.

2.1.1 Linguagem duma MT

Uma palavra $x \in \Sigma^\star$ é **aceite** por uma máquina de Turing M se com palavra x na fita e no estado inicial s_0, a máquina usando a função de transição δ, M entra num **estado final**. **Note-se** que mal a máquina atinja um estado final a computação pára, independentemente de ter lido ou não todos os símbolos de entrada. Por outro lado, se para um estado não final e um símbolo lido pela cabeça, a função de transição δ não estiver definida a máquina pára sem aceitar os dados (a palavra x). Pode ainda acontecer que a máquina nunca pare ... caso em que também não aceita a palavra x. Formalmente,

A *linguagem aceite por* M, é

$$L(M) = \{x \mid \ x \in \Sigma^\star \text{ e } s_0 x \overset{M^\star}{\vdash} x_1 s x_2 \text{ para algum } s \in F, \text{ e } x_1, x_2 \in \Gamma^\star\}$$

Pode-se definir aceitação por uma MT de outros modos, mas que são equivalentes. Por exemplo, apenas que a máquina páre (mesmo que não esteja num estado final).

Exemplo 2.1 *A máquina de Turing definida por,*

$$M = (\{s_0, s_1, s_2, s_3\}, \{0,1\}, \{0,1,\bullet, X\}, \bullet, s_0, \{s_4\})$$

$$\begin{array}{lll}
\delta(s_0,0) = (s_2, X, \rightarrow) & \delta(s_0,1) = (s_1, X, \rightarrow) & \delta(s_0, X) = (s_0, X, \rightarrow) \\
\delta(s_0,\bullet) = (s_4, \bullet, \leftarrow) & \delta(s_2,0) = (s_2, 0, \rightarrow) & \delta(s_2,1) = (s_3, X, \leftarrow) \\
\delta(s_2, X) = (s_2, X, \rightarrow) & \delta(s_1,1) = (s_1, 1, \rightarrow) & \delta(s_1,0) = (s_3, X, \leftarrow) \\
\delta(s_1, X) = (s_1, X, \rightarrow) & \delta(s_3,0) = (s_3, 0, \leftarrow) & \delta(s_3,1) = (s_3, 1, \leftarrow) \\
\delta(s_3, X) = (s_3, X, \leftarrow) & \delta(s_3,\bullet) = (s_0, \bullet, \rightarrow). &
\end{array}$$

cujo diagrama está na Figura 1, aceita a linguagem

$$L = \{x \in \{0,1\}^\star \mid x \textit{ tem igual número de } 1\textit{'s e de } 0\textit{'s}\}$$

No estado s_0 se a cabeça da máquina lê 0 (1), a máquina entra no estado s_2 (s_1), substitui o 0 (1) por X – indicando que o "apagou"– e procura um 1 (0), deslocando-se para a direita. No estado s_0 se encontra um caracter "branco", $\bullet$, entra no estado final s_4 e pára (significa que aceitou os dados!). Se no estado s_2 (s_1) encontra um 1 (um 0) substituí-o por um símbolo de fita X -indicando que o "apagou"– e desloca-se para a esquerda, entrando no estado s_3 até encontrar um "branco". Quando encontra um "branco" passa para o estado s_0. E repete o ciclo enquanto houverem 0's ou/e 1's. Termina com sucesso no estado s_4. Caso contrário, termina num estado não final. Esta máquina pára sempre!

Exemplo 2.2 *A máquina de Turing definida por*

$$M = (\{s_0, s_1, s_2, s_3, s_4\}, \{0,1\}, \{0,1,X,Y,\bullet\}, \delta, s_0, \bullet, \{s_4\})$$

onde,

$$\begin{array}{ll}
\delta(s_0,0) = (s_1, X, \rightarrow) & \delta(s_0, Y) = (s_3, Y, \rightarrow) \\
\delta(s_1,0) = (s_1, 0, \rightarrow) & \delta(s_1,1) = (s_2, Y, \leftarrow) \\
\delta(s_1, Y) = (s_1, Y, \rightarrow) & \delta(s_2,0) = (s_2, 0, \leftarrow) \\
\delta(s_2, X) = (s_0, X, \rightarrow) & \delta(s_2, Y) = (s_2, Y, \leftarrow) \\
\delta(s_3, Y) = (s_3, Y, \rightarrow) & \delta(s_3,\bullet) = (s_4, \bullet, \rightarrow)
\end{array}$$

aceita a linguagem

$$L = \{0^n 1^n \mid \ n \geq 1\}$$

Verifica!

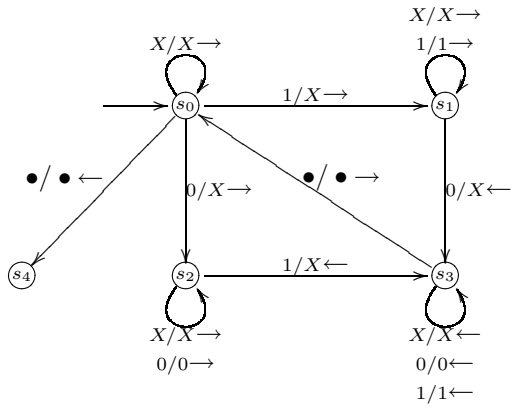


Figura 1: Máquina de Turing para $\{x \in \{0,1\}^\star \mid x \text{ tem igual número de 1's e de 0's}\}$q

Exemplo 2.3 *Sendo* $L = \{ww^r \mid w \in \{0,1\}^\star\}$ *onde* w^r *corresponde ao inverso de* w *, a máquina de Turing a seguir apresentada aceita* L*.*

$$M = (\{s_0, s_1, s_2, s_4, s_5, s_6, s_7\}, \{0,1\}, \{0,1,X,\bullet\}, \delta, s_0, \bullet, \{s_6\})$$

$$\begin{array}{lll}
\delta(s_0,0) = (s_1, X, \rightarrow) & \delta(s_0,1) = (s_2, X, \rightarrow) & \delta(s_0, X) = (s_0, X, \rightarrow) \\
\delta(s_0,\bullet) = (s_6, \bullet, \rightarrow) & \delta(s_1,0) = (s_1, 0, \rightarrow) & \delta(s_1, 1) = (s_1, 1, \rightarrow) \\
\delta(s_1,\bullet) = (s_3, \bullet, \leftarrow) & \delta(s_2,0) = (s_2, 0, \rightarrow) & \delta(s_2, 1) = (s_2, 1, \rightarrow) \\
\delta(s_2,\bullet) = (s_4, \bullet, \leftarrow) & \delta(s_3,0) = (s_5, \bullet, \leftarrow) & \delta(s_3, 1) = (s_7, 1, \rightarrow) \\
\delta(s_3, X) = (s_7, \bullet, \rightarrow) & \delta(s_4,0) = (s_7, 1, \rightarrow) & \delta(s_4, 1) = (s_5, \bullet, \leftarrow) \\
\delta(s_4, X) = (s_7, X, \rightarrow) & \delta(s_5,0) = (s_5, 0, \leftarrow) & \delta(s_5, 1) = (s_5, 1, \leftarrow) \\
\delta(s_5, X) = (s_0, X, \rightarrow) & &
\end{array}$$

Se $x \in L$ *a máquina pára no estado* s_6*.*

Exercício 2.4 *Descreve uma máquina de Turing que aceite a linguagem* $\{0^n 1 0^n \mid n \geq 1\}$*.* $\diamond$

Exemplo 2.5 *Descreve uma máquina de Turing que aceite a linguagem* $\{\mathtt{a}^{n^2} \mid n \geq 1\}$*.*

Ideia*: ir gerando no fim da fita os quadrados e ir verificando se o número de* a*s é igual.*

1. *Como gerar os* n^2*? Temos de ter* n *e* n^2*...Porquê?. Vou ter sempre* n *e* $n^2 - n$*. Os símbolos* c *são* n *e os* b *são* $n^2 - n$*: Por exemplo, para* a^{16} *e a testar para* $n = 3$*:*

   aaaaaaaaaaaaaaaacccbbbbbb

   (a) *Começar por gerar* 1 *e* $1 = 1^2$

   (b) *Para gerar* $(n+1)^2$*:*

   Hipótese 1: *Incrementar o* n *e copiar o valor* n *vezes...*

   Hipótese 2: *Notar que* $(n+1)^2 = n^2 + 2n + 1$*. Então:*

   i. *tenho de copiar* n *duas vezes e acrescentar mais um símbolo. :* $n + n^2 - n + 2n + 1 = (n+1)^2$

      aaaaaaaaaaaaaaaacccbbbbbbDDDEEEb

ii. e depois mudar um b em c (porque tenho agora $n+1$)

`aaaaaaaaaaaaaaaaccccbbbbbbbbbbbb`

Hipótese 3: *Calcular a diferença $2n+1$ sem cópias*

2. *Como comparar? Usar a MT do 0^n1^n, não distinguindo os* `c` *de* `b`

Exercício 2.6 *Implementar as máquinas de Turing descrita no exemplo anterior. Simular a execução da máquina e observar que seu o tempo de execução (= número de passos). A terceira é pelo menos $O(n^2)$ mais eficiente...* $\diamond$

2.1.2 Linguagens recursivamente enumeráveis e recursivas

Se $x \in L(M)$ então M pára quando atinge um estado final. Caso contrário, M pode não parar ou parar num estado não final.

Uma linguagem diz-se **recursivamente enumerável** (r.e) ou **semi-decidível** se é aceite por uma máquina de Turing. A classe de linguagens recursivamente enumeráveis designa-se por $\mathcal{SD}$.

Note-se que sendo M uma máquina de Turing, o problema $x \in L(M)$? não é decidível (i.e computável). Se for verdade, sendo x a sequência de entrada (dados) de M, esta pára num número finito de movimentos (passos). Mas se $x \notin L(M)$, M pode não parar e portanto não se obtém resposta.

Designam-se por **recursivas** ou **decidíveis** as linguagens para as quais existe uma máquina de Turing que a reconhece, isto é, que pára para todos os dados e que pára num estado final se a palavra dada pertence à linguagem e pára num estado não final se a palavra dada não pertence à linguagem. A classe de linguagens recursivas designa-se por $\mathcal{D}$.

Proposição 2.7 $\mathcal{D} \subseteq \mathcal{SD}$.

Dem. Se $L \in \mathcal{D}$ então existe uma MT M que a reconhece. E portanto em particular aceita L. $\square$

Contudo, existem linguagens que não são sequer *semi-decidíveis* ...

Nota sobre problemas e linguagens Embora normalmente se use indiferentemente os termos recursiva/decidível e recursivamente enumerável/semi-decidível os primeiros devem aplicar-se a linguagens e os segundos a propriedades dessas linguagens.

Seja P uma propriedade sobre palavras e L uma linguagem, então:

P é decidível	$\Leftrightarrow$	$\{x \mid P(x)\}$ é recursiva
L é recursiva	$\Leftrightarrow$	$x \in L$ é decidível
P é semi-decidível	$\Leftrightarrow$	$\{x \mid P(x)\}$ é r.e.
L é r.e	$\Leftrightarrow$	$x \in L$ é semi-decidível

2.1.3 Máquinas reconhecedoras

Podemos, também, considerar máquinas de Turing *reconhecedoras* de linguagens, isto é, máquinas

$$M = (S, \Sigma, \Gamma, \bullet, \delta, s_0, \bullet, s_a, s_r)$$

onde há só dois estados finais, um estado de aceitação, s_a e um estado de rejeição, s_r. Neste caso, as máquinas param para quaisquer dados. Como veremos, isto equivale à máquina de Turing calcular uma função total de $\Sigma^\star$ em $\{0,1\}$.

2.1.4 Máquinas de Turing calculadoras de funções parciais

Uma máquina de Turing pode ser vista como uma calculadora de funções parciais dos inteiros nos inteiros:

$$f : \mathbb{N}^k \longrightarrow_p \mathbb{N}$$

Suponhamos que os inteiros estão codificados em unário. Então, um inteiro $i \geq 0$ é representado por 1^i. Se uma função tiver k argumentos $i_1, \ldots, i_k$, a sequência de entrada pode ser $1^{i_1}0 \ldots 01^{i_k}$. Se a máquina de Turing pára com a sequência 1^m na fita, então $f(i_1, \ldots, i_k) = m$.

Uma função parcial f diz-se **recursiva** se existir uma máquina de Turing que calcula a função para todos os valores em que está definida não terminando a execução caso contrário.

Uma função total f diz-se *recursiva total* se existir uma máquina de Turing que calcula a função para todos os valores do argumento, isto é, a MT pára para quaisquer argumentos.

Note-se que as funções recursivas totais correspondem a linguagens recursivas.

Exemplo 2.8 *Seja*

$$m \dot{-} n = \begin{cases} m - n & \text{se } m \geq n \\ 0 & \text{caso contrário} \end{cases}$$

A máquina de Turing $M = (\{s_0, s_1, \ldots, s_6\}, \{0,1\}, \{0,1,\bullet\}, \delta, s_0, \bullet, \emptyset)$ *onde*

$$\begin{array}{lll}
\delta(s_0,1) = (s_1, \bullet, \rightarrow) & \delta(s_1,1) = (s_1, 1, \rightarrow) & \delta(s_1,0) = (s_2, 0, \rightarrow) \\
\delta(s_2,0) = (s_2, 0, \rightarrow) & \delta(s_2,1) = (s_3, 0, \leftarrow) & \delta(s_3,1) = (s_3, 1, \leftarrow) \\
\delta(s_3,0) = (s_3, 0, \leftarrow) & \delta(s_3,\bullet) = (s_0, \bullet, \rightarrow) & \delta(s_2,\bullet) = (s_4, \bullet, \leftarrow) \\
\delta(s_4,0) = (s_4, \bullet, \leftarrow) & \delta(s_4,1) = (s_4, 1, \leftarrow) & \delta(s_4,\bullet) = (s_6, 1, \rightarrow) \\
\delta(s_0,0) = (s_5, \bullet, \rightarrow) & \delta(s_5,0) = (s_5, \bullet, \rightarrow) & \delta(s_5,1) = (s_5, \bullet, \rightarrow) \\
\delta(s_5,\bullet) = (s_6, \bullet, \rightarrow) & &
\end{array}$$

calcula a função $m \dot{-} n$, isto é com dados 1^m01^n calcula $1^{m \dot{-} n}$. A máquina vai sucessivamente apagando um 1 da representação de m e substituindo um 1 por 0 na representação de n, até que uma das seguintes condições ocorre:

- quando procura um 1 na representação de n, não o encontra. Neste caso, $m > n$. Então substitui $n + 1$ $0's$ por um 1 e por n símbolos $\bullet$, deixando $m \dot{-} n$ 1s na fita.
- quando começa um ciclo não encontra nenhum 1. Neste caso, $m < n$. Então apaga toda a fita.

Exemplo 2.9 *A seguinte máquina de Turing permite calcular a função* $f(n) = n + 1$*, supondo os inteiros codificados em "unário".*

$$M = (\{s_0, s_1\}, \{0,1\}, \{0,1,\bullet\}, \delta, s_0, \bullet, \{s_2\})$$

$\delta(s_0, 1) = (s_0, 1, \rightarrow)$	*move-se para a direita*
$\delta(s_0, \bullet) = (s_1, 1, \leftarrow)$	*muda o primeiro branco por* 1
$\delta(s_1, 1) = (s_1, 1, \leftarrow)$	*move-se para a esquerda*
$\delta(s_1, \bullet) = (s_2, \bullet, \rightarrow)$	*pára no estado* s_2 *com a cabeça no primeiro dígito do resultado*

Exercício 2.10 *Descreve máquinas de Turing que permitam calcular as seguintes funções, supondo os inteiros codificados em "unário":*

(i) $f(n) = n^2$

(ii) $f(n, m) = n + m$

◇

Dum modo geral uma máquina de Turing M calcula uma função parcial f de $\Sigma^\star$ em $\Sigma^\star$ se dada como sequência de entrada $x \in \Sigma^\star$ a configuração final de M é $f(x)s\bullet$, para um estado final s.

Exemplo 2.11 *Descrever uma máquina de Turing que reconhece* $\{a^n \mid n \text{ é primo}\}$

Vamos ver como se pode implementar uma máquina de Turing que com dados a^n *pára num estado final se* n *é primo e num estado não final caso contrário. Supomos* $n > 2$*.*

Embora haja algoritmos mais eficientes, podemos determinar se um inteiro n é primo se não for divisível por nenhum inteiro entre 2 e n-1.

Então, basta

- *gerar (em unário) os inteiros k de 2 a n-1*
- *determinar se k divide n: para tal podemos usar subtracções sucessivas*

Se $n = 13$ *então a fita no inicio tem:*

$aaaaaaaaaaaaa \bullet \bullet \ldots$

Para gerar $k = 2..13$ *coloca-se um separador no fim dos dados e a seguir o primeiro valor de k. Fica:*

$aaaaaaaaaaaaabaa \bullet \bullet \ldots$

Nota *que para obter esta (e as seguintes) configurações são necessários muitos movimentos da MT!*

Para determinar se 13 é divisível por 2 basta ir subtraindo (como no exemplo anterior) a^k *a* a^n*, mas sem eliminar* a^n*:*

$Xaaaaaaaaaaaabaa \bullet \bullet \ldots$

$Xaaaaaaaaaaaab Xa \bullet \bullet \ldots$

$XXaaaaaaaaaaab Xa \bullet \bullet \ldots$

$XXaaaaaaaaaaab XX \bullet \bullet \ldots$

No fim da primeira subtracção tem de se restaurar a^k

$XXXaaaaaaaaaabXX \bullet \bullet \ldots$

$XXXaaaaaaaaaabXa \bullet \bullet \ldots$

e continuar até ter a^n *todo marcado. Aí , se* k *divide* n *então* n *não é primo.*

Caso contrário incrementa-se de uma unidade k*. Se* $k = n$ *então termina e* n *é primo, senão volta ao processo anterior.*

Exercício 2.12 *Implementa a máquina de Turing descrita. Simula a execução da máquina e observa que seu o tempo de execução (número de passos).* $\diamond$

Exercício 2.13 *Escreve uma máquina de Turing que utilize o crivo de Eratóstenes para reconhecer a mesma linguagem.* $\diamond$

Função característica duma linguagem Seja $L \subseteq \Sigma^\star$ uma linguagem. A **função característica** da linguagem L é uma função $f : \Sigma^\star \to \{0, 1\}$ tal que $f(x) = 1$ se $x \in L$ e $f(x) = 0$ se $x \notin L$.

Exercício 2.14 *Mostra que a função característica de uma linguagem é*

1. *recursiva total sse a linguagem é recursiva.*
2. *recursiva sse a linguagem é recursivamente enumerável.*

$\diamond$

2.2 Comparação com outros modelos de computação

2.2.1 Máquinas de Turing e Autómatos Finitos

Proposição 2.15 *Se* L *é regular* ($L \in \mathcal{R}$) *então* L *é recursiva.* ($L \in \mathcal{D}$).

Dem. Seja $L = L(A)$ e $A = (S, \Sigma, \delta, s_0, F)$ um AFD então,

$$M = (S \cup \{s_f\}, \Sigma, \Sigma \cup \{\bullet\}, \delta_M, s_0, \bullet, \{s_f\})$$

e

- $\delta_M(s, a) = (s', a, \rightarrow)$ se $\delta(s, a) = s'$
- $\delta_M(s, \bullet) = (s_f, \bullet, \rightarrow)$, se $s \in F$.

Então M reconhece L. (Verifica!) $\square$

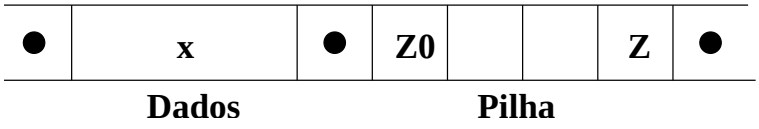


Figura 2: Máquina de Turing que simula um autómato de pilha

2.2.2 Máquinas de Turing e Autómatos de Pilha

Proposição 2.16 *Se L é independente de contexto ($L \in \mathcal{I_C}$) então L é recursiva. ($L \in \mathcal{D}$).*

Dem. (Ideia da demonstração) Dado um autómato de pilha $P = (S, \Sigma, \Gamma, \delta, Z_0, F)$ que aceita L por estados finais constrói-se uma MT M que dado x, simula P com dados x:

- começa por colocar no fim dos dados, o símbolo Z_0, que representa o topo da pilha.
- volta para trás e processa os dados: estando no estado s, para cada símbolo lido a, tem de verificar qual o símbolo do topo da pilha (fim direito dos caracteres não brancos) (Z), e se $\delta(s, a, Z) = (s', \gamma)$, escreve γ ao contrário a partir da posição em que Z estava, passa para o estado s' e anda para trás para ler o próximo símbolo de entrada. Se $\gamma = \epsilon$, apaga o último caracter (escreve um branco).
- quando terminar os dados e se estiver num estado final de P, muda para o estado (novo) final de M.

Nota, que tem de se introduzir vários estados extra para "percorrer a fita"entre os dados e a pilha...e tem de se ir marcando os símbolos dos dados já lidos. □

Exercício 2.17 *Termina a demonstração anterior...* ◇

2.2.3 Resumo

Os modelos de computação podem ser caracterizados em termos do tipo de memória que possuem e como se tem acesso a ela. Em particular comparando as máquinas de Turing com os autómatos finitos e de pilha, temos o seguinte quadro:

Memória	Acesso	Modelo de Computação	Gramáticas	Linguagens
Finita	Fixo	Autómatos Finitos $AFD \equiv AFND \equiv AFND\epsilon$	Regulares	$\mathcal{R}$
Ilimitada	Primeiro elemento da Pilha	Autómatos de Pilha	Ind. de Contexto	$\mathcal{I_C}$
Ilimitada	Sequencial (não restrito)	Máquinas de Turing $MTD \equiv MTND$	Tipo 0	$\mathcal{SD}$

com

$$\mathcal{R} \subset \mathcal{I_C} \subset \mathcal{SD}$$

Em resumo, todos os problemas que se podem resolver usando métodos computacionais (programas em linguagens de programação de uso geral) podem ser resolvidos por uma máquina de Turing ... é tudo uma questão de paciência... como se exemplifica nos exercícios a seguir propostos!

Exercício 2.18 *Descreve uma máquina de Turing que*

(a) dada uma palavra x não vazia de $\{0,1\}^\star$ calcule o dobro do número binário por ela representado depois de eliminar os seus zeros não significativos.

Inicial :	•	0	0	1	0	1	•	•	•
Final :	•	1	0	1	0	•	•	•	•

(b) dados dois números x e y em unário, separados por um zero, calcule a sua soma em unário. Por exemplo se esses números fossem 3 e 2 as configurações da fita deveriam ser

Inicial :	•	1	1	1	0	1	1	•	•
Final :	•	1	1	1	1	1	•	•	•

Sugestão: Procurar o 0 e se a seguir não estiver um "branco", substituir o 0 por um 1. Se não encontrar o 0 os dados estão errados... A seguir localizar o último 1 e substituir por um "branco".

(c) dados dois números x e y em unário separados por um zero calcule o seu produto. Por exemplo se esses números fossem 3 e 2 as configurações da fita deveriam ser

Inicial :	•	•	1	1	1	0	1	1	•	•
Final :	•	•	1	1	1	1	1	1	•	•

Uma solução:

$$M = (\{q_0, \ldots, q_{14}\}, \{1,0\}, \{0,1,\bullet,s,a\}, \delta, q_0, \bullet, \{q_{14}\})$$

$\delta(q_0,1) = (q_1,1,\leftarrow)$ $\quad \delta(q_1,\bullet) = (q_2,s,\rightarrow)$ $\quad \delta(q_2,\bullet) = (q_3,\bullet,\leftarrow)$ $\quad \delta(q_2,s) = (q_2,s,\rightarrow)$
$\delta(q_2,1) = (q_2,1,\rightarrow)$ $\quad \delta(q_2,0) = (q_2,0,\rightarrow)$ $\quad \delta(q_2,a) = (q_2,a,\rightarrow)$ $\quad \delta(q_3,1) = (q_4,\bullet,\leftarrow)$
$\delta(q_3,0) = (q_9,0,\leftarrow)$ $\quad \delta(q_4,1) = (q_4,1,\leftarrow)$ $\quad \delta(q_4,0) = (q_5,0,\leftarrow)$ $\quad \delta(q_5,s) = (q_8,s,\rightarrow)$
$\delta(q_5,1) = (q_6,a,e)$ $\quad \delta(q_5,a) = (q_5,a,e)$ $\quad \delta(q_6,\bullet) = (q_7,1,\rightarrow)$ $\quad \delta(q_6,s) = (q_6,s,e)$
$\delta(q_6,1) = (q_6,1,e)$ $\quad \delta(q_7,s) = (q_7,s,\rightarrow)$ $\quad \delta(q_7,1) = (q_7,1,\rightarrow)$ $\quad \delta(q_7,0) = (q_5,0,e)$
$\delta(q_7,a) = (q_7,a,\rightarrow)$ $\quad \delta(q_8,\bullet) = (q_3,\bullet,e)$ $\quad \delta(q_8,1) = (q_8,1,\rightarrow)$ $\quad \delta(q_8,0) = (q_8,0,\rightarrow)$
$\delta(q_8,a) = (q_8,1,\rightarrow)$ $\quad \delta(q_9,s) = (q_{10},\bullet,\rightarrow)$ $\quad \delta(q_9,1) = (q_9,1,e)$ $\quad \delta(q_10,\bullet) = (q_{11},\bullet,\rightarrow)$
$\delta(q_{10},1) = (q_{10},\bullet,\rightarrow)$ $\quad \delta(q_{10},0) = (q_{10},\bullet,\rightarrow)$ $\quad \delta(q_{10},a) = (q_{10},\bullet,\rightarrow)$ $\quad \delta(q_{11},\bullet) = (q_{12},\bullet,e)$
$\delta(q_{12},\bullet) = (q_{12},\bullet,e)$ $\quad \delta(q_{12},1) = (q_{13},1,e)$ $\quad \delta(q_{13},1) = (q_{13},1,e)$ $\quad \delta(q_{13},\bullet) = (q_{14},\bullet,d)$

Esta máquina depois de efectuar o produto coloca a cabeça no primeiro dígito desse valor em unário (estados q_{11} a q_{14}).

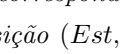
(d) $\star$ calcule a soma em binário de dois números em binário dados.

(e) $\star$ calcule o triplo em binário de um número em binário dado.

(f) $\star$ calcule o produto em binário de dois números em binário dados.

(g) $\star$ simule o "funcionamento"de um autómato finito determinístico de alfabeto $\{0,1\}$. Suponha que a descrição do autómato é dada na parte inicial da fita. Cada estado é representado ou por $[X]$ ou por $[X0]$ conforme é ou não final, sendo X um número em unário.

Inicial :	•	[	1	0	]	0	[11]	[10]	1	[10]	[11]	0	[11]	[11]	1	[10]	0	0	1	0	1	0	•	
Final :	•	[	1	0	]	0	[11]	[10]	1	[10]	[11]	0	[11]	[11]	1	[10]	<	1	>	:	1	1	•	

onde [11] *corresponde a*  *e analogamente para os outros estados.*

Cada transição (Est, a, Est') *é representada por* $[X]a[X']$ *em que* $[X]$, *e* $[X']$ *representam* Est *e* Est' *respectivamente, com* $a \in \{0, 1\}$. *As transições de um mesmo estado estão em posições consecutivas. Se o autómato encravar a máquina deve escrever na fita* $< 0 >$. *Senão escreve* $< 0 >:X$ *ou* $< 1 >:X$ *conforme a sequência é ou não aceite, sendo* X *o número do estado do autómato depois de ler a palavra. A palavra encontra-se na fita imediatamente após a descrição do autómato e deve ser também nessa posição que o resultado é escrito depois de apagar a palavra.*

Notar que na figura cada caracter ocupa uma cela distinta na fita, tendo-se marcado apenas as duas primeiras celas.

◇

2.3 Técnicas para escrever Máquinas de Turing

Pode ser conveniente supor que o controlo finito e a fita tem alguma estrutura. Isto não altera em nada o modelo da máquina de Turing básica.

Múltiplas pistas Os símbolos da fita tem *pistas*, isto é, são tuplos de símbolos que a cabeça reconhece simultaneamente: [a,X]. Neste caso um dos símbolos pode servir de marca, sem se perder a informação de qual o símbolo que estava.

Exemplo: Se cada célula com os dados for da forma $[a, \bullet]$, $a \in \Sigma$ e os brancos $[\bullet, \bullet]$, durante o processamento podemos marcar as células com $[a, X]$, $X \in \Gamma$, sem apagar o a...

Controlo finito com memória finita Os estados podem ser da forma $[s, A]$, onde A representa um (ou mais) símbolo de Γ que se pretende memorizado (por exemplo, para recordar o último símbolo lido). Isto reduz o número de estados necessários e torna o seu significado mais explícito.

Embora os simuladores de MTs usuais normalmente não implementam estas técnicas, são úteis para a demonstração de resultados. E se quiserem podem fazer um simulador que as implemente...

2.4 Variações sobre as Máquinas de Turing

Várias modificações (restrições/extensões) da máquina de Turing básica são possíveis sem que o modelo de computação obtido seja nem mais nem menos poderoso.

Entre elas considerem-se as seguintes restrições/extensões:

1. Alfabeto.
   (a) Codificação de um alfabeto noutro com pelo menos dois símbolos. Dê exemplos e conclua que o alfabeto não reduz a potência. Em particular podemos considerar máquinas de Turing em que o alfabeto tem apenas dois símbolos, para além do branco.
2. Aridade das funções
   (a) Codificação de pares de inteiros em inteiros. Considere a função bijectiva p de $\mathbf{N} \times \mathbf{N}$ em $\mathbf{N}$
   $$p(x, y) = \frac{(x + y)(x + y + 1)}{2} + x$$

(b) Codificações bijectivas de $\mathbf{N}^n$ em $\mathbf{N}$ usando a função p definida atrás.

Conclusão: As funções de aridade maior que 1 podem ser codificadas em funções de aridade 1.

3. Movimentos da cabeça
   (a) Máquinas com movimentos $\{\leftarrow, \rightarrow\}$ (esquerda/direita)
   (b) Máquinas com movimentos $\{\leftarrow, -, \rightarrow\}$ (esquerda/não mexe/direita)
4. Tamanho da fita
   (a) Máquinas com uma fita duplamente infinita
   (b) Máquinas com uma fita semi-infinita
5. Número e tipo de fitas
   (a) Máquinas com 1 fita RW
   (b) Máquinas com k fitas RW
   (c) Máquinas com uma fita de escrita de saída (W) e uma fita RW de trabalho.
   (d) Máquinas com uma fita de leitura (R), com os dados e k fitas RW de trabalho (eventualmente uma fita (W) de saída) (*MT off-line*).

Vamos apenas analisar a equivalência dos seguintes modelos: 3a e 3b;4b e4a; 5a e 5b.

2.4.1 Máquinas com movimentos $\{\leftarrow, -, \rightarrow\}$

Exceptuando a terceira componente da definição de δ estas máquinas são definidas como as MT básicas.

Se M_1 é uma MT com movimentos $\{\leftarrow, \rightarrow\}$ também é uma MT com movimentos $\{\leftarrow, -, \rightarrow\}$. Inversamente, suponhamos que $M_2 = (S_2, \Sigma_2, \Gamma_2, \delta_2, s_2, \bullet, F_2)$ é uma máquina com movimentos $\{\leftarrow, -, \rightarrow\}$. Vamos construir uma MT M_1, apenas com movimentos $\{\leftarrow, \rightarrow\}$, tal que se L é aceite por M_2 então, L é aceite por M_1. Quando a cabeça de M_2 fica parada, M_1 deve entrar num estado especial e movimentar-se uma célula para direita e outra para a esquerda, voltando à mesma célula de tal modo que simule o facto de M_2 não se mexer. Para tal, os estados de M_1 vão ser da forma $[s, X]$ ou $[s, Y]$ para cada estado $s \in S_2$, sendo os estados $[s, X]$ usados sempre que a cabeça de M_2 se movimente e os estados $[s, Y]$ usados quando a cabeça de M_2 não se mova.

Formalmente, tem-se $M_1 = (S_1, \Sigma_1, \Gamma_1, \delta_1, [s_2, X], \bullet, F_1)$ onde

$S_1 = \{[s, X], [s, Y] \mid s \in S_2\}$, $F_1 = F_2 \times \{X, Y\}$

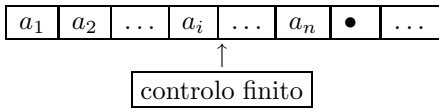
$\Sigma_1 = \Sigma_2$, $\Gamma_1 = \Gamma_2$

δ_1 define-se por, $\forall a \in \Gamma_1,\ s \in S_2$,:

1. $\delta_1([s, X], a) = ([s', X], a', \rightarrow)$, se $\delta_2(s, a) = (s', a', \rightarrow)$
2. $\delta_1([s, X], a) = ([s', X], a', \leftarrow)$ se $\delta_2(s, a) = (s', a', \leftarrow)$
3. $\delta_1([s, X], a) = ([s', Y], a', \rightarrow)$ se $\delta_2(s, a) = (s', a', -)$
4. $\delta_1([s, Y], a) = ([s, X], a, \leftarrow)$

2.4.2 Máquinas com uma fita semi-infinita

Uma MT com uma fita semi-infinita é definida como a máquina de Turing básica com a excepção de que existe uma célula mais à esquerda e portanto δ está limitada nos seus movimentos para a esquerda. No início supomos que as n células mais à esquerda contém a sequência de entrada ("dados") e as restantes o caracter de fita "branco".



Seja $X_1 \cdots X_{i-1} s X_i \cdots X_n$ uma configuração. Se $\delta(s, X_i) = (s', Y, D)$ e $i = 1$ então $D = \rightarrow$ obrigatoriamente.

A noção de configuração e da relação $\overset{M}{\vdash}$ são definidas analogamente, supondo-se a não existência de "brancos" desnecessários.

Exercício 2.19

Formaliza as noções associadas a uma MT com uma fita semi-infinita. $\diamond$

Teorema 2.20 *L é uma linguagem aceite por uma MT com uma fita duplamente infinita se e só se é aceite por uma MT com uma fita semi-infinita.*

Dem. ($\Leftarrow$) Uma MT M_2 com uma fita duplamente infinita pode simular facilmente uma MT com uma fita semi-infinita, M_1. Basta que M_2 marque a célula à esquerda da cabeça no início e depois simule a MT M_1.

($\Rightarrow$) Seja $M_2 = (S_2, \Sigma_2, \Gamma_2, \delta_2, s_2, \bullet, F_2)$ uma MT com fita duplamente infinita. Vamos construir M_1 com uma fita semi-infinita para a direita. Podemos imaginar que a fita de M_1 é dividida ao meio em duas pistas, de modo a que cada célula pode conter dois símbolos. Uma pista S correspondente às células da fita de M_2 à direita da célula onde inicialmente se encontra a cabeça e a outra pista I correspondente às células à esquerda dessa célula. Por exemplo:

Fita de M_2:

...	a_{-2}	a_{-1}	a_0	a_1	a_2	...

Fita de M_1:

a_0	a_1	a_2	...
$\triangleright$	a_{-1}	a_{-2},	...

onde a_0 é a célula que inicialmente está debaixo da cabeça de M_2. A primeira célula da fita de M_1 contém o símbolo $\triangleright$, na "pista" inferior, para indicar que é a célula mais à esquerda. O controlo finito de M_1 indicará se a cabeça de M_2 está debaixo duma célula correspondente à parte superior ou inferior da fita de M_1. Enquanto a cabeça de M_2 estiver à direita da célula inicial, a cabeça de M_1 "trabalhará" na parte superior da sua fita e movimentar-se-á como a de M_2. Se a cabeça de M_2 estiver à esquerda da célula inicial, a cabeça de M_1 actuará na parte infeior da fita e movimentar-se-á em sentido contrário ao da cabeça de M_2. Note-se que a noção de pistas é apenas conceptual, não havendo nenhuma alteração na definição de MT básica.

Formalmente seja $M_1 = (S_1, \Sigma_1, \Gamma_1, \delta_1, s_1, [\bullet, \bullet], F_1)$ onde,

$S_1 = \{[s, S], [s, I] \mid s \in S_2\} \cup \{s_1\}$

$\Sigma_1 = \{[a, \bullet] \mid a \in \Sigma_2\}$

$\Gamma_1 = \{[X, Y] \mid X \in \Gamma_2, Y \in \Gamma_2 \cup \{\triangleright\}\}$
$F_1 = \{[s, S], [s, I] \mid s \in F_2\}$

A função $\delta_1 : S_1 \times \Gamma_1 \longrightarrow_p S_1 \times \Gamma_1 \times \{\leftarrow, \rightarrow\}$ é definida por:

1. Para cada $a \in \Sigma_1 \cup \{\bullet\}$,

$$\delta_1(s_1, [a, \bullet]) = ([s, S], [X, \triangleright], \rightarrow) \qquad \text{se } \delta_2(s_2, a) = (s, X, \rightarrow)$$

2. Para cada $a \in \Sigma_1 \cup \{\bullet\}$,

$$\delta_1(s_1, [a, \bullet]) = ([s, I], [X, \triangleright], \rightarrow) \qquad \text{se } \delta_2(s_2, a) = (s, X, \leftarrow)$$

3. Para cada $[X, Y] \in \Gamma_1$, com $Y \neq \triangleright$ e $m = \rightarrow$ ou $m = \leftarrow$,

$$\delta_1([s, S], [X, Y]) = ([s', S], [Z, Y], m) \qquad \text{se } \delta_2(s, X) = (s', Z, m)$$

4. Para cada $[X, Y] \in \Gamma_1$, com $Y \neq \triangleright$ e se $m = \rightarrow$ então $\bar{m} = \leftarrow$ e se $m = \leftarrow$ então $\bar{m} = \rightarrow$,

$$\delta_1([s, I], [X, Y]) = ([s', I], [X, Z], m) \qquad \text{se } \delta_2(s, Y) = (s', Z, \bar{m})$$

5.

$$\delta_1([s, S], [X, \triangleright]) = \delta_1([s, I], [X, \triangleright]) \quad = \quad ([s', P], [Y, \triangleright], \rightarrow) \text{ se } \delta_2(s, X) = (s', Y, m)$$

onde $P = S$ se $m = \rightarrow$ e $P = I$ se $m = \leftarrow$

□

2.4.3 Máquinas com k fitas RW

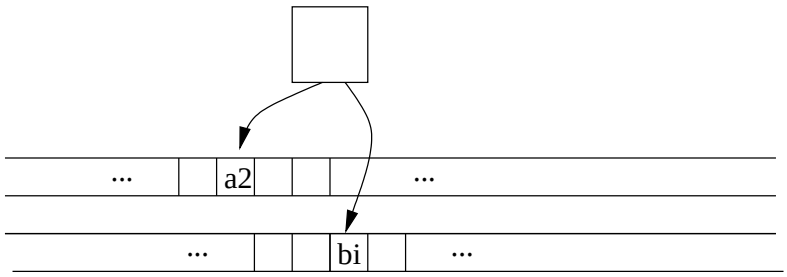


Figura 3: Máquina de Turing com 2 fitas

Uma máquina de Turing com k fitas, $k > 1$, consiste num controlo finito com k cabeças, uma para cada uma das k fitas. Vamos considerar que cada fita é duplamente infinita. Num movimento, dependendo do estado do controlo finito e dos símbolos lidos em cada uma das k fitas pelas k cabeças, a máquina

1. muda de estado
2. escreve um símbolo em cada uma das células que estão debaixo de cada cabeça
3. move cada cabeça, independentemente, para a esquerda ou para a direita ou não mexe

Inicialmente a sequência de entrada encontra-se na primeira fita e todas as outras estão em branco. Os restantes conceitos são análogos aos duma MT com uma só fita.

Exercício 2.21 *Descreve formalmente uma MT com k fitas e as noções de configuração, relação* $\overset{M}{\vdash}$ *e de linguagem aceite por uma MT de k fitas.* ◇

Teorema 2.22 *Se uma linguagem L é aceite por uma MT com k fitas, então L é aceite por uma MT com uma única fita.*

Dem. Suponhamos que L é aceite por uma MT com k fitas M_1. Podemos construir uma MT, M_2 com uma só fita e com $2k$ pistas, duas pistas para cada fita. Uma pista contém o conteúdo da fita correspondente de M_1 e a outra está em branco excepto para a célula actualmente lida pela cabeça correspondente de M_1. Esta célula contém um indicador, que supomos ser o símbolo X. Por exemplo se $k = 2$, a fita de M_2 podia ser

cabeça 1		•	X	...	•	...	•	
fita 1		a_1	a_2	...	a_i	...	a_m	
cabeça 2		•	•	...	X	...	•	
fita 2		b_1	b_2	...	b_i	...	b_m	

se em M_1 a cabeça da fita 1 estivesse no símbolo a_2 e e a da fita 2 no símbolo b_i.

Formalmente, ter-se-á $\Sigma_2 \subseteq (\{X, \bullet\} \times \Sigma_1)^k$.

O controlo finito de M_2 guardará para além do estado de M_1 o número de indicadores (símbolos X) à direita da cabeça de M_2 (n_i). Cada movimento de M_1 é simulado percorrendo a fita de M_2 da esquerda para a direita, para saber quais os símbolos actualmente lidos pelas k cabeças de M_1 e depois da direita para a esquerda, para actualizar esses valores e movimentar convenientemente os "indicadores". Inicialmente supõe-se que a cabeça de M_2 está na célula mais à esquerda que contém um indicador. Então, a cabeça de M_2 percorre a fita para a direita, e para cada célula com indicador "memoriza" o símbolo lido e diminui n_i. Esta memorização pode ser feita considerando que os estados de M_2 guardam também estes símbolos, i.e cada estado de M_2 corresponde a uma combinação de um estado de M_1 k-símbolos de Γ_1 e o número de indicadores. Quando $n_i = 0$ então M_2 tem informação suficiente para determinar o movimento de M_1. Então, a cabeça de M_2 percorre a fita da esquerda para a direita, e sempre que encontra um indicador modifica o símbolo lido e movimenta o indicador de acordo com o movimento da cabeça correspondente em M_1. Quando já actualizou todos os indicadores, M_2 muda o estado correspondente a M_1. Se o novo estado de M_1 é um estado final, o de M_2 também o é. □

Exercício 2.23 *Para $k = 2$ formaliza a máquina M_2 construída na demonstração anterior.* ◇

Exercício 2.24 *Estima o número de movimentos efectuados por M_2 para cada movimento de M_1. Analisa a eficiência do uso de MTs com k fitas, $k > 1$, em relação a MTs só com uma fita.* ◇

Exercício 2.25 *Descreve uma MT com 2 fitas que aceite a seguinte linguagem:*

$$L = \{wcw^r \mid w \in \{0,1\}^\star\}$$

e compare-a com a descrita no exemplo 2.3. Sugestão: Copiar os dados para a segunda fita e comparar os símbolos, em cada fita movimentando as cabeças em direcções opostas. ◇

2.4.4 Máquinas de Turing não determinísticas

Uma máquina de Turing é *não determinística* (MTND) se dado um estado e um símbolo lido, existirem várias hipóteses para o movimento seguinte, isto é,

$$\delta \subseteq (S \times \Gamma) \times (S \times \Gamma \times \{\leftarrow, \rightarrow\})$$

Uma MT não determinística aceita os dados se existe uma sequência de escolhas de movimentos que conduza a um estado de aceitação.

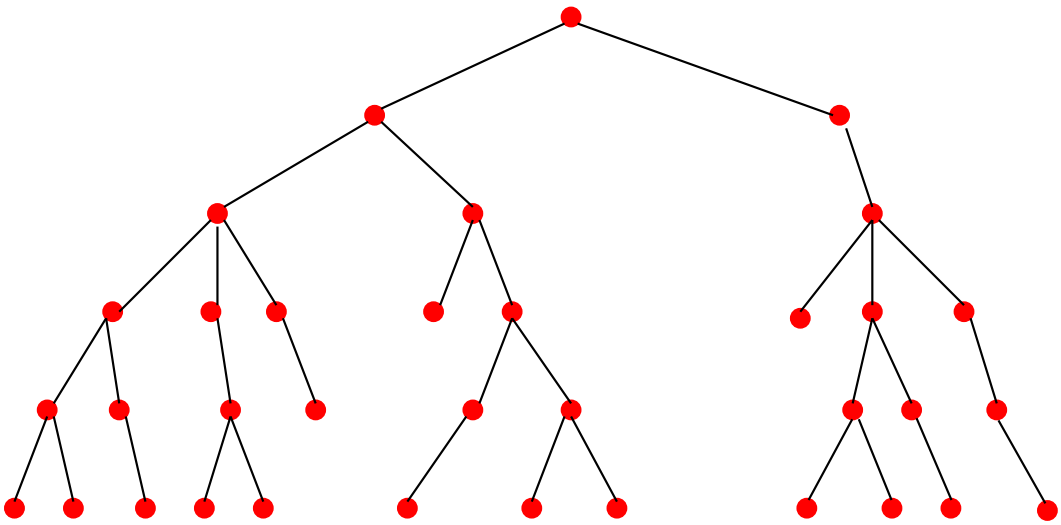


Figura 4: Passos de computação não-determinística

Uma $MTND$ não aceita os dados se nenhuma sequência de escolhas conduz a um estado final ou se não pára.

Analogamente, definem-se os conceitos de função calculada ou linguagem reconhecida).

Em contraste, as MT definidas anteriormente denominam-se *determinísticas*.

Contudo a adição de não determinismo não torna estas MTs um modelo de computação mais poderoso que as MT determinísticas.

Teorema 2.26 *Se uma linguagem L é aceite por uma máquina de Turing não determinística, M_1, então L é aceite por alguma máquina de Turing determinística, M_2.*

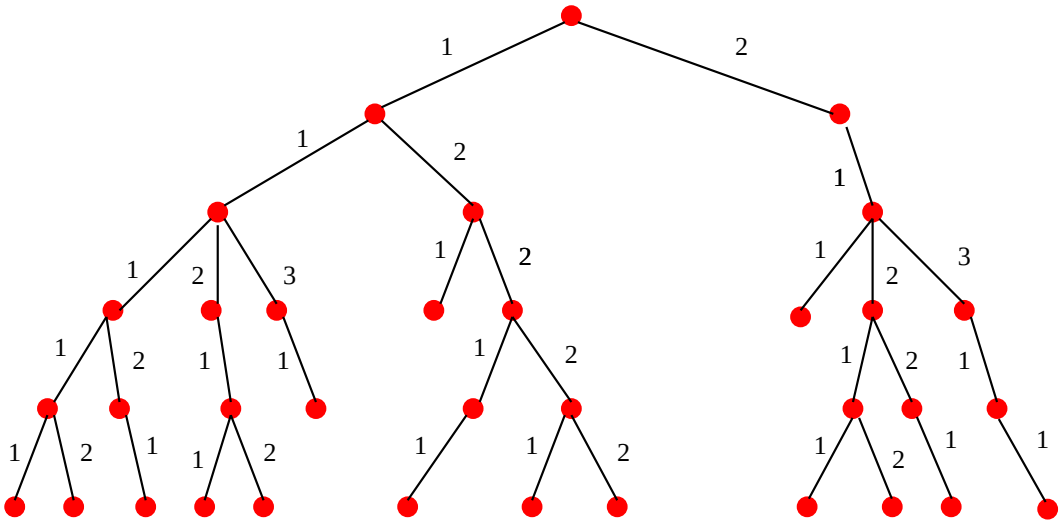
Dem. Para cada estado e símbolo de fita de M_1, existe um número finito de escolhas para o movimento seguinte. Estas escolhas podem ser numeradas 1, 2, ... Seja r o número máximo de escolhas de um par estado-símbolo. Então qualquer sequência finita de escolhas pode ser representada por uma sequência de dígitos de 1 a r.

Vamos construir M_2 com 3 fitas. A primeira contém a sequência de entrada. A segunda, gera sequências de dígitos de 1 a r dum modo sistemático. Por exemplo, por ordem crescente de comprimento e por ordem lexicográfica se de igual comprimento: 1, ..., r, 11,... 1r, ..., 21, ..., 2r, ..., 111, ..., rrr, ...

Para cada sequência gerada na segunda fita, M_2 copia a sequência de entrada para a terceira fita e simula M_1 na terceira fita, usando a sequência da segunda fita para decidir os movimentos de M_1.

Nota que cada sequência de escolhas corresponde a uma sequência de configurações de M_1.

Deste modo M_2 simula sucessivamente todas as sequências de k movimentos de M_1, $k = 1, 2, \ldots$



1,2,11,12,21, 111,112,113,121,122,211,212,213,1111, ...

M_2 explora a árvore de configurações de M_2 em largura.

Se M_1 atinge um estado de final, então M_2 aceita. Se nenhuma sequência de escolhas conduz a M_1 aceitar, M_2 também não aceita. □

Notar que:

- é essencial a ordem porque são geradas a possíveis sequências de escolhas (em largura e não em profundidade) de modo a evitar a simulação de um conjunto de escolhas infinito ...
- A máquina determinística executa num tempo exponencialmente maior do que a máquina não-determística.
- não se sabe se existe uma simulação polinomial de MTND em MTD..

Exercício 2.27 *Formaliza a geração de sequências de dígitos entre* 1 *e* r*, usada na demonstração do teorema anterior. Mostra como é que esta sequência pode codificar os movimentos de* M_1*.* ◇

2.5 Autómatos determinísticos de k-pilhas

$$(S, \Sigma, \Gamma, \delta, Z, s_0, F)$$

$$\delta \in (S \times \Sigma \times \Gamma^k) \times (S, (\Gamma^\star)^k)$$

Dados

- um estado do controlo finito, s
- um símbolo lido ou, alternativamente, ϵ (é determinístico!)

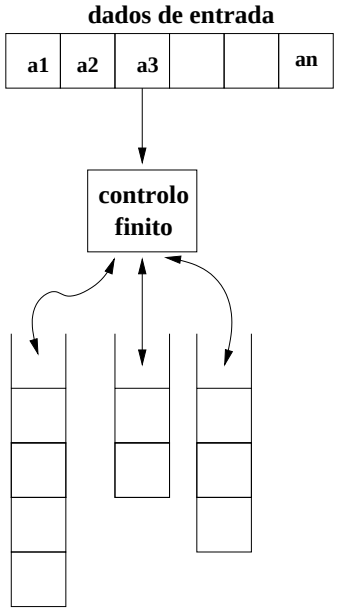


Figura 5: Autómato com 3 pilhas

- os símbolos que se encontram no topo de cada uma das k pilhas

num movimento

- Muda para um novo estado, s'
- substituí o símbolo de topo de cada pilha, por uma sequência de símbolos de pilha

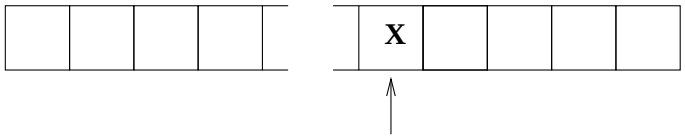
$$\delta(s, a, X_1, \ldots, X_k) = (s', \gamma_1, \ldots, \gamma_k)$$

O autómato de k-pilhas aceita se depois de consumir os dados entra num estado final.

2.5.1 Autómatos de 2-pilhas e MTs

Proposição 2.28 *Se uma linguagem L é aceite por uma máquina de Turing, então L é aceite por um autómato determinístico de 2-pilhas.*

Dem. As 2 pilhas vão simular cada um dos pedaços da fita da MT, já visitada, à esquerda e à direita do símbolo onde está a cabeça.



Para simplificar supomos que os dados são da forma $x\$$ onde $\$ \notin \Sigma$.

Seja M uma máquina de Turing tal que $L = L(M)$.

Construímos um 2-APD, P tal que

1. Cada pilha tem o símbolo inicial Z_0, que não aparece noutras posições
2. P copia os dados x para a primeira pilha. O marcador \$ permite indicar o fim de dados. As restantes transições são por ϵ.
3. P passa os dados para a segunda pilha (de modo a que o símbolo inicial dos dados fique no topo da pilha)
4. P passa para o estado inicial de M
5. P simula o funcionamento de M
   (a) dado o estado s de M, estando a cabeça em X (topo da 2a pilha), P muda para o estado s' de acordo com M
   (b) se M substituí X por Y e move para a direita: P coloca Y da 1a pilha e retira X
   (c) se M substituí X por Y e move para a esquerda: P retira o topo da 1.a pilha, Z, e substitui X por ZY na 2a pilha.

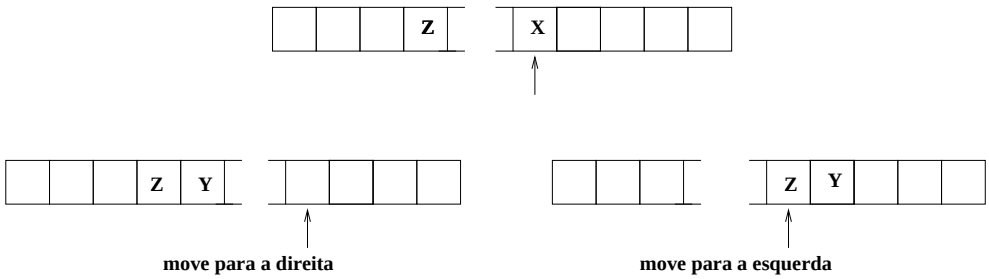


Figura 6: Máquina de Turing que simula um autómato de 2 pilhas

   Excepções apropriadas ocorrem quando X, Y e Z são brancos... quais?
6. P aceita se o novo estado de M for de aceitação. Caso contrário simula outro movimento

□

2.6 Máquinas de Turing e Computadores

Como já viram, um computador pode simular uma máquina de Turing.

Mas, uma máquina de Turing tem uma fita infinita... OK, podemos sempre comprar mais discos e usar o `swap` ...

Para simular um computador numa máquina de Turing (mesmo informalmente) consideramos um modelo abstracto de computador: **RAM** (*random access machine*):

- CPU: unidade lógica e aritmética, unidade lógica de controlo e registos
- Memória: um conjunto **infinito** de posições de memória com endereços 0, 1, 2,... cujo contéudo pode ser qualquer inteiro
- Linguagem máquina: conjunto de instruções
- O programa a executar é guardado em posições de memória

Operação	**Código**	**Operando**	Significado

`LDA`	`0001`	X	Carrega para o `AC` o conteúdo da posição de memória de endereço X
`LDI`	`0010`	X	Carrega para o `AC` o conteúdo da posição de memória cujo o endereço é o conteúdo da posição de memória de endereço X
`STA`	`0011`	X	Guarda o conteúdo do `AC` na posição de memória de endereço X
`STI`	`0100`	X	Guarda o conteúdo do `AC` na posição de memória cujo o endereço é o conteúdo da posição de memória de endereço X
`ADD`	`0101`	X	Adiciona o conteúdo da posição de memória de endereço X ao conteúdo do `AC`
`SUB`	`0110`	X	Subtrai o conteúdo da posição de memória de endereço X do conteúdo do `AC`
`JMP`	`0111`	X	Salta para a instrução cujo endereço é X
`JMZ`	`1000`	X	Salta para a instrução cujo endereço é X, se conteúdo de `AC` é `0`

Exemplo de instruções duma RAM

Máquina de Turing que simule uma RAM Vamos sugerir como construir uma MT com várias fitas, que representam:

1. A memória. Os símbolos `*` e `#` indicam para cada célula o fim do endereço e do contéudo. O símbolo `$` indica o início dos dados.
2. O contador de sequência de programa.
3. O endereço/conteúdo de posição memória da instrução a executar
4. Uma ou mais fitas de trabalho

Simulação do ciclo duma instrução:

1. **Procura** na fita 1 o endereço da instrução que está na 2 e obtém contéudo.
2. **Descodificar a instrução**: os primeiros bits são o código (`LDA`, `ADD`, ...) e os restantes o endereço do operando
3. se a instrução contém o endereço do operando, esse é copiado para a fita 3. O contéudo dessa posição é procurado na fita 1 e copiado para a fita 3.
4. **executa** a instrução consoante a função: copiar, adicionar, modificar o PC,...
5. Senão for `JMP` ou `JMZ`, incrementar 1 ao valor da fita 2 e recomeçar

2.7 Complexidade temporal duma Máquina de Turing

O **tempo de execução** duma máquina de Turing M com dados w é o número de passos efectuados por M até parar.

A **complexidade temporal** de M é a função

$$T(n) = \max \{p \mid p \text{ é o número de passos efectuados por } M \text{ até parar com dados } w \text{ e } |w| = n \}$$

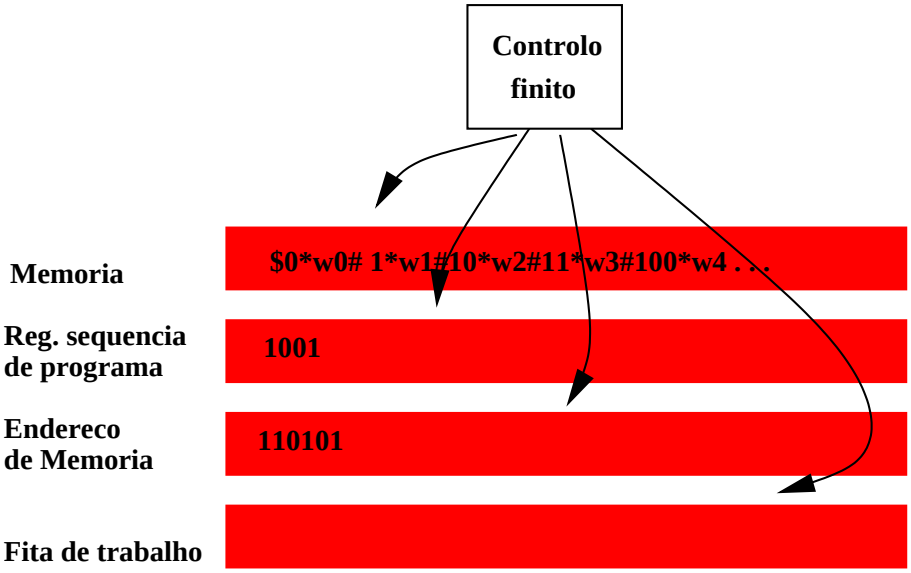


Figura 7: Máquina de Turing que simula um computador

Claro que poderá ser $T(n) = \infty$, para alguns valores de n!

Especialmente importantes são:

- a classe de máquinas de Turing (determinísticas) para as quais $T(n)$ é **polinomial**: os problemas correspondentes podem ser eficientemente resolvidos por computador (*classe* **P**)
- classe de máquinas de Turing não-determinísticas para as quais $T(n)$ é **polinomial**: os problemas correspondentes podem pelo menos ser resolvidos em tempo exponencial por computador (*classe* **NP**)

Modelo de computação	Simulação de n passos numa MTD 1 fita
MTD k-fitas	$O(n^2)$
MTND	$O(c^n)$, $c > 1$
RAM	$O(n^3)$

Não se sabe se as MTNDs têm de ser exponencialmente mais lentas que as MTD, em particular se $\mathbf{P} \neq \mathbf{NP}$! Mais pormenores na disciplina de Complexidade...

3 Indecidibilidade

3.1 Linguagens Indecidíveis (não recursivas)

Linguagens para as quais não existe nenhuma máquina de Turing que as reconheça.

Podem ser:

- recursivamente enumeráveis (r.e): existe uma MT que pára se os dados pertencerem à linguagem, mas pode não parar caso contrário. Equivalentemente, existe uma máquina (de enumeração) que enumera todos os seus elementos.
- não serem r.e

3.1.1 Máquinas Universais de Turing

O poder computacional das máquinas de Turing permite que existam MTs que simulem outras MTs cuja descrição faz parte dos dados.

Considere-se a linguagem constituída pelos pares (M, x) tal que:

1. M é (a codificação em binário de) uma máquina de Turing cujo alfabeto de entrada é $\{0, 1\}$
2. $x \in \{0, 1\}^\star$
3. M aceita x

As máquinas U que aceitam esta linguagem denominam-se Máquinas de Universais de Turing, i.e,

$$L_u = L(U) = \{(M, x) \mid x \in L(M)\}$$

Iremos ver que $w \in L_u$? é um problema indecidível.

3.1.2 Enumeração das palavras de $\{0,1\}^\star$

Podemos considerar a seguinte bijecção f entre $\{0,1\}^\star$ e $\mathbb{N}$:

$$\begin{array}{rcl} \{0,1\}^\star & \xrightarrow{f} & \mathbb{N} \\ \epsilon & & 1 \\ 0 & & 2 \\ 1 & & 3 \\ 00 & & 4 \\ 01 & & 5 \\ 10 & & 6 \\ 11 & & 7 \\ 000 & & 8 \\ \vdots & & \vdots \end{array}$$

$1x$ é a representação binária de $f(x)$. A i-ésima palavra designa-se por x_i.

3.1.3 Codificações de MTs

Vamos codificar qualquer máquina de Turing de alfabeto $\{0,1\}$ numa palavra de $\{0,1\}^\star$. Como vimos que a cada palavra está associado um inteiro i, poderemos falar na i-ésima MT, M_i. Seja

$$M = (\{s_1, s_2, \ldots, s_k\}, \{0,1\}, \{X_1, X_2, \ldots, X_m\}, \delta, s_1, \bullet, \{s_2\})$$

Associamos

- aos estados os inteiros correspondentes (em unário)
- aos símbolos de fita, os inteiros correspondentes (em unário), supondo que $X_1 = 0$, $X_2 = 1$ e $X_3 = \bullet$
- A direcção $\leftarrow$ será referida por D_1 e a $\rightarrow$, por D_2.

Então, uma transição $\delta(s_i, X_j) = (s_k, X_l, D_m)$ será codificada pela palavra:

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

O código da MT M é composto (apenas) pelos códigos das transições C_i separados por um par de 1's:

$$C_1 11 C_2 11 \ldots C_{n-1} 11 C_n$$

Para codificar (M, x) podemos separar o código de M do de x por 111.

Exercício 3.1 *Codifica a máquina* $M = \{\{s_1, s_2, s_3\}, \{0,1\}, \{0,1,\bullet\}, \delta, s_1, \bullet, \{s_2\}\}$

$$\begin{array}{rcl} \delta(s_1, 1) & = & (s_3, 0, \rightarrow) \\ \delta(s_3, 0) & = & (s_1, 1, \rightarrow) \\ \delta(s_3, 1) & = & (s_2, 0, \rightarrow) \\ \delta(s_3, \bullet) & = & (s_3, 1, \leftarrow). \end{array}$$

$\diamond$

3.1.4 Linguagem de diagonalização

Com a codificação apresentada podemos dizer que a i-ésima máquina de Turing M_i é a máquina de Turing cuja codificação é x_i.

Claro que nem todos os i correspondem a codificações de MT's. Mas, podemos supor que esses i correspondem a máquinas de Turing M_i tal que $L(M_i) = \emptyset$.

Seja a linguagem L_d (linguagem de diagonalização)

$$L_d = \{x_i \in \{0,1\}^\star \mid x_i \notin L(M_i)\}$$

L_d é constituída pelas palavras x tal que a MT M cujo código é x não aceita x como dados.

3.1.5 Como se obtém L_d?

Construa-se a tabela em $\mathbb{N} \times \mathbb{N} \longrightarrow \{0,1\}$ tal que:

o par (i,j) tem o valor 1 se a MT M_i aceita x_j e 0, caso contrário.

Por exemplo, poderíamos ter:

$i\backslash j$	1	2	3	4	...
1	0	1	1	0	...
2	1	1	0	0	...
3	0	0	1	1	...
4	0	1	0	1	...
.	.	.	.	.	.
.	.	.	.	.	...

A linha i corresponde a $L(M_i)$: os 1's indicam quais as suas palavras.

Os elementos da diagonal indicam se M_i aceita x_i.

Para construir L_d complementamos a diagonal: 1, 0, 0, 0, ...

E L_d será formada pelos x_j cujo valor é 1.

Mas, então L_d não pode corresponder a nenhuma linha da tabela! Porque é diferente nalguma coluna de todas as linhas da tabela

3.1.6 L_d não é r.e.

Teorema 3.2 *Não existe nenhuma máquina de Turing que aceite L_d. Logo L_d não é recursivamente enumerável.*

Suponhamos que $L_d = L(M)$ para alguma MT M de alfabeto $\{0,1\}$. Então existe pelo menos um código i, tal que $M_i = M$.

Será que $x_i \in L_d$?

- Se $x_i \in L_d$ então M_i aceita x_i. Mas por definição de L_d, $x_i \notin L_d$. ABSURDO!
- Se $x_i \notin L_d$, então M_i não aceita x_i. Mas por definição de L_d, $x_i \in L_d$. ABSURDO!

A contradição resultou de supor que existia M tal que $L_d = L(M)$.

3.2 Complementação de linguagens recursivas e linguagens r.e

O estudo das linguagens complementares pode permitir distinguir se uma linguagem é r.e mas não recursiva!

Teorema 3.3 *Se L é recursiva, $\overline{L}$ é recursiva*

Dem. Suponhamos que $L = L(M)$ para uma MT M que pára sempre. Basta construir uma MT $\overline{M}$ igual a M excepto em que

- os estados finais de M passam a não o ser em $\overline{M}$
- tem um novo estado de final r, sem transições dele.
- para cada par (estado não final de M,símbolo da fita) sem transições em M, acrescenta-se uma transição para o estado r.

□

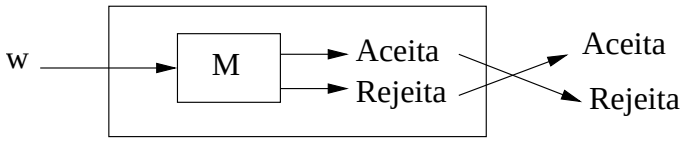


Figura 8: Construção de $\overline{M}$

Teorema 3.4 *Se L e $\overline{L}$ são r.e., então L é recursiva (e portanto, também o é $\overline{L}$)*

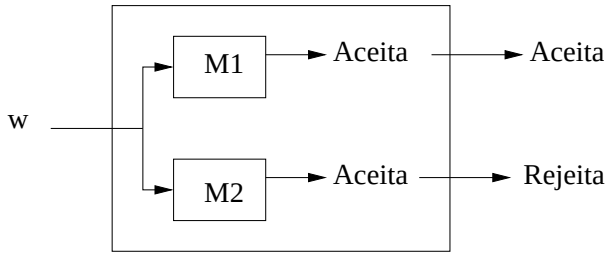


Figura 9: Construção de M a partir de M_1 e M_2

Dem. Seja $L = L(M_1)$ e $\overline{L} = L(M_2)$. Construímos uma MT M que simula em paralelo M_1 e M_2, usando duas fitas e estados cujas componentes são os de M_1 e M_2.

Se com dados w, M_1 aceitar, M aceita. Senão $w \notin L$, mas $w \in \overline{L}$. Então, M_2 tem de aceitar w. Nessa altura M pára e rejeita. Portanto, com todos os dados M pára e $L(M) = L$. Logo L é recursiva. □

3.2.1 Linguagens e Complementos

As linguagens podem dividir-se em:

D recursivas
SD r.e
não-SD não r.e

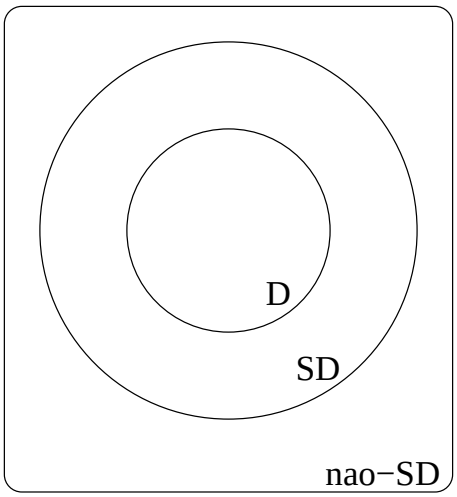


Figura 10: Relação entre linguagens **D**,**SD** e **não-SD**

Há apenas 4 maneiras de uma linguagem L e $\overline{L}$ se situarem no diagrama:

- Ambas L e $\overline{L}$ são recursivas:estão em **D**
- Nem L nem $\overline{L}$ são r.e: estão e, **não-SD**
- L é r.e mas não recursiva e $\overline{L}$ não r.e: L em **SD****D** e $\overline{L}$ em **não-SD**
- $\overline{L}$ é r.e mas não recursiva e L não r.e: $\overline{L}$ em **SD****D** e L em **não-SD**

Exemplo 3.5 *Como L_d é não r.e, podemos concluir que $\overline{L_d}$ não é recursiva, mas poderá ser r.e ou não r.e...*

Exercício 3.6 *Quais os casos que não são possíveis? Dá exemplos de linguagens em cada um dos restantes casos.* $\diamond$

3.2.2 Máquina de Turing Universal, U

Podemos agora descrever uma máquina universal U tal que

$$L_u = L(U) = \{(M, x) \mid x \in L(M)\}$$

Com dados (M, x), U simula M com dados x.

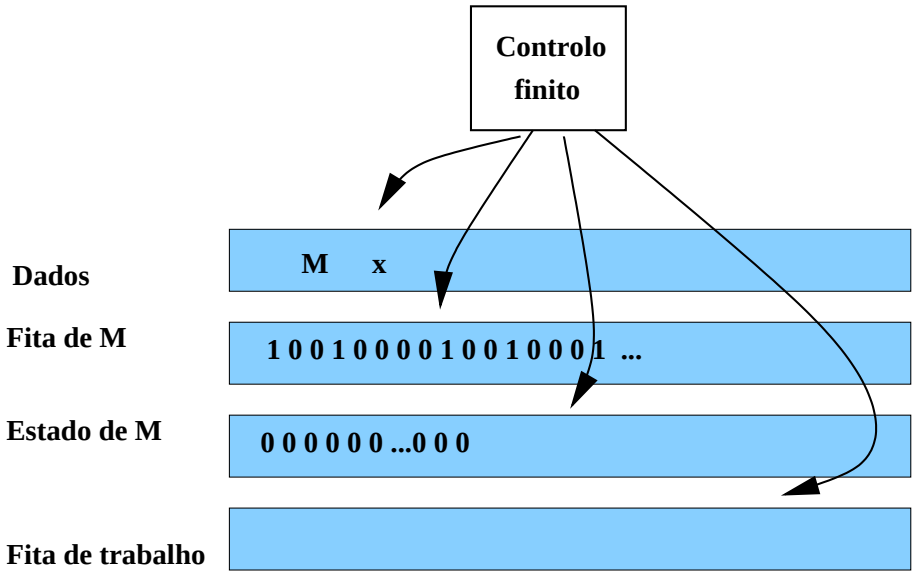


Figura 11: Máquina universal de Turing

1. examina os dados da 1ª fita e verifica se até 111 é um código legítimo para uma MT. Senão pára sem aceitar.
2. copia para a 2ª fita os dados x codificando para símbolos da fita: 10 se for 0 (em x) e 100 se for 1. Coloca a cabeça dessa fita no início dos dados.
3. colocar 0 na 3ª fita.
4. para simular um movimento de M, U procura uma transição $0^i10^j10^k10^l10^m$, tal que 0^i é o valor da 3ª fita, 0^j o símbolo de fita que começa na cabeça da 2ª fita de U. U muda o valor da 3ª fita para 0^k, substituí 0^j por 0^l na 2ª fita e move a cabeça da 2ª fita para o próximo 1 à esquerda ($m = 1$) ou à direita ($m = 2$).
5. Se não houver transições de um estado não final , M pára e U também.
6. Se M pára num estado final, U aceita o par (M, x). Se M não pára, U também não pára.

3.2.3 L_u é r.e mas não é recursiva

A descrição da máquina U garante que L_u3.2.2 é r.e.: U aceita (M, x) se $x \in L(M)$.

Suponhamos que L_u é recursiva. Então $\overline{L_u} = \{(M, x) \mid x \notin L(M)\}$ também o é.

Mas então também teríamos uma MT que aceitava L_d! O que seria absurdo!

Seja $L(M) = \overline{L_u}$. Podemos modificar M para uma MT M' que aceita $L_d = \{(x, x) \mid x \notin L(x)\}$

- M' modifica os dados para $x111x$.
- M' simula M nos novos dados.
- Se x é codificação duma M_i, M verifica se M_i aceita x.

M aceita se e só se M_i não aceita x_i, i.e, se $x_i \in L_d$. Isto é $L(M') = L_d$. **Absurdo!**

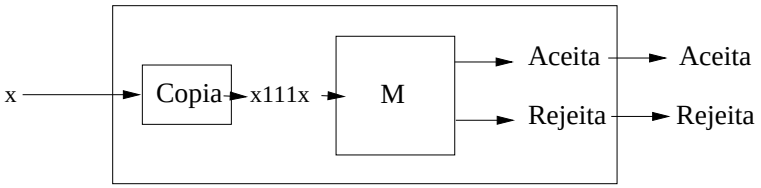


Figura 12: Construção de M' que aceita L_d

3.2.4 O problema da paragem é indecidível

Supondo que uma máquina de Turing aceita por paragem (e não por estado final) podemos ter uma MT M tal que

$$H(M) = \{x \mid M \text{ pára com dados } x\}$$
$$L_h = \{(M, x) \mid x \in H(M)\}$$

denomina-se o **problema da paragem** (*halting problem*) de máquinas de Turing.

Teorema 3.7 *L_h é r.e mas não recursiva.*

Dem. Dada uma MT M e dados x, podemos construir uma MT N exactamente igual a M, excepto que N não pára se M rejeita x. Isto é, N pára com dados x sse M aceita x.

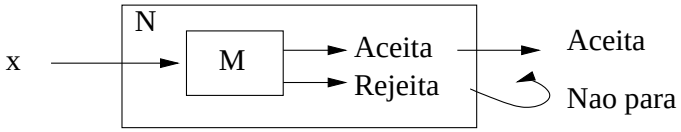


Figura 13: Construção da máquina N

Então, para todo o x, $x \in H(N)$ se e só se $x \in L(M)$. Como L_u é r.e, então L_h é r.e

Mas se L_h fosse recursiva L_u também o era. Suponhamos que existe uma MT K que reconhecia L_h, então vamos ver que existia uma MT K' que reconhecia L_u: dado (M, x), constrói N e determina, usando K, se $x \in H(N)$ (i.e $x \in L(M)$)

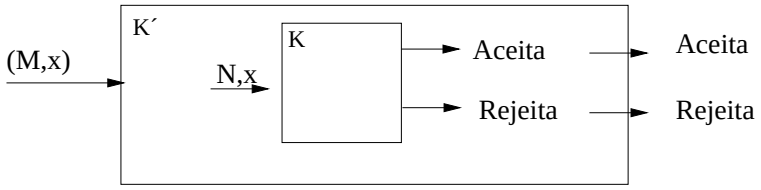


Figura 14: Máquina K' que reconhecia L_u

Mas $x \in L(M)$ se e só se $(M, x) \in L_u$. Então K' reconhecia L_u (seria recursiva). **Absurdo!** Logo, L_h r.e mas não é recursiva. □

3.3 Problemas sobre Máquinas de Turing

Dada uma máquina de Turing será decidível se

(a) tem pelo menos 45 estados?

(b) com a fita vazia executa mais de 45 passos?

(c) quaisquer que sejam os dados executa mais de 45 passos?

(d) aceita a palavra vazia ϵ?

(e) aceita a linguagem vazia?

(f) aceita uma linguagem finita?

(g) aceita uma linguagem regular ou independente de contexto ou recursiva?

Os problemas (a)-(c) são decidíveis e (d)-(g) indecidíveis. Os primeiros são sobre características das MTs e os segundos sobre propriedades das linguagens que aceitam!

(a) é decidível

Dem. Dada uma codificação da MT, contar o número de estados e verificar se é maior que 45. □

(b) é decidível

Dem. Simular a execução da MT a partir do estado inicial, contando os passos, e: ou atinge um estado final em menos de 45 passos; ou o algoritmo termina com a resposta *sim*. □

(c) é decidível

Dem. Em 45 passos a MT só visita um número finito de células à volta da posição inicial da cabeça e o número de palavras de comprimento ≤ 45 é finito. Basta simular os primeiros 45 passos da MT para cada um dos dados com no máximo 45 símbolos. Se para algum, a máquina atingir um estado final, a resposta é *não*. Caso contrário, é *sim*. □

3.3.1 Reduções

Para mostrar um problema P (p.e (d)-(g)) é indecidível, mostramos que se houvesse um algoritmo para decidir P, esse algoritmo podia ser usado para decidir um outro problema que já sabemos ser indecidível, p.e, o problema da paragem

$$L_h = \{(M, x) \mid M \text{ pára com dados } x\}.$$

Então, como L_h é indecidível, esse problema P também o tem de ser. Isto é, vamos reduzir L_h a P.

Uma redução de P_1 para P_2 é um algoritmo que converte instâncias dum problema de decisão P_1 em instâncias dum problema de decisão P_2 tal que ambas as respostas são *sim* ou ambas as respostas são *não*.

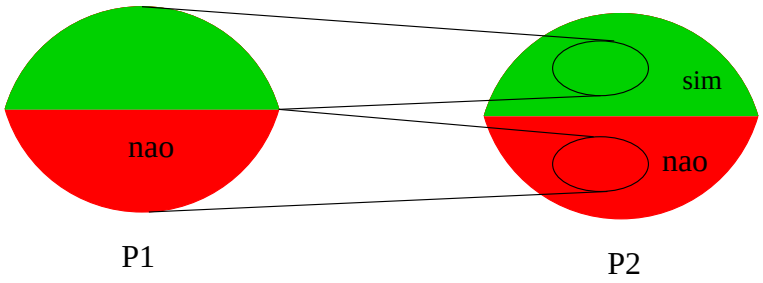


Figura 15: Redução entre problemas que transforma instâncias positivas em positivas e negativas em negativas

3.3.2 Determinar se uma MT aceita ϵ é indecidível

Seja $L_\epsilon = \{M \mid \epsilon \in L(M)\}$. Vamos reduzir L_h a L_ϵ, isto é, encontrar um algoritmo σ tal que $w \in L_h$ sse $\sigma(w) \in L_\epsilon$

Seja (M, x) e pretendemos saber se $(M, x) \in L_h$. Construímos uma nova MT M' (com x e M *built-in*) que com dados y:

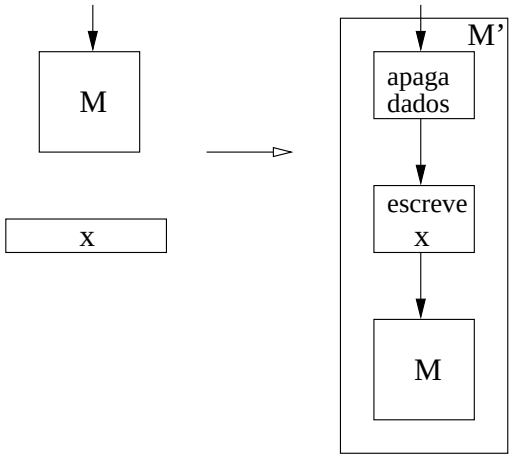


Figura 16: Redução de L_h a L_ϵ

1. apaga y
2. escreve x na fita
3. executa M com dados x
4. aceita se M parar

$$L(M') = \begin{cases} \Sigma^\star & \text{se } M \text{ pára com } x \\ \emptyset & \text{se } M \text{ não pára com } x \end{cases}$$

σ é a função computável $(M, x) \longrightarrow M'$

Então se L_ϵ fosse decidível tínhamos um algoritmo para decidir o problema da paragem:

Dado M e x construímos M' e M' aceita ϵ sse M pára com dados x. Absurdo !

Usando o mesmo processo mostra-se que (e)-(f) são indecidíveis.

3.3.3 Redução entre linguagens

Dadas $L_1 \subseteq \Sigma^\star$ e $L_2 \subseteq \Delta^\star$, uma **redução** de L_1 a L_2 é uma função total computável (algoritmo) $\sigma : \Sigma^\star \longrightarrow \Delta^\star$ tal que para todo $x \in \Sigma^\star$:

$$x \in L_1 \Leftrightarrow \sigma(x) \in L_2$$

Para σ existe uma máquina de Turing total (que pára sempre) e que com dados x pára com $\sigma(x)$ na fita.

Diz-se que L_1 é redutível a L_2,

$$L_1 \leq_m L_2$$

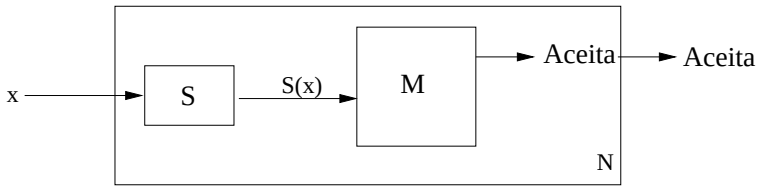
A relação $\leq_m$ é transitiva. Verifica!

Teorema 3.8

(a) Se $L_1 \leq_m L_2$ e L_2 é r.e então L_1 é r.e. Equivalentemente, se $L_1 \leq_m L_2$ e L_1 não é r.e então L_2 também não.

(b) Se $L_1 \leq_m L_2$ e L_2 é recursiva então L_1 é recursiva. Equivalentemente, se $L_1 \leq_m L_2$ e L_1 não é recursiva então L_2 também não.

Dem: (a) Se $L_1 \leq_m L_2$, seja σ a função computável tal que $x \in L_1 \Leftrightarrow \sigma(x) \in L_2$. Como L_2 é r.e, seja M uma MT tal que $L_2 = L(M)$. Construímos uma MT N que com dados x:



- calcula $\sigma(x)$
- executa M com dados $\sigma(x)$
- aceita, se M aceita

$$\begin{aligned} N \text{ aceita } x &\Leftrightarrow M \text{ aceita } \sigma(x) \\ &\Leftrightarrow \sigma(x) \in L_2 \\ &\Leftrightarrow x \in L_1 \end{aligned}$$

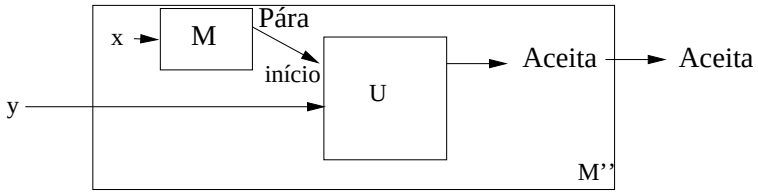
Logo, L_1 é r.e

(b) Sela $L_1 \leq_m L_2$ e L_2 recursiva. Então $\overline{L_2}$ também é recursiva e $\overline{L_1} \leq_m \overline{L_2}$ (podemos usar o mesmo σ). Como tanto L_2 como $\overline{L_2}$ são r.e, por (a), tanto L_1 e $\overline{L_1}$ são r.e. Logo L_1 é recursiva.

3.3.4 Determinar se uma MT aceita $L \in \mathcal{R} \cup \mathcal{I}_\mathcal{C} \cup \mathcal{D}$ é indecidível

Seja $L_{rid} = \{M \mid L(M) \in \mathcal{R} \cup \mathcal{I}_\mathcal{C} \cup \mathcal{D}\}$ e seja A uma linguagem r.e mas não recursiva (p.e L_h ou $\overline{L_u}$). Seja U uma máquina que aceita A. Vamos reduzir o complementar do problema da paragem $\overline{L_h}$ a L_{rid}.

Dado M e x construímos uma M'' de 2-fitas que com dados y:



$$L(M'') = \begin{cases} A & \text{se } M \text{ pára com } x \\ \emptyset & \text{se } M \text{ não pára com } x \end{cases}$$

1. Guarda y numa fita
2. Escreve x noutra fita (com as informações do seu controlo finito)
3. Executa M com dados x (com as informações do seu controlo finito)
4. Se M pára com dados x, então M'' executa uma MT U que aceita A com dados y
5. M" aceita, se U aceita.

Como $\emptyset \in \mathcal{R} \cup \mathcal{I}_\mathcal{C} \cup \mathcal{D}$ e $A \notin \mathcal{R} \cup \mathcal{I}_\mathcal{C} \cup \mathcal{D}$ então $\overline{L_h} \leq_m L_{rid}$

sendo σ a função computável $(M, x) \longrightarrow M''$

Logo, se L_{rid} fosse decidível L_h, também seria. Absurdo!.

3.4 Propriedades de $\mathcal{SD}$

Vamos ver que quase todas as propriedades das linguagens r.e são indecidíveis!

Uma **propriedade** P das linguagens r.e é um subconjunto de $\mathcal{SD}$, constituído pelas linguagens L que têm essa propriedade. Por exemplo:

ser vazio é a apenas o conjunto $\{\emptyset\}$

ser independentes de contexto é o conjunto $\mathcal{I}_\mathcal{C}$ das linguagens que são geradas por gramáticas independentes de contexto.

Podemos associar a P uma função característica $C_P : \mathcal{SD} \longrightarrow \{0, 1\}$

$$C_P(L) = \begin{cases} 0 & \text{se } L \notin P \\ 1 & \text{se } L \in P \end{cases}$$

Uma propriedade é **trivial** se é vazia ($\emptyset$) ou todas as linguagens r.e ($\mathcal{SD}$). Se uma propriedade P é **não trivial** existem L e L' distintos tal que $L \in P$ e $L' \notin P$.

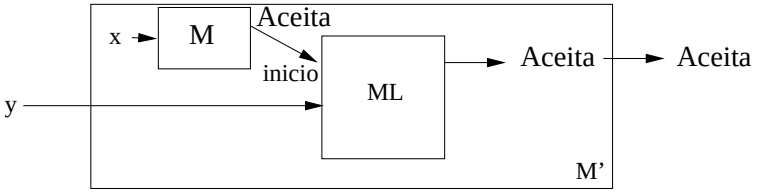
Para representar um conjunto de linguagens de modo finito consideramos as máquinas de Turing que as aceitam.

Se P é uma propriedade de $\mathcal{SD}$, L_P é conjunto de códigos das MT M_i tal que $L(M_i) \in P$. Dizer que uma propriedade P é decidível é dizer que L_P é decidível.

Teorema 3.9 (Teorema de Rice) *Todas as propriedades não triviais de linguagens de $\mathcal{SD}$ são indecidíveis.*

Dem:

Seja P uma tal propriedade. Suponhamos que $\emptyset \notin P$. Então existe $L \neq \emptyset$ tal que $L \in P$. Seja M_L uma MT que aceita L. Vamos ver que $L_u \leq_m L_P$. Dado (M, x) o algoritmo de redução constrói uma $M' = \sigma((M, x))$



$$L(M') = \begin{cases} \emptyset & \text{se } M \text{ não aceita } x \\ L & \text{se } M \text{ aceita } x \end{cases}$$

M' com dados y:

- Guarda y numa fita.
- Escreve M e x noutra fita e simula M com dados x. Nota que M e x estão *built-in* em M'.
- Se M não aceita x, M' não faz mais nada e $L(M') = \emptyset$. Então o código de M' não pertence a L_P.
- Se M aceita x, então M' simula M_L com os seus dados y, e $L(M') = L$. Como $L \in P$, o código de M' pertence a L_P.

Temos: $(M, x) \in L_u$ sse $M' = \sigma((M, x)) \in L_P$

Como L_u não é recursivo, L_P também não o é. Logo P é indecidível.

Se $\emptyset \in P$ então considerámos $\overline{P}$ e pelo método anterior concluímos que $\overline{P}$ é indecidível.

Mas $\overline{L_P} = L_{\overline{P}}$ (porque todas MT aceitam uma linguagem r.e.) e então L_P também não é recursivo (Porquê?). Logo P é indecidível.

Exercício 3.10 *Mostra que o conjunto dos códigos das MT que aceitam todos os dados que são palindromos (e eventualmente outros dados) é indecidível.* $\diamond$

3.5 Alguns problemas sobre L.I.C.

Problemas decidíveis para LICs:

- Dada uma gramática independente de contexto G, $L(G) = \emptyset$?

- Dada uma gramática independente de contexto G, $L(G)$ é finita?
- Dada uma gramática independente de contexto G e $x \in \Sigma^\star$, $x \in L(G)$?

Problemas indecidíveis para LICs

- Dada uma linguagem independente de contexto L, $\overline{L}$ é LIC?
- Dada uma gramática independente de contexto G, G é ambígua?
- Dada uma linguagem independente de contexto L, L é determinística?
- Dada uma gramática independente de contexto G, $L(G) = \Sigma^\star$?
- Dadas duas l.i.c L_1 e L_2, $L_1 \cap L_2 = \emptyset$? Ou $L_1 = L_2$?

3.5.1 Determinar se uma G.I.C gera $\Sigma^\star$ é indecidível

Seja $L_\Sigma = \{G \mid G \text{ é i.c. e } L(G) = \Sigma^\star\}$

Por redução ao problema complementar da paragem: $\overline{L_h} \leq_m L_\Sigma$

Histórias de computação válidas Seja $M = (\delta, S, \Sigma, \Gamma, \delta, s_0, F)$ uma MT (com fita semi-infinita) e os dados x. Uma configuração de M é uma palavra $X_1 \cdots X_{i-1} s X_i \cdots X_n$ onde s é o estado e X_i o símbolo que está a ser lido.

Uma história de computação válida de M com x é uma sequência de configurações α_i separadas por um caracter especial $\# \notin \Gamma \cup S$ tal que:

$$\#\alpha_0\#\alpha_1\#\ldots\#\alpha_N\#$$

- α_0 é configuração inicial de M com dados x: $s_0 x$
- α_N é uma configuração de paragem (de aceitação ou não)
- α_{i+1} segue num passo de α_i, i.e., $\alpha_i \overset{M}{\vdash} \alpha_{i+1}$, para $0 \leq 1 \leq N-1$

Se M não pára em x não existe história de computação válida. Define-se

$$VALCOMPS(M,x) = \{\text{histórias de computação válidas de } M \text{ em } x\}$$

Podemos codificar as configurações como palavras de $\Delta = \{\#\} \cup \Gamma \cup S$, e então $VALCOMPS(M,x) \subseteq \Delta^\star$ e

$$VALCOMPS(M,x) = \emptyset \Leftrightarrow M \text{ não pára com dados } x$$

e sendo

$$\overline{VALCOMPS(M,x)} = \Delta^\star \setminus VALCOMPS(M,x)$$

$$\overline{VALCOMPS(M,x)} = \Delta^\star \Leftrightarrow M \text{ não pára com dados } x$$

Facto: $\overline{VALCOMPS(M,x)}$ é L.I.C. ! E temos um algoritmo que constrói uma gramática G, a partir de M e x, tal que:

$$M \text{ não pára com dados } x \Leftrightarrow L(G) = \Delta^\star$$

Temos então:

$$\overline{L_h} \leq_m L_\Sigma$$

Como $\overline{L_h}$ não é r.e, concluímos que L_Σ não é r.e.

$\overline{VALCOMPS(M,x)}$ é L.I.C. Se $z \in VALCOMPS(M,x)$ então

1. z começa e acaba por # e é da forma $\#\alpha_0\#\alpha_1\#\ldots\#\alpha_N\#$, com $\alpha_i \in (\Gamma \cup S)^\star$
2. cada α_i é uma palavra em que exactamente um símbolo é de S
3. α_0 é a configuração inicial s_0x
4. O estado de α_N é um estado final ou um do qual não há transições (podemos sempre supor que só há um desses estados de paragem)
5. $\alpha_i \overset{M}{\vdash} \alpha_{i+1}$, para $0 \leq 1 \leq N-1$

Seja, para $0 \leq i \leq 5$,

$$A_i = \{x \in \Delta^\star \mid x \text{ satisfaz } i\}$$

Então, $VALCOMPS(M,x) = \cap_{0\leq i\leq 5} A_i$ e $\overline{VALCOMPS(M,x)} = \cup_{0\leq i\leq 5}\overline{A_i}$

Vamos ver que $\overline{A_i}$, $0 \leq i \leq 5$ são L.I.C

As linguagens A_1, A_2, A_3 e A_4 são regulares. Logo os seus complementares também.

A_1: é a linguagem $\{\#\}\Delta^\star\{\#\}$

A_2: é apenas necessário verificar que entre dois #'s existe apenas um símbolo de S. Pode-se construir um autómato finito...

A_3: é a linguagem regular $\{\#s_0x\#\}\Delta^\star$

A_4: basta testar se em α_N está um estado de paragem (final ou não). Podemos construir um autómato finito não determinístico...

$\overline{A_5}$ é L.I.C. Suponhamos um $z \in \Delta^\star$ que satisfaz 1 a 4, e considere-se uma subpalavra sua da forma $\#\alpha\#\beta\#$. Se $\alpha \overset{M}{\vdash} \beta$, as duas configurações tem de concordar excepto para alguns símbolos perto da posição da cabeça e as diferenças têm de ser consistentes em relação a δ, p.e.:

$$\#\ a\ b\ a\ s\ a\ b\ a\ b\ b\ \#\ a\ b\ s'\ a\ b\ b\ a\ b\ b\ \#$$

se $\delta(s,a) = (s', b, \leftarrow)$ em M

Para verificar a consistência de α e β em relação a δ, basta verificar a consistência das subpalavras comprimento 4 de α e de β, correspondentes, i.e. que distem o mesmo número de símbolos, do símbolo # esquerdo mais próximo (supondo a fita semi-infinita). Por exemplo:

$$a\ s\ a\ b \qquad s'\ a\ b\ b$$

distam 3 símbolos do símbolo # esquerdo mais próximo e são consistentes. Se as sequências à mesma distância forem iguais são sempre consistentes. P.e. $d = 7$

$$b\ a\ b\ b \qquad b\ a\ b\ b$$

O conjunto dos pares (z_1, z_2), com $z_i \in \Delta^\star$, $|z_i| = 4$, $i = 1, 2$ e z_1 consistente com z_2 é finito e pode ser listado.

E, para quaisquer configurações α e β, $\alpha \overset{M}{\vdash} \beta$ sse todas as subpalavras correspondentes de α e β, de comprimento 4, forem consistentes.

Assim, para verificar que $\alpha \overset{M}{\vdash} \beta$ **é falso**, basta encontrar uma subpalavra de α que não seja consistente (em relação a δ) como a correspondente de β.

Podemos então descrever um autómato de pilha não determinístico que aceita $\overline{A_5}$, verificando se existe i tal que α_{i+1} não segue num passo de α_i, segundo δ. O autómato lê z e

- escolhe um α_i
- escolhe uma subpalavra $u \in \alpha_i$ com $|u| = 4$
  - ao escolher, guarda na pilha os símbolos de α_i até u
  - memoriza u no controlo finito
  - lê z até ao primeiro $\#$
  - percorre α_{i+1} tirando elementos que introduziu na pilha, para determinar uma subpalavra $v \in \alpha_{i+1}$ correspondente a u
  - verifica a consistência de u e v, usando os símbolos guardados no controlo finito. Se u e v não forem consistentes aceita z

Por exemplo, se $\delta(s, a) = (s', b, \rightarrow)$ e z contiver a subpalavra:

$$\#\ a\ b\ a\ a\ b\ s\ a\ b\ b\ \#\ a\ b\ a\ a\ \ s'\ b\ b\ b\ \#$$

e $u = b\ s\ a\ b$ o autómato descrito descobre o erro. Verifica!

Temos então demonstrado:

Teorema 3.11 *Dada uma g.i.c G é indecidível se $L(G) = \Sigma^\star$.*

Usando o facto de $\overline{VALCOMPS(M, x)}$ ser L.I.C. também se demonstra:

Teorema 3.12 *Sendo G_1 e G_2 duas gramáticas i.c. e R uma linguagem regular, os seguintes problemas são indecidíveis:*

(a) $L(G_1) = L(G_2)$
(b) $L(G_1) \subseteq L(G_2)$
(c) $L(G_1) = R$
(d) $R \subseteq L(G_1)$

4 Bibiografia recomendada

[HMU00, Cap. 8-9], [Pap94, Cap. 2],[Dew93, Cap. 28] [Koz97]

Referências

[Dew93] A. K. Dewdney. *The (New) Turing Omnibus – 66 Excursions in Computer Science*. W. H. Freeman and Company, 1993.

[HMU00] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 2nd edition, 2000.

[Koz97] Dexter C. Kozen. *Automata and Computability*. Undergraduate texts in computer science. Springer, 1997.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[Tui36] Alan Tuing. On computable numbers, with an application to the entscgeidungsproblem. In *The Undecidable*. Raven Press, 1936.