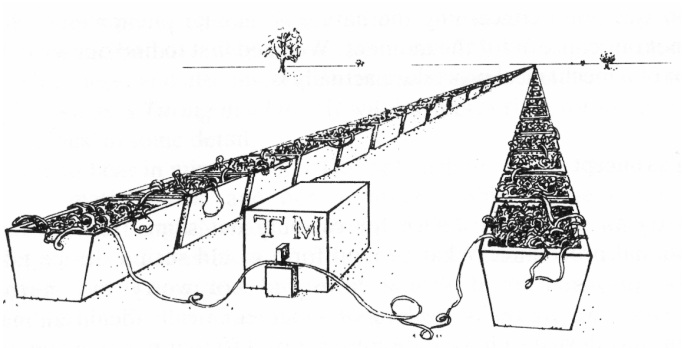
Capítulo 7

Máquinas de Turing



(Penrose, The Emperor's New Mind, Oxford Univ. Press, 1989)

A especificação formal de um problema envolve geralmente um ou mais parâmetros cujas concretizações correspondem aos possíveis casos particulares, a que chamamos *instâncias* do problema. Vejamos alguns exemplos.

1. Podemos associar o parâmetro n, de domínio $\mathbb{N}$, ao problema de decidir se um número natural é ou não perfeito, sendo cada instância deste problema um número natural.
2. As instâncias do problema de determinar os zeros de uma função quadrática consistem nas possíveis concretizações dos três parâmetros reais a, b e c que especificam cada função definida por $ax^2 + bx + c$.
3. Uma instância do problema de determinar se dois quaisquer grafos conexos são ou não isomorfos é um par (G_1, G_2), constituído pela concretização dos dois parâmetros do problema, G_1 e G_2, cada um deles pertencente ao conjunto dos grafos conexos.

Uma *solução* de um problema por intermédio de um "algoritmo" consiste em especificar um plano detalhado e exequível, que seja aplicável a qualquer instância

desse problema e que, ao ser aplicado a uma qualquer instância, conduza à solução particular correspondente.

É claro que um problema pode ter várias soluções algorítmicas, enquanto cada algoritmo é geralmente "desenhado" para resolver um só problema.

À especificação das operações elementares que os algoritmos podem utilizar, bem como dos recursos que estes podem aceder (incluindo as formas de os aceder e manipular) chamamos um *modelo de computação*.

Por exemplo, podemos olhar as classes dos autómatos finitos e dos autómatos de pilha como dois modelos distintos de computação. Em cada um destes modelos, um autómato pode ser visto como uma especificação de um "algoritmo" que permite resolver o problema de decidir se uma dada palavra pertence ou não a uma linguagem.

Nestes dois modelos, os "tipos de algoritmos" que se podem especificar são distintos, o que se traduz em diferentes classes de problemas (linguagens) que os modelos conseguem resolver (decidir).

Os autómatos finitos permitem resolver aqueles problemas de decisão em que a informação necessária para o resolver é finita, independente das instâncias e fixa à priori.

Já no modelo dos autómatos de pilha, a informação necessária para resolver um determinado problema pode ser gerada durante a "fase de execução do algoritmo", embora o acesso à informação seja condicionado pelas regras de manipulação de uma pilha.

Por exemplo, sabemos que o problema de decidir se uma dada palavra é ou não um palíndromo não é resolúvel por autómatos finitos mas já o é por autómatos de pilha. Por outro lado, o problema de decidir se um palavra é da forma ww não é resolúvel no modelo dos autómatos de pilha mas tem uma solução simples num modelo de autómatos que em vez de uma pilha tenham uma fila. De igual modo, o problema de decidir se uma palavra é da forma $a^n b^n c^n$, para algum número natural n, apresenta uma solução simples num modelo de autómatos que tenham duas pilhas.

Dizemos que o modelo dos autómatos finitos é mais fraco do que o modelo dos autómatos de pilha ou que o modelo dos autómatos de pilha é mais forte do que o modelo dos autómatos finitos.

Uma vez que as classes dos problemas resolúveis pelo modelo de autómatos com uma fila e pelo modelo dos autómatos com duas pilhas são idênticas, dizemos que estes dois modelos têm o mesmo poder ou que são equivalentes. Embora aparentemente bastante distintos, é um facto que os modelos dos autómatos com duas pilhas, dos autómatos com uma fila e das máquinas de Turing, ao qual se dedica este capítulo, são todos equivalentes.

Na verdade, qualquer modelo de computação que seja "razoável" e "suficientemente geral" será equivalente ao modelo das máquinas de Turing, no sentido em que todos esses modelos têm capacidade para resolver os mesmos problemas (decidem as mesmas linguagens).

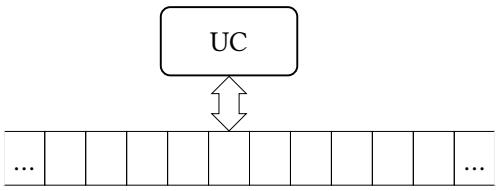
Hoje em dia, é uma ideia comum e aceite que os computadores não conseguem resolver todos os problemas matemáticos, por mais bem especificados que sejam, mesmo se munidos de recursos de memória ilimitados e/ou tempo arbitrariamente grande. Entre muitos outros, vamos destacar três famosos problemas que são algoritmicamente insolúveis: o "entscheidungsproblem", o problema da paragem e o décimo problema de Hilbert.

Para chegar a esta conclusão, foram fundamentais os trabalhos pioneiros de vários matemáticos, produzidos na década de 30 do século XX, dedicados a clarificar e formalizar a noção, até então intuitiva, do que é um algoritmo.

Motivados pelo ambicioso programa para a axiomatização e total formalização do conhecimento matemático, lançado por Hilbert no início do século XX, estes cientistas dedicaram uma parte significativa das suas vidas ao estudo da computabilidade. Entre outros, salientamos os importantes contributos prestados por Alan Mathison Turing (Reino Unido, 1912 - 1854) — com as máquinas de Turing, Stephen Cole Kleene (Estados Unidos da América, 1909 - 1994) — com as funções recorrentes, Kurt Gödel (República Checa, 1906 - 1978) — com as funções μ-recorrentes, Alonzo Church (Estados Unidos da América, 1903 - 1995) — com o cálculo-λ, Emil Leon Post (Polónia , 1897 - 1954) — com os sistemas de Post e Moses Schönfinkel (Ucrânia, 1888 - 1942) — com a lógica combinatória.

7.1 Máquinas de Turing deterministas

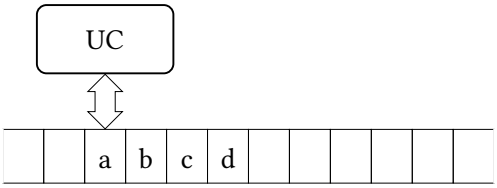
Existem muitos modelos de máquinas de Turing, quase todos equivalentes. Como modelo básico vamos adoptar o modelo com uma única fita ilimitada nos dois sentidos.



Uma máquina de Turing é constituída por três componentes principais: uma fita, uma "cabeça" de leitura/escrita e uma Unidade de Controlo de estados (UC). A fita da máquina, dividida num número infinito de células, serve como espaço ilimitado de consulta/armazenamento de informação. A "cabeça" de leitura/escrita move-se ao longo da fita, célula a célula, tanto para a esquerda como para a direita. Sob o comando da Unidade de Controlo, a "cabeça" de leitura/escrita começa por ler o símbolo presente na célula actual (aquela que está sob a "cabeça"). Em seguida,

substitui-o por um novo símbolo ou simplesmente apaga-o e, finalmente, move-se para uma célula adjacente.

A máquina é inicialmente preparada, inscrevendo na fita uma palavra, um símbolo por célula, ficando as restantes células vazias (diremos que cada uma destas células não preenchidas contém um *símbolo branco*). A "cabeça" de leitura/escrita é então posicionada sobre a célula que contém primeiro símbolo da palavra e a unidade de controlo assume o *estado inicial*.



Com base no estado actual da unidade de controlo e no símbolo actual (sob a "cabeça" de leitura/escrita), uma máquina de Turing muda de uma *configuração* para a seguinte quando executa as três acções seguintes:

i. transita para outro estado (eventualmente, para o mesmo);

ii. escreve um símbolo na fita;

iii. move a "cabeça" de leitura/escrita uma posição para a esquerda ($\triangleleft$) ou uma posição para a direita ($\triangleright$).

Definição 7.1.1 Máquina de Turing determinista.

Uma **máquina de Turing determinista** (MT) é um sêxtuplo ordenado

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

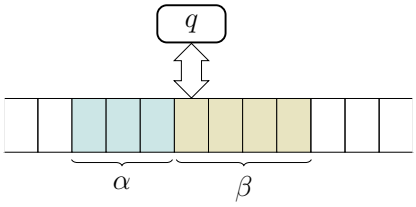
onde:

- Q é um conjunto finito de estados;
- Σ é um conjunto finito de símbolos de entrada;
- Γ é um conjunto finito de símbolos de fita, com $\Sigma \subset \Gamma$;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\triangleleft, \triangleright\}$ é a função (parcial) de transição de estado;
- $s \in Q$ é o estado inicial;
- $\square \in \Gamma \setminus \Sigma$ é o símbolo branco;
- $F \subseteq Q$ é o conjunto de estados de aceitação.

7.1.1 Transições e configurações

Uma MT evolui ao longo do tempo, numa sequência de passos discretos, mudando de configuração em configuração até que, eventualmente, pára.

Cada momento no funcionamento de uma máquina de Turing é completamente caracterizado pelo estado actual da máquina, pela posição da "cabeça" de leitura/escrita e também pelo conteúdo da fita, conforme ilustrado na figura seguinte.



Definição 7.1.2 Configuração de uma MT.

A **descrição instantânea** ou **configuração** de uma máquina de Turing

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

é uma expressão da forma $\alpha q \beta$, com $q \in Q$ e $\alpha, \beta \in \Gamma^\star$, onde

- q é o estado actual da máquina;
- α é a sequência de símbolos à esquerda da "cabeça" de leitura/escrita — as células *à esquerda* do primeiro símbolo de α *apenas contêm símbolos brancos* e o primeiro símbolo de α não é o símbolo branco;
- β é a sequência de símbolos à direita da "cabeça" de leitura/escrita — o primeiro símbolo de β é o símbolo sob a "cabeça" de leitura/escrita; nas células *à direita* do último símbolo de β *apenas existem símbolos brancos* e o último símbolo de β não é o símbolo branco.

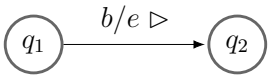
Um passo na evolução de uma MT é denotada por $\alpha q \beta \vdash \alpha' q' \beta'$ e corresponde à passagem da configuração $\alpha q \beta$, num dado instante, para a configuração $\alpha' q' \beta'$, no instante seguinte.

Denotamos ainda por $\alpha q \beta \vdash^\star \alpha' q' \beta'$ uma sequência de zero ou mais passos, partindo de uma configuração $\alpha q \beta$, num dado instante, para a configuração $\alpha' q' \beta'$, num instante posterior.

Definição 7.1.3 Passo à direita.

Um **passo à direita** numa MT consiste em

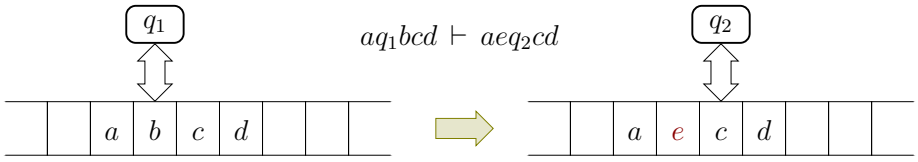
1. transitar para outro estado;
2. escrever um símbolo na fita;
3. mover a "cabeça" de leitura/escrita uma posição para a direita.



Se o estado actual da MT for q_1, o símbolo actual for b e se estiver definida a transição

$$\delta(q_1, b) = (q_2, e, \triangleright)$$

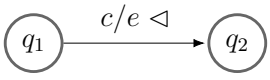
então o estado da MT passa a ser q_2, o símbolo b é substituído pelo símbolo e e a "cabeça" de leitura/escrita move-se para a célula imediatamente à direita.



Definição 7.1.4 Passo para a esquerda.

Um **passo à esquerda** numa MT consiste em

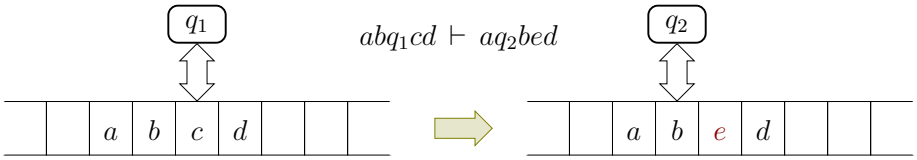
1. transitar para outro estado;
2. escrever um símbolo na fita;
3. mover a "cabeça" de leitura/escrita uma posição para a esquerda.



Se o estado estado actual da MT for q_1, o símbolo actual for c e se estiver definida a transição

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

então a MT passa ao estado q_2, o símbolo c é substituído pelo símbolo e e a "cabeça" de leitura/escrita move-se para a célula imediatamente à esquerda.



7.1.2 Configurações de paragem e reconhecimento de palavras

Partindo da configuração inicial, sw, para alguma palavra $w \in \Sigma^\star$, uma MT atinge eventualmente uma configuração em que a função de transição não está definida para o estado e símbolo de fita. Nesse momento, a *máquina pára* e se o estado for de aceitação então a palavra w é aceite, caso contrário é rejeitada.

Definição 7.1.5 Configuração de paragem.

Uma máquina de Turing M está numa **configuração de paragem** se a descrição instantânea é da forma $\alpha q a \beta$ e $\delta(q, a)$ não está definida.

Se, para além disso,

- $q \in F$ então a máquina está numa **configuração de aceitação**;
- $q \notin F$ então a máquina está numa **configuração de rejeição**.

Partindo da configuração inicial, uma máquina de Turing pode não chegar jamais a atingir uma configuração de paragem. Neste caso a máquina *não pára* para a palavra w inicialmente escrita na fita, situação que é denotada por $sw \vdash^\star \infty$.

Definição 7.1.6 Palavra Reconhecida por uma MT.

Uma palavra $w \in \Sigma^\star$ é reconhecida (ou aceite) por uma máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, s, F)$ se existem $f \in F$ e $\alpha, \beta \in \Gamma^\star$ tais que $sw \vdash^\star \alpha f \beta$ e além disso $\alpha f \beta$ é uma configuração de paragem.

Definição 7.1.7 Linguagem reconhecida por uma MT.

A **linguagem reconhecida** por uma máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, s, F)$ é

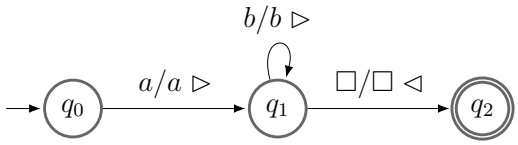
$$\mathcal{L}(M) = \{w \in \Sigma^\star : w \text{ é aceite por } M\}.$$

Se L é a linguagem reconhecida por uma máquina de Turing M então:

- para $w \in L$, a máquina M pára num estado de aceitação;
- para $w \notin L$, a máquina M pára num estado de rejeição ou não pára.

Exemplo 7.1.8

A figura seguinte representa uma máquina de Turing para reconhecer a linguagem associada à expressão regular ab^*.



O conjunto de estados da MT é $Q = \{q_0, q_1, q_2\}$, o estado inicial é q_0 e o único estado de aceitação é q_2. O alfabeto da linguagem é $\Sigma = \{a, b\}$ e o alfabeto da fita é $\Gamma = \{a, b, \square\}$.

A sequência de configurações que conduzem à aceitação da palavra abb é

$$q_0abb \vdash aq_1bb \vdash abq_1b \vdash abbq_1 \vdash abbq_2$$

e como esta última configuração é de paragem e o estado q_2 é de aceitação, a palavra é aceite.

A sequência de configurações que conduzem à rejeição da palavra aba é

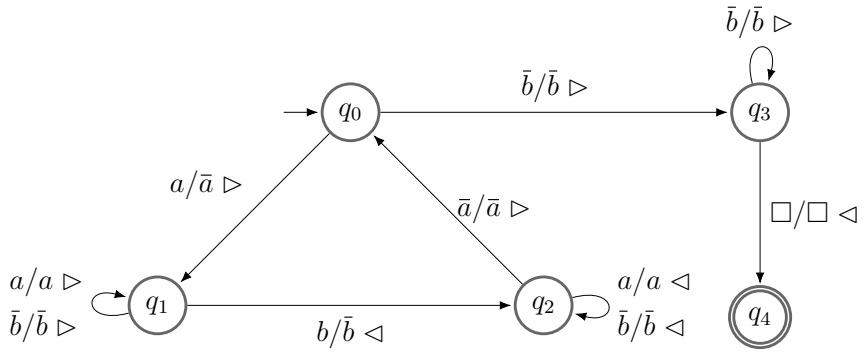
$$q_0aba \vdash aq_1ba \vdash abq_1a$$

e como esta última configuração é de paragem, mas o estado q_1 não é de aceitação, a palavra é rejeitada.

Exemplo 7.1.9

Na figura seguinte ilustramos uma máquina de Turing para reconhecer a linguagem independente do contexto $L = \{a^n b^n : n > 0\}$.

O método utilizado para reconhecer as palavras desta linguagem recorre à técnica de marcação dos símbolos da fita, o qual consiste em marcar certos símbolos da fita de modo a assinalar que já foram lidos. Formalmente, cada símbolo marcado é um novo símbolo do alfabeto da fita (que não faz parte do alfabeto da linguagem).



No início do reconhecimento, no estado inicial q_0, o símbolo actual é um dos seguintes:

- $\square$, pelo que a palavra a reconhecer é ε, e a máquina pára e rejeita;
- b, pelo que a palavra não pertence a L, e a máquina pára e rejeita;
- a, pelo que a palavra pode pertencer a L, e a máquina marca o símbolo a, avança para a direita e muda para o estado q_1.

No estado q_1, a máquina avança sucessivamente para a direita, saltando sobre todos os símbolos a, bem como sobre todos os b que já estejam marcados, até encontrar o primeiro b não marcado, na tentativa de o emparelhar com o símbolo a que tinha sido marcado. Se não encontra um b, é porque a palavra não pertence à linguagem e a máquina pára no estado q_1 rejeitando-a. Se encontra um b então marca-o, move-se para a esquerda e transita para o estado q_2.

No estado q_2, a máquina recua na fita até encontrar o símbolo a que foi marcado anteriormente. Em seguida, avança uma posição para a direita e transita para o estado inicial, q_0.

Ao transitar para o estado inicial q_0, o símbolo actual é um dos seguintes:

- a, e a máquina volta a tentar emparelhá-lo com um b não marcado, conforme descrito anteriormente;
- b, pelo que a palavra não pertence a L e a máquina pára e rejeita;
- $\bar{b}$, pelo que não existem mais símbolos a para emparelhar; resta confirmar que todos os outros símbolos da palavra original já foram marcados, pelo que a máquina transita para o estado q_3.

No estado q_3, a máquina avança sucessivamente para a direita, saltando sobre todos os símbolos b marcados. Se encontra algum símbolo não marcado então pára e rejeita a palavra. Se encontra o símbolo branco, então passa ao estado de aceitação q_4, parando e aceitando a palavra inicialmente colocada na fita.

Como exemplo, a sequência de configurações que conduzem à aceitação da palavra $aabb$ é

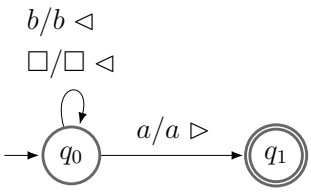
$$\begin{aligned} q_0aabb &\vdash \bar{a}q_1abb \vdash \bar{a}aq_1bb \vdash \bar{a}q_2a\bar{b}b \vdash q_2\bar{a}a\bar{b}b \vdash \bar{a}q_0a\bar{b}b \\ &\vdash \bar{a}\bar{a}q_1\bar{b}b \vdash \bar{a}\bar{a}\bar{b}q_1b \vdash \bar{a}\bar{a}q_2\bar{b}\bar{b} \vdash \bar{a}q_2\bar{a}\bar{b}\bar{b} \vdash \bar{a}\bar{a}q_0\bar{b}\bar{b} \vdash \bar{a}\bar{a}\bar{b}q_3\bar{b} \\ &\vdash \bar{a}\bar{a}\bar{b}\bar{b}q_3 \vdash \bar{a}\bar{a}\bar{b}q_4\bar{b} \end{aligned}$$

Finalmente, a sequência de configurações que conduzem à rejeição da palavra aba é

$$q_0aba \vdash \bar{a}q_1ba \vdash q_2\bar{a}\bar{b}a \vdash \bar{a}q_0\bar{b}a \vdash \bar{a}\bar{b}q_3a$$

Exemplo 7.1.10

Na figura seguinte ilustramos uma máquina de Turing para reconhecer a linguagem associada à expressão regular $a(a+b)^*$:



Esta máquina apresenta duas características peculiares patentes nas máquinas de Turing:

- A primeira é que, ao verificar que o primeiro símbolo da palavra a reconhecer é a, a máquina muda para o estado q_1 e pára. Como pára num estado de aceitação, mesmo sem ter lido o resto da palavra, a palavra é aceite!
- A segunda é que quando a palavra a reconhecer começa pelo símbolo b ou é a palavra vazia, a máquina desloca-se sucessivamente para a esquerda e não pára. Uma vez que não pára, tanto a palavra vazia como qualquer palavra que comece por b não é aceite.

7.1.3 Linguagens recorrentes e recorrentemente enumeráveis

Conforme ilustrado no Exemplo 7.1.10, p. 198 existe a possibilidade de uma máquina de Turing não parar com algumas palavras, dizendo nesse caso que essas palavras não são aceites e que não pertencem à linguagem reconhecida pela máquina. Como veremos, nem sempre é possível construir uma máquina de Turing que pare sempre (tanto para as palavras da linguagem como para as palavras que não são da linguagem).

Importa assim, estabelecer a distinção entre:

1. a classe das linguagens que são reconhecíveis por máquinas de Turing que param sempre (para qualquer palavra);
2. a classe das linguagens que são reconhecíveis por máquinas de Turing que podem não parar para alguma palavra que não seja da linguagem (embora parem sempre para as palavras da linguagem).

Definição 7.1.11 Linguagem recorrentemente enumerável.

Uma linguagem L é **recorrentemente enumerável** se existe uma máquina de Turing M que a reconhece, i.e., $L = \mathcal{L}(M)$.

A origem do termo "recorrentemente enumerável" será clarificada na Secção 7.4, p. 215. Por agora, é suficiente reter que uma máquina de Turing que reconhece uma linguagem recorrentemente enumerável pode não parar com palavras que não pertencem à linguagem.

Definição 7.1.12 Linguagem recorrente.

Uma linguagem L é **recorrente** se existe uma máquina de Turing M que a reconhece e que além disso *pára sempre*, ou seja,

- se $w \in L$ então M pára num estado de aceitação;
- se $w \notin L$ então M pára num estado de rejeição.

Uma linguagem recorrente diz-se também **decidível** no sentido em que existe um "algoritmo" (, uma máquina de Turing que pára sempre) que a decide (i.e., que resolve o problema de decidir se uma palavra pertence ou não à linguagem).

Claramente qualquer linguagem recorrente é também recorrentemente enumerável. No entanto, a recíproca não é verdadeira, como veremos na Secção 7.5, p. 219.

Teorema 7.1.13

Sejam L_1 e L_2 linguagens sobre um alfabeto Σ.

Se L_1 e L_2 são recorrentes (respectivamente, recorrentemente enumeráveis) então $L_1 \cup L_2$ e $L_1 \cap L_2$ são recorrentes (respectivamente, recorrentemente enumeráveis).

Demonstração.

Sejam M_1 e M_2 máquinas de Turing que decidem, respectivamente, as linguagens recorrentes L_1 e L_2.

Podemos combinar sequencialmente as máquinas M_1 e M_2 numa única máquina de Turing $M_\cup$. Dada uma palavra w, a máquina $M_\cup$ simula M_1 com w e, no caso de M_1 rejeitar w, simula M_2 com w. Se pelo menos uma delas aceitar w então a

máquina $M_\cup$ também aceita w. Esta máquina, assim definida, decide $L_1 \cup L_2$.

De forma análoga, podemos combinar sequencialmente as máquinas M_1 e M_2 numa única máquina de Turing $M_\cap$. Dada uma palavra w, a máquina $M_\cap$ simula M_1 com w e, em seguida, simula M_2 com w. Se ambas aceitarem w então $M_\cap$ também a aceita, caso contrário, rejeita-a. Esta máquina, assim definida, decide $L_1 \cap L_2$.

Sejam agora M_1 e M_2 máquinas de Turing que reconhecem as linguagens recorrentemente enumeráveis L_1 e L_2.

Podemos combinar em paralelo as máquinas M_1 e M_2 numa única máquina de Turing $M_\cup$. Dada uma palavra w, a máquina $M_\cup$ executa alternadamente um passo de M_1 e um passo de M_2 com w. Se uma delas chegar a aceitar w então a máquina $M_\cup$ também aceita w. Se ambas rejeitarem w então a máquina $M_\cup$ pára e também rejeita w. Se a máquina $M_\cup$ não parar, isso significa que nem M_1 nem M_2 param com a palavra w, pelo que w não pertence nem a L_1 nem a L_2, ou seja, não pertence a $L_1 \cup L_2$. Concluímos que a máquina $M_\cup$, assim definida, reconhece $L_1 \cup L_2$.

Podemos combinar sequencialmente as máquinas M_1 e M_2 numa única máquina de Turing $M_\cap$. Dada uma palavra w, a máquina $M_\cap$ começa por simular M_1 com w. Se a máquina M_1 não pára com w então também $M_\cap$ não pára, pelo que a palavra w não é aceite. Se M_1 pára com w e a rejeita então também $M_\cap$ pára com w e rejeita-a. Se M_1 pára com w e a aceita então é iniciada a simulação de M_2 com w. Se a máquina M_2 não pára com w então também $M_\cap$ não pára, pelo que a palavra w não é aceite. Se M_2 pára com w e a rejeita então também $M_\cap$ pára com w e a rejeita. Se M_2 pára com w e a aceita então também $M_\cap$ pára com w e a aceita. A máquina $M_\cap$, assim definida, reconhece $L_1 \cap L_2$.

Teorema 7.1.14

Se L é uma linguagem recorrente então $\overline{L}$ é recorrente.

Demonstração.

Seja M uma máquina de Turing que decide uma linguagem recorrente L. Com base em M, construímos uma máquina de Turing $\overline{M}$ semelhante a M, com a única diferença de os estados de aceitação de $\overline{M}$ serem os estados de rejeição de M.

Se uma linguagem e a sua linguagem complementar são ambas recorrentemente enumeráveis então também são ambas decidíveis.

Teorema 7.1.15

Se uma linguagem L e a sua linguagem complementar, $\overline{L}$, são recorrentemente enumeráveis então também L e $\overline{L}$ são recorrentes.

Demonstração.

Sejam M e $\overline{M}$ máquinas de Turing, respectivamente reconhecedoras de linguagens recorrentemente enumeráveis L e $\overline{L}$.

Vamos mostrar que L é recorrente, construindo uma máquina de Turing que a decide. Aplicando o Teorema 7.1.14, p. 200 concluímos que também $\overline{L}$ é recorrente.

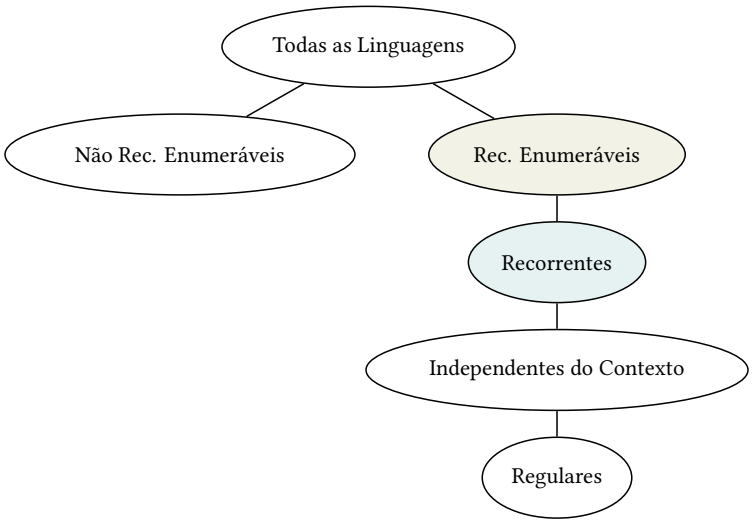
Podemos combinar em paralelo as máquinas M e $\overline{M}$ numa máquina M_D que, dada uma palavra w, executa alternadamente um passo de M e um passo de $\overline{M}$. Uma vez que w pertence a L ou a $\overline{L}$, é garantido que a máquina M_D tem de parar e aceitar ou rejeitar w após um certo número de passos.

Quando M_D pára após um passo de M, se M aceita w então M_D aceita w, senão M_D rejeita w. Por outro lado, quando M_D pára após um passo de $\overline{M}$, se $\overline{M}$ aceita w então M_D rejeita w, senão M_D aceita w.

Na Secção 7.5, p. 219 apresentaremos uma linguagem recorrentemente enumerável que não é recorrente. Pelo Teorema 7.1.15, p. 200, o complementar dessa linguagem não pode ser recorrentemente enumerável. Assim, a classe das linguagens recorrentemente enumeráveis não é fechada para a operação do complementar.

Deixamos como exercícios as demonstrações de que a classe das linguagens recorrentemente enumeráveis é fechada para a operação estrela de Kleene e para homomorfismos.

Sintetizamos no esquema seguinte a Hierarquia de Linguagens estudadas até ao momento:



Finalizamos esta secção com uma prova indirecta de que existem linguagens não recorrentemente enumeráveis. A demonstração da não enumerabilidade do conjunto dos subconjuntos de um conjunto infinito enumerável, enunciada no lema seguinte, emprega a mesma técnica de diagonalização que George Cantor utilizou

para provar que o conjunto dos números reais não é enumerável [20].

Lema 7.1.16

Seja X um conjunto infinito enumerável. Então o conjunto dos subconjuntos de X, $\mathcal{P}(X)$, não é enumerável.

Teorema 7.1.17

A classe das linguagens recorrentemente enumeráveis é enumerável.

Demonstração.

Começamos por enunciar um conjunto de propriedades cuja demonstração, longa e sobretudo técnica, sugerimos como exercício:

- Qualquer conjunto finito pode ser codificado como uma palavra binária.
- Qualquer par de palavras binárias pode ser codificado como uma palavra binária.
- Qualquer n-uplo de palavras binárias pode ser codificado como uma palavra binária.
- Qualquer matriz de palavras binárias pode ser codificada como uma palavra binária.

Dado que as máquinas de Turing são especificadas por um sêxtuplos ordenados de objectos finitos, qualquer máquina de Turing pode ser codificada como uma palavra binária. Dependendo da codificação, existem palavras binárias que não correspondem a qualquer máquina de Turing, mas tal não afecta a conclusão que pretendemos: a classe de todas as máquinas de Turing tem a mesma cardinalidade que o conjunto de todas as palavras binárias.

Uma vez que o conjunto $\{0,1\}^\star$ é infinito enumerável, também a classe das linguagens que são reconhecidas por máquinas de Turing é enumerável.

Corolário 7.1.18

Existem linguagens não recorrentemente enumeráveis.

Demonstração.

Uma linguagem L é um subconjunto de $\Sigma^\star$. Para além disso, o conjunto $\Sigma^\star$ é infinito enumerável. Assim, a classe de todas as linguagens sobre o alfabeto Σ, $\mathcal{P}(\Sigma^\star)$, não é enumerável.

Pelo teorema anterior, a classe de todas as linguagens recorrentemente enumeráveis é apenas enumerável. Logo, têm de existir linguagens não recorrentemente enumeráveis.

7.2 Computabilidade e enumeradores

No contexto actual, o interesse das máquinas de Turing reside essencialmente no problema do reconhecimento de linguagens. No entanto, o conceito original, pensado por Alan Turing, era o de um dispositivo de cálculo, a que chamou "a-machine".

O trabalho de Turing versava sobretudo sobre a questão da computabilidade dos números reais e das funções inteiras, questões essas que vamos agora abordar.

7.2.1 Funções computáveis

Um número é Turing-computável se existe uma máquina de Turing que o "calcula" com uma qualquer precisão (finita), ou seja, partindo de uma fita que contém um dado valor $n \in \mathbb{N}$, a máquina consegue escrever na fita uma aproximação desse número com n dígitos de precisão.

São computáveis todos os números racionais, todos os números algébricos (aqueles que são raízes de polinómios com coeficientes inteiros), como por exemplo $\sqrt{2}$, bem como vários números transcendentes como o π ou ainda o número de Euler e.

Existem no entanto números não computáveis, como por exemplo a constante ω de Chaitin (Gregory Chaitin, Estados Unidos da América, 1947 -). Na verdade, uma vez que conjunto dos números computáveis é enumerável existem "inumeráveis" números não computáveis.

Sem modificar a definição, podemos encarar as máquinas de Turing como avaliadoras de funções: para cada palavra inicialmente colocada na fita (o argumento da função), a máquina executa uma sequência de passos, atingindo eventualmente uma configuração de paragem. O valor da função para o argumento dado é o conteúdo da fita na configuração de paragem (ou apenas parte do conteúdo, seguindo alguma convenção fixada à priori).

Existem funções às quais não é possível associar uma máquina de Turing. Às funções que possuem uma máquina de Turing que as avalie no sentido exposto acima, chamamos funções Turing-computáveis.

Definição 7.2.1 Função Turing-computável.

Uma função $f : D \subseteq \Sigma^\star \to \Gamma^\star$ é *Turing-computável*, ou simplesmente computável, se existir alguma máquina de Turing

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

tal que para toda a palavra $x \in D$,

$$sx \overset{\star}{\vdash} qf(x)$$

para alguma configuração de paragem $qf(x)$, com $q \in F$.

Exemplo 7.2.2

Pretendemos construir uma máquina de Turing para calcular a diferença própria entre dois números inteiros e positivos

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

Utilizamos a *representação unária* para codificar os números inteiros não negativos: $n \equiv 0^n = 00 \cdots 0$.

As codificações dos argumentos da função, m e n, vão ser separadas pelo símbolo 1 e a máquina de Turing deverá cumprir a especificação

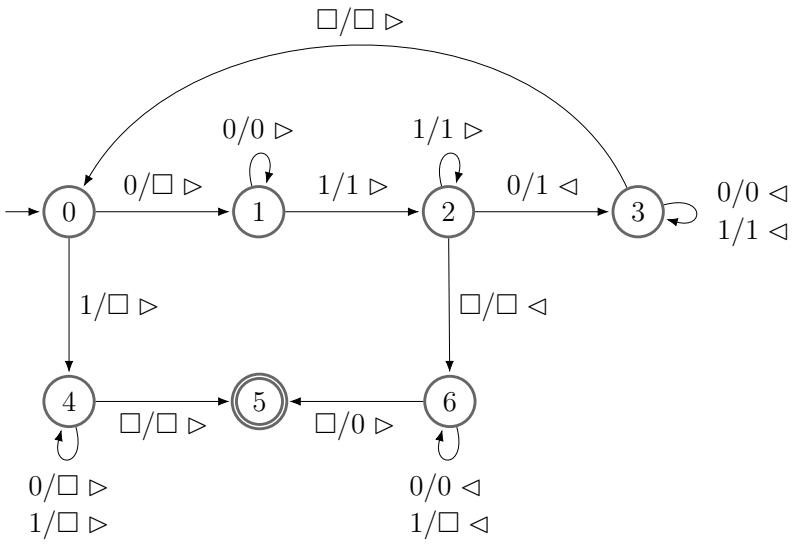
$$0^m 1 0^n \overset{\star}{\vdash} 0^{m \ominus n}$$

Esta notação indica que, partindo da configuração inicial $q_0 0^m 1 0^n$, a máquina de Turing deverá atingir a configuração de paragem $q 0^{m \ominus n}$, onde q é um estado de aceitação.

A estratégia utilizada consiste em:

- substituir cada 0 de m por $\square$ e cada 0 de n por 1;
- se $m \geq n$ (em q_6) os $(n+1)$ 1´s são substituídos por $\square$ e é reposto um 0 em falta;
- se $m < n$ (em q_4) todos os 0´s e 1´s são substituídos por $\square$.

O seguinte diagrama de transições esquematiza a solução.



Exemplo 7.2.3

Pretendemos construir uma máquina de Turing para calcular a função INC : $\mathbb{N}_0 \to \mathbb{N}$ definida por

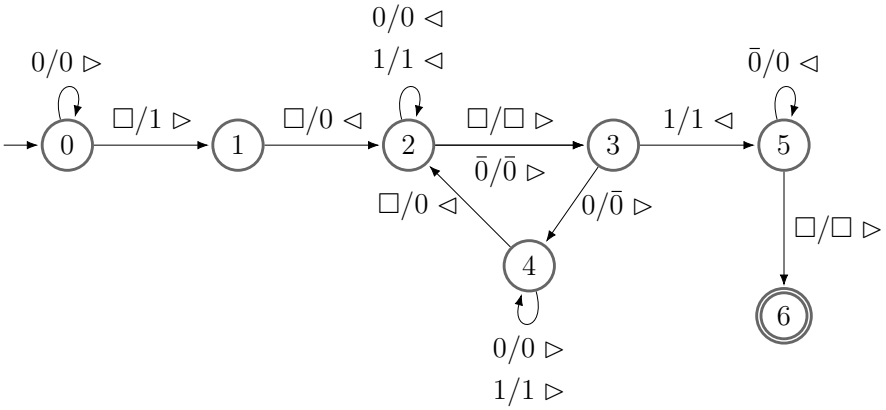
$$\text{INC}(n) = n + 1.$$

Vamos usar a *representação unária* $n \equiv 0^n$ para codificar os números inteiros não negativos e a máquina de Turing cumpre a especificação

$$0^n \vdash^\star 0^n 1 0^{n+1}$$

A estratégia utilizada consiste em:

- colocar um 1 a seguir aos 0's;
- para cada 0 não marcado antes do 1 marcar o 0 e acrescentar mais um 0 depois do 1;
- quando não existirem mais zeros por marcar à esquerda do 1 desmarcá-los todos.



A relação entre problemas de decisão e problemas de computação é tornada precisa no estudo da complexidade computacional de problemas. Para já importa reter que qualquer problema de decisão se pode converter num problema de computação em que a função toma valores 0 ou 1. Reciprocamente, o problema de computação de uma função f pode ser visto como um problema de decisão sobre a linguagem constituída pelas palavras que são pares da forma $(x, f(x))$.

7.2.2 A máquina de Turing universal

É possível codificar a definição de cada máquina de Turing M, de forma única, como uma palavra $\langle M \rangle \in \{0,1\}^\star$. Por exemplo, codificamos os estados em unário como sequências de 0's e usamos sequências de 1's para demarcar as várias componentes da definição.

Assim, é possível definir uma máquina de Turing U que aceita como entrada um par constituído pela descrição $\langle M \rangle$ de uma qualquer máquina de Turing M e por uma palavra w e que simula o funcionamento de M com a palavra w:

- Se M pára num estado de aceitação então U pára num estado de aceitação e aceita w;
- Se M pára num estado de rejeição então U pára num estado de rejeição e rejeita w;
- Se M não pára com w então U também não pára com w.

A máquina U, assim definida, é conhecida como a *Máquina de Turing Universal.*

7.2.3 Enumeradores

Temos olhado para as máquinas de Turing como reconhecedoras de linguagens. Vamos agora ver como uma máquina de Turing pode ser encarada como um dis-

positivo gerador de uma linguagem.

Um *enumerador* é essencialmente uma máquina de Turing com duas fitas: uma é uma fita normal de leitura/escrita que serve como auxiliar de cálculo, enquanto a outra é uma fita de saída, apenas de escrita.

Um enumerador começa a funcionar com as suas duas fitas vazias e, passo a passo, vai escrevendo na fita de saída palavras $w \in \Sigma^\star$ separadas por um símbolo especial $\# \notin \Sigma$.

Os símbolos na fita de saída nunca são alterados uma vez que a "cabeça" de escrita avança sempre para a direita (para um espaço em branco) sempre que escreve um símbolo da palavra ou o símbolo separador de duas palavras. A cabeça de escrita da fita de saída permanece na mesma posição ($\triangledown$) quando está posicionada sob um espaço em branco e não existe nenhum símbolo para escrever.

A *linguagem gerada pelo enumerador* não é mais do que o conjunto de todas as palavras que (potencialmente) são escritas na fita de saída. Se um enumerador atinge uma configuração de paragem após um número finito de passos, então a linguagem gerada é finita. No entanto, se nunca chegar a atingir uma configuração de paragem então gera, potencialmente, um número infinito de palavras.

Um enumerador pode ser definido formalmente de várias maneiras distintas, todas elas equivalentes. A que vamos adoptar é a seguinte:

Definição 7.2.4 Enumerador de uma linguagem.

Um enumerador é um quíntuplo ordenado

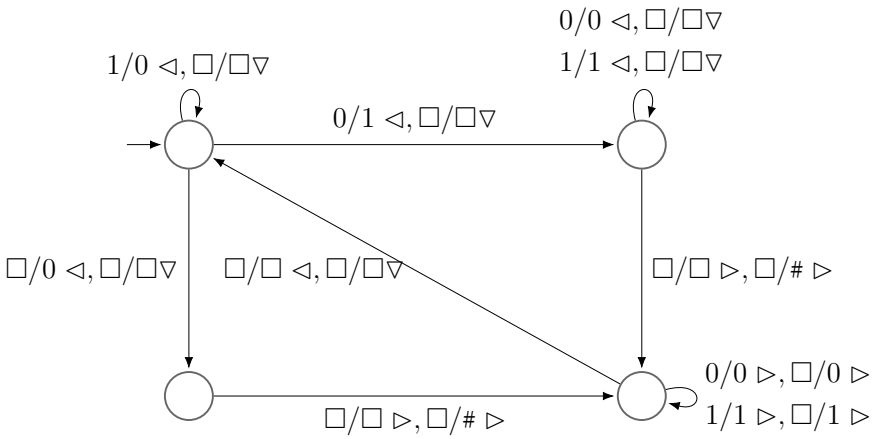
$$E = (Q, \Sigma, \Gamma, \delta, s)$$

onde:

- Q é um conjunto finito de *estados*;
- Σ é o *alfabeto de saída*;
- Γ é *alfabeto de trabalho*, com $\Sigma \subset \Gamma$;
- $\square \in \Gamma \setminus \Sigma$ é o símbolo *branco*;
- $\# \notin \Gamma$ é o símbolo *separador*;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\triangleleft, \triangleright\} \times \big((\Sigma \cup \{\#\}) \times \{\triangleright\} \bigcup \{\square\} \times \{\triangledown\}\big)$.

Exemplo 7.2.5 Um enumerador de $\{0,1\}^\star$.

O diagrama seguinte apresenta um enumerador da linguagem $\{0,1\}^\star$, listando as palavras binárias em ordem lexicográfica: $\varepsilon\#0\#1\#00\#01\#10\#11\#\cdots$.



Teorema 7.2.6

Se L é uma linguagem recorrente então possui um enumerador*, isto é, existe uma máquina de Turing que lista apenas as palavras de L.*

Demonstração.

Se L é uma linguagem recorrente sobre um alfabeto Σ então existe uma máquina de Turing M para decidir L. É também possível definir um enumerador $\widetilde{M}$ de $\Sigma^\star$.

Podemos combinar as máquinas M e $\widetilde{M}$, definindo um novo enumerador que escreve sucessivamente cada uma das palavras de $\Sigma^\star$ na fita de leitura/escrita, associada à MT que decide a linguagem. Na fita de saída são escritas apenas as palavras da linguagem reconhecidas pela máquina M. O procedimento seguinte sintetiza o modo de funcionamento combinado, passo a passo, das duas máquinas.

Procedimento 7.2.7 Máquina de Turing → Enumerador (1).

Entrada: uma máquina de Turing, M, que decide $\mathcal{L}(M)$.

Saída: um enumerador de $\mathcal{L}(M)$.

preparar um enumerador, $\widetilde{M}$, de $\Sigma^\star$.

para $i = 1, 2, \ldots$

$\widetilde{M}$ escreve w_i na fita de M;

M decide se w_i pertence ou não à linguagem:

se M pára num estado de aceitação

então w_i é escrita na fita de saída;

senão (M pára num estado de rejeição)

w_i é ignorada.

Notamos que o procedimento anterior não funciona quando a máquina de Turing M apenas reconhece L, uma vez que pode não parar para alguma palavra $w_i \notin L$. No entanto, temos o seguinte resultado:

Teorema 7.2.8

Se L é uma linguagem recorrentemente enumerável então possui um enumerador.

Demonstração.

Sejam M uma máquina de Turing que reconhece a linguagem $L \subseteq \Sigma^\star$ e $\widetilde{M}$ um enumerador de $\Sigma^\star$. Definimos uma nova máquina de Turing (um enumerador) que executa o procedimento a seguir explicitado.

Procedimento 7.2.9 Máquina de Turing $\rightarrow$ Enumerador (2).

Entrada: uma máquina de Turing, M, que reconhece $\mathcal{L}(M)$.

Saída: um enumerador de $\mathcal{L}(M)$.

preparar um enumerador, $\widetilde{M}$, de $\Sigma^\star$.

para $i = 1, 2, \ldots$

$\widetilde{M}$ acrescenta w_i à fita auxiliar;

para $j = 1, 2, \ldots, i$

w_j é copiada da fita auxiliar para a fita de M;

M executa apenas i passos;

se M pára num estado de aceitação ao fim de i passos

então w_j é escrita na fita de saída.

Teorema 7.2.10

Se L possui um enumerador então L é recorrentemente enumerável.

Demonstração.

Seja $\widetilde{M}$ um enumerador de L. Baseado neste, definimos uma nova máquina de Turing M que no início contém apenas a palavra w na fita.

Em cada passo do procedimento, o enumerador escreve uma nova palavra à direita da palavra w, separada por um símbolo branco. Em seguida, estas duas palavras são comparadas símbolo a símbolo, atingindo um estado de paragem e reconhecimento quando a comparação permite concluir que são iguais.

O procedimento seguinte descreve esta construção.

Procedimento 7.2.11 Enumerador → Máquina de Turing.

Entrada: um enumerador, $\widetilde{M}$, de uma linguagem L.

Saída: uma máquina de Turing, M, que reconhece L.

Definir uma máquina de Turing, M que, dada uma qualquer palavra w inscrita na sua fita executa as seguintes acções:

para $i = 1, 2, \ldots$

$\widetilde{M}$ escreve w_i na fita de M a seguir a w;

M compara as duas palavras, verificando se são iguais:

se $w = w_i$

então M pára (num estado de aceitação) e aceita w;

senão M apaga a palavra w_i da sua fita.

Usando os resultados anteriores (ver a construção da máquina de Turing universal) é possível também definir um enumerador de todas as máquinas de Turing partindo de um enumerador de $\{0,1\}^\star$.

Procedimento 7.2.12 Enumerador das máquinas de Turing.

Entrada: um enumerador $\widetilde{M}$ de $\{0,1\}^\star$.

Saída: um enumerador $\widetilde{\widetilde{M}}$ de todas as máquinas de Turing.

Para $i = 1, 2, \ldots$

$\widetilde{M}$ escreve x_i na fita auxiliar de $\widetilde{\widetilde{M}}$;

se x_i é uma codificação válida de máquina de Turing

então escreve x_i na fita de saída de $\widetilde{\widetilde{M}}$;

senão $\widetilde{\widetilde{M}}$ apaga x_i da sua fita auxiliar.

7.3 A tese de Turing

Existem muitos modelos de máquinas de Turing. Por exemplo,

- com a opção de não movimento: $\{\lhd, \rhd, \triangledown\}$;
- com uma fita semi-infinita: a fita prolonga-se apenas para direita de uma certa posição inicial;
- com múltiplas fitas de trabalho;
- com uma fita de "output" apenas para escrita, para além da fita normal de leitura/escrita;
- com fitas multi-dimensionais;
- "off-line": para além da fita normal de leitura escrita, existe uma fita de "input" que é apenas de leitura;
- não deterministas ...

Qualquer um desses modelos, desde que "razoável" e "não restritivo em demasia", será equivalente ao modelo de máquina de Turing que adoptámos, no sentido em que ambos reconhecem, decidem e computam as mesmas classes de linguagens.

7.3.1 Máquinas de Turing com mais do que uma fita

Podemos pensar em generalizar a definição de máquina de Turing (com uma fita) para máquinas de Turing com várias fitas de leitura/escrita.

Se a máquina tem $k \geq 2$ fitas então para cada fita existe uma "cabeça" de leitura/escrita e a transição de um estado para outro depende de todos os símbolos que estejam sob as "cabeças".

Assim, passo a passo, é escrito um novo símbolo em cada uma das fitas e cada uma das "cabeças" move-se independentemente das restantes, quer para a esquerda ($\triangleleft$), quer para a direita ($\triangleright$) ou simplesmente não se move ($\triangledown$).

Essencialmente uma máquina de Turing determinista com $k \geq 2$ fitas difere de uma máquina de Turing determinista (com uma só fita) apenas na assinatura da função de transição, a qual passa a ser

$$\delta \colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{\triangleleft, \triangledown, \triangleright\}^k.$$

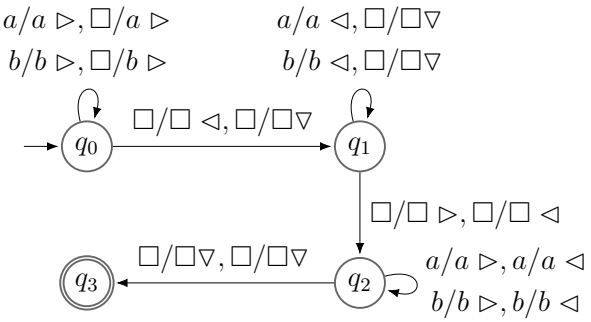
Exemplo 7.3.1

Pretendemos construir uma máquina de Turing com duas fitas que reconheça a linguagem dos palíndromos

$$\{w \in \{a,b\}^\star \colon w = w^{-1}\}$$

Inicialmente, a primeira fita contém a palavra w e a sua "cabeça" de leitura está sobre a primeira letra da palavra. A segunda fita está inicialmente vazia e a sua "cabeça" de leitura encontra-se numa posição arbitrária.

A ideia do funcionamento da máquina consiste em começar por copiar a palavra w para a segunda fita (estado q_0). Em seguida, a "cabeça" da primeira fita move-se para a esquerda até ao primeiro símbolo da palavra enquanto a "cabeça" da segunda fita se posiciona sobre o último símbolo da palavra na segunda fita (estado q_1). Finalmente, a máquina começa a testar se os símbolos sob as "cabeças" são iguais, movendo a "cabeça" da primeira fita da esquerda para direita ao mesmo tempo que a "cabeça" da segunda fita se move da direita para a esquerda (estado q_2).



Teorema 7.3.2

Qualquer máquina de Turing com $k \geq 2$ fitas pode ser convertida numa máquina de Turing com uma fita.

O teorema anterior, cuja demonstração deixamos ao cuidado do leitor, permite concluir que qualquer problema que seja resolúvel por uma MT com duas ou mais fitas é também resolúvel por uma MT com uma só fita. Assim, o modelo da MT com $k \geq 2$ fitas não é mais poderoso que o modelo da MT com uma fita.

Para além disso, uma vez que qualquer MT com uma fita é simulável com uma MT com k fitas, bastando para isso utilizar uma das k fitas disponíveis e ignorar as $k-1$ restantes, podemos concluir que os dois modelos são equivalentes. Contudo, a utilização de múltiplas fitas permite construir e descrever máquinas de Turing mais simples do que as que se obteriam usando apenas uma fita.

Exemplo 7.3.3

Sejam M_1 e M_2 máquinas de Turing que decidem as linguagens L_1 e L_2, respectivamente. Podemos decidir a concatenação da linguagem L_1 com L_2 com uma máquina de Turing, M, com três fitas.

A palavra de entrada w é escrita na primeira fita da máquina M. A cabeça de leitura/escrita da primeira fita posiciona-se sobre o primeiro símbolo de w ou sobre um símbolo branco, no caso em que $w = \varepsilon$.

No início de cada etapa, constituída por quatro passos, a posição da cabeça de leitura/escrita define uma partição da palavra de entrada w: $w = uv$, onde u é a palavra à esquerda da cabeça e v é a palavra à direita, incluindo a posição de leitura.

De uma etapa para a seguinte, a cabeça de leitura da primeira fita avança uma posição na sequência de símbolos que constituem a palavra w, o que permite testar em cada etapa uma das possíveis partições $w = uv$.

A segunda fita serve para simular o funcionamento da máquina M_1 com os prefixos u de w, que são copiados da primeira fita. O conteúdo desta segunda fita é apagado no final de cada simulação.

A terceira fita serve para simular o funcionamento da máquina M_2 com os sufixos v de w que são copiados da primeira fita. O conteúdo desta terceira fita é apagado no final de cada simulação.

O algoritmo é assim:

(1) copiar o prefixo actual u de w da primeira para a segunda fita de M;

copiar o sufixo actual v de w da primeira para a terceira fita de M;

simular M_1 com u presente na segunda fita de M;

se M_1 aceita u

então simular M_2 com v presente na terceira fita de M;

se M_2 aceita v

então M pára e aceita w;

se chegou ao fim de w

então M pára e rejeita w;

senão M avança uma posição em w e volta (1).

7.3.2 A Tese de Church-Turing (1936)

Desde as primeiras décadas do século XX que diversos modelos de computação têm sido propostos e (quase) todos eles mostraram ser equivalentes ao modelo da máquina de Turing determinista. Entenda-se esta equivalência no sentido em que as funções que são computáveis pelos diversos modelos são também computáveis por máquinas de Turing e vice-versa.

Embora alguns modelos de computação sejam mais poderosos que o modelo da máquina de Turing, estes são considerados não razoáveis. É do consenso geral que um modelo razoável deve satisfazer as seguintes propriedades:

1. A especificação da computação deve traduzir-se numa sequência finita de instruções;
2. Cada uma das instruções deve ser realizável num número finito de passos ou em tempo finito;
3. As instruções devem ser realizáveis de forma determinista de modo a que o seu resultado seja claro, não ambíguo e previsível.

Intuitivamente, qualquer especificação de uma computação que satisfaça os requisitos anteriores pode ser simulada por uma máquina de Turing. Este é essencialmente o conteúdo da chamada tese de Church-Turing

> Uma função $f\colon D \subseteq \mathbb{N} \to \mathbb{N}$ é computável por um algoritmo (um procedimento que termina sempre) se e só se é computável por uma máquina de Turing.

O conteúdo desta tese emergiu inicialmente, com uma abordagem diferente, no trabalho sobre o cálculo-λ de Church. No entanto, não era um resultado tão evidente conforme reconhecido mais tarde por Gödel:

> ... But I was completely convinced only by Turing's paper.
>
> —Gödel: letter to Kreisel of May 1, 1968.

O próprio Church afirmou a clara vantagem da formulação da tese usando as máquinas de Turing:

> ... has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately — i.e., without the necessity of proving preliminary theorems.
>
> —Alonzo Church: Review of Turing, 1937

É claro que a tese de Church-Turing não pode ser provada como se fosse um teorema. Poderá eventualmente vir a ser refutada se, algum dia, alguém desenvolver um modelo de computação que obtenha o consenso de ser razoável e que, para além disso, permita computar alguma função não computável por uma máquina de Turing, ou então resolver (decidir) algum problema que não seja resolúvel (decidível) por uma máquina de Turing.

Assim, não é necessário construir uma máquina de Turing para argumentar que uma determinada função é computável. É suficiente apresentar uma demonstração de que a função é realizável num qualquer modelo de computação conveniente (desde que equivalente). Reciprocamente, se queremos provar que uma função não é computável basta elaborar uma demonstração num qualquer modelo de computação que seja razoável (e equivalente). Em ambas os casos a tese de Church-Turing apresenta-se como um importante suporte empírico para garantir que as mesmas conclusões ocorrerão em qualquer modelo de computação que possamos considerar.

7.4 Problemas decidíveis

Existem muitos problemas que podem ser formulados naturalmente como problemas de decisão. Por exemplo:

- Será que podemos decidir se duas quaisquer gramáticas independentes do contexto geram a mesma linguagem?
- Será que podemos decidir se uma qualquer gramática independente do contexto é ambígua?
- Será que podemos decidir se a linguagem complementar de uma qualquer linguagem independente do contexto é independente do contexto?
- Será que podemos decidir se a intersecção de duas quaisquer linguagens independentes do contexto é ainda independente do contexto?

- Será que podemos decidir se uma qualquer gramática independente do contexto gera a linguagem universal?

Por mais surpreendente que possa parecer, nenhum dos problemas acima referido tem solução. São todos problemas indecidíveis, pelo que não existe nenhum método geral para os resolver (que possa ser aplicado a todas as instâncias do problema). Na próxima secção veremos como foi possível chegar a estes resultados.

Por outro lado, existem muitos problemas que são decidíveis. Vamos listar alguns problemas na área das linguagens formais que são decidíveis (i.e., que têm algoritmos que os resolvem).

7.4.1 Decidir se um autómato finito reconhece uma palavra

Proposição 7.4.1

A linguagem $\text{ACC}_{AFD} = \{\langle A, w\rangle : A \text{ é um AFD que aceita } w\}$ *é decidível.*

Demonstração.

Basta construir uma máquina de Turing, M_{AFD}, que dado uma codificação $\langle A, w\rangle$ de um AFD A e de uma palavra w:

- *simula* A com a palavra w;
- *se* a simulação termina num estado de aceitação do AFD *então* M_{AFD} aceita, *senão* rejeita.

Proposição 7.4.2

A linguagem $\text{ACC}_{AFND} = \{\langle A, w\rangle : A \text{ é um AFND que aceita } w\}$ *é decidível.*

Demonstração.

Basta construir uma máquina de Turing, M_{AFND} que dada uma codificação $\langle A, w\rangle$ de um AFND A e de uma palavra w:

- *converte* a descrição $\langle A\rangle$ de A numa descrição $\langle B\rangle$ de um AFD B equivalente a A;
- *simula* B com a palavra w, usando a máquina M_{AFD} definida na demonstração da Proposição 7.4.1, p. 216;
- *se* a simulação termina num estado de aceitação do AFD

  então M_{AFND} aceita, *senão* rejeita.

7.4.2 Decidir se uma palavra satisfaz uma expressão regular

Proposição 7.4.3

A linguagem $\text{ACC}_{ER} = \{\langle R, w\rangle : w$ *satisfaz a expressão regular* $R\}$ *é decidível.*

Demonstração.

Basta construir uma máquina de Turing, M_{ER} que dada uma codificação $\langle R, w\rangle$ de uma ER R e de uma palavra w:

- *converte* a descrição $\langle R\rangle$ de R numa descrição $\langle A\rangle$ de um AFD A que reconhece a linguagem $\mathcal{L}(R)$;
- *simula* A com a palavra w, usando a máquina M_{AFD} definida na demonstração da Proposição 7.4.1, p. 216;
- *se* a simulação termina num estado de aceitação do AFD

  então M_{ER} aceita, *senão* rejeita.

7.4.3 Decidir se a linguagem reconhecida por um autómato finito determinista é vazia

Proposição 7.4.4

A linguagem $\text{E}_{AFD} = \{\langle A\rangle : A$ *é um AFD tal que* $\mathcal{L}(A) = \emptyset\}$ *é decidível.*

Demonstração.

Basta construir uma máquina de Turing que, dada a codificação $\langle A\rangle$ de um AFD A, testa se pelo menos um dos estados de aceitação é atingível a partir do estado inicial:

- *marca* o estado inicial de A;
- *repete* até que não sejam marcados mais estados ou que seja marcado um estado de aceitação:
  - *marca* todos os estados q para os quais existe $a \in \Sigma$ tal que $\delta(p, a) = q$ para algum estado p já marcado;
- *se* algum dos estados de aceitação foi marcado *então* rejeita, *senão* aceita.

7.4.4 Decidir se dois autómatos finitos deterministas reconhecem a mesma linguagem

Proposição 7.4.5

A linguagem $\text{EQ}_{AFD} = \{\langle A, B\rangle : A \text{ e } B \text{ são AFD tais que } \mathcal{L}(A) = \mathcal{L}(B)\}$ *é decidível.*

Demonstração.

Basta construir uma máquina de Turing que dada uma codificação $\langle A, B\rangle$ de dois quaisquer AFD A e B:

- *constrói* uma descrição $\langle C\rangle$ de um AFD C tal que $\mathcal{L}(C) = \left(\mathcal{L}(A) \cap \overline{\mathcal{L}(B)}\right) \cup \left(\overline{\mathcal{L}(A)} \cap \mathcal{L}(B)\right)$
- *testa* se $\mathcal{L}(C) = \emptyset$.

7.4.5 Decidir se uma gramática independente do contexto gera uma dada palavra

Proposição 7.4.6

A linguagem $\text{ACC}_{GIC} = \{\langle G, w\rangle : G \text{ é uma GIC que gera } w\}$ *é decidível.*

Demonstração.

Relembramos que qualquer palavra w de tamanho $|w| = n$ gerada por uma GIC na forma normal de Chomsky tem uma derivação em $2n - 1$ ou menos passos.

Assim, construímos uma máquina de Turing, M_{GIC} que dada a codificação $\langle G, w\rangle$:

- *converte* G na forma Normal de Chomsky;
- *lista* todas as derivações com $2n - 1$ ou menos passos.
- *se* w estiver na lista *então* aceita *senão* rejeita.

7.4.6 Decidir se a linguagem gerada por uma gramática independente do contexto é vazia

Proposição 7.4.7

A linguagem $\text{E}_{GIC} = \{\langle G\rangle : G \text{ uma GIC tal que } \mathcal{L}(G) = \emptyset\}$ *é decidível.*

Demonstração.

Basta construir uma máquina de Turing, M que dada uma codificação $\langle G\rangle$ *testa* se a variável inicial é geradora.

7.4.7 Decidir se um autómato de pilha reconhece uma dada palavra

Proposição 7.4.8

A linguagem $\text{ACC}_{AP} = \{\langle A, w\rangle : A \text{ é um autómato de pilha que aceita } w\}$ *é decidível.*

Demonstração.

Basta construir uma máquina de Turing, M_{AP} que dada uma codificação $\langle A, w\rangle$:

- *constrói* $\langle G\rangle$, onde G é uma GIC tal que $\mathcal{L}(A) = \mathcal{L}(G)$;
- *se* M_{GIC} aceita a entrada $\langle G, w\rangle$

  então M_{AP} aceita

  senão rejeita.

7.5 Problemas indecidíveis

> ... Is the axiom of the solvability of every problem a peculiar characteristic of mathematical thought alone, or is it possibly a general law inherent in the nature of the mind, that all questions which it asks must be answerable? ... This conviction of the solvability of every mathematical problem is a powerful incentive to the worker. We hear within us the perpetual call: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no *ignoramibus*.
>
> —Extracto da apresentação de Hilbert, "Mathematical Problems", International Congress of Mathematicians, Paris, 1900.

7.5.1 O Entscheidungsproblem

A palavra "Entscheidungsproblem" significa "problema de decisão" e está associado ao matemático David Hilbert, que o apresentou em 1928 no "Sixth International Congress of Mathematicians, Bologna". Na altura, Hilbert estaria convicto de que todos os problemas matemáticos teriam solução. Podemos condensar o enunciado deste problema, originalmente colocado de um modo mais abrangente e ambicioso do que o aqui apresentado, na seguinte questão:

> Existirá um procedimento para determinar, numa sequência finita de passos, se uma dada afirmação especificada na linguagem da lógica/aritmética de primeira ordem é, ou não, demonstrável a partir dos axiomas de Peano e usando apenas as regras da lógica de primeira ordem?

Notamos que a existência de um tal procedimento permitiria, por exemplo, dar resposta à conjectura de Goldbach, um famoso problema ainda em aberto na área da Teoria de Números sobre a possibilidade de decompor os números naturais ≥ 4 na soma de dois números primos:

$$\forall k > 1, \exists p, q \geq 2 \colon 2k = p + q \wedge \text{primo}(p) \wedge \text{primo}(q).$$

O problema de decisão foi resolvido em 1935/1936, passada menos de uma década da sua formulação, pelos matemáticos Alan Turing em Cambridge e Alonzo Church em Princeton. A resposta ao problema, surpreendente por ser negativa, abalou profundamente a visão da época sobre os fundamentos da matemática espelhados no programa de Hilbert.

Apesar dos formalismos utilizados por Turing e Church serem completamente distintos, podemos considerar que ambos seguiram a mesma abordagem geral. Essencialmente, começaram por propor uma definição rigorosa de algoritmo e, em seguida, mostraram que as instâncias de um problema que sabiam ser não decidível eram representáveis como instâncias do "Entscheidungsproblem".

Esse problema indecidível é o Problema da Paragem, sobre o qual nos debruçaremos em seguida. Assim, se o "Entscheidungsproblem" tivesse uma resposta positiva também o problema da paragem a teria, o que, conforme Turing e Church provaram, não é o caso.

7.5.2 O problema da paragem

> Se Deus é omnipotente então pode criar uma pedra que nem mesmo Ele pode erguer.
>
> —"Introduction to Computer Theory", Daniel I.A. Cohen, 1991

Um das mais famosas questões associadas à noção de algoritmo consiste em saber se existe ou não um algoritmo genérico que consiga verificar se um qualquer outro procedimento pára ou não com todas as instâncias do problema que esse procedimento pretende resolver.

Esta questão, conhecida como o Problema da Paragem, foi formulada por Church usando o seu formalismo do cálculo-λ e também por Turing usando as suas "a--machines". Seguindo a abordagem de Turing, o problema em causa é:

> Existirá um Algoritmo (MT) que decida se qualquer Procedimento (MT) pára, com qualquer instância?

Este problema pode especificar-se como um problema de decisão sobre uma linguagem formal:

> Será que a linguagem
>
> $$\text{HALT} = \{\langle M, w\rangle : M \text{ é uma MT que pára com } w\}$$
>
> é decidível?

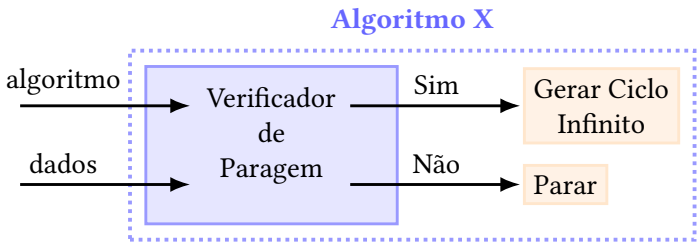
Vamos argumentar que este problema não é decidível, por redução ao absurdo e de modo semi-formal.

Admita-se que existe uma máquina de Turing para decidir este problema. O funcionamento de uma tal máquina, a que chamamos um verificador de paragem, é ilustrado na figura seguinte.

Dadas uma qualquer descrição de uma máquina de Turing $\langle M\rangle$ e uma qualquer palavra w, o verificador de paragem é uma máquina de Turing M_{H} tal que:

- quando M pára com w, M_{H} pára com a resposta Sim (num estado de aceitação).
- quando M não pára com w, M_{H} pára com a resposta Não (num estado de rejeição).

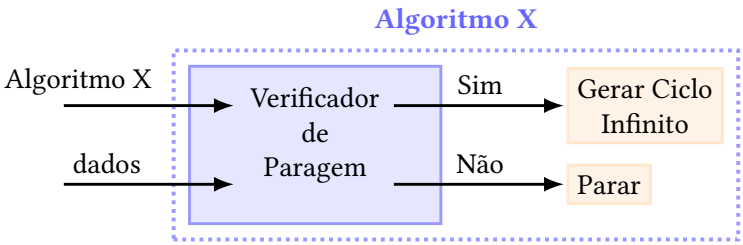
Existindo o verificador de paragem, é também possível definir o "Algoritmo X" (MT) ilustrado em seguida:



Assim, dadas uma qualquer descrição de uma máquina de Turing $\langle M \rangle$ e uma qualquer palavra w, o Algoritmo X é uma máquina M_X tal que:

- quando M pára com w, M_X não pára.
- quando M não pára com w então M_X pára.

Mas, atendendo a que o "algoritmo" (MT) de entrada é qualquer, podemos considerar a situação seguinte:



Esta situação é claramente *ABSURDA*!

De facto, com a entrada $\langle M_X \rangle$ e uma qualquer palavra w

- se M_X pára com w então M_X não pára.
- se M_X não pára com w então M_X pára.

Concluímos que *o Problema da Paragem não é decidível*, ou seja,

> *A linguagem HALT não é recorrente!*

7.5.3 Uma linguagem não recorrentemente enumerável

Prova-se, de forma semelhante à demonstração da inexistência de um algoritmo para resolver o Problema da Paragem, que a linguagem

$$\text{ACC}_{\text{MT}} = \{\langle M, w \rangle : M \text{ é uma MT que aceita } w\}$$

não é recorrente.

No entanto ACC_{MT} é recorrentemente enumerável. A demonstração passa por considerar a máquina de Turing universal, anteriormente definida.

Dados $\langle M, w\rangle$, onde M é uma MT e w é uma palavra, a máquina de Turing universal:

simula M com a entrada w;

se M atinge um estado de aceitação então aceita;

se M atinge um estado de rejeição então rejeita.

Note-se que a máquina de Turing Universal pode não parar: basta que M não pare!

Por outro lado, se L e $\overline{L}$ são recorrentemente enumeráveis então também L e $\overline{L}$ são decidíveis (basta executar M_L e $M_{\overline{L}}$ em paralelo). Concluímos que:

A linguagem $\overline{ACC_{MT}}$ não é recorrentemente enumerável.

7.5.4 O 10º Problema de Hilbert

Poderemos argumentar que o problema da paragem é um problema artificial, numa vã tentativa de minimizar a sua importância. Poderemos ainda pensar que a existência de uma resposta afirmativa para o "Entscheidungsproblem" seria certamente "boa demais para poder ser verdade". No entanto, existem muitos problemas, talvez mais concretos, que são indecidíveis.

Um desses problemas é o 10º problema de Hilbert sobre a existência de um método para determinar as soluções inteiras das equações diofantinas.

Uma equação diofantina é uma equação polinomial que se pode reduzir à forma

$$p(x_1, x_2, \ldots, x_n) = 0$$

onde p é um polinómio em duas ou mais variáveis com coeficientes inteiros.

Por exemplo, uma das soluções da equação diofantina $x^3 + y^3 = z^3 + w^3$ é $x = 1$, $y = 12$, $z = 9$ e $w = 10$.

Proposto por Hilbert em 1900, este problema foi alvo de um intenso estudo pelos matemáticos norte-americanos Julia Robinson (1919 - 1985), Martin Davis (1929 -) e Hilary Putnam (1926 -2016). A demonstração da sua indecidibilidade resistiu no entanto até 1970, altura em que o matemático russo Yuri Matijasevič (1947 -), com apenas 22 anos de idade, conseguiu a proeza de reduzi-lo ao problema da paragem.

Não vamos demonstrar que o 10º Problema de Hilbert é indecidível, mas apenas que é recorrentemente enumerável, o que é bastante mais simples.

Atendendo à Tese de Church-Turing, vamos usar uma "linguagem de alto nível" para delinear um procedimento que permita reconhecer a linguagem

$$\text{HILB} = \{\langle p\rangle : p \text{ é um polinómio de coeficientes inteiros com uma raiz inteira}\}.$$

Dado um polinómio de coeficientes inteiros, p, o procedimento testa de "forma sistemática" os elementos de $\mathbb{Z}^n$, calculando para cada $(x_1, x_2, \ldots, x_n) \in \mathbb{Z}^n$ o valor do polinómio $p(x_1, x_2, \ldots, x_n)$. Se esse valor for zero o procedimento termina com a resposta afirmativa (a equação diofantina tem uma solução inteira).

Se o polinómio tiver uma raiz inteira, ao percorrer de "forma sistemática" os elementos de $\mathbb{Z}^n$, o procedimento encontrará a raiz e parará. Se o polinómio não tiver raízes inteiras, o procedimento não irá parar.

Traduzindo este procedimento para uma máquina de Turing, chegaremos a uma máquina que reconhece a linguagem HILB, pelo que esta é recorrentemente enumerável.

Deixamos como exercício o desenvolvimento de um algoritmo para percorrer os elementos de $\mathbb{Z}^n$ de "forma sistemática".

7.5.5 O Teorema de Rice

Vimos alguns exemplos de problemas indecidíveis. Vamos mostrar agora que muitos problemas de decisão que envolvem máquinas de Turing são indecidíveis.

Vamos provar que é indecidível toda e qualquer propriedade $\mathcal{P}$ que seja satisfeita por pelo menos uma máquina de Turing, que não seja satisfeita por todas as máquinas de Turing e que, para além disso, o facto de uma máquina de Turing, M satisfazer ou não a propriedade $\mathcal{P}$ depende apenas da linguagem $\mathcal{L}(M)$ e não das possíveis implementações (máquinas de Turing que decidem a linguagem).

Consideremos a linguagem (das codificações) de todas as máquinas de Turing deterministas sobre um mesmo alfabeto de entrada (por exemplo $\Sigma = \{0, 1\}$):

$$\mathcal{M} = \{\langle M \rangle : M \text{ é uma MT sobre o alfabeto de entrada } \Sigma\}. \tag{7.5.1}$$

Teorema 7.5.1 Teorema de Rice.

Consideremos o conjunto $\mathcal{M}$ definido por (7.5.1) e seja $\mathcal{P} \subset \mathcal{M}$ uma linguagem tal que:

1. *$\mathcal{P} \neq \emptyset$, ou seja, existe pelo menos uma máquina de Turing M tal que $\langle M \rangle \in \mathcal{P}$;*
2. *$\mathcal{P} \neq \mathcal{M}$, ou seja, existe pelo menos uma máquina de Turing M tal que $\langle M \rangle \notin \mathcal{P}$;*
3. *Dadas duas quaisquer* MT M_1 *e* M_2 *tais que* $\mathcal{L}(M_1) = \mathcal{L}(M_2)$, *ou*

$$\langle M_1 \rangle \in \mathcal{P} \text{ e } \langle M_2 \rangle \in \mathcal{P},$$

ou

$$\langle M_1 \rangle \notin \mathcal{P} \text{ e } \langle M_2 \rangle \notin \mathcal{P}.$$

Nestas condições, a linguagem $\mathcal{P}$ é indecidível.

Demonstração.

Assumindo que $\mathcal{P}$ é decidível, vamos mostrar que o problema da paragem, HALT, também o é, o que é falso, como já vimos.

Seja $M_\emptyset$ uma máquina de Turing tal que $\mathcal{L}(M_\emptyset) = \emptyset$. (Basta definir uma MT com transições do estado inicial para um estado de rejeição, qualquer que seja o símbolo de entrada.)

Caso 1. Considerando que $\langle M_\emptyset \rangle \notin \mathcal{P}$, fixemos uma máquina de Turing M_1 tal que $\langle M_1 \rangle \in \mathcal{P}$. Notamos que a máquina M_1 existe uma vez que $\mathcal{P} \neq \emptyset$.

Para cada máquina de Turing M e para cada palavra w, consideremos a máquina de Turing M_w que cumpre a seguinte especificação:

para cada palavra x

M_w simula a máquina de Turing M com a entrada w;

se M pára com w

então M_w simula M_1 com a entrada x;

se M_1 pára num estado de aceitação

então M_w pára num estado de aceitação;

senão (M_1 pára num estado de rejeição ou não pára)

M_w pára num estado de rejeição ou não pára.

Vejamos qual é a linguagem reconhecida pela máquina M_w:

- Se M pára com a entrada w, ou seja, se $\langle M, w\rangle \in$ HALT então $\mathcal{L}(M_w) = \mathcal{L}(M_1)$;
- Se M não pára com a entrada w, ou seja, se $\langle M, w\rangle \notin$ HALT então $\mathcal{L}(M_w) = \emptyset = \mathcal{L}(M_\emptyset)$.

Atendendo a que $\langle M_\emptyset\rangle \notin \mathcal{P}$, a que $\langle M_1\rangle \in \mathcal{P}$ e usando a terceira condição do Teorema de Rice, concluímos que:

- Se $\langle M, w\rangle \in$ HALT então $\langle M_w\rangle \in \mathcal{P}$;
- Se $\langle M, w\rangle \notin$ HALT então $\langle M_w\rangle \notin \mathcal{P}$.

Uma vez que estamos a assumir que a linguagem $\mathcal{P}$ é decidível, existe uma máquina de Turing $M_\mathcal{P}$ que a decide, isto é, $M_\mathcal{P}$ pára com qualquer palavra de entrada e

- Se $\langle M\rangle \in \mathcal{P}$ então $M_\mathcal{P}$ aceita $\langle M\rangle$;
- Se $\langle M\rangle \notin \mathcal{P}$ então $M_\mathcal{P}$ rejeita $\langle M\rangle$.

Usando a máquina $M_\mathcal{P}$, vamos construir uma máquina H que, dada uma qualquer palavra da forma $\langle M, w\rangle$:

constrói a descrição da máquina M_w;

simula a máquina $M_\mathcal{P}$ com a entrada $\langle M_w\rangle$;

se $M_\mathcal{P}$ pára num estado de aceitação

então H pára num estado de aceitação;

senão ($M_\mathcal{P}$ pára num estado de rejeição)

H pára num estado de rejeição.

Notamos que a máquina H pára sempre e além disso:

- Se $\langle M, w\rangle \in$ HALT então $M_\mathcal{P}$ aceita $\langle M_w\rangle$ e portanto H aceita $\langle M, w\rangle$;
- Se $\langle M, w\rangle \notin$ HALT então $M_\mathcal{P}$ rejeita $\langle M_w\rangle$ e portanto H rejeita $\langle M, w\rangle$;

Assim, H é uma máquina de Turing que decide o problema da paragem, o que é absurdo, pelo que $\mathcal{P}$ não pode ser decidível.

Caso 2. No caso em que $\langle M_\emptyset\rangle \in \mathcal{P}$, começamos por observar que a linguagem complementar $\overline{\mathcal{P}}$ também satisfaz as condições do teorema de Rice. De forma análoga ao Caso 1, provamos que $\overline{\mathcal{P}}$ é indecidível e, pelo Teorema 7.1.14, p. 200, concluímos que $\mathcal{P}$ é também indecidível.

Exemplo 7.5.2

Consideremos o problema de decidir se a palavra vazia é reconhecida por uma máquina de Turing:

$$\mathrm{ACC}_{\mathrm{MT}}(\varepsilon) = \{\langle M\rangle : \varepsilon \in \mathcal{L}(M)\}.$$

Existem máquinas de Turing que reconhecem a palavra vazia e nem todas as máquinas de Turing reconhecem a palavra vazia. Assim, as duas primeiras condições do Teorema de Rice são satisfeitas pela linguagem $\mathrm{ACC}_{\mathrm{MT}}(\varepsilon)$.

Quanto à terceira condição, se duas máquinas de Turing reconhecem a mesma linguagem então, em particular, ou ambas reconhecem a palavra vazia ou então nenhuma delas reconhece a palavra vazia.

Concluímos que a linguagem $\mathrm{ACC}_{\mathrm{MT}}(\varepsilon)$ é indecidível, uma vez que satisfaz as três condições do Teorema de Rice.

7.6 Máquinas de Turing não deterministas

No seu inovador artigo de 1936, para além das "a-machines", Turing define as "choice-machines", actualmente designadas por máquinas de Turing não deterministas.

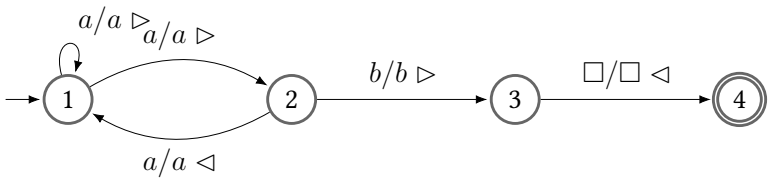
Uma máquina de Turing não determinista é definida de forma semelhante a uma máquina de Turing determinista, excepto que a função de transição tem assinatura

$$\delta : Q \times \Gamma \to \mathcal{P}\left(Q \times \Gamma \times \{\triangleleft, \triangleright\}\right)$$

Podemos ver o funcionamento de uma máquina de Turing não determinista como uma árvore que se ramifica nas diferentes possibilidades de transição. Se uma dessas ramificações conduzir a um estado de aceitação então a palavra inicialmente colocada na fita é reconhecida.

Exemplo 7.6.1 Uma máquina de Turing Não Determinista.

A figura seguinte ilustra uma máquina de Turing não determinista que reconhece a linguagem associada à expressão regular a^*ab.



O reconhecimento da palavra $w = aab$ corresponde aos possíveis caminhos numa **árvore de derivação** cuja raiz é a configuração inicial e que se ramifica de acordo com as diferentes possibilidades de transição, conforme ilustrado na Figura 7.6.2, p. 228.

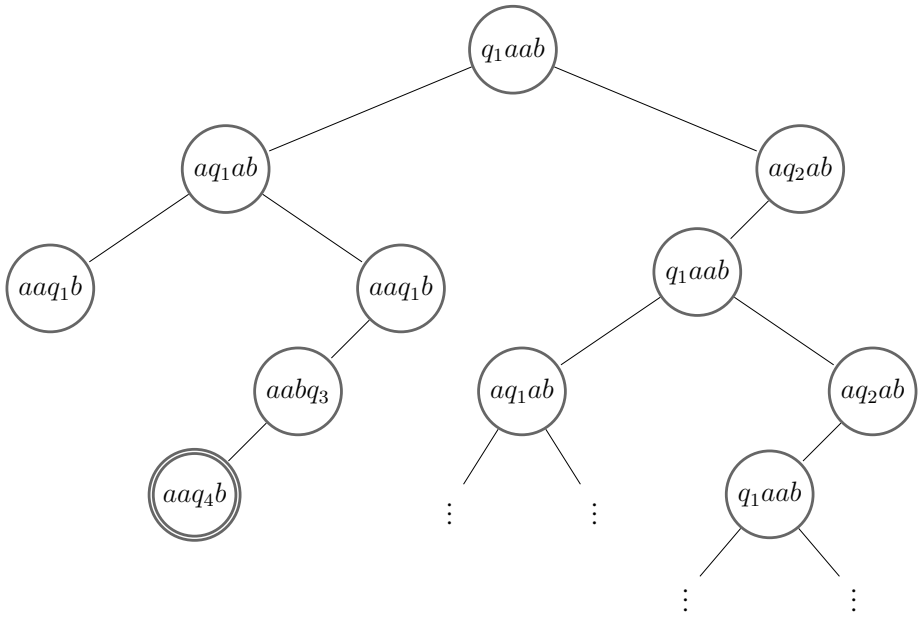


Figura 7.6.2 Árvore de derivação para a palavra aab.

Notamos que alguns dos ramos desta árvore terminam em configurações de paragem num estado de não aceitação, enquanto outros terminam em configurações de paragem no estado de aceitação. Notamos ainda que esta árvore é infinita, pois existem ramificações que correspondem à possibilidade de a máquina não parar. No entanto, uma vez que pelo menos um dos ramos conduz a uma configuração de paragem num estado de aceitação, dizemos que a palavra \(aab\) é reconhecida por esta MT não determinista.

Chegamos à mesma conclusão para qualquer palavra w que satisfaz a expressão regular $(a + b)^*ab$ e, para além disso, as palavras que não são dessa forma, não são aceites por esta máquina de Turing (a árvore de reconhecimento não contém nenhum nó que corresponda a uma configuração de paragem e aceitação).

O reconhecimento de uma linguagem por intermédio de uma máquina de Turing não determinista define-se de forma semelhante ao reconhecimento por uma máquina determinista.

Seja $M = (Q, \Sigma, \Gamma, \delta, s, F)$ uma máquina de Turing não determinista.

Uma palavra $w \in \Sigma^\star$ é reconhecida por M se existe pelo menos uma sequência

de configurações que parte da configuração inicial e conduz a uma configuração de paragem num estado de aceitação, isto é, existem $f \in F$ e $\alpha, \beta \in \Gamma^\star$ tais que $sw \vdash^\star \alpha f \beta$ e além disso $\alpha f \beta$ é uma configuração de paragem.

Notamos que uma palavra $w \in \Sigma^\star$ não é reconhecida por M quando cada um dos ramos na árvore de derivação de w ou é infinito ou, sendo finito, termina numa folha que corresponde a uma configuração de não aceitação.

Uma palavra $w \in \Sigma^\star$ é rejeitada por M se a árvore de derivação de w é finita (isto é, todos os ramos são finitos) e todas as folhas correspondem a configurações de rejeição.

Assim, a linguagem reconhecida por M, denotada por $\mathcal{L}(M)$, é constituída por todas as palavras que são reconhecidas por M.

Finalmente, uma linguagem L é decidível por M se e só se M reconhece L e rejeita a linguagem $\Sigma^\star \setminus L$.

Vamos simular uma qualquer máquina de Turing não determinista, M_N usando uma máquina de Turing determinista, M_D, com 3 fitas:

Fita 1 é uma fita só de leitura que contém a palavra de entrada w;

Fita 2 é a fita de trabalho, onde é simulada a execução de M_N em um dos ramos da árvore de derivação de w;

Fita 3 contém uma palavra que representa o actual ramo da árvore que está a ser simulado.

O conjunto de transições possíveis a partir de cada estado $q \in Q_N$ e símbolo de fita $a \in \Gamma_N$, $\delta_N(q, a)$, é sempre finito. Seja k o número máximo de transições a partir de qualquer estado, ao qual corresponde o máximo de ramificações a partir de um qualquer nó numa qualquer árvore de derivação. O alfabeto da fita 3 é $\Omega = \{1, 2, \ldots, k\} \cup \{\square\}$ e podemos usar as palavras de $v \in \Omega$, em ordem lexicográfica, para representar os caminhos nas árvores de derivação desde a raiz até um nó. Estas palavras são designadas **endereços** e a palavra vazia representa a raiz e os restantes endereços representam as sucessivas travessias em largura na árvore de derivação.

Para a máquina de Turing não determinista no Exemplo 7.6.1, p. 227, o número máximo de transições a partir de um estado é $k = 2$, pelo que as árvores de derivação são binárias. A configuração de aceitação representada na árvore da Figura 7.6.2, p. 228 corresponde ao endereço 1211 conforme ilustrado na Figura 7.6.3, p. 230.

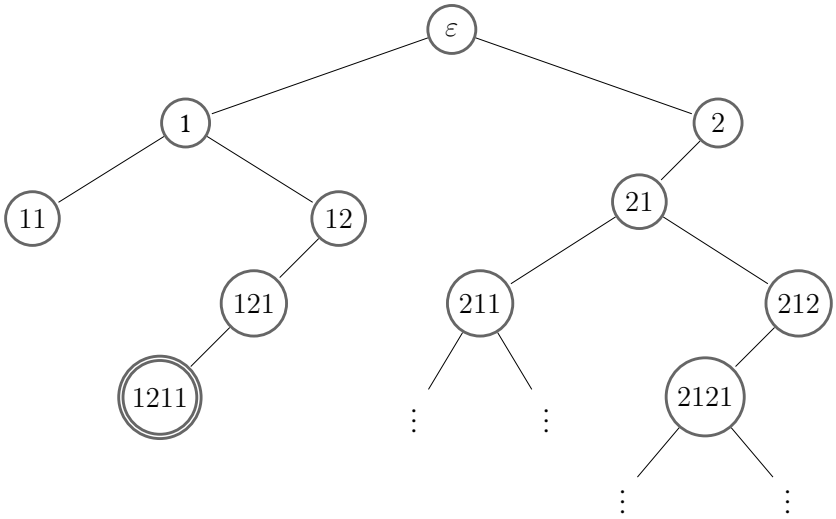


Figura 7.6.3 Travessia em largura de uma árvore de derivação.

Notamos que alguns endereços não são válidos e correspondem a configurações impossíveis, as quais devem ser ignoradas pelo simulador da máquina não determinista.

Procedimento 7.6.4 Simulador de uma MT não determinista.

Inicialmente a fita 1 contém a palavra w a reconhecer e as fitas 2 e 3 estão vazias.

Passo 1. Copiar a palavra w da fita 1 para a fita 2;

Passo 2. Usar a fita 2 para simular a máquina não determinista num ramo da árvore de derivação, consultando em cada passo o símbolo actual sob a cabeça da fita 3 para escolher a transição a realizar de entre o conjunto de transições possíveis a partir do estado actual e do símbolo actual na fita 2.

Se o símbolo actual na fita 3 for o espaço em branco (a simulação chegou a uma folha da árvore), saltar para o passo 3;

Se o símbolo corresponder a uma transição não definida, saltar para o passo 3;

Se a configuração após a transição for de rejeição, saltar para o passo 3;

Se a configuração após a transição for de aceitação, aceitar a palavra de entrada w e terminar;

Passo 3. Substituir o endereço inscrito na fita 3 pelo endereço seguinte em ordem lexicográfica;

Saltar para o passo 1.

A demonstração de que é possível simular qualquer máquina de Turing não determinista por uma determinista consiste, essencialmente, na formalização do procedimento acima indicado, tendo em conta que as máquinas de Turing deterministas com mais do que uma fita podem ser simuladas por máquinas de Turing deterministas com apenas uma fita.

Concluímos que o modelo das máquinas de Turing não deterministas é computacionalmente equivalente ao modelo das máquinas de Turing deterministas, o que permite estabelecer os seguintes resultados, cujas demonstrações deixamos ao cuidado do leitor.

Lema 7.6.5

Se L é uma linguagem reconhecida por uma máquina de Turing não determinista então é recorrentemente enumerável, isto é, existe uma máquina de Turing determinista que a reconhece.

Teorema 7.6.6

L é uma linguagem reconhecida por uma máquina de Turing não determinista se e só se é recorrentemente enumerável.

Teorema 7.6.7

L é uma linguagem decidível por uma máquina de Turing não determinista se e só se é decidível por uma máquina de Turing determinista.

Em jeito de conclusão, constatamos que a máquina de Turing determinista apresentada para simular uma máquina não determinista realiza uma pesquisa em largura pela árvore de derivação, percorrendo as sucessivas possibilidades em ordem crescente do número de passos, até encontrar uma sequência que termine numa configuração de aceitação.

Assim, o número de passos necessários para que a máquina de Turing determinista atinja uma configuração de aceitação de uma palavra é em geral exponencial no número de passos da sequência mais curta que permite aceitar a mesma palavra com a máquina de Turing não determinista.

Pensa-se que este crescimento exponencial é uma propriedade inerente às simulações de máquinas de Turing não deterministas por máquinas deterministas, o que nos remete para um universo de novas questões, em nada triviais, sobre a complexidade dos problemas/linguagens decidíveis, no que concerne ao tempo despendido

no reconhecimento das palavras (número de passos), bem como quanto ao espaço utilizado durante o reconhecimento (número de células lidas/escritas).

Munidos dos muitos conceitos, propriedades e q.b. de formalismo interiorizado ao longo destes 7 capítulos, continuaremos no seguinte com as bases da Teoria da Complexidade, as suas questões e respostas e os seus problemas em aberto, como a famosa conjectura P $\stackrel{?}{=}$ NP.

7.7 Exercícios

1. Considere o alfabeto $\Sigma = \{a, b\}$. Desenvolva máquinas de Turing para:

- **(a)** Trocar a´s por b´s numa palavra;
- **(b)** Apagar a palavra situada à direita da posição actual;
- **(c)** Apagar a palavra situada à esquerda da posição actual;
- **(d)** Apagar a palavra inscrita na fita, assumindo que a posição actual corresponde a um qualquer símbolo da palavra;
- **(e)** Apagar o símbolo de uma palavra inscrito na posição actual, deslocando para a esquerda a palavra situada à direita da posição actual;
- **(f)** Apagar o símbolo na posição actual, deslocando para a direita a palavra situada à esquerda da posição actual;
- **(g)** Inserir um símbolo branco na posição actual, deslocando para a direita a palavra situada à direita da posição actual (inclusive);
- **(h)** Inserir um símbolo branco na posição actual, deslocando para a esquerda a palavra situada à esquerda da posição actual (inclusive);
- **(i)** Duplicar uma palavra;
- **(j)** Reverter uma palavra.

2. Mostre que as seguintes linguagens são recursivas, construindo máquinas de Turing que as decidam.

- **(a)** $\{a^n b^m c^n : m \geq 2n \geq 0\}$;
- **(b)** $\{a^n b^m c^n : 0 \leq n < 2n\}$.

3. Construa máquinas de Turing para:

- **(a)** converter um número em binário num número em unário;

  Sugestão. Comece por desenvolver uma subrotina para calcular o dobro de um número em unário. Em seguida, use as propriedades $d(w0) = 2d(w)$ e $d(w1) = 2d(w) + 1$ para converter o número da

esquerda para a direita. Exemplo:

$$\begin{gathered}\overline{1}011\#1\\ \overline{1}\overline{0}11\#11\\ \overline{1}\overline{0}\overline{1}1\#11111\\ \overline{1}\overline{0}\overline{1}\overline{1}\#11111111111\end{gathered}$$

(b) converter um número em unário num número em binário;

Sugestão. Realize sucessivas divisões por 2 (marcando um em cada dois 1´s) tendo em atenção se o resto é 0 ou 1. Para simplificar, coloque o resultado à esquerda do número dado. Exemplo:

$$\begin{gathered}\#11111111111\\ 1\#X1X1X1X1X1X\\ 11\#XXX1XXX1XXX\\ 011\#XXXXXXX1XXX\\ 1011\#XXXXXXXXXXX\end{gathered}$$

(c) somar dois números em unário;

Sugestão. É um problema simples de cópia das duas sequências de 1´s;

(d) subtrair dois números em unário;

Sugestão. Basta ir copiando e marcando 1´s até que uma das sequências se esgote.

(e) somar dois números em binário;

Sugestão. Neste problema é preciso ir calculando o bit de transporte para a próxima posição, começando na direita (bits menos significativos). É também preciso ter em atenção que os números podem ter tamanhos diferentes. Por exemplo, dados $x = 1110$ e $y = 0011$, vem que $t_0 = 0$, $z_0 = x_0 + y_0 + t_0 = 0 + 1 + 0 = 1$ e $t_1 = 0$; $z_1 = x_1+y_1+t_1 = 1+1+0 = 0$ e $t_2 = 1$; $z_2 = x_2+y_2+t_2 = 1+0+1 = 0$ e $t_3 = 1$; $z_3 = x_3 + y_3 + t_3 = 1 + 0 + 1 = 0$ e $t_4 = 1$; Logo, $z = x + y = 10001$.

(f) subtrair dois números em binário;

Sugestão. Problema semelhante ao da alínea anterior. Como representar números negativos?

4. Construa máquinas de Turing para decidir as seguintes linguagens (especifique a representação escolhida, estabeleça a caracterização formal da MT, tabela e diagrama de transições):

(a) $(a+b)a(a+b)^*$

Sugestão. Saltar a primeira letra, verificar se a segunda letra é um a e parar (aceitando ou rejeitando).

(b) $\{w \in \{a,b\}^\star : w = w^{-1}\}$

Sugestão. Ir comparando a letra mais à esquerda ainda não marcada com a letra mais à direita ainda não marcada (marcando as duas). Preste atenção aos palíndromos ímpares!

(c) $\{ww^{-1} : w \in \{a,b\}^\star\}$

Sugestão. Semelhante à alínea anterior. Note que as palavras da linguagem são apenas os palíndromos pares!

(d) $\{w \in \{0,1\}^\star : \#_0(w) = \#_1(w)\}$

Sugestão. Implemente o seguinte algoritmo:

(1) Procurar um 0 não marcado;

Se não encontra então

Procurar um 1 não marcado

Se não encontra então aceita

senão rejeita;

senão marcar o 0 encontrado;

(2) Procurar um 1 não marcado;

Se não encontra então rejeita;

senão marca o 1 encontrado;

volta ao passo 1

(e) $\{a^n b^n a^n : n \geq 1\}$

Sugestão. Para cada a não marcado antes dos b´s marcar um b e marcar um a depois dos b´s. Repetir, tendo em atenção que no decorrer deste processo qualquer uma das 3 sequências se pode esgotar. Teste a máquina para as palavras ε, a, b, ab, ba, aa, $aaba$, $abba$, $abaa$.

(f) $\{a^n b^n c^n : n \geq 0\}$

Sugestão. Semelhante à alínea anterior. Tenha em atenção que a palavra vazia faz parte da linguagem!

5. Construa máquinas de Turing que cumpram as seguintes especificações:

(a) $1^n \vdash 1^{n^2}$;

(b) $1^n 0 1^m \vdash 1^n 0 1^m 0 1^{nm}$;

(c) $1^n \vdash 1^{2^n}$;

(d) $a^n b^m \vdash c^{\min(m,n)}$;

(e) $a^n b^m \vdash c^{\max(m,n)}$

(f) $1^n \vdash F(n)$, onde $F(n)$ é a sequência de Fibonnaci definida por: $F(0) = 0$, $F(1) = 1$ e $F(n) = F(n-1) \cdot F(n-2)$, para $n \geq 2$, onde $\cdot$ denota a operação de concatenação de palavras binárias.

6. Averigue se a classe das linguagens recorrentes (recorrentemente enumeráveis) é fechada para:

(a) a concatenação;

(b) a operação estrela de Kleene;

Sugestão. Resolução semelhante à da alínea anterior, considerando agora todas as possíveis partições de uma palavra w em n partes ($n = 1, 2, \ldots, |w|$).

(c) imagens direta e inversa por intermédio de um homomorfismo.

Sugestão. A classe das linguagens recorrentemente enumeráveis é fechada para homomorfismos e para a imagem inversa por intermédio de um homomorfismo. No entanto, a classe das linguagens recorrentes não é fechada para homomorfismos (embora seja fechada para a imagem inversa por intermédio de um homomorfismo).

7. Considere linguagens recorrentemente enumeráveis $L_1, L_2, \ldots, L_n$, com $n \geq 2$, definidas sobre o mesmo alfabeto Σ, as quais formam uma partição de $\Sigma^\star$ (são disjuntas duas a duas e a sua união é $\Sigma^\star$). Nestas condições, mostre que são todas recorrentes.

8. Uma MT entra em ciclo com uma palavra w se a sequência de configurações associada à palavra w tem uma configuração repetida (e portanto um número infinito de repetições dessa configuração).

Seja $L \subseteq \Sigma^\star$ uma linguagem reconhecida por uma máquina de Turing M tal que para qualquer palavra $w \in \Sigma^\star$ a máquina M ou aceita w ou rejeita w ou entra ciclo com w. Mostre que L é decidível.

Sugestão. Basta construir uma MT que detecte configurações repetidas,

com o auxílio de uma MT com duas fitas: uma fita principal usada para simular a execução da máquina original com uma palavra e uma fita auxiliar para guardar as sucessivas configurações.

No final de cada passo, o simulador acrescenta a configuração à fita auxiliar e verifica, em seguida, se esta contém duas configurações repetidas. Em caso afirmativo, pára e rejeita a palavra.

9. Mostre que as linguagens seguintes são decidíveis.

(a) $\{\langle A\rangle : A \text{ é um AFD e } \mathcal{L}(A) \text{ é infinita}\}$.

Sugestão. Como determinar se um estado de aceitação é atingível a partir do estado inicial? Como determinar se existe um ciclo que começa e termina num estado de aceitação?

(b) Sendo $L = \{w \in \{a,b\}^\star : \#_a(w) \text{ é ímpar}\}$,

$$\{\langle A\rangle : A \text{ é um AFD sobre } \{a,b\} \text{ e } \mathcal{L}(A) \cap L = \emptyset\}.$$

Sugestão. A linguagem L é regular. Logo, também $\mathcal{L}(A) \cap L$ é regular ....

(c) Considerando o alfabeto binário, $\Sigma = \{0,1\}$,

$$\{\langle G\rangle : G \text{ é uma GIC e } \{1\}^\star \cap \mathcal{L}(G) \neq \emptyset\}.$$

Sugestão. Como é que se constrói uma GIC geradora da intersecção de uma linguagem regular com uma GIC?

10. Desenvolva um procedimento que, dado $n \in \mathbb{N}$, permita enumerar os elementos de $\mathbb{Z}^n$ de uma forma sistemática. Por exemplo, um procedimento que permita listar, para $k = 0, 1, 2, \ldots$, todos os elementos do hipercubo $H_k \subset \mathbb{Z}^n$ de centro na origem e com lados de comprimento $2k$.

Sugestão. Investigue o método combinatório das estrelas e barras ("stars and bars").

11. Mostre que o problema de aceitação

$$\text{ACC}_{\text{MT}} = \{\langle M, w\rangle : M \text{ é uma máquina de Turing que aceita } w\}$$

é indecidível.

12. Por redução do problema ACC_{MT} mostre que os seguintes problemas são indecidíveis:

(a) $\text{E}_{\text{MT}} = \{\langle M\rangle : M \text{ é uma máquina de Turing e } \mathcal{L}(M) = \emptyset\}$;

(b) $\text{REG}_{\text{MT}} = \{\langle M\rangle : M \text{ é uma máquina de Turing e } \mathcal{L}(M) \text{ é regular}\}$;

(c) $\text{FIN}_{\text{MT}} = \{\langle M\rangle : M \text{ é uma máquina de Turing e } \mathcal{L}(M) \text{ é finita}\}$;

(d) $\text{ALL}_{\text{MT}} = \{\langle M\rangle : M \text{ é uma máquina de Turing e } \mathcal{L}(M) = \Sigma^\star\}$.

13. Por redução do problema E_{MT} mostre que

$$\text{EQ}_{\text{MT}} = \{\langle M_1, M_2\rangle : M_1 \text{ e } M_2 \text{ são MT e } \mathcal{L}(M_1) = \mathcal{L}(M_2)\}$$

é indecidível.

14. Mostre que a linguagem $\overline{\text{HALT}}$ não é recorrentemente enumerável.

Sugestão. Basta argumentar de forma análoga à usada na Secção 7.5, p. 219 para demonstrar que $\overline{\text{ACC}_{\text{MT}}}$ não é recorrentemente enumerável.

15. Considere o problema

$$\text{EQ}_{\text{MT}} = \{\langle M_1, M_2\rangle : M_1 \text{ e } M_2 \text{ são MT e } \mathcal{L}(M_1) = \mathcal{L}(M_2)\}.$$

(a) Mostre que EQ_{MT} não é recorrentemente enumerável.

(b) Mostre que $\overline{\text{EQ}_{\text{MT}}}$ não é recorrentemente enumerável.

16. Aplique o Teorema de Rice para provar que as linguagens referidas no Exercício 7.7.12, p. 236 não são recorrentes.

17. Considere a linguagem $L = \{ww : w \in \{a, b\}^\star\}$.

Este exercício é de longa resolução. Teste o funcionamento das máquinas de Turing que desenhar com palavras pequenas, por exemplo, ε, a, aa, ab, $abab$, $abba$, ...

(a) Construa uma máquina de Turing não determinista, M_N que reconheça L, mas que não decida L.

Sugestão. Desenhe uma MT que: (i) de forma não determinista encontre o meio da palavra colocada na fita, marcando a primeira parte da palavra; (ii) verifique se a primeira parte (marcada) é igual à segunda parte (não marcada). Para além disso, como queremos que a máquina não decida a linguagem, esta não deverá parar para pelo menos uma palavra que não seja da forma ww.

(b) Esboce as árvores de derivação pela máquina M_N associadas às palavras $abbabb$ e $abbaba$.

Sugestão. Uma vez que as árvores de derivação não vão ser finitas, concentre-se apenas nos ramos que conduzem à aceitação / rejeição das palavras.

(c) Construa uma máquina de Turing não determinista que decida L.

Sugestão. Basta modificar a máquina anterior tratando os casos em

que não pára, de modo a que, nesses casos, páre num estado de rejeição.

(d) Construa uma máquina de Turing determinista que decida L.

Sugestão. Desenhe uma MT que: (i) calcule de forma determinista o meio da palavra, por exemplo, transformando

$$w = s_1 s_2 \cdots s_n s_{n+1} \cdots s_{2n-2} s_{2n-1} s_{2n}$$

em

$$w = \bar{s}_1 \bar{s}_2 \cdots \bar{s}_n \bar{\bar{s}}_{n+1} \cdots \bar{\bar{s}}_{2n-2} \bar{\bar{s}}_{2n-1} \bar{\bar{s}}_{2n};$$

(ii) verifique se a primeira parte é igual à segunda parte.