Teoria da Computação

Dep. Matemática – Instituto Superior Técnico

Carlos Caleiro
F. Miguel Dionísio
Paula Gouveia
Jaime Ramos
João Rasga

Março 2023

Índice

Prelúdio

Texto de apoio à disciplina de Teoria da Computação da Licenciatura em Engenharia Informática e de Computadores do IST, em complemento da bibliografia recomendada:

M. Sipser, *Introduction to the Theory of Computation, Second Edition.* Thomson Course Technology, 2006.

A todos os interessados numa introdução à teoria da computabilidade e da complexidade computacional.

Estrutura do texto

Introdução à Teoria das Linguagens, à Teoria da Computabilidade, e depois à Teoria da Complexidade, dos modelos computacionais mais simples, como os autómatos, até às máquinas de Turing. Exercícios seleccionados resolvidos.

Emulador

Para acompanhar os alunos na melhor compreensão das matérias será útil utilizar um emulador de autómatos finitos (por exemplo, o que é disponibilizado em `https://automatonsimulator.com`), e principalmente um emulador de máquinas de Turing, nomeadamente o que está disponível na página da disciplina.

Agradecimentos

- Manuel Biscaia Martins
- Guilherme Ramos

1

Preliminares

"Without mathematics one wanders endlessly in a dark labyrinth."

Galileo Galilei, *Il Saggiatore*, 1623

1.1 Conjuntos, funções e cardinalidade

Um conjunto é uma colecção de elementos, sem repetições nem ordem relativa. Como é usual, usamos $a \in A$ para denotar que um elemento *a pertence* a um conjunto A, e $A \subseteq B$ para indicar que todos os elementos do conjunto A estão também no conjunto B, isto é, A é *subconjunto* de B. O conjunto vazio, que denotamos por $\emptyset$ é particularmente interessante, pois $a \notin \emptyset$ qualquer que seja a, pelo que $\emptyset \subseteq B$ qualquer que seja B. Obviamente, também, $A \subseteq A$, pelo que usamos $A \subsetneq B$ quando a inclusão é *estrita*, isto é, existe $b \in B$ tal que $b \notin A$.

Adoptamos a notação usual para conjuntos de números:

- naturais $\mathbb{N} = \{1, 2, 3, 4, \dots\}$, ou $\mathbb{N}_0 = \{0, 1, 2, 3, 4, \dots\}$;
- inteiros $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$;
- racionais $\mathbb{Q}$;
- reais $\mathbb{R}$, reais não negativos $\mathbb{R}_0^+$, ou reais positivos $\mathbb{R}^+$.

Usamos $\wp(A)$ para representar o conjunto das *partes* de A, isto é, o conjunto de todos os subconjuntos de A. Por exemplo, o conjunto $\{1, 2\}$ tem exactamente 4 subconjuntos, $\wp(\{1, 2\}) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$.

Usamos $A \cap B$ para representar a *intersecção* dos conjuntos A e B, isto é, o conjunto dos elementos que pertencem a ambos. Analogamente, usamos $A \cup B$ para representar a *união* dos conjuntos A e B, isto é, o conjunto dos elementos que pertencem a pelo menos um dos dois conjuntos. Como se espera, podemos intersectar ou unir mais do que dois conjuntos. Como é usual, também usamos a *complementação*, sendo que $A \setminus B$ denota o subconjunto de A que contém todos os seus elementos que não pertencem a B.

Consideramos ainda o *produto cartesiano* de conjuntos $A \times B = \{(a, b) : a \in A, b \in B\}$, que se estende de maneira óbvia a mais conjuntos, resultando em triplos, quádruplos, e vectores com mais posições, em vez de apenas pares. Usamos A^k para denotar o produto cartesiano de A por si próprio $k \in \mathbb{N}$ vezes. Obviamente, $A^1 = A$.

Denotamos por $f : A \to B$ uma *função* do conjunto A para o conjunto B. Uma função pode ser *parcial* e estar indefinida para alguns elementos de A. Denotamos por $\mathrm{dom}(f) \subseteq A$ o *domínio* de definição de f. A função f associa a cada $a \in \mathrm{dom}(f)$ um elemento $f(a) \in B$. O *contra-domínio* de f é conjunto $\mathrm{cod}(f) = \{f(a) : a \in \mathrm{dom}(f)\}$. Quando $\mathrm{dom}(f) = A$ dizemos que f é uma função *total*, ou simplesmente uma *aplicação*.

Se $f : A \to B$ e $g : B \to C$ são funções então define-se a função composta $(g \circ f) : A \to C$ tal que $(g \circ f)(a) = g(f(a))$ se $a \in \mathrm{dom}(f)$ e $f(a) \in \mathrm{dom}(g)$.

Uma função $f : A \to B$ diz-se *injectiva* se associa diferentes elementos de B a todos os elementos de A, ou seja, f é total e se $f(a_1) = f(a_2)$ para $a_1, a_2 \in A$ então $a_1 = a_2$. Uma função $f : A \to B$ diz-se *sobrejectiva* se $\mathrm{cod}(f) = B$. Uma função $f : A \to B$ diz-se *bijectiva* se é injectiva e sobrejectiva.

Se $f : A \to B$ é injectiva, pode definir-se uma função *inversa* $f^{-1} : B \to A$ tal que $\mathrm{dom}(f^{-1}) = \mathrm{cod}(f)$ e, para cada $b \in \mathrm{dom}(f^{-1})$, $f^{-1}(b)$ é o único elemento $a \in A$ tal que $f(a) = b$. Obviamente, $(f^{-1} \circ f)(a) = a$ para qualquer $a \in A$.

Uma *relação* é um subconjunto R de $A \times B$, ou em geral de um produto cartesiano. Para um conjunto A e $k \in \mathbb{N}$, $R \subseteq A^k$ diz-se uma *relação k-ária, ou de aridade k*, sobre A.

Usamos $\#A$ para denotar o *cardinal* do conjunto A, ou seja, o *tamanho* do conjunto. Para conjuntos finitos, o cardinal é como usual um número em $\mathbb{N}_0$. No entanto, para conjuntos infinitos, as coisas são mais complicadas. Sem entrar na discussão de assuntos não triviais da matemática moderna, como a *hipótese do contínuo*, definimos que $\#A \leq \#B$ se existe uma função injectiva $f : A \to B$. Dizemos que $\#A = \#B$ se $\#A \leq \#B$ e $\#B \leq \#A$. Escrevemos ainda $\#A < \#B$ se $\#A \leq \#B$ mas $\#A \neq \#B$.

Um conjunto A diz-se *contável* se $\#A \leq \#\mathbb{N}$. Um conjunto A diz-se *infinito* se $\#\mathbb{N} \leq \#A$. Um conjunto A diz-se *numerável* se é infinito e contável, ou seja, $\#A = \#\mathbb{N}$.

Proposição 1.1.
Um conjunto A é numerável se e só se existe uma função $h : A \to \mathbb{N}$ bijectiva.

Dem.: Se o conjunto A é contável e infinito, temos funções injectivas $f : A \to \mathbb{N}$ e $g : \mathbb{N} \to A$. Considere-se a função $h : A \to \mathbb{N}$ definido por $h(a) = \#\{a' \in A : f(a') \leq f(a)\}$ e verifique-se que é bijectiva.

Se temos uma bijecção $h : A \to \mathbb{N}$, obviamente que h é injectiva e portanto A é contável. Por outro lado, h^{-1} é total, já que h é sobrejectiva, e $h^{-1}(n_1) = h^{-1}(n_2) = a$ implica que $h(a) = n_1 = n_2$, concluindo-se que h^{-1} é injectiva e portanto A é infinito. □

Há muitos conjuntos interessantes que são numeráveis, desde logo $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Z}$, $\mathbb{Q}$.

No entanto, também há conjuntos infinitos com cardinalidades maiores. É o caso dos números reais, ou seja, $\#\mathbb{N} < \#\mathbb{R}$.

1.2 Alfabetos e linguagens

Definição 1.2. ALFABETO, PALAVRA, LINGUAGEM
Um *alfabeto* Σ é um conjunto finito e não-vazio (de *símbolos*).
Uma *palavra* sobre Σ é uma sequência finita de elementos do alfabeto. Denotamos por Σ^* o conjunto de todas as palavras sobre Σ (*fecho de Kleene*).
Uma *linguagem* sobre Σ é um subconjunto de Σ^*. Se L é uma linguagem sobre Σ, a linguagem *complementar* de L é $\overline{L} = \Sigma^* \setminus L$. Denotamos por $\mathcal{L}^\Sigma$ o conjunto de todas as linguagens sobre Σ. △

Por exemplo, se $\Sigma = \{0, 1\}$ então Σ^* contém 4 palavras de comprimento 2: as palavras 00, 01, 10 e 11. O conjunto $\{0, 1\}^*$ é um conjunto infinito, que contém exactamente 2^n palavras de comprimento n para cada $n \in \mathbb{N}_0$. Em particular, contém uma palavra de comprimento 0, que não tem nenhum símbolo do alfabeto, dita a *palavra vazia*. Como não podemos visualizá-la, usaremos ϵ para nos referirmos à palavra vazia. Note-se que $\epsilon \in \Sigma^*$ para qualquer alfabeto Σ, pois a palavra vazia é a única que podemos construir sem usar qualquer símbolo.

Usamos w^R para denotar a palavra w invertida. Por exemplo, se $w = 1011$, então $w^R = 1101$.

Proposição 1.3.
Se Σ é um alfabeto então $\#\Sigma^* = \#\mathbb{N}$.

Dem.: Exercício. □

Usamos . para concatenar palavras. Obviamente, $w.\epsilon = \epsilon.w = w$. Por exemplo, 000.111 representa a palavra 000111. Deste modo, podemos escrever $0.w$ para denotar uma palavra que começa por 0, ou mesmo $0.w.11$ para denotar uma palavra que começa por 0 e termina com 11. Dizemos que u é um prefixo de $u.v$, e analogamente que v é um seu sufixo. Omitiremos a operação de concatenação . sempre que não traga ambiguidade.

Usamos $|w|$ para representar o comprimento de uma palavra w. Claramente, $|010| = 3$, $|\epsilon| = 0$ e $|uv| = |u| + |v|$. Dado $n \in \mathbb{N}$, se $n \leq |w|$, usamos também w_n para representar o n-ésimo símbolo da palavra w. Assim, qualquer palavra w é tal que $w = w_1 w_2 \dots w_{|w|}$. Por exemplo, se $w = 010$, tem-se $w_1 = 0$, $w_2 = 1$ e $w_3 = 0$.

Dados $w \in \Sigma^*$ e $n \in \mathbb{N}_0$, denotamos por w^n a palavra em $\{w\}^*$ que tem comprimento $n \times |w|$. Por exemplo, $w^0 = \epsilon$ é a palavra vazia, $w^1 = w$, e w^2 é a palavra ww.

O conjunto de todas as palavras sobre $\{0, 1\}$ que contêm o mesmo número de 0s e de 1s é uma linguagem. Claramente, 100110 ou ϵ pertencem à linguagem, mas 111110 ou 010 não pertencem.

Noutro extremo, podemos entender a língua portuguesa, por exemplo, como uma linguagem sobre o alfabeto formado pelos caracteres de A a Z, maiúsculos

e minúsculos, acentuados ou não, com ou sem cedilhas, juntamente com o hífen e outros sinais de pontuação.

Dadas as linguagens $L_1, L_2 \in \mathcal{L}^\Sigma$, definimos a *concatenação* das linguagens como sendo a linguagem $L_1.L_2 = \{uv : u \in L_1, v \in L_2\}$. Definimos também o *fecho de Kleene* de uma linguagem L como sendo a linguagem $L^* = \{u_1u_2 \dots u_n : n \in \mathbb{N}_0 \text{ e } u_1, \dots, u_n \in L\}$.

Atentemos agora na cardinalidade de $\mathcal{L}^\Sigma$.

Proposição 1.4.
Se Σ é um alfabeto então $\#\mathcal{L}^\Sigma > \#\mathbb{N}$.

Dem.: Suponhamos, por absurdo, que $\#\mathcal{L}^\Sigma \leq \#\mathbb{N}$. Facilmente se verifica que $\mathcal{L}^\Sigma$ é infinito: se $a \in \Sigma$ então $\{a\}$, $\{aa\}$, $\{aaa\}$, ... são todas linguagens distintas e, portanto, a função $f : \mathbb{N} \to \mathcal{L}^\Sigma$ tal que $f(n) = \{a^n\}$ para cada $n \in \mathbb{N}$ é injectiva. O conjunto $\#\mathcal{L}^\Sigma$ seria assim numerável, e existiria uma função $L : \mathbb{N} \to \mathcal{L}^\Sigma$ bijectiva.

Existe também uma função $w : \mathbb{N} \to \Sigma^*$ bijectiva porque sabemos, pela Proposição 1.3, que Σ^* é numerável.

A argumentação seguinte é usualmente conhecida como *diagonalização*.

Poderíamos então considerar a linguagem $A \subseteq \Sigma^*$ tal que, para cada $i \in \mathbb{N}$, $w(i) \in A$ se e só se $w(i) \notin L(i)$. É fácil verificar que $A \neq L(i)$, qualquer que seja $i \in \mathbb{N}$. Nomeadamente, a palavra $w(i)$ distingue as duas linguagens pois pertence a A apenas e só se não pertence $L(i)$. No entanto, $A \in \mathcal{L}^\Sigma$, o que contradiz a sobrejectividade de L. □

Analogamente, definimos $\mathcal{F}^\Sigma$ como o conjunto de todas as funções $f : \Sigma^* \to \Sigma^*$.

Proposição 1.5.
Se Σ é um alfabeto então $\#\mathcal{F}^\Sigma > \#\mathbb{N}$.

Dem.: Semelhante, exercício. □

Podemos usar alfabetos muito simples para representar objectos arbitrários. Por exemplo, podemos representar números naturais usando várias notações:

unária:	$\epsilon, 1, 11, 111, 1111, \dots$
binária:	$0, 1, 10, 11, 100, 101, 110, 111, 1000, \dots$
decimal:	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, \dots$

Podemos também representar outros objectos. Por exemplo, podemos usar 1110100010010001001100 para representar um grafo (orientado) circular com 3 nós (ver Figura 1.1).

Separemos a sequência: 111.0.100.010.010.001.001.100. A sequência inicial de 1s indica-nos o número de nós do grafo (3 nós). O primeiro zero serve apenas como delimitador. A seguir temos várias sequências de comprimento 3 (o número de nós do grafo); 100 e 010 representam a aresta do nó 1 para o nó 2, depois 010 e 001 representam a aresta do nó 2 para o nó 3, e finalmente 001 e 100 representam a aresta do nó 3 para o nó 1. Há obviamente outras formas de representar grafos, até mais sucintas, mas adoptaremos esta representação e chamá-la-emos de *canónica*.

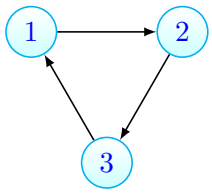


Figura 1.1: Grafo circular com 3 nós.

Portanto, à partida, uma palavra não tem um significado determinado, pois é apenas uma sequência de símbolos. O significado da palavra emergirá apenas da forma como for usada, em cada contexto. Neste sentido, não temos *a priori* como dizer se a palavra 10 representa a quantidade 'dez' em notação decimal, a quantidade 'dois' em notação binária, ou um grafo com apenas um nó e sem arestas.

Observe-se ainda que alfabetos com 2 símbolos (por exemplo $\{0,1\}$, que será de certa forma o nosso alfabeto fetiche) são suficientes, em geral, para representar linguagens sobre alfabetos mais sofisticados. Dado um alfabeto Σ com um máximo de 2^k símbolos, é óbvio que podemos usar sequências binárias distintas, de comprimento k, para representar cada um dos símbolos de Σ.

1.3 Demonstrações: indução, redução ao absurdo

Seja A um conjunto. Uma *propriedade* sobre A é uma aplicação $\mathcal{P} : A \to \{0,1\}$, em que $\mathcal{P}(a) = 1$ significa que a propriedade é verdadeira para $a \in A$, e $\mathcal{P}(a) = 0$ significa que a propriedade é falsa para $a \in A$. Diz-se que a propriedade *se verifica* em A, ou que é *verdadeira* em A, se for verdadeira para todos os elementos de A.

Exemplo 1.6.
Considere-se a propriedade $\mathcal{P}_1$ sobre $\mathbb{N}$ tal que

$$\mathcal{P}_1(n) = \begin{cases} 1 & \text{se } n \text{ é par} \\ 0 & \text{caso contrário} \end{cases}$$

Não é muito difícil concluir que a propriedade é verdadeira para alguns valores de $\mathbb{N}$ mas não para todos. △

Exemplo 1.7.
Considere-se a propriedade $\mathcal{P}_2$ sobre $\mathbb{N}$ tal que

$$\mathcal{P}_2(n) = \begin{cases} 1 & \text{se } n^2 = (n-1)^2 + 2n - 1 \\ 0 & \text{caso contrário} \end{cases}$$

Neste caso, a propriedade é verdadeira para todos os valores de $\mathbb{N}$. △

Exemplo 1.8.
Considere-se a propriedade $\mathcal{P}_3$ sobre grafos tal que

$$\mathcal{P}_3(g) = \begin{cases} 1 & \text{se a soma dos graus de todos os vértices de } g \text{ é um número par} \\ 0 & \text{caso contrário} \end{cases}$$

Esta propriedade é verdadeira para todos os grafos. △

Por uma questão de conveniência, é usual escrever a propriedade apenas para o caso verdadeiro. Por exemplo, no caso do Exemplo 1.8, a propriedade pode ser apresentada como "para qualquer grafo, a soma dos graus dos seus vértices é um número par".

Uma *demonstração* é uma justificação de que uma propriedade se verifica, isto é, que é verdadeira em todos os pontos do domínio. Existem diferentes técnicas para demonstrar uma propriedade.

Uma demonstração pode ser *directa*. Neste caso, a demonstração consiste numa sequência de passos devidamente justificados que permitem justificar a veracidade da propriedade.

Exemplo 1.9.
Seja U um conjunto. A seguinte propriedade é sempre verdadeira:

$$\overline{A \cup B} = \overline{A} \cap \overline{B}, \text{ quaisquer que sejam os conjuntos } A, B \subseteq U$$

em que $\overline{X}$ denota o conjunto $U \setminus X$. Deixa-se como exercício escrever a aplicação correspondente a esta propriedade.

Para demonstrar a propriedade, basta demonstrar que: (i) $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$; (ii) $\overline{A \cup B} \supseteq \overline{A} \cap \overline{B}$. De (i) e (ii) conclui-se imediatamente que os conjuntos $\overline{A \cup B}$ e $\overline{A} \cap \overline{B}$ têm de ser iguais. A demonstração de (i) e (ii) apresentada de seguida recorre apenas aos conceitos de teoria dos conjuntos anteriormente referidos.

Para demonstrar (i), seja x um elemento de $\overline{A \cup B}$. Usando a definição de conjunto complementar, sabe-se que x é um elemento de U mas que x não pertence a $A \cup B$. Se x não pertence a $A \cup B$ então, por definição de união de conjuntos, x não pertence a A nem a B. Recorrendo novamente à definição de conjunto complementar, conclui-se que x pertence a $\overline{A}$ e também a $\overline{B}$. Logo, por definição de intersecção de conjuntos, x também pertence a $\overline{A} \cap \overline{B}$. Então, todo o elemento de $\overline{A \cup B}$ é também elemento de $\overline{A} \cap \overline{B}$, ou seja, a propriedade (i) é verdadeira.

Deixa-se como exercício a demonstração da propriedade (ii). △

No exemplo anterior apresenta-se uma demonstração directa de uma propriedade. Cada um dos passos foi justificado rigorosamente, neste caso com base em definições de teoria de conjuntos. Mas alguns dos passos poderiam ter sido justificados por propriedades estabelecidas anteriormente.

Uma outra técnica de demonstração é a demonstração por *redução ao absurdo*, ou por *contradição*. Neste caso, assume-se que a propriedade que se pretende demonstrar é falsa e tenta-se chegar a uma *contradição*.

Exemplo 1.10.
Seja U um conjunto. A propriedade seguinte é sempre verdadeira:

$$A \subseteq A \cup B, \text{ quaisquer que sejam } A, B \subseteq U.$$

Suponha-se que a propriedade é falsa. Então, por definição de subconjunto, existe um elemento x que pertence a A e que não pertence a $A \cup B$. Mas se x

não pertence a $A \cup B$, por definição de união de conjuntos, x não pertence a A nem a B. Neste momento, chegou-se a uma contradição porque, por um lado x pertence a A, mas por outro lado, x não pertence a A. Então, a hipótese inicial de que A não está contido em $A \cup B$ é falsa, ou seja, a propriedade em causa é verdadeira. △

Uma demonstração pode ainda ser feita por *indução*. Neste caso, no entanto, é necessário que o conjunto sobre o qual a propriedade está definida tenha uma propriedade especial: o conjunto tem de ser *indutivo*. No que se segue, considera-se um conjunto indutivo particular, o conjunto dos números naturais. Obviamente podemos considerar outros conjuntos indutivos mas tal objectivo cai fora do âmbito deste texto.

Seja então $\mathcal{P}$ uma propriedade sobre o conjunto dos números naturais, $\mathbb{N}_0$. Pretende-se demonstrar que $\mathcal{P}(n) = 1$ para todo o $n \in \mathbb{N}_0$, isto é, pretende-se demonstrar que $\mathcal{P}(0) = 1$, $\mathcal{P}(1) = 1$, $\mathcal{P}(2) = 1$, ... Uma demonstração por indução é constituída por duas partes: a *base de indução* e o *passo de indução*. A base de indução consiste em demonstrar que a propriedade é verdadeira para 0, isto é, que $\mathcal{P}(0)$ é verdadeira. O passo de indução consiste em demonstrar que, qualquer que seja $k \in \mathbb{N}_0$, se $\mathcal{P}(k)$ é verdadeira então $\mathcal{P}(k+1)$ também é verdadeira. A $\mathcal{P}(k)$ chama-se a *hipótese de indução* e a $\mathcal{P}(k+1)$ chama-se a *tese de indução*. É fácil concluir que se se demonstrar a base de indução e o passo de indução para uma determinada propriedade então essa propriedade é verdadeira para todos os números naturais $\mathbb{N}_0$. A propriedade é verdadeira para 0 por que tal foi demonstrado na base de indução. O passo de indução permite concluir que se a propriedade é verdadeira para 0 então também é verdadeira para 1. E, se é verdadeira para 1, então também é verdadeira para 2. E assim sucessivamente para todos os números naturais.

Exemplo 1.11.
Considere-se a propriedade $\mathcal{P}(n)$ seguinte que permite calcular a soma dos termos de uma progressão aritmética:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

Demonstra-se, por indução, que esta propriedade é verdadeira para todos os números naturais. A estrutura de uma demonstração por indução é sempre a mesma.
Base de indução: $\mathcal{P}(0)$ é verdadeira.

Neste caso, é necessário demonstrar que $\sum_{i=0}^{0} i = \frac{0(0+1)}{2}$ é uma asserção verdadeira. Mas $\sum_{i=0}^{0} i = 0$ e $\frac{0(0+1)}{2} = 0$ logo a asserção verifica-se trivialmente.

De seguida, demonstra-se o passo de indução: assume-se que a hipótese de indução é verdadeira e demonstra-se a veracidade da tese de indução.

Hipótese de indução: $\mathcal{P}(k)$ é verdadeira, isto é, $\sum_{i=0}^{k} i = \frac{k(k+1)}{2}$.

Tese de indução: $\mathcal{P}(k+1)$ é verdadeira, isto é, $\sum_{i=0}^{k+1} i = \frac{(k+1)(k+2)}{2}$.

A demonstração da tese de indução é simples de obter.

$$\begin{aligned}\sum_{i=0}^{k+1} i &= \left(\sum_{i=0}^{k} i\right) + (k+1) \\ &= \frac{k(k+1)}{2} + (k+1) \qquad (\dagger) \\ &= \frac{k(k+1) + 2(k+1)}{2} \\ &= \frac{(k+1)(k+2)}{2}\end{aligned}$$

O passo (†) é justificado por pela hipótese de indução. △

O método anterior também pode ser aplicado a propriedades definidas sobre o conjunto $\mathbb{N}$. Neste caso, apenas há que perceber que a base de indução consiste em estabelecer a veracidade de $\mathcal{P}(1)$. Aliás, dado um número natural b e denotando por $\mathbb{N}_b$ o conjunto de todos os números naturais maiores do que ou iguais a b, i.e, $\mathbb{N}_b = \{k \in \mathbb{N} : k \geq b\}$, é possível demonstrar por indução a veracidade de uma propriedade definida sobre $\mathbb{N}_b$. Tal como no caso $\mathbb{N}$, a única coisa a fazer é considerar como base de indução $\mathcal{P}(b)$.

1.4 Notação assintótica

O comportamento assintótico de funções pode ser expresso recorrendo à notação de Bachmann-Landau.

Definição 1.12. NOTAÇÃO ASSINTÓTICA
Dadas funções totais $f, g : \mathbb{N} \to \mathbb{R}_0^+$ diz-se que:

- $f = \mathcal{O}(g)$ se existem $c \in \mathbb{R}^+$ e $m \in \mathbb{N}$ tais que $f(n) \leq cg(n)$ para todo o natural $n > m$ (f diz-se *assintoticamente menor ou igual a* g, ou *dominada por* g);
- $f = \Theta(g)$ se $f = \mathcal{O}(g)$ e $g = \mathcal{O}(f)$ (f diz-se *assintoticamente igual a* g, ou *da ordem de* g);
- $f = o(g)$ se qualquer que seja a constante real $c \in \mathbb{R}^+$ existe $m \in \mathbb{N}$ tal que $f(n) \leq cg(n)$ para todo o natural $n > m$ (f diz-se *assintoticamente menor do que* g, ou *desprezável perante* g). △

Obviamente, $f = o(g)$ implica $f = \mathcal{O}(g)$.

Notação
A utilização da notação de Bachmann-Landau (*big-Oh notation*) não é uniforme. Por exemplo, há autores que escrevem $f \in \mathcal{O}(g)$ em vez de $f = \mathcal{O}(g)$ considerando que $\mathcal{O}(g)$ é uma classe de funções. É também usual ver esta definição apresentada para a avaliação da função em n, isto é, escrever $f(n) = \mathcal{O}(g(n))$ em vez de $f = \mathcal{O}(g)$. No entanto, é importante realçar que é a função f que é assintoticamente menor ou igual à função g e não a avaliação de f em n que é

assintoticamente menor ou igual à avaliação de g em n. Isto não impede que, depois de apresentada a definição formal do conceito, se apresentem convenções que facilitem s sua utilização. No que se segue, vamos muitas vezes confundir funções com a sua avaliação em n e escrever $n^2+1 = \mathcal{O}(n^2)$, em vez de $f = \mathcal{O}(g)$ para funções f e g tais que $f(x) = x^2 + 1$ e $g(x) = x^2$.

A proposição seguinte permite obter uma caracterização dos conceitos anteriores usando a noção de limite.

Proposição 1.13.
Sejam $f, g : \mathbb{N} \to \mathbb{R}_0^+$.

1. *Se* $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = r \in \mathbb{R}_0^+$ *então* $f = \mathcal{O}(g)$.

2. *Se* $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = r \in \mathbb{R}^+$ *então* $f = \Theta(g)$.

3. $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$ *se e só se* $f = o(g)$.

A demonstração desta proposição é deixada como exercício.

Quando $f = \mathcal{O}(g)$ há funções g para as quais é usual introduzir terminologia específica. Quando $f = \mathcal{O}(1)$ dizemos que f é *constante*. Quando $f = \mathcal{O}(\log n)$ dizemos que f é *logarítmica*. Quando $f = \mathcal{O}(n^2)$, dizemos que f é *quadrática*. Quando $f = \mathcal{O}(n^3)$, dizemos que f é *cúbica*. Em geral, quando $f = \mathcal{O}(n^c)$, para alguma constante $c > 1$, dizemos que f é *polinomial*. Quando $f = \mathcal{O}(c^n)$, para alguma constante $c > 1$, dizemos que f é *exponencial*. Finalmente, quando $f = \mathcal{O}(n!)$ dizemos que f é *factorial*. Adicionalmente, a ordem pela qual o comportamento assintótico destas funções foi apresentado estabelece uma hierarquia entre funções. Por exemplo, uma função quadrática também é cúbica, mas o inverso não é necessariamente verdade. Voltaremos a este assunto mais à frente.

Exercícios

1 Conjuntos, funções e cardinalidade

1. Sejam $f : A \to B$ e $g : B \to C$ duas funções. Demonstre que se $(g \circ f)$ é injectiva então f é injectiva, e que se $(g \circ f)$ é sobrejectiva então g é sobrejectiva.

2. Será que $A \subsetneq B$ implica que $\#A < \#B$? Justifique.

3. Mostre que se $A \subseteq B$ e $\#A < \#B$ então $B \setminus A \neq \emptyset$.

4. Demonstre que $\mathbb{N}_0, \mathbb{Z}, \mathbb{N} \times \mathbb{N}, \mathbb{Q}$ são conjuntos numeráveis.

5. Sejam A, B conjuntos.

   (a) Mostre que $\#A = \#\mathbb{N}$ se e só se existe uma função $h : A \to \mathbb{N}$ bijectiva.

   (b) (*Teorema de Cantor-Bernstein*) Demonstre que $\#A = \#B$ se e só se existe uma função $h : A \to B$ bijectiva.

6. Demonstre que $\#\mathbb{N} < \#\mathbb{R}$.

7. Seja A um conjunto.

   (a) Mostre A não é equipotente a $\wp(A)$.

   (b) (*Teorema de Cantor*) Demonstre que $\#A < \#\wp(A)$.

2 Alfabetos e linguagens

1. Seja L uma linguagem. Mostre que $L^* = \bigcup_{i\in\mathbb{N}_0} L^i$ onde $L^0 = \{\epsilon\}$ e $L^{n+1} = L.L^n$.

2. Dado um alfabeto Σ, mostre como representar cada linguagem em $\mathcal{L}^\Sigma$ como uma linguagem em $\mathcal{L}^{\{0,1\}}$, bem como cada função em $\mathcal{F}^\Sigma$ como uma função em $\mathcal{F}^{\{0,1\}}$.

3. Mostre que $\{0,1\}^*$ é numerável. Sugestão: tire partido da representação dos naturais em notação binária.

4. (*Proposição 1.3*) Seja Σ um alfabeto. Mostre que $\#\Sigma^* = \#\mathbb{N}$.

5. Seja Σ um alfabeto. Mostre que $\#\Sigma^* = \#\{0,1\}^*$, $\#\mathcal{L}^\Sigma = \#\mathcal{L}^{\{0,1\}}$ e $\#\mathcal{F}^\Sigma = \#\mathcal{F}^{\{0,1\}}$.

6. Use o método da diagonalização para demonstrar que o conjunto das palavras infinitas sobre $\{0,1\}$ não é numerável.

7. Use o método da diagonalização para demonstrar que o conjunto de todas as funções totais de $\{0,1\}^*$ em $\{0,1\}$ não é numerável.

8. Use o método da diagonalização para demonstrar que o conjunto de todas as funções totais de $\mathbb{N}_0$ em $\mathbb{N}_0$ não é numerável.

9. Seja Σ um alfabeto. Mostre que $\mathcal{L}^\Sigma$ não é numerável.

10. Seja Σ um alfabeto. Mostre que $\#\mathcal{L}^\Sigma = \#\mathbb{R}$.

3 Demonstrações: indução, redução ao absurdo

1. Demonstre as seguintes propriedades:

   (a) $2^n < n!$ para todo o $n \in \mathbb{N}_4$;

   (b) $2^n > n^2$ para todo o $n \in \mathbb{N}_5$.

2. A *sucessão de Fibonacci* $f = (f_n)_{n\in\mathbb{N}_0}$ define-se por recorrência como se segue: $f_0 = 0$, $f_1 = 1$ e $f_n = f_{n-1} + f_{n-2}$, para $n \geq 2$. Demonstre, por indução, as seguintes propriedades da sucessão de Fibonacci:

(a) $f_{n+1}f_{n-1} = f_n^2 + (-1)^n$ para todo o $n \geq 1$;

(b) $f_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$ para todo o $n \in \mathbb{N}_0$.

4 Comportamento assintótico de funções

1. Demonstre que:

   (a) $n + 5 = \mathcal{O}(n)$ e $n + 5 = \mathcal{O}(n^2)$;

   (b) $n + 5 = \Theta(n)$ mas $n + 5 \neq \Theta(n^2)$.

2. Seja $p : \mathbb{N} \to \mathbb{R}_0^+$ tal que $p(n)$ é um polinómio de grau k. Mostre que $p(n) = \mathcal{O}(n^{k'})$ qualquer que seja $k' \geq k$.

3. Sejam $f, g : \mathbb{N} \to \mathbb{R}_0^+$.

   (a) Demonstre que se $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = r \in \mathbb{R}_0^+$ então $f = \mathcal{O}(g)$.

   (b) Demonstre que se $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = r \in \mathbb{R}^+$ então $f = \Theta(g)$.

4. Demonstre que:

   (a) $2^{n+3} = \mathcal{O}(2^n)$;

   (b) $2^{n+3} = \Theta(2^n)$;

   (c) $n! = \mathcal{O}(n^n)$;

   (d) $\log(n^2 + 1) = \mathcal{O}(\log n)$;

   (e) $\log(n^2 + 1) = \Theta(\log n)$;

   (f) $\log(n!) = \mathcal{O}(n \log(n))$.

5. Considere as funções monótonas $f, g : \mathbb{N} \to \mathbb{R}_0^+$ e $h : \mathbb{N} \to \mathbb{N}$. Demonstre que se $f = \mathcal{O}(n^{k_1})$, $g = \mathcal{O}(n^{k_2})$ e $h = \mathcal{O}(n^{k_3})$ então:

   (a) $f + g = \mathcal{O}(n^{\max\{k_1,k_2\}})$;

   (b) $f \times g = \mathcal{O}(n^{k_1+k_2})$;

   (c) $f \circ h = \mathcal{O}(n^{k_1 \times k_3})$.

6. Sejam $f, g, h, f_1, f_2, g_1, g_2 : \mathbb{N} \to \mathbb{R}_0^+$, e $max(\{g_1, g_2\})$ a função que a cada $n \in \mathbb{N}$ faz corresponder $max(\{g_1(n), g_2(n)\})$. Demonstre que:

   (a) se $f = \mathcal{O}(g)$ e $g = \mathcal{O}(h)$ então $f = \mathcal{O}(h)$;

   (b) se $f_1 = \mathcal{O}(g_1)$ e $f_2 = \mathcal{O}(g_2)$ então $f_1 + f_2 = \mathcal{O}(max(\{g_1, g_2\}))$;

   (c) se $f_1 = \mathcal{O}(g)$ e $f_2 = \mathcal{O}(g)$ então $f_1 + f_2 = \mathcal{O}(g)$;

   (d) se $f_1 = \mathcal{O}(g_1)$ e $f_2 = \mathcal{O}(g_2)$ então $f_1 \times f_2 = \mathcal{O}(g_1 \times g_2)$.

7. Sejam $f, g, h, : \mathbb{N} \to \mathbb{R}_0^+$. Demonstre que:

   (a) $f = \Theta(g)$ se e só se $g = \Theta(f)$;

   (b) se $f = \Theta(g)$ e $g = \Theta(h)$ então $f = \Theta(h)$.

2

Autómatos e linguagens

"What we cannot speak about we must pass over in silence."
Ludwig Wittgenstein, *Tractatus Logico-Philosophicus*, 1921

2.1 Linguagens regulares

Recorde-se que uma linguagem é um conjunto $L \subseteq \Sigma^*$ de palavras sobre um alfabeto Σ. Esta secção é dedicada às linguagens ditas *regulares*, com características muito simples, aos modelos computacionais que as reconhecem e suas propriedades. Apesar de muito simples, as linguagens regulares e seus reconhecedores, os autómatos finitos, têm inúmeras aplicações. Por exemplo, o léxico de uma linguagem de programação é constituído por conjuntos de palavras de diferentes categorias: palavras reservadas, identificadores e constantes numéricas, entre outras. Cada um destes conjuntos é uma linguagem regular, e a análise lexical, a primeira fase de um compilador, envolve o reconhecimento destas palavras através de autómatos finitos. Os autómatos finitos são úteis para detectar a existência de certas sequências de símbolos (padrões) em sequências de símbolos mais longas, e como tal também relevantes em muitos outros contextos que vão desde os editores de texto ao processamento de linguagem natural. Um dos mais conhecidos algoritmos para este tipo de tarefa, o algoritmo de Knuth-Morris-Pratt, pode ser descrito através de um autómato finito.

2.1.1 Autómatos finitos deterministas

O modelo computacional mais simples que consideraremos é o *autómato finito determinista*.

Definição 2.1. Autómato finito determinista
Um *autómato finito determinista*, ou apenas AFD, é formado por:

- um alfabeto Σ;
- um conjunto finito de *estados* Q;

- um estado designado $q_{\text{in}} \in Q$, dito *estado inicial*;
- um conjunto designado de estados $F \subseteq Q$, ditos *estados finais*;
- uma *função de transição* $\delta : Q \times \Sigma \to Q$. △

Qualquer AFD tem pelo menos um estado, o estado inicial, e pode ter um qualquer número de estados finais, em particular, nenhum. Note-se que a função de transição de um AFD não tem de ser necessariamente uma função total. Usa-se notação $\delta(q,a)\downarrow$ para indicar que a função de transição δ atribui um valor ao par (q,a) e usa-se a notação $\delta(q,a)\uparrow$ para indicar que δ não está definida para o par (q,a). Quando $\delta(q,a)\downarrow$ diz-se que existe uma transição a partir de q associada ao símbolo a e, se $\delta(q,a) = p$, diz-se que existe uma transição de q para p associada ao símbolo a. Quando $\delta(q,a)\uparrow$ diz-se que a partir de q não existe transição associada ao símbolo a.

A ideia subjacente a um AFD como reconhecedor de uma linguagem é muito simples: para determinar se uma dada palavra w pertence à linguagem deve verificar-se se a computação iniciada em q_{in} transitando sucessivamente de estado de acordo com δ e cada um dos símbolos de w conduz, ou não, a um estado final.

Exemplo 2.2.
Considere-se o AFD $D = (\Sigma, Q, q_{\text{in}}, F, \delta)$ em que

- $\Sigma = \{a, b, c\}$
- $Q = \{q_{\text{in}}, q_1, q_2\}$
- $F = \{q_1\}$
- $\delta : Q \times \Sigma \to Q$ é representada através da tabela

δ	a	b	c
q_{in}	q_1		
q_1	q_1	q_2	q_2
q_2	q_1	q_2	q_2

Este AFD pode ser também representado através de um grafo orientado, como ilustrado na Figura 2.1. Os vértices correspondem aos estados, e as setas e suas etiquetas representam as transições. O estado inicial é identificado através de uma seta sem origem mas com destino, e o estado final é identificado por dois círculos concêntricos.

Tome-se a palavra *aba*, por exemplo. De acordo com a função de transição δ, o primeiro símbolo da palavra, o símbolo a, permite transitar do estado inicial q_{in} para o estado q_1. O segundo símbolo, b, permite transitar do estado q_1 para o estado q_2. Por fim, o último símbolo, a, permite transitar de q_2 para q_1. Como q_1 é estado final de D, diz-se que *aba* é aceite por D, ou que *aba* pertence à linguagem reconhecida por D. De modo semelhante se conclui que a, *aca* e *aabcbaa* são outros exemplos de palavras aceites por D. Por sua vez, a

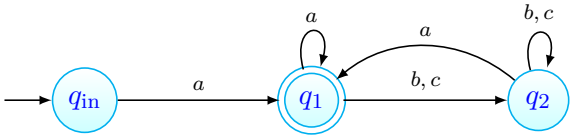


Figura 2.1: AFD para a linguagem $\{x_1 \dots x_n \in \{a,b,c\}^* : n \in \mathbb{N}, x_1 = x_n = a\}$.

palavra ab não é aceite por D porque após a transição associada ao seu último símbolo, b, o AFD se encontra no estado q_2, e este estado não é final. O mesmo acontece com qualquer palavra com comprimento maior que 1 que termine em b ou c. Também não são aceites por D palavras começadas por b ou c, pois não existem transições associadas a estes símbolos a partir do estado inicial. É fácil concluir que as palavras aceites por D são precisamente todas as palavras sobre $\{a,b,c\}$ que começam e terminam em a.

A noção de caminho num AFD e de palavra associada a um caminho é relevante neste contexto. Por exemplo, a sequência de estados $q_{\text{in}}q_1q_2q_1$ é um caminho de D, ao qual está associada a palavra aba, porque $\delta(q_{\text{in}}, a) = q_1$, $\delta(q_1, b) = q_2$, e $\delta(q_2, a) = q_1$. Uma outra palavra associada a este caminho é aca. A sequência $q_2q_2q_1q_2q_1$ é um outro exemplo de caminho de D, e $baca$ é uma das palavras que lhe está associada. Observe-se que uma palavra pode estar associada a vários caminhos de um AFD, mas se uma palavra está associada a um caminho que começa num dado estado, então não está associada a mais nenhum caminho que comece nesse estado. △

Cada AFD define uma linguagem sobre o seu alfabeto Σ. Para caracterizar rigorosamente esta linguagem é útil considerar a extensão a Σ^* da função de transição do AFD.

Definição 2.3. FUNÇÃO DE TRANSIÇÃO ESTENDIDA DE AFD
Seja $D = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFD. A *função de transição estendida* de D é a função $\delta^* : Q \times \Sigma^* \to Q$ tal que

$$\delta^*(q,w) = \begin{cases} q & \text{se } w = \epsilon \\ \delta^*(\delta(q,a),w') & \text{se } w = a.w',\ \delta(q,a)\downarrow \text{ e } \delta^*(\delta(q,a),w')\downarrow \\ \text{indefinido} & \text{caso contrário} \end{cases}$$

para cada $q \in Q$, $a \in \Sigma$ e $w \in \Sigma^*$. △

Observe-se que se $\delta^*(q,w) = q'$ e $\delta^*(q',w') = q''$ então $\delta^*(q,w.w') = q''$. Quando $\delta^*(q,w) = q'$, tal significa que existe em D um caminho que tem início em q e termina em q' ao qual está associada a palavra w.

Definição 2.4. PALAVRA ACEITE E LINGUAGEM RECONHECIDA POR AFD
Seja $D = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFD. A palavra $w \in \Sigma^*$ diz-se *aceite* por D se $\delta^*(q_{\text{in}}, w) \in F$. O conjunto

$$L(D) = \{w \in \Sigma^* : \delta^*(q_{\text{in}}, w) \in F\}$$

é a *linguagem reconhecida por* D, ou *linguagem de* D. △

A linguagem reconhecida por D é o conjunto de todas as palavras aceites por D, ou seja, as palavras associadas às computações que partem do estado inicial e que terminam num estado final.

Exemplo 2.5.
Na Figura 2.2 está representado um AFD que reconhece a linguagem das palavras sobre o alfabeto $\{a, b, c\}$ que terminam em ab. △

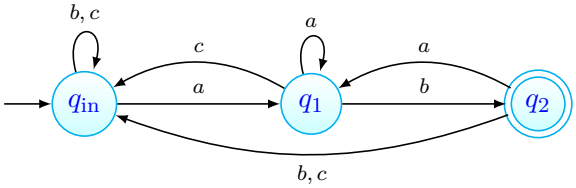


Figura 2.2: AFD para a linguagem $\{wab : w \in \{a, b, c\}^*\}$.

Exemplo 2.6.
Na Figura 2.3 está representado um AFD que reconhece a linguagem das palavras sobre o alfabeto $\{0, 1\}$ que começam em 01 mas não terminam em 01. △

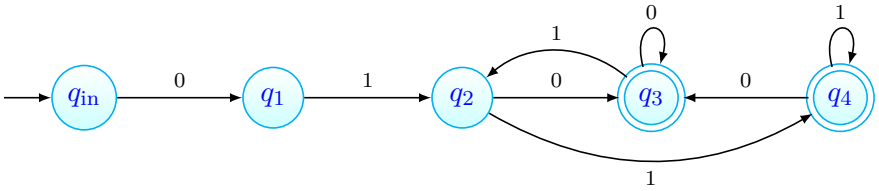


Figura 2.3: AFD para $\{w \in \{0, 1\}^* : w$ começa, mas não termina, em $01\}$.

Exemplo 2.7.
Na Figura 2.4 está representado um AFD que reconhece a linguagem das palavras sobre o alfabeto $\{a, b, c\}$ que têm um número par de as e no máximo um c. △

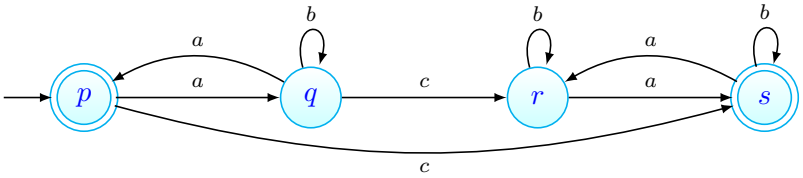


Figura 2.4: AFD para $\{w \in \{a, b, c\}^* : w$ tem nº par de as e no máximo um $c\}$.

Exemplo 2.8.
O AFD à esquerda na Figura 2.5 não aceita nenhuma palavra, pelo que se diz que é vacuoso. O AFD vacuoso canónico, à direita na Figura 2.5, tem um único estado, não tem estados finais e a função de transição não está definida para nenhum elemento de $Q \times \Sigma$. △

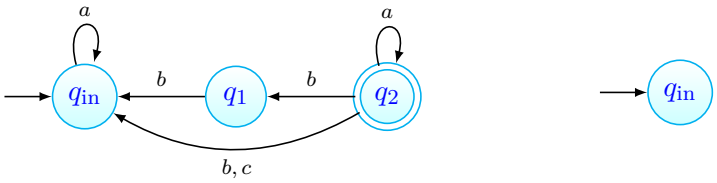


Figura 2.5: AFD vacuoso (à esquerda) e AFD vacuoso canónico (à direita).

Note-se que se o estado inicial de um AFD D for também estado final, a linguagem reconhecida por D inclui pelo menos uma palavra, a palavra vazia ϵ. Esta é, aliás, a única forma de ϵ ser aceite por um AFD.

Observe-se ainda que se o conjunto dos estados finais de um AFD D é vazio, nenhuma palavra é aceite por D. A linguagem reconhecida pelo AFD é neste caso a linguagem vazia, isto é, $L(D) = \emptyset$.

Do ponto de vista da linguagem reconhecida por um AFD, o nome que se dá aos estados é irrelevante. Assim, dados dois AFDS, pode sempre assumir-se, sem perda de generalidade, que os respectivos conjuntos de estados são disjuntos.

Como vimos, a função de transição de um AFD não tem de ser total. Veja-se, por exemplo, os AFDS apresentados nos Exemplos 2.2, 2.6 e 2.7. No entanto, é muito simples obter um AFD total, isto é, com função de transição total, que reconhece a mesma linguagem. Basta para tal adicionar ao AFD um novo estado não-final, para onde se definem todas as transições antes indefinidas, como se ilustra no exemplo abaixo.

Exemplo 2.9.
O AFD total apresentado na Figura 2.6 reconhece a mesma linguagem que o AFD do Exemplo 2.2. △

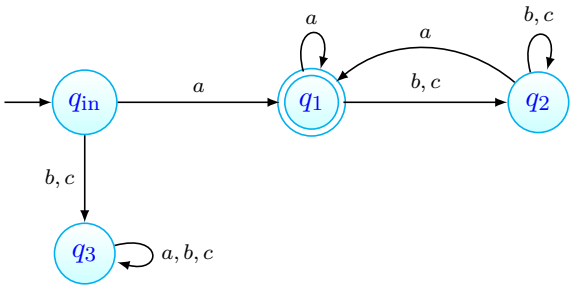


Figura 2.6: AFD total para $\{x_1 \dots x_n \in \{a, b, c\}^* : n \in \mathbb{N}, x_1 = x_n = a\}$.

Dada um certa linguagem L, diz-se que D é um AFD para L se D é um AFD e $L(D) = L$. Diz-se que uma certa linguagem L é reconhecida por um AFD sempre que exista um AFD para L.

As linguagens que são reconhecidas por AFDS têm uma designação especial.

Definição 2.10. LINGUAGEM REGULAR
Uma linguagem $L \subseteq \Sigma^*$ diz-se *regular* se existe um AFD D com alfabeto Σ tal que $L(D) = L$. Denota-se por $\mathcal{REG}^\Sigma$ o conjunto de todas as linguagens regulares com alfabeto Σ. △

Usaremos apenas $\mathcal{REG}$, em vez de $\mathcal{REG}^\Sigma$, sempre que o alfabeto esteja subentendido ou não seja importante no contexto.

As linguagens referidas nos Exemplos 2.2, 2.5, 2.6 e 2.7 são linguagens regulares. Nem todas as linguagens são regulares. Como se verá adiante na Secção 2.1.4, a linguagem das palavras sobre o alfabeto $\{a, b\}$ do tipo $a^n b^n$, com $n \in \mathbb{N}_0$, é um exemplo de linguagem não regular.

2.1.2 Equivalência e minimização

Dois AFDS são equivalentes quando, mesmo sendo diferentes, reconhecem a mesma linguagem.

Definição 2.11. AFDS EQUIVALENTES
Dois AFDS D_1 e D_2 com o mesmo alfabeto são *equivalentes* se $L(D_1) = L(D_2)$. △

Por exemplo, o AFD apresentado na Figura 2.7 é equivalente ao AFD do Exemplo 2.2.

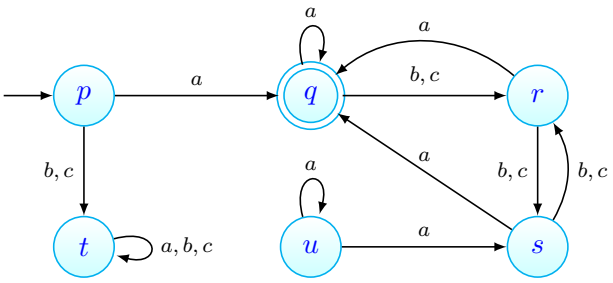


Figura 2.7: AFD equivalente ao do Exemplo 2.2.

Para analisar a questão da equivalência de autómatos é necessário entender as propriedades dos seus estados.

Definição 2.12. ESTADO ACESSÍVEL, PRODUTIVO, ÚTIL E INÚTIL
Seja $D = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFD. Diz-se que um estado $q \in Q$ é

- *acessível* se existe $w \in \Sigma^*$ tal que $\delta^*(q_{\text{in}}, w) = q$;
- *produtivo* se existe $w \in \Sigma^*$ tal que $\delta^*(q, w) \in F$;
- *útil* se é estado acessível e é estado produtivo;
- *inútil* se não é estado útil. △

No AFD da Figura 2.7 existem caminhos que começam no estado inicial e terminam, por exemplo, em r. Mas não existe nenhum caminho que comece no estado inicial e termine em u. O estado r é assim um estado acessível, mas o estado u não é acessível. Como existem caminhos que começam em u e terminam no estado final, mas não existe nenhum caminho que comece em t e termine no estado final, o estado u é produtivo, mas o estado t não é produtivo.

As transições de e para estados que não são acessíveis, bem como as transições de e para estados que não são produtivos, são irrelevantes do ponto de vista da linguagem reconhecida pelo AFD. Em particular, se não existirem estados finais acessíveis, então o AFD é vacuoso. O mesmo acontece se o estado inicial não for produtivo.

Apresenta-se seguidamente um algoritmo que permite calcular os estados notáveis acima referidos, isto é, os estados acessíveis, os estados produtivos e os estados úteis e inúteis de um AFD.

Definição 2.13. ALGORITMO DE PROCURA DE ESTADOS NOTÁVEIS (APEN) O algoritmo de procura de estados notáveis (APEN) é o seguinte:

ENTRADA: AFD $D = (\Sigma, Q, q_{in}, F, \delta)$
SAÍDA: *o tuplo de conjuntos* $APEN(D) = (Ac, Prd, Ut, In)$

1. $A := \{q_{in}\}$;
2. $Aux := \bigcup_{a\in\Sigma}\{\delta(q_{in}, a)\}$;
3. *enquanto* $Aux \not\subseteq A$
   3.1. $A := A \cup Aux$;
   3.2. $Aux := \bigcup_{a\in\Sigma}\{\delta(p, a) : p \in Aux\}$;
4. $Ac := A$;
5. $A := F$;
6. $Aux := \bigcup_{a\in\Sigma}\{p : \delta(p, a) \in F\}\}$;
7. *enquanto* $Aux \not\subseteq A$
   7.1. $A := A \cup Aux$;
   7.2. $Aux := \bigcup_{a\in\Sigma}\{p : \delta(p, a) \in Aux\}$;
8. $Prd := A$;
9. $Ut := Ac \cap Prd$;
10. $In := Q \backslash (Ac \cap Prd)$.

△

A execução do APEN termina sempre e calcula o conjunto de todos os estados acessíveis, produtivos, úteis e inúteis do AFD de entrada.

Proposição 2.14.
A execução do APEN termina sempre e, dado um AFD $D = (\Sigma, Q, q_{in}, F, \delta)$, *se* $APEN(D) = (Ac, Prd, Ut, In)$ *então* Ac, Prd, Ut, In *são, respectivamente, os conjuntos de estados acessíveis, produtivos, úteis e inúteis de* D.

Dem. (esboço): A terminação de qualquer dos ciclos do algoritmo é garantida pelo facto de $A \subseteq Q$ ser um conjunto de estados, que A cresce a cada passo de

qualquer dos ciclos, pelo que não pode crescer indefinidamente pois Q é finito. Aliás, ambos os ciclos terminam no máximo ao fim de $\#Q - 1$ iterações.

A correcção do algoritmo advém dos seguinte factos, simples: (1) no final de k iterações do primeiro ciclo do algoritmo tem-se necessariamente que $A = \{\delta^*(q_{\text{in}}, w) : w \in \Sigma^* \text{ com } |w| \leq k\}$; (2) um estado $q \in Q$ é acessível se e só existe $w \in \Sigma^*$ com $|w| < \#Q$ tal que $\delta^*(q_{\text{in}}, w) = q$.

Relativamente aos estados produtivos, o argumento é semelhante: (1) no final de k iterações do segundo ciclo do algoritmo tem-se necessariamente que $A = \{q \in Q : \delta^*(q, w) \in F \text{ para algum } w \in \Sigma^* \text{ com } |w| \leq k\}$; (2) um estado $q \in Q$ é produtivo se e só se existe $w \in \Sigma^*$ com $|w| < \#Q$ tal que $\delta^*(q, w) \in F$.

Os estados úteis e inúteis, estão portanto, também correctos. □

No intuito de analisar a equivalência de autómatos, é claro que podemos ignorar todos os seus estados inúteis (com a excepção eventual do estado inicial, caso o AFD seja vacuoso, como no Exemplo 2.8). No entanto isto não basta, em geral, pelo que é necessário analisar em mais detalhe as propriedades dos estados dos autómatos (vide Exemplo 2.20).

Definição 2.15. ESTADOS EQUIVALENTES E DISTINGUÍVEIS
Diz-se que dois estados p e q de um AFD $D = (\Sigma, Q, q_{\text{in}}, F, \delta)$ são

- *equivalentes* se, para cada $w \in \Sigma^*$, $\delta^*(p, w) \in F$ se e só se $\delta^*(q, w) \in F$;
- *distinguíveis* se não são estados equivalentes. △

Os estados p e q são distinguíveis precisamente quando existe uma palavra $w \in \Sigma^*$ que os distingue, isto é, tal que $\delta^*(p, w) \in F$ e $\delta^*(q, w) \notin F$, ou vice-versa. Em particular, se p é estado final e q não é estado final, p e q são estados distinguíveis, pois a palavra ϵ distingue-os, dado que $\delta^*(p, \epsilon) = p \in F$ e $\delta^*(q, \epsilon) = q \notin F$. Descrever-se-ão adiante outras situações que permitem identificar directamente estados distinguíveis, e indirectamente estados equivalentes.

No AFD da Figura 2.7 os estados r e s são equivalentes, mas todos os outros pares de estados úteis (distintos) são distinguíveis. O estado p é distinguível de q porque a palavra ϵ, por exemplo, os distingue. Por razões semelhantes r e s são também distinguíveis de q. A palavra ba distingue r de p pois $\delta^*(r, ba) \in F$ e $\delta^*(p, ba)\uparrow$, e distingue s de p pelo mesmo motivo.

Quando num AFD se encontram dois estados distintos equivalentes, o AFD pode ser simplificado, no sentido em que é possível colapsá-los e obter um AFD equivalente. As transições para o estado que deixa de existir são substituídas por transições para o estado equivalente cuja designação é mantida. Na proposição seguinte enuncia-se e prova-se esta propriedade dos AFDS.

Proposição 2.16.
Seja $D = (\Sigma, Q, q_{in}, F, \delta)$ um AFD. Sejam p e q estados distintos e equivalentes de D tais que $p \neq q_{in}$. É equivalente a D o AFD

$$D' = (\Sigma, Q', q_{in}, F', \delta')$$

em que

- $Q' = Q \backslash \{p\}$;
- $F' = F \backslash \{p\}$;
- $\delta' : Q' \times \Sigma \to Q'$ *é tal que*

$$\delta'(q', a) = \begin{cases} \delta(q', a) & \text{se } \delta(q', a) \in Q' \\ q & \text{se } \delta(q', a) = p \\ \text{indefinido} & \text{se } \delta(q', a)\!\uparrow \end{cases}$$

para cada $q' \in Q'$ *e* $a \in \Sigma$.

Dem.: Exercício. □

Note-se que o requisito $p \neq q_{\text{in}}$ no enunciado da proposição anterior não implica perda de generalidade pois, como existe um único estado inicial, dados dois estados distintos um deles será necessariamente não inicial.

Como se viu, se p é um estado final de um AFD e q não é estado final então p e q são distinguíveis mas existem outras situações que também permitem identificar estados distinguíveis.

Proposição 2.17.
Seja $D = (\Sigma, Q, q_{in}, F, \delta)$ *um* AFD *e sejam* $p, q \in Q$ *e* $a \in \Sigma$.

1. *Se* $p \in F$ *e* $q \notin F$ *então* p *e* q *são estados distinguíveis.*
2. *Se* p *é produtivo e* q *não é produtivo então* p *e* q *são estados distinguíveis.*
3. *Se* $\delta(p, a)$ *é produtivo e* $\delta(q, a)\!\uparrow$ *então* p *e* q *são estados distinguíveis.*
4. *Se* p' *e* q' *são estados distinguíveis,* $\delta(p, a) = p'$ *e* $\delta(q, a) = q'$ *então* p *e* q *são estados distinguíveis.*

Dem.: Exercício. □

Para facilitar a exposição, introduz-se as noções e notações seguintes. Um *par não ordenado* de estados é um conjunto de estados com cardinalidade 2. Denota-se indiferentemente por

$$[p, q] \quad \text{ou} \quad [q, p]$$

o par não ordenado de estados constituído pelos estados distintos p e q. Denota-se por Prs_D o conjunto de todos os pares não ordenados de estados de D.

Apresenta-se de seguida um algoritmo que permite identificar todos os pares de estados distinguíveis de um AFD, e por exclusão de partes também os pares de estados equivalentes.

Definição 2.18. ALGORITMO DE PROCURA DE ESTADOS DISTINGUÍVEIS (APED)
O algoritmo de procura de estados distinguíveis (APED) é o seguinte:

ENTRADA: AFD $D = (\Sigma, Q, q_{in}, F, \delta)$
SAÍDA: *conjunto* $APED(D) = Dst$

1. $\Delta := \{[p,q] : p \in F \text{ e } q \notin F\} \cup$
   $\{[p,q] : p \text{ produtivo e } q \text{ não produtivo}\} \cup$
   $\{[p,q] : \text{existe } a \in \Sigma \text{ tal que } \delta(p,a) \text{ produtivo e } \delta(q,a)\uparrow\}$;
2. $Aux := \{[p,q] \notin \Delta : \text{existe } a \in \Sigma \text{ tal que } [\delta(p,a), \delta(q,a)] \in \Delta\}$;
3. *enquanto* $\Delta \cup Aux \neq Prs_D$ *e* $Aux \neq \emptyset$
   3.1. $\Delta := \Delta \cup Aux$;
   3.2. $Aux := \{[p,q] \notin \Delta : \text{existe } a \in \Sigma \text{ tal que } [\delta(p,a), \delta(q,a)] \in Aux\}$;
4. $Dst := \Delta \cup Aux$.

△

A execução do APED termina sempre porque, como existe um número finito de estados, existem também um número finito de pares de estados e, em particular, um número finito de pares de estados distinguíveis. Garante-se também que, quando termina a sua execução, só foram identificados pares de estados distinguíveis e foram identificados todos os pares nessas condições.

Considere-se o AFD D que se obtém removendo todos os estados inúteis do AFD da Figura 2.7. O algoritmo APED aplicado a D constrói o conjunto $APED(D) = \{[p,q],[q,s],[q,r],[p,r],[p,s]\}$. Os detalhes desta construção serão apresentados adiante. Resulta assim que r e s são identificados como os únicos estados equivalentes de D. Aplicado ao AFD D do Exemplo 2.5, o APED constrói o conjunto $APED(D) = \{[q_{\text{in}}, q_1], [q_{\text{in}}, q_2], [q_1, q_2]\}$, que permite concluir que neste AFD não existem estados distintos equivalentes.

Proposição 2.19.
A execução do APED termina sempre e, dado um AFD $D = (\Sigma, Q, q_{in}, F, \delta)$, *se* $APED(D) = Dst$ *então* Dst *é precisamente o conjunto dos pares de estados distinguíveis de* D.

Dem.: Exercício. □

Exemplo 2.20. Seja D o AFD representado na Figura 2.8. Para encontrar

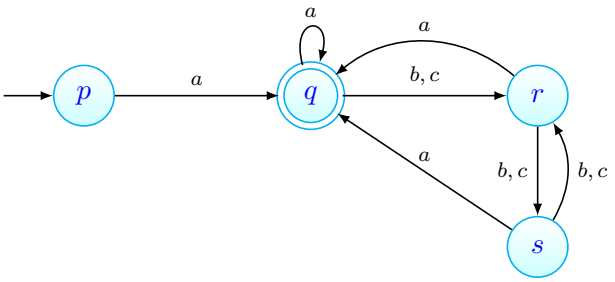


Figura 2.8: AFD com um par de estados equivalentes.

todos os pares de estados distinguíveis de D, o APED calcula o valor inicial de Δ como indicado na instrução 1. Para tal são considerados primeiro todos os pares de estados em que um seja final e outro não. São eles, neste caso,

$$[p, q],\ [q, s],\ [q, r]$$

Consideram-se de seguida os pares de estados em que um deles é produtivo e outro não. Neste caso não existem tais pares porque todos os estados são produtivos. Por último, são identificados pares de estados tais que a partir de um deles exista uma transição associada a um símbolo para um estado produtivo, e a partir do outro não exista transição associada a este símbolo. Neste caso encontram-se os pares

$$[p, q],\ [p, r],\ [p, s]$$

Em particular, $\delta(r, b) = s$ e s é produtivo mas $\delta(p, b)\uparrow$. Os outros casos são semelhantes. Consequentemente, o valor de Δ calculado pela instrução 1 é

$$\Delta = \{[p, q], [q, s], [q, r], [p, r], [p, s]\}.$$

O APED verifica agora se a partir dos estados distinguíveis encontrados até ao momento se podem identificar novos pares distinguíveis, isto é, é calculado o valor inicial de Aux, como indicado na instrução 2. Por exemplo, dado que o par $[p, s] \in \Delta$, há que procurar estados q' e q'' e um símbolo i do alfabeto tais que $\delta(q', i) = p$ e $\delta(q'', i) = s$. Neste caso não existem q' e q'' nessas condições, pelo que o par $[p, s]$ não permite encontrar mais pares de estados distinguíveis. Procedendo de modo semelhante para os outros pares em Δ, não se encontram mais pares de estados distingíveis. Isto significa que na instrução 2 o valor que é atribuído a Aux é o conjunto vazio.

Dado que $Aux = \emptyset$, a guarda do ciclo correspondente à instrução 3 é falsa. É então executada a instrução 4 que atribui a $APED(D)$ o valor de $\Delta \cup Aux$. Assim, o conjunto dos pares de estados calculado pelo APED é

$$APED(D) = \{[p, q], [q, s], [q, r], [p, r], [p, s]\}$$

que, pela Proposição 2.19, é o conjunto de todos os pares de estados distinguíves de D. Logo, o conjunto de pares de estados equivalentes de D é $\{[r, s]\}$.

Para facilitar a execução deste algoritmo com papel e lápis, pode utilizar-se uma tabela que se vai preenchendo à medida que se vão identificando pares de estados distinguíveis. A tabela, no caso deste AFD, é

q			
r			
s			
	p	q	r

sendo construída por forma a permitir referenciar cada par de estados de D. Nas linhas desta tabela encontram-se todos os estados, com a excepção do estado p, e nas colunas encontram-se todos os estados, com a excepção do estado s. À medida que se vão identificando pares de estados distinguíveis, estes vão sendo assinalados na tabela no local apropriado usando, por exemplo, o símbolo $\times$.

Os primeiros pares a serem identificados são os correspondentes ao caso em que um deles é estado final e outro não que, como se viu atrás, são $[p, q]$, $[q, s]$ e $[q, r]$ pelo que se obtém

q	$\times$		
r		$\times$	
s		$\times$	
	p	q	r

De seguida, identificaram-se estados distinguíveis pelo facto de um ter uma transição associada a um símbolo do alfabeto para um estado produtivo e outro não ter transição associada a esse símbolo: são os pares $[p, q]$, $[p, r]$, $[p, s]$. Após referenciar estes pares na tabela, obtém-se

q	$\times$		
r	$\times$	$\times$	
s	$\times$	$\times$	
	p	q	r

Nesta tabela estão assim assinalados todos os pares que constituem o valor inicial de Δ calculado pela instrução 1.

O passo seguinte implica considerar cada par $[p', q']$ já identificado (par assinalado com $\times$) e verificar se a partir dele se conseguem identificar novos pares de estados $[p'', q'']$ pelo facto de $\delta(p'', i) = p'$ e $\delta(q'', i) = q'$ para algum símbolo i do alfabeto. Após a análise de cada um desses pares, ele é referenciado na tabela usando $\otimes$, por exemplo, e os novos pares de estados assim identificados (se existirem) são assinalados na tabela com $\times$, como anteriormente. Começando pelo par $[p, q]$, por exemplo, conclui-se que nenhum novo par é identificado, pelo que a tabela resultante é

q	$\otimes$		
r	$\times$	$\times$	
s	$\times$	$\times$	
	p	q	r

Prosseguindo de modo análogo para os restantes casos, não se encontram mais pares de estados distingíveis pelo que

q	$\otimes$		
r	$\otimes$	$\otimes$	
s	$\otimes$	$\otimes$	
	p	q	r

é a tabela resultante. Nesta tabela está representada toda a informação relativa aos valores das variáveis Δ e Aux após a execução da instrução 2 do APED. Todos os pares de estados que foram sendo identificados como distinguíveis têm a indicação (com $\otimes$) de que foram devidamente analisados para se determinar se novos pares se poderiam identificar a partir deles. Isto significa que não é possível identificar mais estados distinguíveis. Esta situação corresponde ao facto de, como se viu, o corpo do ciclo relativo à instrução 3 não ser executado

porque o valor de Aux é o conjunto vazio. A tabela final é assim a indicada acima. Os pares de estados distinguíveis que são identificados pelo algoritmo são todos os assinalados. △

Exemplo 2.21. Seja D o AFD representado na Figura 2.9. Apresentam-se de

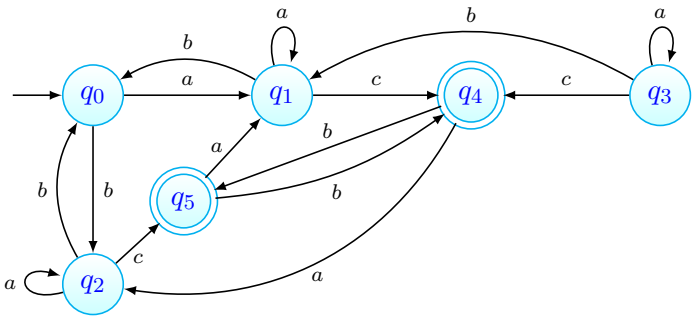


Figura 2.9: AFD com dois pares de estados equivalentes.

forma abreviada as diferentes fases da aplicação do APED a D. Os pares de estados distinguíveis encontrados na fase inicial, isto é, os pares encontrados ao calcular o valor de Δ executando a instrução 1 do APED, são os seguintes:

- os pares $[q_0, q_4]$, $[q_1, q_4]$, $[q_2, q_4]$, $[q_3, q_4]$, $[q_0, q_5]$, $[q_1, q_5]$, $[q_2, q_5]$ e $[q_3, q_5]$

  são identificados pelo facto de um deles ser estado final e o outro não

- os pares $[q_0, q_1]$, $[q_0, q_2]$ e $[q_0, q_3]$

  são identificados pelo facto de a partir de q_0 não existir transição associada a c, mas a partir de q_1, q_2 e q_3 existirem transições associadas a c para estados produtivos (de modo análogo se pode concluir que q_0 é distinguível de q_4 e de q_5, mas como estes pares de estados já foram identificados, não é agora necessário repeti-los).

Assinalando estes pares na tabela obtém-se

q_1	×				
q_2	×				
q_3	×				
q_4	×	×	×	×	
q_5	×	×	×	×	
	q_0	q_1	q_2	q_3	q_4

Procuram-se agora encontrar novos pares como indica a instrução 2 do APED. Considere-se, para começar, o par $[q_0, q_1]$. Como $\delta(q_1, b) = q_0$ e $\delta(q_3, b) = q_1$, identifica-se um novo par: $[q_1, q_3]$. De modo análogo se identifica também $[q_2, q_3]$. Mais nenhum novo par é identificado quando se analisam os restantes pares identificados na fase inicial. Isto significa que $Aux = \{[q_1, q_3], [q_2, q_3]\}$ após a execução da instrução 2. A tabela resultante é

q_1	$\otimes$				
q_2	$\otimes$				
q_3	$\otimes$	$\times$	$\times$		
q_4	$\otimes$	$\otimes$	$\otimes$	$\otimes$	
q_5	$\otimes$	$\otimes$	$\otimes$	$\otimes$	
	q_0	q_1	q_2	q_3	q_4

Há agora que analisar os pares $[q_1, q_3]$ e $[q_2, q_3]$ para verificar se a partir destes se identificam novos pares, o que corresponde à execução das instruções 3.1 e 3.2 do APED. Neste caso não são identificados novos pares e tabela resultante é

q_1	$\otimes$				
q_2	$\otimes$				
q_3	$\otimes$	$\otimes$	$\otimes$		
q_4	$\otimes$	$\otimes$	$\otimes$	$\otimes$	
q_5	$\otimes$	$\otimes$	$\otimes$	$\otimes$	
	q_0	q_1	q_2	q_3	q_4

a partir da qual se pode concluir que não é possível encontrar mais estados distinguíveis. Logo, o novo valor de Aux é o conjunto vazio. A guarda do ciclo é agora falsa, pelo que se segue a execução da instrução 4 de que resulta

$$APED(D) = \{[q_0, q_1], [q_0, q_2]], [q_0, q_3], [q_0, q_4], [q_0, q_5], [q_1, q_3], [q_1, q_4], [q_1, q_5], [q_2, q_3], [q_2, q_4], [q_2, q_5], [q_3, q_4], [q_3, q_5]\}.$$

Decorre da Proposição 2.19 que este conjunto é precisamente o conjunto de todos os pares de estados distinguíveis de D. Consequentemente, o conjunto de pares de estados equivalentesde de D é $\{[q_1, q_2], [q_4, q_5]\}$. △

Para além de permitir calcular o conjunto dos pares de estados distinguíveis de um AFD e, consequentemente, o conjunto dos seus pares de estados equivalentes, o APED pode também ser usado para determinar se dois AFDS D_1 e D_2 são ou não equivalentes. Basta para tal integrar os dois autómatos num só e determinar se os estados iniciais de cada um deles são equivalentes. Recorde-se que pode assumir-se, sem perda de generalidade, que os conjuntos dos estados dos dois autómatos são disjuntos.

Definição 2.22. ALGORITMO DE TESTE À EQUIVALÊNCIA DE AFDS (ATEQ) O algoritmo de teste à equivalência de AFDS (ATEQ) é o seguinte:

ENTRADA: AFDS $D_1 = (\Sigma, Q_1, q_{in}^1, F_1, \delta_1)$ *e* $D_2 = (\Sigma, Q_2, q_{in}^2, F_2, \delta_2)$
tais que $Q_1 \cap Q_2 = \emptyset$
SAÍDA: *valor Booleano* $ATEQ(D_1, D_2) = b$

1. *construir o* AFD $D = (\Sigma, Q_1 \cup Q_2, q_{in}^1, F_1 \cup F_2, \delta)$ *em que*

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 \text{ e } \delta_1(q,a)\downarrow \\ \delta_2(q,a) & \text{se } q \in Q_2 \text{ e } \delta_2(q,a)\downarrow \\ \text{indefinido} & \text{caso contrário} \end{cases}$$

2. $Dst := APED(D)$;
3. se $[q_{in}^1, q_{in}^2] \in Dst$ então $b =$ **false**
senão $b =$ **true**.

△

A execução do ATEQ termina sempre e identifica como equivalentes dois AFDS se e só se eles são equivalentes, porque nenhuma palavra distingue os seus estados iniciais.

Proposição 2.23.
A execução do ATEQ termina sempre e, dados AFDS D_1 *e* D_2 *com o mesmo alfabeto,* $ATEQ(D_1, D_2) =$ **true** *se e só se* D_1 *e* D_2 *são equivalentes.*

Dem.: Exercício. □

Exemplo 2.24. Designe-se aqui por D_1 o AFD do Exemplo 2.21 e por D_2 o AFD representado na Figura 2.10, ambos com alfabeto $\Sigma = \{a, b, c\}$.

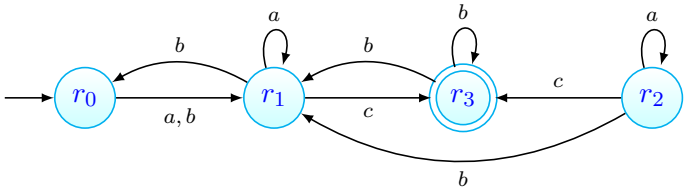


Figura 2.10: AFD equivalente ao do Exemplo 2.21.

Usa-se o ATEQ para determinar se D_1 e D_2 são ou não equivalentes. No primeiro passo, D_1 e D_2 são integrados num só AFD $D = (\Sigma, Q, q_0, F, \delta)$, cujos detalhes se deixam ao cuidado do leitor. Calcula-se depois o conjunto dos pares de estados distinguíveis de D, usando o APED. Ilustram-se sucintamente as diferentes fases da construção. Os pares de estados distinguíveis encontrados na fase inicial do APED são os indicados na tabela seguinte:

q_1	×								
q_2	×								
q_3	×								
q_4	×	×	×	×					
q_5	×	×	×	×					
r_0		×	×	×	×	×			
r_1	×				×	×	×		
r_2	×				×	×	×		
r_3	×	×	×	×			×	×	×
	q_0	q_1	q_2	q_3	q_4	q_5	r_0	r_1	r_2

De seguida,

- a partir de $[q_0, q_1]$ identificam-se $[q_1, q_3]$ e $[q_2, q_3]$
- a partir de $[q_0, r_1]$ identificam-se $[q_1, r_2]$ e $[q_2, r_2]$
- a partir de $[q_1, r_0]$ identifica-se $[q_3, r_1]$
- a partir de $[r_0, r_1]$ identifica-se $[r_1, r_2]$

e dos outros pares identificados inicialmente não resultam novos pares.

No passo seguintes, são considerados estes novos pares e conclui-se que a partir deles não se identificam mais novos pares. A tabela final obtida é

q_1	⊗								
q_2	⊗								
q_3	⊗	⊗	⊗						
q_4	⊗	⊗	⊗	⊗					
q_5	⊗	⊗	⊗	⊗					
r_0		⊗	⊗	⊗	⊗	⊗			
r_1	⊗			⊗	⊗	⊗	⊗		
r_2	⊗	⊗	⊗		⊗	⊗	⊗	⊗	
r_3	⊗	⊗	⊗	⊗			⊗	⊗	⊗
	q_0	q_1	q_2	q_3	q_4	q_5	r_0	r_1	r_2

Identificados todos os pares de estados distinguíveis, conclui-se que q_0 e r_0 são estados equivalentes. Assim, D_1 e D_2 são equivalentes. △

Estas técnicas são também aplicáveis à minimização de AFDS.

Definição 2.25. AFD MÍNIMO
Um AFD D diz-se *mínimo* se não existe nenhum AFD equivalente a D com menos estados do que D. △

Para construir um AFD mínimo equivalente a um AFD dado vai ser necessário considerar um certa partição do conjunto Q dos seus estados. Uma partição de Q é um conjunto de subconjuntos não vazios de Q, disjuntos dois a dois e cuja união é Q. A partição que é relevante para a construção do AFD mínimo é a partição induzida pelos estados equivalentes do AFD. Cada elemento da partição é o conjunto dos estados que são equivalentes a um dado estado do AFD.

Definição 2.26. PARTIÇÃO INDUZIDA PELOS ESTADOS EQUIVALENTES
Seja D um AFD com conjunto de estados Q. A *partição de Q induzida pelos estados equivalentes* é o conjunto $\{C[q] : q \in Q\}$ em que $C[q] = \{q\} \cup \{p \in Q : p \text{ e } q \text{ equivalentes}\}$ para cada $q \in Q$. △

São relevantes as seguintes propriedades da partição induzida pelos estados equivalentes.

Proposição 2.27.
Seja $D = (\Sigma, Q, q_{in}, F, \delta)$ um AFD e seja C um elemento da partição de Q induzida pelos estados equivalentes.

1. *Se em C existe um estado final então C só tem estados finais.*
2. *Se em C existe um estado produtivo então C só tem estados produtivos.*
3. *Se $\delta(q,a)\uparrow$ para algum $q \in C$ e algum $a \in \Sigma$ então, para cada $q' \in C$, $\delta(q',a)\uparrow$ ou $\delta(q',a)$ não é produtivo.*
4. *Se $\delta(q,a) = p$ para algum $q \in C$, algum $p \in C$ e algum $a \in \Sigma$ então, para cada $q' \in C$, alguma das seguintes condições é verdadeira:*
   - *$\delta(q',a) = p'$ e p equivalente a p';*
   - *$\delta(q',a)\uparrow$ e p' não é produtivo.*

Dem.: Exercício. □

Exemplo 2.28. Recorde-se o AFD D apresentado na Figura 2.9. No Exemplo 2.21 concluiu-se que $\{[q_1,q_2],[q_4,q_5]\}$ é o conjunto de pares de estados equivalente de D. Seguindo a Definição 2.26 tem-se

- $C[q_0] = \{q_0\}$;
- $C[q_3] = \{q_3\}$;
- $C[q_1] = C[q_2] = \{q_1, q_2\}$;
- $C[q_4] = C[q_5] = \{q_4, q_5\}$.

A partição do conjunto de estados de D induzida pelos estados equivalentes é assim $\{\{q_0\},\{q_3\},\{q_1,q_2\},\{q_4,q_5\}\}$. △

Dado um AFD D, mostra-se agora como construir $min(D)$, um AFD mínimo equivalente ao AFD dado. No caso do AFD ser vacuoso, $min(D)$ retém apenas o estado inicial de D. Caso contrário, há que começar por eliminar de D todos os seus eventuais estados inúteis e depois encontrar todos os estados equivalentes do AFD resultante, que se colapsam.

Definição 2.29. MINIMIZAÇÃO DE AFD
Dado um AFD $D = (\Sigma, Q, q_{\text{in}}, F, \delta)$, a *minimização* de D é o AFD $min(D)$ construído como se segue:

- se D é vacuoso então $min(D)$ é o AFD vacuoso que tem apenas o estado inicial q_{in}, sem estados finais, e com função de transição totalmente indefinida;
- se D não é vacuoso, $min(D) = (\Sigma, Q_{\text{m}}, q_{\text{in}}^{\text{m}}, F_{\text{m}}, \delta_{\text{m}})$ em que
  - $Q_{\text{m}} = \{C_0, C_1, \ldots, C_n\}$ é a partição do conjunto dos estados úteis de D induzida pelos estados equivalentes, com $q_{\text{in}} \in C_0$;
  - $q_{\text{in}}^{\text{m}} = C_0$;
  - $F_{\text{m}} = \{C_i \in Q_{\text{m}} : C_i \cap F \neq \emptyset\}$;
  - $\delta_{\text{m}} : Q_{\text{m}} \times \Sigma \to Q_{\text{m}}$ é tal que

$$\delta_{\text{m}}(C_i, a) = \begin{cases} C_j & \text{se } \delta(q,a) \in C_j \text{ para algum } q \in C_i \\ \text{indefinido} & \text{se } \delta(q,a)\uparrow \text{ para algum } q \in C_i \end{cases}$$

  para cada $1 \leq i \leq n$ e $a \in \Sigma$. △

A minimização de D resulta de facto num AFD mínimo equivalente a D.

Proposição 2.30.
Dado um AFD D, $min(D)$ *é um* AFD *mínimo equivalente a* D.

Dem.: Exercício. □

Pode agora concluir-se que se um AFD não vacuoso não tem estados inúteis e não tem estados distintos equivalentes então o AFD é mínimo.

Exemplo 2.31. Considere-se de novo o AFD D referido no Exemplo 2.21. Este autómato não é vacuoso mas tem um estado inútil, o estado q_3. A construção de $min(D)$ é como se segue:

(i) o conjunto dos estados úteis de D é $Q_m = \{q_0, q_1, q_2, q_4, q_5\}$;

(ii) o conjunto dos pares de estados de Q_m equivalentes é $\{[q_0], [q_1, q_2], [q_4, q_5]\}$;

(iii) o conjunto anterior induz a seguinte partição de Q_m

$$\{\{q_0\}, \{q_1, q_2\}, \{q_4, q_5\}\}$$

(iv) a minimização de D é o AFD $min(D)$ representado na Figura 2.11, no qual $C_0 = \{q_0\}$, $C_1 = \{q_1, q_2\}$ e $C_2 = \{q_4, q_5\}$. △

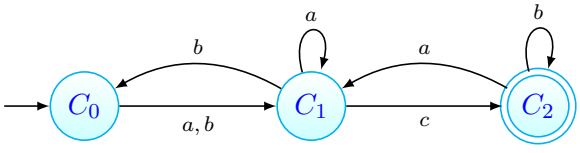


Figura 2.11: Minimização do AFD do Exemplo 2.21.

2.1.3 Autómatos finitos não-deterministas

Estuda-se agora um outro tipo de autómatos finitos: os autómatos finitos não-deterministas. Existem duas diferenças fundamentais nos autómatos finitos não-deterministas. A primeira reside no facto de poderem existir duas ou mais transições associadas a um mesmo símbolo a partir de um único estado. A designação *não-determinista* advém precisamente desta característica. A segunda diferença reside no facto de ser permitido efectuar transições entre estados sem que nenhum símbolo do alfabeto a elas esteja associado, as chamadas transições-ϵ, ou movimentos-ϵ.

Definição 2.32. AUTÓMATO FINITO NÃO-DETERMINISTA
Um *autómato finito não-determinista*, ou apenas AFND$^\epsilon$, é formado por:

- um alfabeto Σ;
- um conjunto finito de estados Q;
- um estado inicial $q_{\text{in}} \in Q$;
- um conjunto de estados finais $F \subseteq Q$;
- uma função de transição (total) $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \wp(Q)$.

Designa-se por autómato finito não-determinista *sem movimentos-ϵ*, ou apenas AFND, qualquer autómato finito não-determinista tal que $\delta(q, \epsilon) = \emptyset$ para cada $q \in Q$. Nesse caso, convenciona-se que a função de transição é simplesmente do tipo $\delta : Q \times \Sigma \to \wp(Q)$. △

Exemplo 2.33. Considere-se o AFND$^\epsilon$ $A^\epsilon = (\Sigma, Q, q_{\text{in}}, F, \delta)$ em que

- $\Sigma = \{a, b, c\}$
- $Q = \{q_{\text{in}}, q_1, q_2, q_3, q_4, q_5\}$
- $F = \{q_5\}$
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \wp(Q)$ é representada através da tabela

δ	a	b	c	ϵ
q_0	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_1, q_3\}$
q_1	$\{q_2\}$	$\{q_1\}$	$\{q_1\}$	$\emptyset$
q_2	$\{q_1\}$	$\{q_2\}$	$\{q_2\}$	$\{q_5\}$
q_3	$\{q_3\}$	$\{q_3\}$	$\{q_3, q_4\}$	$\{q_4\}$
q_4	$\emptyset$	$\emptyset$	$\{q_5\}$	$\emptyset$
q_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

que, tal como anteriormente, pode também ser representado através de um grafo orientado, como ilustrado na Figura 2.12. Tome-se a palavra ab. Não existe nenhuma transição a partir do estado q_{in} associada ao símbolo a. Mas, a partir de q_{in}, pode transitar-se quer para q_1, quer para q_3, através de um movimento-ϵ, e, com mais um movimento-ϵ, para q_4. Isto permite que o autómato possa

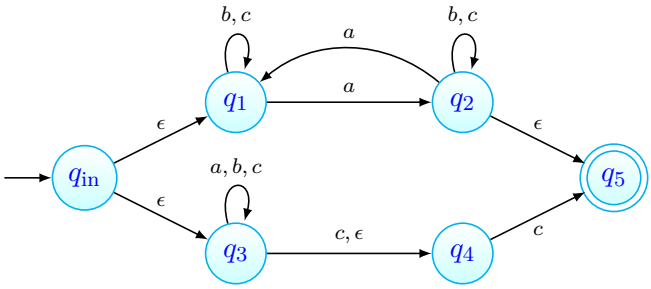


Figura 2.12: AFND$^\epsilon$ para $\{w \in \{a,b,c\}^* :$ tem nº ímpar de as ou termina em $c\}$.

agora realizar uma transição associada ao primeiro símbolo da palavra. Pode realizar quer a transição de q_1 para q_2, quer a transição de q_3 para próprio q_3. Há que considerar de seguida o próximo símbolo da palavra. Existem transições associadas a b quer a partir de q_2, quer a partir de q_3. Ambas conduzem de novo aos próprios estados q_2 e q_3. Este símbolo é o último da palavra, mas nenhum destes estados é estado final. No entanto, através de um movimento-ϵ pode transitar-se para q_5 a partir de q_2, e para q_4 a partir de q_3. O estado q_4 também não é estado final, mas, como q_5 é estado final, a palavra ab é aceite por A^ϵ. De igual modo se conclui que as palavras a, aac e $babacc$, por exemplo, são aceites por A^ϵ. Por sua vez, as palavras b, aa e $abab$ não são aceites por A^ϵ. É fácil concluir que as palavras aceites por A^ϵ são precisamente as palavras sobre o alfabeto $\{a,b,c\}$ que têm um número ímpar de as, ou que terminam em c.

A noção de caminho num AFND$^\epsilon$ e de palavra associada a um caminho é também aqui relevante. Por exemplo, a sequência de estados $q_{\text{in}}q_1q_2q_2q_5$ é um caminho de A^ϵ, ao qual está associada a palavra ab, porque $q_1 \in \delta(q_{\text{in}}, \epsilon)$, $q_2 \in \delta(q_1, a)$, $q_2 \in \delta(q_2, b)$, e $q_5 \in \delta(q_2, \epsilon)$. A palavra ac é outra palavra associada a este caminho. A sequência $q_2q_2q_2q_1q_2$ é um outro exemplo de caminho de A^ϵ, e $bcaa$ é uma das palavras que lhe está associada. Observe-se que, contrariamente ao que acontece nos AFDS, uma palavra pode estar associada a vários caminhos que começam num mesmo estado. Por exemplo, ao caminho $q_{\text{in}}q_3q_3q_3$ está também associada a palavra ab. △

Exemplo 2.34. Na Figura 2.13 encontra-se representado um AFND que reconhece a linguagem das palavras sobre o alfabeto $\{0,1\}$ nas quais o antepenúltimo símbolo e o penúltimo símbolo são iguais. △

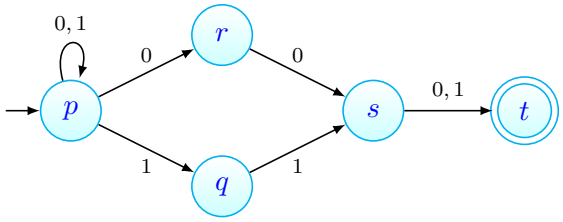


Figura 2.13: AFND para a linguagem $\{wxxz : w \in \{0,1\}^*, x, z \in \{0,1\}\}$.

A ideia subjacente a um AFND$^\epsilon$ como reconhecedor de uma linguagem é muito simples: para determinar se uma dada palavra w pertence à linguagem deve verificar-se se alguma computação iniciada em q_{in} transitando sucessivamente de estado de acordo com δ e cada um dos símbolos de w conduz, ou não, a um estado final. Ao contrário dos AFDS, esta computação pode não ser única.

O não-determinismo permite uma certa compactação dos estados necessários, permitindo modelos mais simples, como se ilustra de seguida. Veremos adiante se esta capacidade traz, ou não, outras novidades. O AFD representado na Figura 2.14 é equivalente ao AFND na Figura 2.13. Ambos reconhecem a linguagem das palavras sobre o alfabeto $\{0,1\}$ nas quais o antepenúltimo símbolo e o penúltimo símbolo são iguais. Enquanto o AFND tem 5 estados e 8 transições, o AFD, que é mínimo (verifique), tem 9 estados e 18 transições.

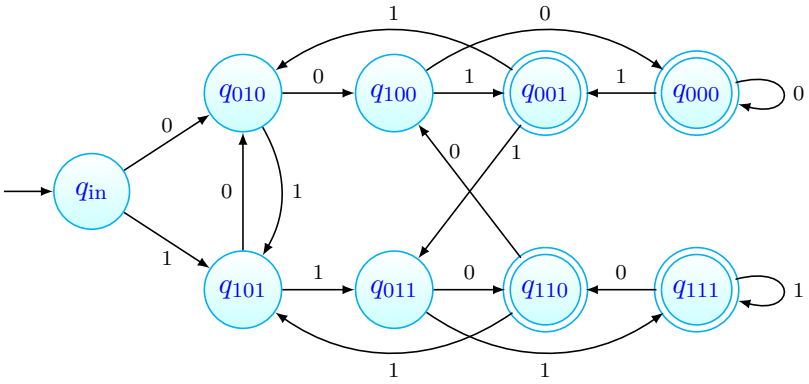


Figura 2.14: AFD para a linguagem $\{wxxz : w \in \{0,1\}^*, x, z \in \{0,1\}\}$.

Definição 2.35. FECHO-ϵ DE UM ESTADO DE AFND$^\epsilon$
Seja $A^\epsilon = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFND$^\epsilon$. O *fecho-ϵ* de um estado $q \in Q$ de A^ϵ é o conjunto $q^\epsilon \subseteq Q$ definido como se segue:

- $q \in q^\epsilon$;
- se $q' \in q^\epsilon$ então $\delta(q', \epsilon) \subseteq q^\epsilon$.

Dado um subconjunto C de Q, C^ϵ é o conjunto $\bigcup_{q \in C} q^\epsilon$. △

O fecho-ϵ do estado q é o conjunto de estados constituído pelo próprio estado q e por todos os estados de cada caminho do AFND$^\epsilon$ que comece por q e a que esteja associada a palavra ϵ. Observe-se que se não existirem movimentos-ϵ então $q^\epsilon = \{q\}$ para cada estado q.

Define-se agora a função de transição estendida de um AFND$^\epsilon$.

Definição 2.36. FUNÇÃO DE TRANSIÇÃO ESTENDIDA DE AFND$^\epsilon$
Seja $A^\epsilon = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFND$^\epsilon$. A *função de transição estendida* de A^ϵ é a função $\delta^* : Q \times \Sigma^* \to \wp(Q)$ tal que

$$\delta^*(q,w) = \begin{cases} q^\epsilon & \text{se } w = \epsilon \\ \bigcup_{q' \in q^\epsilon} \left(\bigcup_{q'' \in \delta(q',a)} \delta^*(q'', w') \right) & \text{se } w = a.w' \end{cases}$$

para cada $q \in Q$, $a \in \Sigma$ e $w \in \Sigma^*$. △

O facto de $p \in \delta^*(q, w)$ significa que em A^ϵ existe pelo menos um caminho que começa em q e termina em p e a que está associada a palavra w.

Definição 2.37. PALAVRA ACEITE E LINGUAGEM RECONHECIDA POR AFND$^\epsilon$ Seja $A^\epsilon = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFND$^\epsilon$. A palavra $w \in \Sigma^*$ diz-se *aceite* por A^ϵ se $\delta^*(q_{\text{in}}, w) \cap F \neq \emptyset$. O conjunto

$$L(A^\epsilon) = \{w \in \Sigma^* : \delta^*(q_{\text{in}}, w) \cap F \neq \emptyset\}$$

é a *linguagem reconhecida por* A^ϵ, ou *linguagem de* A^ϵ. △

Uma palavra w é aceite por A^ϵ se existe um caminho de A^ϵ ao qual está associada w que começa no estado inicial e termina num estado final. A linguagem reconhecida por A^ϵ é o conjunto de todas as palavras aceites por A^ϵ.

Exemplo 2.38. Recorde-se o AFND$^\epsilon$ A^ϵ apresentado no Exemplo 2.33. Tem-se, por exemplo, que

$$q^\epsilon_{\text{in}} = \{q_{\text{in}}, q_1, q_3, q_4\} \qquad q^\epsilon_2 = \{q_2, q_5\} \qquad q^\epsilon_3 = \{q_3, q_4\}$$

e

$$\begin{aligned}
\delta^*(q_{\text{in}}, ab) &= \textstyle\bigcup_{q' \in q^\epsilon_{\text{in}}} (\bigcup_{q'' \in \delta(q', a)} \delta^*(q'', b)) \\
&= (\textstyle\bigcup_{q'' \in \{q_2\}} \delta^*(q'', b)) \cup (\bigcup_{q'' \in \{q_3\}} \delta^*(q'', b)) \\
&= \delta^*(q_2, b) \cup \delta^*(q_3, b) \\
&= (\textstyle\bigcup_{q' \in q^\epsilon_2} (\bigcup_{q'' \in \delta(q', b)} \delta^*(q'', \epsilon))) \cup (\bigcup_{q' \in q^\epsilon_3} (\bigcup_{q'' \in \delta(q', b)} \delta^*(q'', \epsilon))) \\
&= (\textstyle\bigcup_{q'' \in \{q_2\}} \delta^*(q'', \epsilon)) \cup (\bigcup_{q'' \in \{q_3\}} \delta^*(q'', \epsilon)) \\
&= \delta^*(q_2, \epsilon) \cup \delta^*(q_3, \epsilon) \\
&= q^\epsilon_2 \cup q^\epsilon_3 \\
&= \{q_2, q_3, q_4, q_5\}.
\end{aligned}$$

Dado que $\delta^*(q_{\text{in}}, ab) = \{q_2, q_3, q_4, q_5\}$ e q_5 é estado final, a palavra ab pertence à linguagem reconhecida por A^ϵ, como já se havia referido no Exemplo 2.33. △

Uma questão relevante neste ponto é a de saber se os AFND$^\epsilon$s têm ou não mais poder expressivo que os AFNDS, e se estes têm ou não mais poder expressivo que os AFDS. Será que existe alguma linguagem que é reconhecida por um AFND$^\epsilon$ mas nenhum AFND a reconhece? Será que existe alguma linguagem que é reconhecida por um AFND mas nenhum AFD a reconhece?

Na verdade, nestes modelos computacionais muito simples, vamos ver que o não-determinismo e os movimentos-ϵ não trazem expressividade acrescida, ou seja, se uma linguagem é reconhecida por um AFND$^\epsilon$ então existe também um AFND que a reconhece e existe também um AFD que a reconhece. As afirmações recíprocas são obviamente verdadeiras dado que um AFND é um caso particular de AFND$^\epsilon$ e um AFD pode ser visto como um caso particular de AFND. Um AFND é um caso particular de AFND$^\epsilon$ no qual não existem movimentos-ϵ. É assim trivial obter um AFND$^\epsilon$ que reconheça a linguagem de um AFND dado:

basta estender a função de transição atribuindo aos pares (q, ϵ) o conjunto vazio. Um AFD pode ser visto como o caso particular de um AFND em que, para cada estado, existe no máximo uma transição associada a cada símbolo do alfabeto.

Começamos por analisar como eliminar os movimentos-ϵ de um AFND$^\epsilon$, explorando a construção dos fechos-ϵ dos estados, por forma a obter um AFND equivalente.

Definição 2.39. AFND RESULTANTE DE AFND$^\epsilon$
Dado um AFND$^\epsilon$ $A^\epsilon = (\Sigma, Q, q_{\text{in}}, F, \delta)$, o AFND *resultante* é dado por $\mathit{afnd}(A^\epsilon) = (\Sigma, Q, q_{\text{in}}, F', \delta')$ em que

- $F' = \{q \in Q : q^\epsilon \cap F \neq \emptyset\}$;
- $\delta' : Q \times \Sigma \to \wp(Q)$ é tal que

  $$\delta'(q, a) = \bigcup_{q' \in q^\epsilon} \left(\bigcup_{q'' \in \delta(q', a)} q''^\epsilon\right)$$

  para cada $q \in Q$ e $a \in \Sigma$. △

A linguagem reconhecida por $\mathit{afnd}(A^\epsilon)$ é precisamente igual a $L(A^\epsilon)$.

Proposição 2.40.
Dado um AFND$^\epsilon$ A^ϵ, *tem-se* $L(\mathit{afnd}(A^\epsilon)) = L(A^\epsilon)$.

Dem. (esboço): O resultado segue, observando que se $A^\epsilon = (\Sigma, Q, q_{\text{in}}, F, \delta)$ e $\mathit{afnd}(A^\epsilon) = (\Sigma, Q, q_{\text{in}}, F', \delta')$, então $\delta'^*(q, w) = \delta^*(q, w)$ para quaisquer $q \in Q$ e $w \in \Sigma^*$. □

Exemplo 2.41. Recorde-se o AFND$^\epsilon$ $A^\epsilon = (\{a, b, c\}, \{q_{\text{in}}, \ldots, q_5\}, q_{\text{in}}, \{q_5\}, \delta)$ apresentado no Exemplo 2.33. Para construir o autómato $\mathit{afnd}(A^\epsilon)$ há que determinar o conjunto F' dos seus estados finais e a sua função de transição δ'. Conclui-se que $F' = \{q_2, q_5\}$ porque os únicos estados de A^ϵ cujos fechos-ϵ incluem q_5, o único estado final de A^ϵ, são q_2 e q_5 (verifique). Como exemplo, calculam-se de seguida alguns valores de δ':

- $\begin{aligned}\delta'(q_{\text{in}}, a) &= \bigcup_{q' \in q_{\text{in}}^\epsilon} \left(\bigcup_{q'' \in \delta(q', a)} q''^\epsilon\right) \\ &= \bigcup_{q' \in \{q_{\text{in}}, q_1, q_3, q_4\}} \left(\bigcup_{q'' \in \delta(q', a)} q''^\epsilon\right) \\ &= \left(\bigcup_{q'' \in \{q_2\}} q''^\epsilon\right) \cup \left(\bigcup_{q'' \in \{q_3\}} q''^\epsilon\right) \\ &= q_2^\epsilon \cup q_3^\epsilon \\ &= \{q_2, q_5\} \cup \{q_3, q_4\} \\ &= \{q_2, q_3, q_4, q_5\}\end{aligned}$
- $\begin{aligned}\delta'(q_2, a) &= \bigcup_{q' \in q_2^\epsilon} \left(\bigcup_{q'' \in \delta(q', a)} q''^\epsilon\right) \\ &= \bigcup_{q' \in \{q_2, q_5\}} \left(\bigcup_{q'' \in \delta(q', a)} q''^\epsilon\right) \\ &= \bigcup_{q'' \in \{q_1\}} q''^\epsilon \\ &= q_1^\epsilon \\ &= \{q_1\}.\end{aligned}$

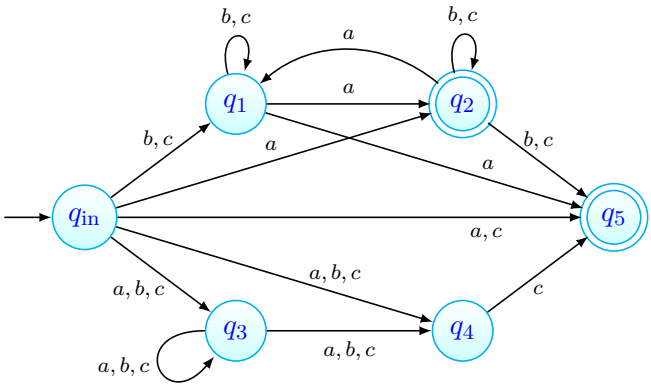


Figura 2.15: AFND equivalente ao AFND$^\epsilon$ do Exemplo 2.33.

Os outros casos obtêm-se de forma semelhante. O autómato $afnd(A^\epsilon)$ é então o apresentado na Figura 2.15. △

Vejamos agora como eliminar o não-determinismo de um AFND por forma a obter um AFD equivalente.

Definição 2.42. AFD RESULTANTE DE AFND
Dado um AFND $A = (\Sigma, Q, q_{\text{in}}, F, \delta)$, o AFD *resultante de* A é dado por $afd(A) = (\Sigma, \wp(Q), \{q_{\text{in}}\}, F', \delta')$ em que

- $F' = \{C \subseteq Q : C \cap F \neq \emptyset\}$;
- $\delta' : \wp(Q) \times \Sigma \to \wp(Q)$ é tal que

$$\delta'(C, a) = \bigcup_{q \in C} \delta(q, a)$$

para cada $C \subseteq Q$ e $a \in \Sigma$. △

Cada estado do autómato $afd(A)$ é um conjunto de estados do AFND A. O conjunto dos estados Q' de $afd(A)$ é finito porque o conjunto dos estados de A é finito. Note-se que o AFD resultante pode conter estados inúteis, facilmente removíveis.

Proposição 2.43.
Dado um AFND A, *tem-se* $L(afd(A)) = L(A)$.

Dem. (esboço): O resultado segue, observando que se $A = (\Sigma, Q, q_{\text{in}}, F, \delta)$ e $afd(A) = (\Sigma, \wp(Q), \{q_{\text{in}}\}, F', \delta')$, então $\delta'^*(C, w) = \bigcup_{q \in C} \delta^*(q, w)$ para qualquer $C \subseteq Q$ e $w \in \Sigma^*$. □

Exemplo 2.44. Seja $A = (\{0, 1\}, \{p, q, r, s, t\}, p, \{t\}, \delta)$ o AFND apresentado no Exemplo 2.34. Vai construir-se o autómato $afd(A)$. O conjunto dos estados deste autómato é $\wp(\{p, q, r, s, t\})$ e o seu estado inicial é $\{p\}$. Os estados finais são todos os subconjunto de $\{p, q, r, s, t\}$ que incluem t. Como exemplo, calculam-se alguns valores de δ':

- $\delta'(\{p\},0) = \bigcup_{q' \in \{p\}} \delta(q',0)$
  $= \delta(p,0)$
  $= \{p,r\}$

- $\delta'(\{p,r\},0) = \bigcup_{q' \in \{p,r\}} \delta(q',0)$
  $= \delta(p,0) \cup \delta(r,0)$
  $= \{p,r\} \cup \{s\}$
  $= \{p,r,s\}$

Os outros casos obtêm-se de forma semelhante. Muitos dos estados de $afd(A)$ são inúteis. Na Figura 2.16 está representado o autómato $afd(A)$, já expurgado dos seus estados inúteis (confirme). Observe-se que este autómato coincide com o apresentado na Figura 2.14. △

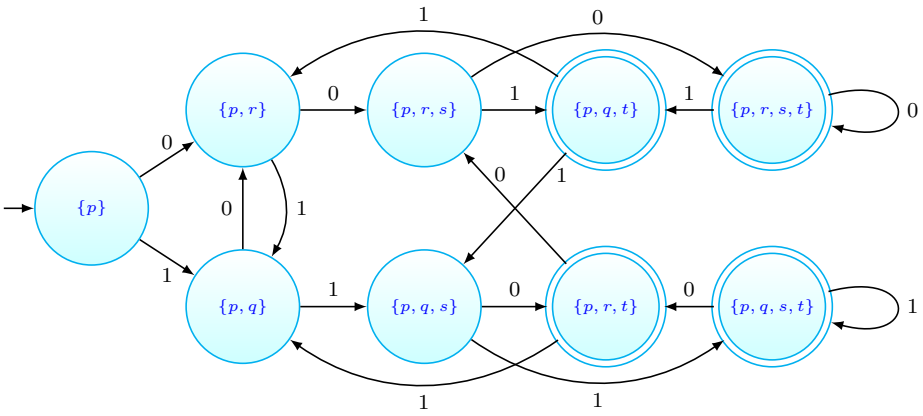


Figura 2.16: AFD equivalente ao AFND do Exemplo 2.34.

Note-se em conclusão que apesar de os modelos serem equivalentes, ou seja, que as linguagens reconhecidas por AFNDS$^\epsilon$ ou AFNDS são necessariamente linguagens regulares, é claro que a função de transição estendida de um AFD é bem mais simples. No entanto, para além de os AFNDS e AFNDS$^\epsilon$ terem em geral menos estados e transições que AFDS equivalentes, os movimentos-ϵ permitem a construção modular de reconhecedores para certas linguagens, como veremos adiante.

2.1.4 Propriedades das linguagens regulares

A classe das linguagens regulares é muito bem comportada, suportando uma quantidade de construções. Comecemos por analisar as suas propriedades lógicas.

Proposição 2.45.
Seja Σ um alfabeto e $L, L_1, L_2 \subseteq \Sigma^*$ linguagens regulares. Então, $\overline{L}$ e $L_1 \cap L_2$ são linguagens regulares.

Dem.: Seja $D = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFD tal que $L(D) = L$. Assuma-se, sem perda de generalidade que a função de transição de D é total, e considere-se o AFD *dual* $\overline{D} = (\Sigma, Q, q_{\text{in}}, Q \setminus F, \delta)$. É imediato verificar que $L(\overline{D}) = \overline{L(D)} = \overline{L}$.

Analogamente, sejam $D_1 = (\Sigma, Q_1, q^1_{\text{in}}, F_1, \delta_1)$ e $D_2 = (\Sigma, Q_2, q^2_{\text{in}}, F_2, \delta_2)$ AFDS tais que $L(D_1) = L_1$ e $L(D_2) = L_2$. Considere-se o AFD *produto* $D_1 \times D_2 = (\Sigma, Q_1 \times Q_2, (q^1_{\text{in}}, q^2_{\text{in}}), F_1 \times F_2, \delta)$ onde $\delta : Q \times \Sigma \to Q$ é tal que, para cada $(q_1, q_2) \in Q$ e $a \in \Sigma$:

- $\delta((q_1, q_2), a) = \begin{cases} (\delta_1(q_1, a), \delta_2(q_2, a)) & \text{se } \delta_1(q_1, a)\downarrow \text{ e } \delta_2(q_2, a)\downarrow \\ \text{indefinido} & \text{caso contrário} \end{cases}$.

É imediato verificar que $L(D_1 \times D_2) = L(D_1) \cap L(D_2)$. □

Obviamente, daqui conclui-se que a classe das linguagens regulares não só é fechada para intersecções e complementações, mas também para outras operações lógicas, como união, ou complementação relativa, já que, por exemplo, $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$ e $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$.

Exemplo 2.46.
Seja D o AFD (total) apresentado no Exemplo 2.5 para a linguagem L das palavras sobre $\{a, b, c\}$ que terminam em ab. Na Figura 2.17 está representado o seu autómato dual $\overline{D}$, o qual reconhece a linguagem $\overline{L}$ das palavras sobre $\{a, b, c\}$ que não terminam em ab. △

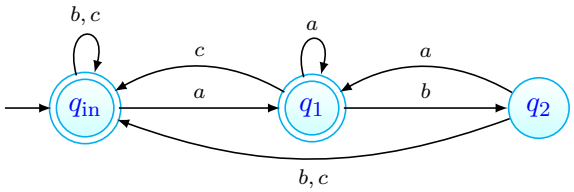


Figura 2.17: AFD dual do AFD do Exemplo 2.5.

Exemplo 2.47.
Recorde-se o AFD $\overline{D}$ construído no Exemplo 2.46 para a linguagem $\overline{L}$ das palavras sobre $\{a, b, c\}$ que não terminam em ab. Na Figura 2.18, o AFD D' à esquerda reconhece a linguagem L' das palavras sobre $\{a, b, c\}$ que têm um número par de as. O AFD à direita é o autómato produto $\overline{D} \times D'$ que reconhece a linguagem $\overline{L} \cap L'$, isto é, a linguagem das palavras sobre $\{a, b, c\}$ que não terminam em ab e têm um número par de as. △

Na verdade, é possível obter algumas destas, e outras, propriedades de fecho da classe das linguagens regulares de forma simples, recorrendo a AFNDS$^\epsilon$ em vez de AFDS.

Proposição 2.48.
Seja Σ um alfabeto e $L, L_1, L_2 \subseteq \Sigma^*$ linguagens regulares. Então, $L_1 \cup L_2$, $L_1.L_2$ e L^* são linguagens regulares.

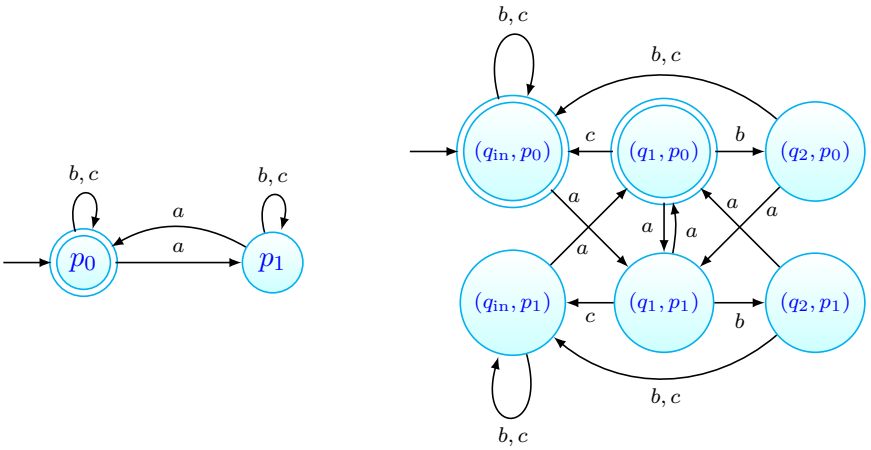


Figura 2.18: AFD para a linguagem $\{w \in \{a,b,c\}^* : w \text{ tem nº par de } as\}$ (à esquerda) e exemplo de AFD produto (à direita).

Dem.: Sejam $A_1^\epsilon = (\Sigma, Q_1, q_{\text{in}}^1, F_1, \delta_1)$ e $A_2^\epsilon = (\Sigma, Q_2, q_{\text{in}}^2, F_2, \delta_2)$ AFNDS$^\epsilon$ tais que $L(A_1^\epsilon) = L_1$ e $L(A_2^\epsilon) = L_2$ e, sem perda de generalidade, $Q_1 \cap Q_2 = \emptyset$.

Considere-se o AFND$^\epsilon$ *soma* $A_1^\epsilon + A_2^\epsilon = (\Sigma, Q, q_{\text{in}}, F_1 \cup F_2, \delta)$ em que:

- $Q = \{q_{\text{in}}\} \cup Q_1 \cup Q_2$ com $q_{\text{in}} \notin Q_1 \cup Q_2$;
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \wp(Q)$ é tal que, para cada $q \in Q$ e $a \in \Sigma \cup \{\epsilon\}$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 \\ \delta_2(q,a) & \text{se } q \in Q_2 \\ \{q_{\text{in}}^1, q_{\text{in}}^2\} & \text{se } q = q_{\text{in}} \text{ e } a = \epsilon \\ \emptyset & \text{se } q = q_{\text{in}} \text{ e } a \neq \epsilon \end{cases}.$$

É simples verificar que $L(A_1^\epsilon + A_2^\epsilon) = L(A_1^\epsilon) \cup L(A_1^\epsilon) = L_1 \cup L_2$.

Considere-se também o AFND$^\epsilon$ *sequencial* $A_1^\epsilon.A_2^\epsilon = (\Sigma, Q_1 \cup Q_2, q_{\text{in}}^1, F_2, \delta)$ em que:

- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \wp(Q)$ é tal que, para cada $q \in Q$ e $a \in \Sigma \cup \{\epsilon\}$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 \backslash F_1, \text{ ou } q \in F_1 \text{ e } a \neq \epsilon \\ \delta_1(q,\epsilon) \cup \{q_{\text{in}}^2\} & \text{se } q \in F_1 \text{ e } a = \epsilon \\ \delta_2(q,a) & \text{se } q \in Q_2 \end{cases}.$$

É simples verificar que $L(A_1^\epsilon.A_2^\epsilon) = L(A_1^\epsilon).L(A_1^\epsilon) = L_1.L_2$.

Por fim, seja $A^\epsilon = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFND$^\epsilon$ tal que $L(A^\epsilon) = L$, e considere-se o AFND$^\epsilon$ *iteração* $(A^\epsilon)^* = (\Sigma, Q', s, F', \delta')$ em que:

- $Q' = \{s\} \cup Q$ com $s \notin Q$;
- $F' = \{s\} \cup F$;

- $\delta' : Q' \times (\Sigma \cup \{\epsilon\}) \to \wp(Q')$ é tal que, para cada $q \in Q'$ e $a \in \Sigma \cup \{\epsilon\}$:

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{se } q \in Q \backslash F \\ \delta(q,\epsilon) \cup \{s\} & \text{se } q \in F \text{ e } a = \epsilon \\ \delta(q,a) & \text{se } q \in F \text{ e } a \neq \epsilon \\ \{q_{\text{in}}\} & \text{se } q = s \text{ e } a = \epsilon \\ \emptyset & \text{se } q = s \text{ e } a \neq \epsilon \end{cases} .$$

É simples verificar que $L((A^\epsilon)^*) = L(A^\epsilon)^* = L^*$. □

Exemplo 2.49.
Recorde-se o autómato com alfabeto $\{a, b, c\}$ para a linguagem das palavras que terminam em ab apresentado no Exemplo 2.5, bem como o autómato com o mesmo alfabeto para a linguagem das palavras que têm um número par de as apresentado no Exemplo 2.47. Na Figura 2.19 está representado o AFND$^\epsilon$ soma que resulta destes autómatos, o qual reconhece a união das suas linguagens. Designou-se por q o estado inicial do novo autómato. △

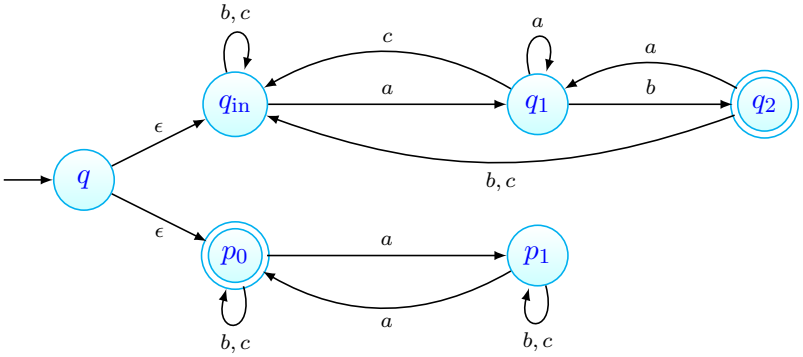


Figura 2.19: Exemplo de AFND$^\epsilon$ soma.

Exemplo 2.50.
Recorde-se o autómato para a linguagem das palavras nas quais o antepenúltimo e o penúltimo símbolos são iguais apresentado no Exemplo 2.34, bem como o autómato para a linguagem das palavras que têm um número par de as apresentado no Exemplo 2.47. Sem perda de generalidade pode assumir-se que ambos têm alfabeto $\{0, 1, a, b, c\}$. Na Figura 2.20 está representado o AFND$^\epsilon$ sequencial que resulta destes autómatos que reconhece a concatenação da linguagem do primeiro autómato com a linguagem do segundo. △

Exemplo 2.51.
Recorde-se o autómato apresentado no Exemplo 2.5 para a linguagem L das palavras sobre $\{a, b, c\}$ que terminam em ab. Na Figura 2.21 está representado o AFND$^\epsilon$ iteração que resulta deste autómato, o qual reconhece a linguagem L^*, o fecho de Kleene de L. △

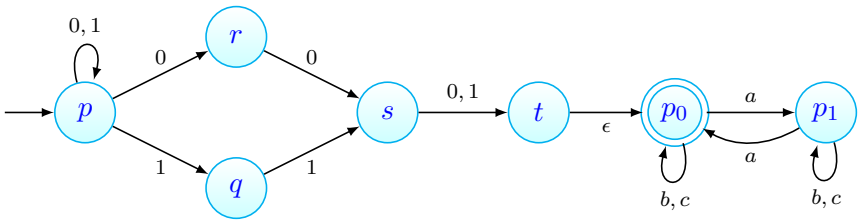


Figura 2.20: Exemplo de AFND$^{\epsilon}$ sequencial.

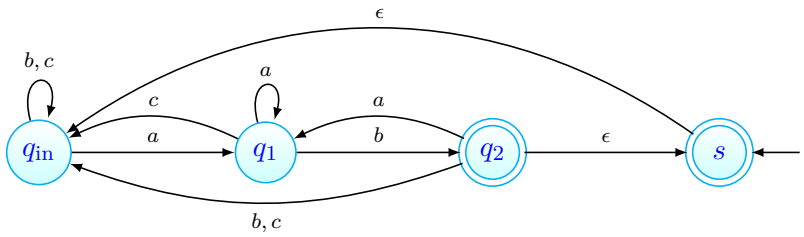


Figura 2.21: Exemplo de AFND$^{\epsilon}$ iteração.

Os resultados anteriores são cruciais para a ideia de *expressão regular*, uma forma compacta de definir linguagens regulares.

Definição 2.52. EXPRESSÕES REGULARES
Seja Σ um alfabeto. O *conjunto das expressões regulares sobre* Σ é o conjunto R_Σ definido indutivamente como se segue:

- $\emptyset \in R_\Sigma$;
- $w \in R_\Sigma$ para cada $w \in \Sigma^*$;
- se $\alpha_1, \alpha_2 \in R_\Sigma$ então $(\alpha_1 + \alpha_2) \in R_\Sigma$ (*soma*);
- se $\alpha_1, \alpha_2 \in R_\Sigma$ então $(\alpha_1.\alpha_2) \in R_\Sigma$ (*concatenação*);
- se $\alpha \in R_\Sigma$ então $(\alpha^*) \in R_\Sigma$ (*fecho de Kleene*). △

Para simplificar a escrita de expressões regulares omitem-se os parênteses mais exteriores e assume-se que o fecho de Kleene tem precedência sobre a concatenação e esta sobre a soma. Pode também omitir-se o . relativo à operação de concatenação. Assim, por exemplo, pode escrever-se

- $0 + 1$ em vez de $(0 + 1)$;
- $(0 + 1)0 + 1$ em vez de $(((0 + 1).0) + 1)$;
- 10^* em vez de $(1.(0^*))$.

Definição 2.53. LINGUAGEM DE EXPRESSÃO REGULAR, EQUIVALÊNCIA
Dada uma expressão regular $\alpha \in R_\Sigma$, a *linguagem denotada por* α é o conjunto $L(\alpha) \subseteq \Sigma^*$ definido indutivamente como se segue:

- $L(\emptyset) = \emptyset$;

- $L(w) = \{w\}$ para cada $w \in \Sigma^*$;
- $L(\alpha_1 + \alpha_2) = L(\alpha_1) \cup L(\alpha_2)$;
- $L(\alpha_1.\alpha_2) = L(\alpha_1).L(\alpha_2)$;
- $L(\alpha^*) = L(\alpha)^*$.

Expressões regulares α_1 e α_2 dizem-se *equivalentes*, o que se representa por $\alpha_1 = \alpha_2$, precisamente se $L(\alpha_1) = L(\alpha_2)$. △

Proposição 2.54.
Para quaisquer expressões regulares α, β e γ, tem-se:

- $\alpha + \beta = \beta + \alpha$
- $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
- $\alpha(\beta\gamma) = (\alpha\beta)\gamma$
- $\alpha\epsilon = \epsilon\alpha = \alpha$
- $\alpha\emptyset = \emptyset\alpha = \emptyset$
- $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$
- $(\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$
- $\alpha + \alpha = \alpha$
- $\alpha\alpha^* = \alpha^*\alpha$
- $\alpha + \emptyset = \alpha$
- $\alpha^* + \epsilon = \alpha^*$
- $\alpha^* + \alpha\alpha^* = \alpha^*$
- $\epsilon + \alpha\alpha^* = \alpha^*$
- $(\alpha\beta)^* = \epsilon + \alpha(\beta\alpha)^*\beta$
- $\emptyset^* = \epsilon$
- $(\alpha^*)^* = \alpha^*$

Dem.: Exercício. □

O resultado abaixo garante que é possível obter reconhecedores para as linguagens definidas por expressões regulares, de forma modular a partir das construções vistas acima.

Proposição 2.55.
Seja α uma expressão regular. Então, $L(\alpha)$ é uma linguagem regular.

Dem. (esboço): Imediato, a partir da Definição 2.53 e da Proposição 2.48. □

Além do mais, de facto, toda a linguagem regular pode ser denotada por uma expressão regular.

Proposição 2.56.
Seja $L \subseteq \Sigma^*$ uma linguagem regular. Então, existe $\alpha \in R_\Sigma$ tal que $L(\alpha) = L$.

Dem. (esboço): Considera-se um AFD que reconheça a linguagem, a partir do qual se obtém um sistema de equações cujas variáveis são os estados do AFD, e cuja solução constitui uma expressão regular para a linguagem reconhecida no AFD a partir de cada um dos seus estados. A expressão regular α é a que corresponde ao estado inicial do AFD. O sistema de equações resolve-se recorrendo ao facto de que a solução para a equação $X = \beta X + \gamma$ é $X = \beta^*\gamma$. □

Exemplo 2.57.
A partir do AFD D do Exemplo 2.2 obtém-se o seguinte sistema de equações:

$$\begin{cases} q_{\text{in}} = aq_1 \\ q_1 = aq_1 + bq_2 + cq_2 + \epsilon \\ q_2 = bq_2 + cq_2 + aq_1 \end{cases}$$

A primeira equação resulta da transição $\delta(q_{\text{in}}, a) = q_1$, e as demais obtêm-se de modo análogo. A segunda equação inclui ϵ pelo facto de q_1 ser estado final. Segue-se a resolução deste sistema, a qual envolve algumas das propriedades das expressões regulares referidas na Proposição 2.54, bem como o facto de que, sendo β e γ expressões regulares, a solução para a equação $X = \beta X + \gamma$ ser $X = \beta^*\gamma$:

$$\begin{cases} q_{\text{in}} = aq_1 \\ q_1 = aq_1 + bq_2 + cq_2 + \epsilon \\ q_2 = bq_2 + cq_2 + aq_1 \end{cases} \Leftrightarrow \begin{cases} q_{\text{in}} = aq_1 \\ q_1 = aq_1 + (b+c)q_2 + \epsilon \\ q_2 = (b+c)q_2 + aq_1 \end{cases} \Leftrightarrow$$

$$\begin{cases} q_{\text{in}} = aq_1 \\ q_1 = aq_1 + (b+c)q_2 + \epsilon \\ q_2 = (b+c)^*aq_1 \end{cases} \Leftrightarrow \begin{cases} q_{\text{in}} = aq_1 \\ q_1 = aq_1 + (b+c)(b+c)^*aq_1 + \epsilon \\ q_2 = (b+c)^*aq_1 \end{cases} \Leftrightarrow$$

$$\begin{cases} q_{\text{in}} = aq_1 \\ q_1 = (a + (b+c)(b+c)^*a)q_1 + \epsilon \\ q_2 = (b+c)^*aq_1 \end{cases} \Leftrightarrow \begin{cases} q_{\text{in}} = aq_1 \\ q_1 = (a + (b+c)(b+c)^*a)^*\epsilon \\ q_2 = (b+c)^*aq_1 \end{cases} \Leftrightarrow$$

$$\begin{cases} q_{\text{in}} = a(a + (b+c)(b+c)^*a)^* \\ q_1 = (a + (b+c)(b+c)^*a)^* \\ q_2 = (b+c)^*aq_1 \end{cases}$$

A expressão regular α correspondente ao estado inicial é $a(a + (b + c)(b + c)^*a)^*$. Esta expressão, que é equivalente a $a + a(b + c)^*a$ (verifique), denota a linguagem reconhecida por D. △

De todo o modo, como deve ser claro já neste momento, nem todas as linguagens são regulares. Para mostrar que uma linguagem L não é regular há que garantir que não existe nenhum autómato finito que a reconheça, ou expressão regular que a denote. A seguinte proposição enuncia um resultado, conhecido como *lema da bombagem* que é útil para esse efeito.

Proposição 2.58.
Se $L \subseteq \Sigma^*$ é uma linguagem regular então existe $k \in \mathbb{N}$ tal que se $w \in L$ é uma palavra com $|w| \geq k$ então $w = w_1w_2w_3$ em que $w_1, w_2, w_3 \in \Sigma^*$ satisfazem as seguintes condições:

- $w_2 \neq \epsilon$;
- $|w_1w_2| \leq k$;

- $w_1 w_2^i w_3 \in L$, *para cada* $i \in \mathbb{N}_0$.

Dem.: Seja $D = (\Sigma, Q, q_{\text{in}}, F, \delta)$ um AFD tal que para $L(D) = L$, e tome-se $k = \#Q$. Tem-se $k \in \mathbb{N}$, pois Q é necessariamente finito e não vazio. Se L não contém nenhuma palavra de comprimento k ou maior, o resultado fica imediatamente estabelecido. Considere-se agora o caso em que existe $w \in L$ com $|w| = n \geq k$.

Como $\delta^*(q_{\text{in}}, w) \in F$, existe um caminho $p_0 p_1 \dots p_n$ de D ao qual w está associada no qual $p_0 = q_{\text{in}}$ e $p_n \in F$. Este caminho tem $n+1$ estados e portanto, como Q tem apenas k elementos, existe um estado $q \in Q$ que ocorre pelo menos duas vezes neste caminho sendo que, necessariamente, duas ocorrências de q estão no caminho $p_0 p_1 \dots p_k$. Existem assim $j_1, j_2 \in \mathbb{N}_0$ com $j_1 < j_2 \leq k$ tais que $p_{j_1} = p_{j_2} = q$.

Ao caminho $p_{j_1} p_{j_1+1} \dots p_{j_2}$ está associada uma palavra w_2 que é necessariamente não vazia porque $j_1 \neq j_2$. Aos caminhos $p_0 p_1 \dots p_{j_1}$ e $p_{j_2} p_{j_2+1} \dots p_n$ estão associadas palavras w_1 e w_3, respectivamente. Assim, $w = w_1 w_2 w_3$ e $w_1 w_2$ tem no máximo comprimento k.

Como $p_{j_1} = p_{j_2} = q$,

$$p_0 \dots p_{j_1} p_{j_2+1} p_{j_2+2} \dots p_n$$

é ainda um caminho de D e a este caminho está associada a palavra $w_1 w_3$, pelo que $w_1 w_3 = w_1 w_2^0 w_3 \in L$.

Dado $i \in \mathbb{N}$,

$$p_0 \dots p_{j_1} p^1_{j_1+1} \dots p^1_{j_2} p^2_{j_1+1} \dots p^2_{j_2} \dots p^{i-1}_{j_2} p^i_{j_1+1} \dots p^i_{j_2} \, p_{j_2+1} \, p_{j_2+2} \dots p_n$$

em que, para cada $1 \leq r \leq i$, $p^r_{j_1+1} = p_{j_1+1}$, $p^r_{j_1+2} = p_{j_1+2}$,..., $p^r_{j_2} = p_{j_2}$, é ainda um caminho de D. Este caminho é obtido repetindo i vezes a sequência $p_{j_1+1} \dots p_{j_2}$. A este caminho está associada a palavra $w_1 w_2^i w_3$, e portanto $w_1 w_2^i w_3 \in L$. □

Ilustra-se agora a utilização da Proposição 2.58 na demonstração de que não é regular a linguagem das palavras sobre $\Sigma = \{a, b\}$ constituídas por uma sequência de as seguida por uma sequência com igual número de bs, isto é, a linguagem $\{a^n b^n : n \in \mathbb{N}_0\}$.

Exemplo 2.59.
A linguagem $L = \{a^n b^n : n \in \mathbb{N}_0\}$ não é regular. Se L fosse uma linguagem regular então verificaria o resultado enunciado na Proposição 2.58. Sendo k o natural cuja existência é garantida por esse resultado, considere-se $w = a^k b^k \in L$. O resultado garante que $a^k b^k = w_1 w_2 w_3$ em que $w_1 w_2$ tem no máximo k símbolos. Então, $w_1 w_2$ e w_2 são sequências não vazias que apenas têm as, pelo que $w_2 = a^j$ com $j \neq 0$. O resultado implica ainda que, em particular, $w_1 w_3$ é uma palavra desta linguagem. Isto significa que $a^{k-j} b^k$ tem de ser um elemento da linguagem, o que é impossível uma vez que $k - j \neq k$. Conclui-se assim que a linguagem não é regular. △

Outros exemplos de linguagens não regulares, ainda sobre o alfabeto $\Sigma = \{a, b\}$ são a linguagem das palavras nas quais o número de as é igual ao número de bs, a linguagem das palavras nas quais o número de as é o dobro do número de bs, ou a linguagem dos palíndromos. Para reconhecer linguagens não regulares precisaremos de modelos computacionais mais poderosos.

2.2 Linguagens independentes do contexto

Esta secção é dedicada às linguagens ditas *independentes do contexto*, com características mais sofisticadas que as linguagens regulares, e aos modelos computacionais que as reconhecem.

2.2.1 Autómatos de pilha

O modelo computacional que consideraremos é o chamado *autómato de pilha* (em inglês, *push-down automaton*).

Um autómato de pilha é basicamente um AFND$^\epsilon$ ao qual se adiciona uma estrutura adicional: uma pilha. Uma pilha é uma estrutura de armazenamento de dados na qual se pode sempre colocar mais um elemento e cujo acesso segue a disciplina FILO (*First In, Last Out*). Isto significa que, em cada momento, o único elemento a que se tem acesso (para remoção ou consulta) é o último elemento que foi colocado na pilha, o qual é designado por *topo* da pilha. Quando se coloca um elemento na pilha, este passa a constituir o topo da nova pilha. Quando se remove o topo de uma pilha altera-se o seu conteúdo e o novo topo corresponde ao penúltimo elemento que se havia colocado, se existir. Caso contrário, obtém-se uma pilha *vazia*.

Uma transição num autómato de pilha corresponde a uma transição no AFND$^\epsilon$ subjacente e a uma possível alteração da pilha: remoção do topo e/ou introdução de um novo elemento. A pilha proporciona assim uma capacidade de memória adicional.

Definição 2.60. AUTÓMATO DE PILHA
Um *autómato de pilha*, ou apenas AP, é formado por:

- um alfabeto Σ;
- um alfabeto *auxiliar* Γ;
- um conjunto finito de estados Q;
- um estado inicial $q_{\text{in}} \in Q$;
- um conjunto de estados finais $F \subseteq Q$;
- uma função de transição $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to \wp(Q \times (\Gamma \cup \{\epsilon\}))$.

△

Vale a pena entender que num AP, se $(q', b') \in \delta(q, a, b)$ isto significa que existe uma transição do estado q para o estado q' quando o símbolo correspondente da palavra a reconhecer é a, desde que o símbolo no topo da pilha nesse momento seja b, que se remove e substitui por b'. Note-se que é possível ter $a = \epsilon$ como num AFND$^\epsilon$. Também é possível ter $b = \epsilon$, o que não impõe nenhum requisito ao símbolo no topo da pilha, nem a sua remoção, ou $b' = \epsilon$, o que não coloca nenhum símbolo na pilha. Obviamente, a pilha acumula símbolos do alfabeto auxiliar Γ, e é portanto representada por uma palavra em Γ^*.

Definição 2.61. PALAVRA ACEITE E LINGUAGEM RECONHECIDA POR AP
Seja $M = (\Sigma, \Gamma, Q, q_{\text{in}}, F, \delta)$ um AP. A palavra $w \in \Sigma^*$ diz-se *aceite* por M se:

- w pode ser escrita como $a_1 a_2 \dots a_m$, com $a_i \in \Sigma \cup \{\epsilon\}$, $1 \leq i \leq m$;
- existem uma sequência de estados $p_0 p_1 \dots p_m$ e uma sequência de pilhas $\gamma_0 \gamma_1 \dots \gamma_m$, com $p_j \in Q$ e $\gamma_j \in \Gamma^*$ para cada $0 \leq j \leq m$, tais que
  - $p_0 = q_{\text{in}}$ e $\gamma_0 = \epsilon$
  - $(p_{i+1}, b') \in \delta(p_i, a_{i+1}, b)$, com $\gamma_i = \gamma b$, $\gamma_{i+1} = \gamma b'$, para $0 \leq i < m$;
  - $p_m \in F$. △

A *linguagem reconhecida por* M, ou *linguagem de* M, denota-se por $L(M)$ e é constituída por todas as palavras aceites por M.

As linguagens que são reconhecidas por APS têm uma designação especial.

Definição 2.62. LINGUAGEM INDEPENDENTE DO CONTEXTO
Uma linguagem $L \subseteq \Sigma^*$ diz-se *independente do contexto* se existe um AP M com alfabeto Σ tal que $L(M) = L$. Denota-se por $\mathcal{INDC}^\Sigma$ o conjunto de todas as linguagens independentes do contexto com alfabeto Σ. △

De novo, usaremos apenas $\mathcal{INDC}$, em vez de $\mathcal{INDC}^\Sigma$, quando o alfabeto estiver subentendido ou seja irrelevante no contexto.

Exemplo 2.63.
Na Figura 2.22 está representado um AP $M = (\Sigma, \Gamma, Q, q_{\text{in}}, F, \delta)$ com alfabeto $\Sigma = \{a, b\}$ e alfabeto auxiliar $\Gamma = \{a, x\}$. Observe-se a notação usada nas

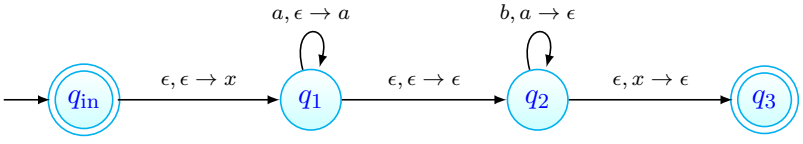


Figura 2.22: AP que reconhece a linguagem $\{a^n b^n : n \in \mathbb{N}_0\}$.

etiquetas das transições: a etiqueta $b, a \to \epsilon$ na transição de q_2 para q_2, por exemplo, denota que $(q_2, \epsilon) \in \delta(q_2, b, a)$. Isto significa que existe uma transição do estado q_2 para o mesmo estado associada ao símbolo b, desde que o símbolo a esteja no topo da pilha, e ainda que da transição resulta que o símbolo no topo da pilha é substituído por ϵ (isto é, a é removido da pilha).

Analisando o autómato conclui-se que transições associadas ao símbolo a podem ser realizadas independentemente do conteúdo da pilha e colocam a no topo da pilha, e que transições associadas ao símbolo b só podem efetuar-se se existir o símbolo a no topo pilha, e removem-no do topo da pilha. O movimento-ϵ a partir de q_{in} coloca o símbolo x na pilha, e o movimento-ϵ que conduz ao estado final só pode realizar-se se x estiver no topo da pilha.

Tome-se a palavra $aabb$. Esta palavra pode ser escrita como $\epsilon aa\epsilon bb\epsilon$. A partir de q_{in} pode transitar-se para q_1 através de um movimento-ϵ, e x é colocado no topo da pilha. Pode agora realizar-se uma transição a partir de q_1 para o próprio q_1 associada ao símbolo a, e colocar a no topo da pilha. O conteúdo da pilha é agora xa. O próximo símbolo da palavra é de novo a, e é realizada a mesma transição de que resulta a pilha xaa. Um movimento-ϵ permite depois transitar para q_2, sem alterar a pilha. O símbolo seguinte da palavra é b, e pode efetuar-se a transição de q_2 para q_2 associada a b porque o símbolo a está no topo da pilha. O topo é removido, e o conteúdo da pilha é agora xa. O mesmo acontece relativamente ao próximo símbolo da palavra, que é de novo b. Como depois o conteúdo da pilha é x, pode realizar-se o movimento-ϵ a partir de q_2 para o estado final q_3. A palavra $aabb$ é assim aceite por M. De igual modo se pode concluir que ϵ, ab e $aaabbb$, por exemplo, são aceites por M.

Por sua vez, a palavra aab não é aceite por M porque após a transição associada ao último b, haverá ainda um a no topo da pilha, o que impede que se venha a realizar o movimento-ϵ para o estado final. Após as transições associadas a a, a pilha tem mais as do que o número de bs na palavra. Também a palavra $aabbb$ não é aceite por M, neste caso porque, após a transição associada ao segundo b, o símbolo a não está no topo da pilha e nunca poderá vir a estar, pelo que o terceiro b não poderá ser reconhecido. Após as transições associadas a a a pilha tem neste caso menos as do que o número de bs na palavra. Palavras só com bs ou só com as também não são aceites por M, tal como palavras nas quais exista um a depois de um b, ou um b depois de um a. Conclui-se então facilmente que as palavras aceites por M são precisamente todas as palavras do tipo $a^n b^n$ com $n \in \mathbb{N}$. A manipulação da pilha assegura que o número de as nas palavras é igual ao número de bs, e as transições entre estados asseguram que bs só podem ocorrer após todos os as.

Pode considerar-se também a noção de caminho num AP e a de palavra associada a um caminho. Por exemplo, a sequência de estados $q_{\text{in}}q_1q_1q_1q_2q_2q_2q_3$ é um caminho de M, ao qual está associada a palavra $aabb$, porque porque $(q_1, x) \in \delta(q_{\text{in}}, \epsilon, \epsilon)$, $(q_1, a) \in \delta(q_1, a, \epsilon)$, $(q_1, a) \in \delta(q_1, a, \epsilon)$, $(q_2, \epsilon) \in \delta(q_1, \epsilon, \epsilon)$, $(q_2, \epsilon) \in \delta(q_2, b, a)$, $(q_2, \epsilon) \in \delta(q_2, b, a)$ e $(q_3, \epsilon) \in \delta(q_2, \epsilon, x)$. △

Deixa-se como exercício ao leitor verificar que todas as linguagens regulares são independentes do contexto.

Como referido no início do capítulo, certos constituintes das linguagens de programação são linguagens regulares, nomeadamente as linguagens das palavras reservadas, ou dos identificadores. Tal não acontece quando se considera toda a linguagem. Nas expressões aritméticas, por exemplo, o número de parênteses direitos tem de ser igual ao número de parênteses esquerdos, e portanto a linguagem das expressões aritméticas não é regular. Tendo como

propósito facilitar a construção de compiladores, os aspetos essenciais da sintaxe das linguagens de programação modernas podem ser especificados através de gramáticas independentes do contexto que, como se verá adiante, descrevem linguagens independentes do contexto.

2.2.2 Propriedades das linguagens independentes do contexto

As linguagens independentes do contexto mantêm algumas boas propriedades das linguagens regulares. Nomeadamente, é possível usar construções semelhantes às obtidas na Secção 2.1.4, mas agora sobre APS, para garantir as seguintes propriedades.

Proposição 2.64.
Seja Σ um alfabeto e $L, L_1, L_2 \subseteq \Sigma^*$ linguagens independentes do contexto. Então, $L_1 \cup L_2$, $L_1.L_2$ e L^* são linguagens independentes do contexto.

Dem.: Exercício. □

No entanto, em geral, a classe das linguagens independentes do contexto não é fechada para intersecções ou complementações, como veremos adiante. Antes, porém, precisamos de compreender que há linguagens que não são independentes do contexto. Para tal dispomos do seguinte resultado, conhecido como *lema da bombagem para linguagens independentes do contexto*.

Proposição 2.65.
Se $L \subseteq \Sigma^*$ é uma linguagem independente do contexto então existe $k \in \mathbb{N}$ tal que se $w \in L$ é uma palavra com $|w| \geq k$ símbolos então $w = w_1w_2w_3w_4w_5$ em que $w_1, w_2, w_3, w_4, w_5 \in \Sigma^*$ satisfazem as seguintes condições:

- $w_2w_4 \neq \epsilon$;
- $|w_2w_3w_4| \leq k$;
- $w_1w_2^iw_3w_4^iw_5 \in L$, *para cada* $i \in \mathbb{N}_0$.

Dem.: Exercício. □

Ilustra-se a utilização da Proposição 2.65 na demonstração de que a linguagem $\{a^nb^nc^n : n \in \mathbb{N}_0\}$ não é independente do contexto.

Exemplo 2.66.
A linguagem $L = \{a^nb^nc^n : n \in \mathbb{N}_0\}$ sobre o alfabeto $\Sigma = \{a, b, c\}$ não é uma linguagem independente do contexto. Se L fosse uma linguagem independente do contexto então verificaria o resultado enunciado na Proposição 2.65. Sendo k o natural cuja existência é garantida por esse resultado, considere-se a palavra $w = a^kb^kc^k \in L$. O resultado garante que $a^kb^kc^k = w_1w_2w_3w_4w_5$ em que $w_2w_3w_4$ tem no máximo k símbolos e w_2 e w_4 não são ambas vazias, e garante também que $w_1w_3w_5 \in L$. Existem três possibilidades a considerar: (i) $w_2w_3w_4$ só tem bs; (ii) $w_2w_3w_4$ inclui pelo menos um a, e portanto não inclui nenhum

c, uma vez que tem no máximo k símbolos; (iii) $w_2w_3w_4$ inclui pelo menos um c e portanto não inclui nenhum a, pela mesma razão.

No caso (i), $w_1w_3w_5 = a^kb^{k-j}c^k$ para algum $j \neq 0$. Então $a^kb^{k-j}c^k \in L$, o que é impossível dado que $k \neq k - j$.

No caso (ii), tem-se que $w_1w_3w_5 = a^{k-j}b^{k-j'}c^k$ em que ou $j \neq 0$ ou $j' \neq 0$. A linguagem L incluiria assim uma sequência com mais c's do que a's ou mais c's dos que b's, o que é impossível.

No caso (iii), tem-se que $w_1w_3w_5 = a^kb^{k-j}c^{k-j'}$ em que ou $j \neq 0$ ou $j' \neq 0$. A linguagem L incluiria assim uma sequência com mais a's do que b's ou mais a's dos que c's, o que é de novo impossível.

Conclui-se que L não é independente do contexto. △

Este exemplo permite-nos confirmar que a classe das linguagens independentes do contexto não é fechada para intersecções, e portanto também não o será para complementações. Para tal basta notar que $\{a^nb^nc^n : n \in \mathbb{N}_0\} = \{a^nb^nc^m : n, m \in \mathbb{N}_0\} \cap \{a^nb^mc^m : m \in \mathbb{N}_0\}$, e que as duas linguagens introduzidas são de facto independentes do contexto.

Outros exemplos de linguagens que não são independentes do contexto, ainda sobre o alfabeto $\Sigma = \{a, b, c\}$, são $\{a^ib^jc^k : i, j, k \in \mathbb{N} \text{ e } i < j < k\}$ ou $\{ww : w \in \{a, b, c\}^*\}$.

2.3 Outras classes de linguagens

Como fica claro, existem linguagens mais exigentes do ponto de vista computacional, que não são regulares, nem mesmo independentes do contexto. Não faremos um estudo detalhado, mas vale a pena compreender algumas classes principais. Esse trabalho deve-se em particular a Noam Chomsky, e foi desenvolvido no contexto do estudo da linguística usando a noção formal de *gramática*.

2.3.1 Gramáticas

Comecemos por introduzir a noção geral de *gramática*, como forma de definir rigorosamente uma linguagem.

Definição 2.67. GRAMÁTICA
Uma *gramática* é formada por:

- um alfabeto Σ, cujos símbolos se dizem *terminais*;
- um conjunto finito V (disjunto de Σ) de *símbolos não-terminais*;
- um símbolo não-terminal designado $S \in V$, dito *símbolo inicial*;
- um conjunto finito de *produções*

$$P \subseteq \{(\alpha_1 X \alpha_2, \beta) : \alpha_1, \alpha_2, \beta \in (V \cup \Sigma)^*, X \in V\}.$$

△

Exemplo 2.68.
A gramática $G = (\Sigma, V, S, P)$ em que

- $\Sigma = \{$os, as, estudantes, eles, elas, participam, estudam, muito, pouco$\}$
- $V = \{\text{F}, \text{SN}, \text{SV}, \text{N}, \text{PR}, \text{V}, \text{D}, \text{ADV}\}$
- $S = \text{F}$
- $P = \{$(F,SN SV), (SN,D N), (SN,PR), (SV,V), (SV,V ADV),
  (D,os),(D,as),(N,estudantes),(PR,eles),(PR,elas),(V,participam)
  (V,estudam),(ADV,muito),(ADV,pouco)$\}$

corresponde a um fragmento simplificado da estrutura sintáctica da língua portuguesa. Numa produção como (F,SN SV), por exemplo, diz-se que F é o lado esquerdo da produção, e que SN SV é o seu lado direito. Outra notação possível é F $\longrightarrow$ SN SV. Produções que têm o mesmo lado esquerdo podem representar-se simultaneamente, separando os respectivos lados direitos por |. Por exemplo, as duas produções da gramática G com lado direito SN podem ser representadas por SN $\longrightarrow$ D N|PR. A frase da língua portuguesa "eles estudam muito", por exemplo, é gerada por esta gramática como se indica de seguida:

F	
SN SV	F $\longrightarrow$ SN SV
PR SV	SN $\longrightarrow$ PR
eles SV	PR$\longrightarrow$ eles
eles V ADV	SV$\longrightarrow$ V ADV
eles estudam ADV	V$\longrightarrow$ estudam
eles estudam muito	ADV$\longrightarrow$ muito

Começa-se com o símbolo inicial, F, e são depois usadas sucessivamente várias produções de G. O lado esquerdo de cada uma delas é substituído pelo respectivo lado direito, obtendo-se uma nova sequência de símbolos terminais e/ou não-terminais. △

Definição 2.69. Aplicação de produção
Sejam $G = (\Sigma, V, S, P)$ uma gramática, $(\alpha, \beta) \in P$ e sejam $w, w' \in (V \cup \Sigma)^*$:

- (α, β) pode ser aplicada a w se α ocorre em w, ou seja, $w = w_1 \alpha w_2$ com $w_1, w_2 \in (V \cup \Sigma)^*$;
- w' resulta de w por *aplicação de* (α, β) se a produção pode ser aplicada a w e w' se obtém substituindo uma ocorrência de α por β em w, nomeadamente $w' = w_1 \beta w_2$. △

Definição 2.70. Derivação em gramática e linguagem gerada
Seja $G = (\Sigma, V, S, P)$ uma gramática. Uma *derivação* de $w \in (V \cup \Sigma)^*$ em G é uma sequência $w_1 w_2 \dots w_n$ de elementos de $(V \cup \Sigma)^*$, com $n \geq 1$, tal que

- $w_1 = S$, $w_n = w$

- w_{i+1} resulta de w_i por aplicação de uma produção de P, para cada $1 \leq i < n$.

A palavra w diz-se *gerada* por G se existe uma derivação de w em G. A *linguagem gerada* pela gramática G, ou linguagem de G, é o conjunto $L(G) = \{w \in \Sigma^* : \ w \text{ é gerada por } G\}$. △

O alcance da noção de gramática é bastante geral, pelo que vale a pena atentar nos seguintes casos particulares.

Definição 2.71. GRAMÁTICA REGULAR, (IN)DEPENDENTE DO CONTEXTO
Uma gramática $G = (\Sigma, V, S, P)$ diz-se:

- *regular* se as produções de P são da forma (X, w) ou (X, wY);
- *independente do contexto* se as produções de P são da forma (X, β);
- *dependente do contexto* se as produções de P são da forma $(\alpha_1 X \alpha_2, \alpha_1 \gamma \alpha_2)$ ou (S, ϵ), caso em que S não pode ocorrer no lado direito de nenhuma produção

com $X, Y \in V$, $w, u, v \in \Sigma^*$, $\alpha_1, \alpha_2, \beta, \gamma \in (V \cup \Sigma)^*$ e $\gamma \neq \epsilon$. △

Exemplo 2.72.
A gramática $G = (\Sigma, V, S, P)$ com $\Sigma = \{0, 1\}$, $V = \{S, X, Y, Z\}$, e P constituído pelas produções

$$S \longrightarrow 01X \quad X \longrightarrow 0Y \,|\, 1Z \quad Y \longrightarrow 0Y \,|\, 1X \,|\, \epsilon \quad Z \longrightarrow 0Y \,|\, 1Z \,|\, \epsilon$$

é uma gramática regular. A linguagem gerada por G é a linguagem sobre $\{0, 1\}$ das palavras que começam em 01, mas não terminam 01. △

Exemplo 2.73.
A gramática $G = (\Sigma, V, S, P)$ com $\Sigma = \{a, b\}$, $V = \{S\}$, e P constituído pelas produções

$$S \longrightarrow aSb \mid \epsilon$$

é uma gramática independente do contexto. A linguagem gerada por G é a linguagem sobre $\{a, b\}$ das palavras do tipo $a^n b^n$, com $n \in \mathbb{N}_0$. △

Exemplo 2.74.
A gramática $G = (\Sigma, V, S, P)$ com $\Sigma = \{a, b, c\}$, $V = \{S, X\}$, e P constituído pelas produções

$$S \longrightarrow aXSc \mid abc \mid \epsilon \qquad Xa \longrightarrow aX \qquad Xb \longrightarrow bb$$

é uma gramática dependente do contexto. A linguagem gerada por G é a linguagem sobre $\{a, b, c\}$ das palavras do tipo $a^n b^n c^n$, com $n \in \mathbb{N}_0$. △

Em inglês usa-se a terminologia *context-free/context-sensitive*, em lugar de independente/dependente do contexto. Como será de esperar, as designações acima não são inocentes.

Proposição 2.75.
Seja $G = (\Sigma, V, S, P)$ uma gramática. Tem-se:

- *se G é uma gramática regular então $L(G)$ é uma linguagem regular;*
- *se G é uma gramática independente do contexto então $L(G)$ é uma linguagem independente do contexto.*

Dem. (esboço): Deixa-se como exercício transformar uma gramática regular num AFND$^{\epsilon}$ para a mesma linguagem, bem como transformar uma gramática independente do contexto num AP para a mesma linguagem. □

O recíproco deste resultado também é verdadeiro.

Proposição 2.76.
Seja $L \subseteq \Sigma^*$ uma linguagem. Tem-se:

- *se L é regular então existe uma gramática regular G tal que $L(G) = L$;*
- *se L é independente do contexto então existe uma gramática independente do contexto G tal que $L(G) = L$.*

Dem. (esboço): Deixa-se como exercício transformar um AFD numa gramática regular para a mesma linguagem, bem como transformar um AP numa gramática independente do contexto para a mesma linguagem. □

As linguagens que são definidas por gramáticas dependentes do contexto tomam essa designação.

Definição 2.77. LINGUAGEM DEPENDENTE DO CONTEXTO
Uma linguagem $L \subseteq \Sigma^*$ diz-se *dependente do contexto* se existe uma gramática dependente do contexto G com alfabeto Σ tal que $L(G) = L$. Denota-se por $\mathcal{DEP}^{\Sigma}$ o conjunto de todas as linguagens dependentes do contexto com alfabeto Σ. △

Tal como antes, usaremos apenas $\mathcal{DEP}$, em vez de $\mathcal{DEP}^{\Sigma}$, sempre que o alfabeto esteja subentendido ou não seja importante no contexto.

Curiosamente, a classe das linguagens dependentes do contexto volta a ter muitas das boas propriedades de fecho das linguagens regulares, incluindo operações Booleanas como intersecção e complementação, que não se verificam para linguagens independentes do contexto. Deixa-se os detalhes ao cuidado do leitor.

2.3.2 A hierarquia de Chomsky

Aos quatro tipos de gramáticas vistos acima, e respectivas linguagens, Chomsky associou a seguinte designação.

TIPO-0: linguagens geradas por gramáticas;

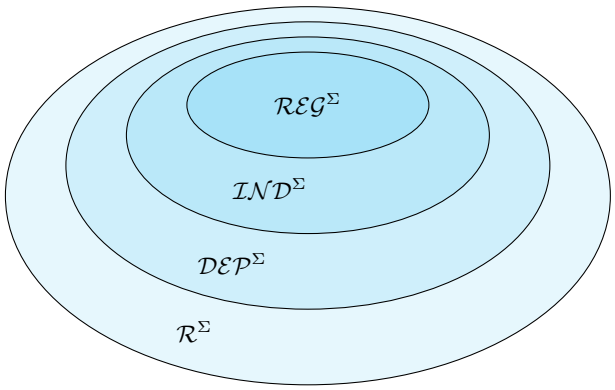
TIPO-1: linguagens geradas por gramáticas dependentes do contexto;

TIPO-2: linguagens geradas por gramáticas independentes do contexto;

TIPO-3: linguagens geradas por gramáticas regulares.

O conjunto das linguagens de **TIPO-0** sobre um alfabeto Σ denotar-se-á por $\mathcal{R}^{\Sigma}$, ou apenas $\mathcal{R}$, por razões que compreenderemos adiante. Na verdade, estudámos modelos computacionais subjacentes às linguagens de **TIPO-3**, os autómatos finitos, e de **TIPO-2**, os autómatos de pilha, mas para além da caracterização por via de gramáticas, não explorámos ainda os modelos computacionais subjacentes aos restantes tipos. Na verdade, as linguagens de **TIPO-0** são precisamente as linguagens reconhecíveis por *máquinas de Turing*, o modelo universal de computação que estudaremos de seguida. As linguagens de **TIPO-1**, dependentes do contexto, são tipicamente associadas a *autómatos lineares* (em inglês, *linear bounded automata*), que correspondem a máquinas de Turing que não podem escrever em espaços. De facto, como veremos, $\mathcal{DEP} = \mathbf{NSPACE}(n)$ de acordo com um conhecido resultado de Kuroda.

A hierarquia de linguagens proposta por Chomsky pode ser visualizada no diagrama abaixo, onde a elipse mais interior corresponde às linguagens de **TIPO-3**, e a mais exterior às linguagens de **TIPO-0**.



Os resultados que já conhecemos, bem como as caracterizações por via de gramáticas, permitem-nos compreender facilmente que

$$\mathcal{REG} \subseteq \mathcal{IND} \subseteq \mathcal{DEP} \subseteq \mathcal{R}.$$

De facto, tem-se mesmo inclusões estritas, isto é

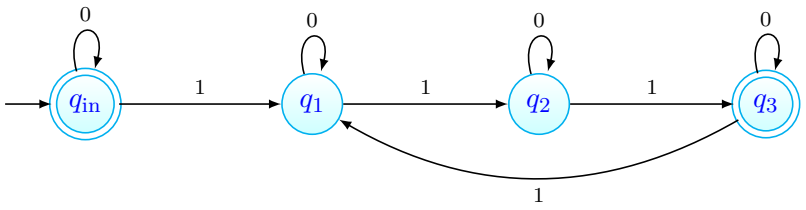
$$\mathcal{REG} \subsetneq \mathcal{IND} \subsetneq \mathcal{DEP} \subsetneq \mathcal{R}.$$

A separação das classes $\mathcal{REG}$ e $\mathcal{IND}$ é garantida pelos Exemplos 2.59 e 2.73. A separação das classes $\mathcal{IND}$ e $\mathcal{DEP}$ é garantida pelos Exemplos 2.66 e 2.74. A separação das classes $\mathcal{DEP}$ e $\mathcal{R}$ será compreendida mais adiante. Põe-se ainda a seguinte questão: haverá linguagens que não são de nenhum destes tipos?

Exercícios

1 Autómatos finitos deterministas

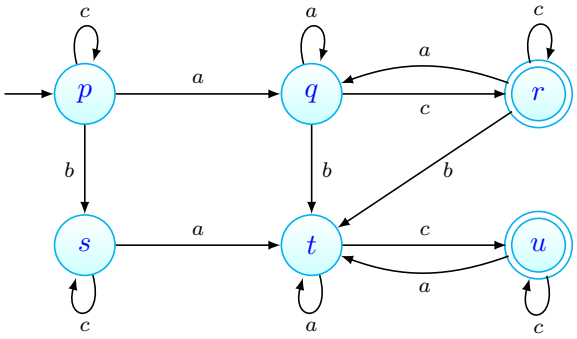
1. Seja D o AFD com alfabeto $\Sigma = \{0, 1\}$, cuja representação gráfica é a seguinte:



   (a) Verifique que as palavras ϵ, 0000 e 10110111 são aceites por D.

   (b) Verifique informalmente que $L(D)$ é o conjunto das palavras sobre Σ nas quais o número de 1s é um múltiplo de 3.

2. Seja D o AFD com alfabeto $\Sigma = \{a, b, c\}$, cuja representação gráfica é a seguinte:



   (a) Verifique que as palavras *aacac* e *abacc* são aceites por D.

   (b) Verifique informalmente que $L(D)$ é o conjunto das palavras sobre Σ que têm pelo menos um a, no máximo um b, e terminam em c.

3. Especifique um AFD que reconheça as seguintes linguagens sobre o alfabeto $\{0, 1\}$:

   (a) a linguagem das palavras cujo comprimento é par;

   (b) a linguagem das palavras que têm 1 em cada posição ímpar;

   (c) a linguagem das palavras que têm um número ímpar de 0s;

   (d) a linguagem das palavras que têm um número ímpar de 0s e um número par de 1s;

   (e) a linguagem das palavras que entre dois 1s consecutivos tenham no máximo um 0;

(f) a linguagem das palavras que têm 1 na penúltima posição;

(g) a linguagem das palavras que têm 1 na antepenúltima posição.

4. Especifique um AFD que reconheça as seguintes linguagens sobre o alfabeto $\{a, b, c\}$:

   (a) a linguagem das palavras que têm exatamente dois as;

   (b) a linguagem das palavras que começam em a e terminam em bc;

   (c) a linguagem das palavras que têm pelo menos um a e pelo menos um b;

   (d) a linguagem das palavras que têm dois bs consecutivos;

   (e) a linguagem das palavras que têm $caab$ como subpalavra;

   (f) a linguagem das palavras nas quais a subpalavra ab não ocorre, ou ocorre exatamente duas vezes;

   (g) a linguagem das palavras que não têm símbolos consecutivos iguais;

   (h) a linguagem das palavras nas quais o primeiro e o último símbolo são iguais;

   (i) a linguagem das palavras nas quais os dois últimos símbolos são iguais;

   (j) a linguagem das palavras que não têm dois bs consecutivos;

   (k) a linguagem das palavras nas quais o primeiro e o último símbolo são diferentes;

   (l) a linguagem das palavras nas quais os dois últimos símbolos são diferentes.

5. Seja $Dig = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Uma constante numérica é constituída por uma parte inteira e uma parte decimal, separadas por um " . ". A parte inteira é 0 ou uma palavra sobre Dig que não começa por 0. A parte decimal é uma palavra sobre Dig. O símbolo " . " pode omitir-se se a parte decimal for vazia. A parte inteira e a parte decimal não podem ser ambas vazias. Especifique um AFD que reconheça a linguagem das palavras sobre o alfabeto $\Sigma = Dig \cup \{., +, -\}$ do tipo w, $+w$ ou $-w$, onde w é uma constante numérica.

6. Mostre que é regular a linguagem constituída pelas palavras sobre o alfabeto $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ que representam, em notação decimal, números naturais que são múltiplos de 3.

7. Mostre que é regular a linguagem constituída pelas palavras sobre o alfabeto $\{0, 1\}$ que representam, em notação binária, números naturais que:

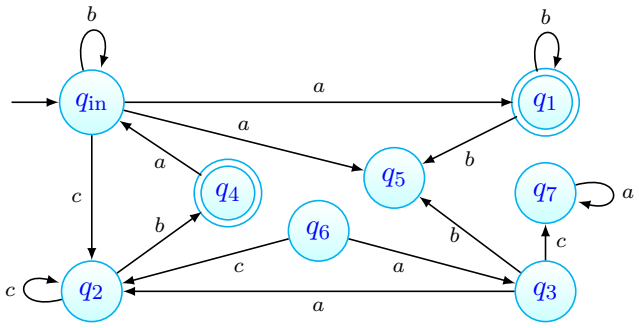
   (a) são múltiplos de 3;

   (b) são múltiplos de 4;

   (c) são múltiplos de 5.

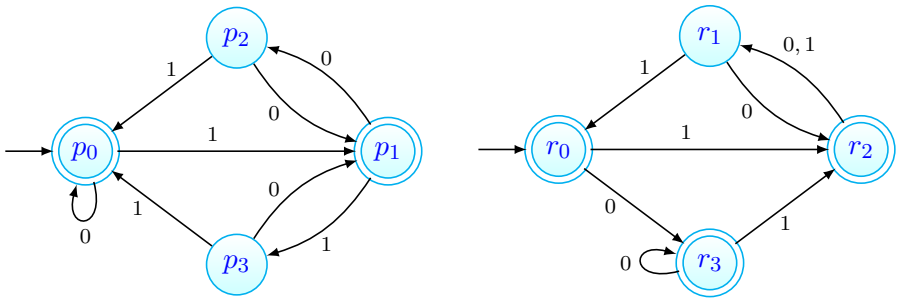
2 Equivalência e minimização de autómatos

1. Considere o AFD D com alfabeto $\Sigma = \{a, b, c\}$, cuja representação gráfica é a seguinte:

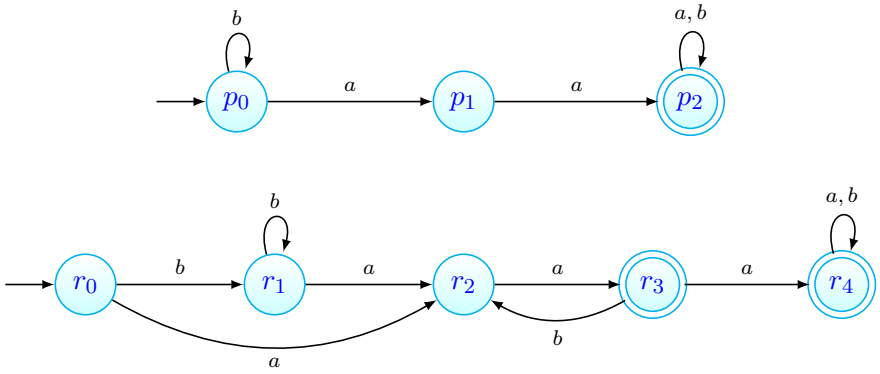


   Indique os estados de D que não são produtivos e os estados de D que não são acessíveia. Indique depois os estados de D que são úteis, e os que são inúteis.

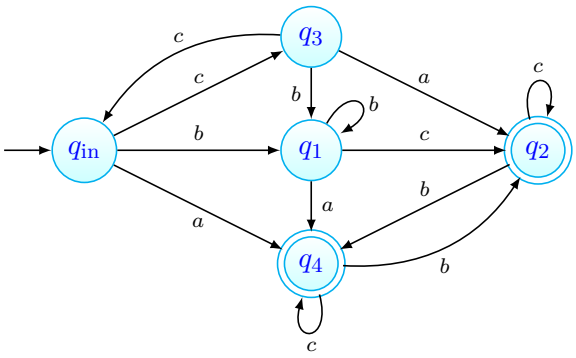
2. Considere os AFDS com alfabeto $\Sigma = \{0, 1\}$ com as representações gráficas indicadas. Estes autómatos são equivalentes?



3. Considere os AFDS com alfabeto $\Sigma = \{a, b\}$ com as representações gráficas indicadas. Estes autómatos são equivalentes?

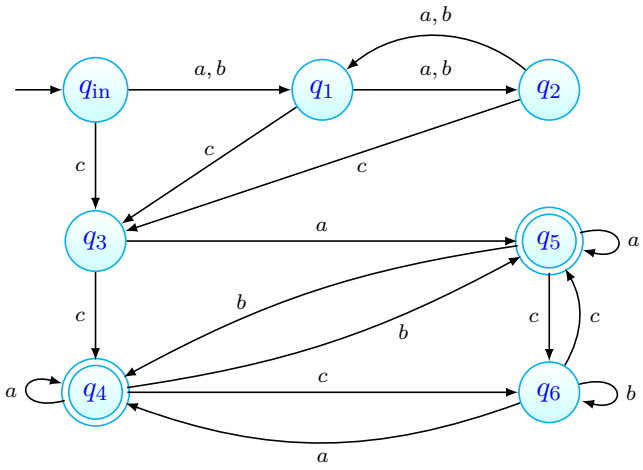


4. Considere o AFD D com alfabeto $\Sigma = \{a, b, c\}$, cuja representação gráfica é a seguinte:



O autómato D é mínimo? Em caso negativo construa $min(D)$.

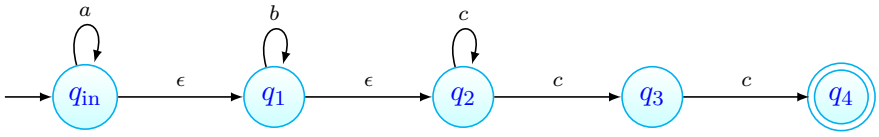
5. Considere o AFD D com alfabeto $\Sigma = \{a, b, c\}$, cuja representação gráfica é a seguinte:



O autómato D é mínimo? Em caso negativo construa $min(D)$.

3 Autómatos finitos não-deterministas

1. Seja A^ϵ o AFND$^\epsilon$ com alfabeto $\Sigma = \{a, b, c\}$, cuja representação gráfica é a seguinte:



   (a) Verifique que as palavras *abcc*, *bbccc* e *acccc* são aceites por A^ϵ.

   (b) Verifique informalmente que $L(A^\epsilon)$ é o conjunto das palavras sobre Σ do tipo $a^i b^j c^k$ com $i, j, k \in \mathbb{N}_0$ e $k \geq 2$.

2. Especifique um AFND$^\epsilon$ que reconheça as seguintes linguagens sobre o alfabeto $\{0, 1\}$:

(a) a linguagem das palavras que têm 1 na penúltima posição;

(b) a linguagem das palavras que têm 1 na antepenúltima posição.

3. Especifique um AFND$^\epsilon$ que reconheça as seguintes linguagens sobre o alfabeto $\{a, b, c\}$:

(a) a linguagem das palavras nas quais os três últimos símbolos são iguais;

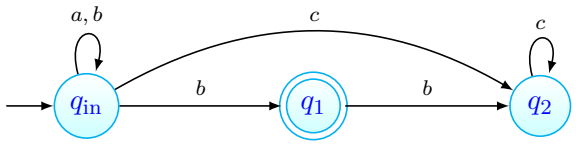
(b) a linguagem das palavras que têm *caab* como subpalavra;

(c) a linguagem das palavras nas quais a subpalavra *ab* não ocorre, ou ocorre exatamente duas vezes;

(d) a linguagem das palavras nas quais o último símbolo ocorre uma única vez em toda a palavra;

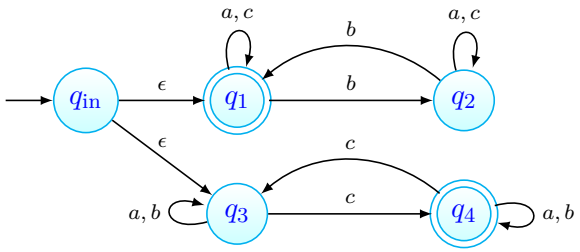
(e) a linguagem das palavras nas quais o último símbolo ocorre pelo menos duas vezes em toda a palavra.

4. Considere o AFND A com alfabeto $\Sigma = \{a, b, c\}$, cuja representação gráfica é a seguinte:



Construa um AFD equivalente a A.

5. Considere o AFND$^\epsilon$ A^ϵ com alfabeto $\Sigma = \{a, b, c\}$, cuja representação gráfica é a seguinte:



Construa um AFD equivalente a A^ϵ.

6. Considere o AFND$^\epsilon$ A^ϵ do Exemplo 3.1. Construa um AFD equivalente a A^ϵ.

4 Expressões regulares

1. Considere a expressão regular $\alpha = a(a+c)^*b(a+b+c)^*(a+c)+c^*(a+b)^*$ sobre o alfabeto $\{a, b, c\}$. De entre as palavras $\emptyset$, ϵ, ab, abc, $abcc$, $cccab$, $aaaa$, $cacb$, $abccb$, indique as que pertencem à linguagem denotada por α.

2. Indique uma expressão regular que denote as seguintes linguagens sobre o alfabeto $\{0, 1\}$:

   (a) a linguagem das palavras que começam em 11;

   (b) a linguagem das palavras que têm 1 na antepenúltima posição;

   (c) a linguagem das palavras que têm pelo menos três 0s;

   (d) a linguagem das palavras nas quais o primeiro e o último símbolo são diferentes;

   (e) a linguagem das palavras cujo comprimento é par;

   (f) a linguagem das palavras que têm um número ímpar de 0s;

   (g) a linguagem das palavras que têm um número ímpar de 0s e um número par de 1s;

   (h) a linguagem das palavras que entre dois 1s consecutivos tenham no máximo um 0;

   (i) a linguagem das palavras que não têm 001 como subpalavra.

3. Indique uma expressão regular que denote a linguagem descrita no Exercício 1.5.

4. Mostre que:

   (a) $b(aa^* + \epsilon)^* + bba^* = (b + bb)a^*$;

   (b) $(aa^*a + caa) + (a^*b + cb) = (a^* + c)(aa + b)$;

   (c) $b^*a^*c + b^*c + b^*a^*cc^* + b^*c^*c = b^*(\epsilon + a)c^*c$.

5. Considere as seguintes expressões regulares sobre o alfabeto $\{a, b, c\}$:

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

   (b) $(ba)^*(c + \epsilon)$;

   (c) $(ab)^* + (aa + c)^*$;

   (d) $(b(cc)^*)^*$.

   Recorra ao resultado estabelecido na Proposição 2.48 para especificar um AFND$^\epsilon$ que reconheça a linguagem denotada por cada uma destas expressões. De seguida, obtenha um AFD equivalente.

5 Propriedades das linguagens regulares

1. Sejam L_1 e L_2 linguagens regulares sobre um alfabeto Σ. Mostre que a linguagem $L_1 \backslash L_2$ é também regular.

2. Mostre que se L é uma linguagem regular, então é também regular a linguagem $\{w^R : w \in L\}$.

3. Mostre que se L é uma linguagem regular, então é também regular a linguagem $\{w \in L :$ não existe nenhum prefixo v de w tal que $v \in L\}$.

4. Use o lema da bombagem para linguagens regulares para demonstrar que não são regulares as linguagens seguintes:

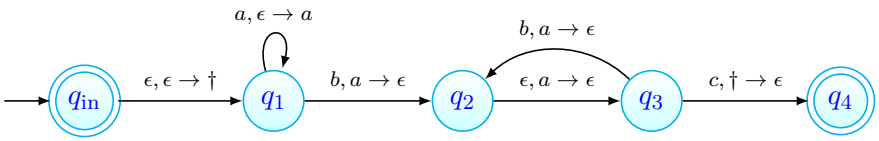
   (a) $L = \{0^{2n}1^n : n \in \mathbb{N}_0\}$;
   (b) $L = \{0^n1^m : m, n \in \mathbb{N}_0 \text{ e } n > m\}$;
   (c) $L = \{0^n1^n0^n : n \in \mathbb{N}_0\}$;
   (d) $L = \{w \in \{0,1\}^* : w$ é um palíndromo$\}$
   (recorde que uma palavra w é um palíndromo se $w = w^R$);
   (e) $L = \{ww : w \in \{0,1\}^*\}$;
   (f) $L = \{w \in \{0,1\}^* :$ em w o número de 0s é igual ao número de 1s$\}$;
   (g) $L = \{w \in \{0,1\}^* :$ em w o número de 0s é o dobro do número de 1s$\}$;
   (h) $L = \{a^{2^n} : n \in \mathbb{N}_0\}$.

5. Mostre que se L_i $(i \in \mathbb{N}_0)$ é uma linguagem regular, então $\bigcup_{i\in\mathbb{N}_0} L_i$ pode não ser uma linguagem regular.

   Sugestão: note que é regular a linguagem singular $\{0^i1^i\}$ com $i \in \mathbb{N}_0$.

6 Autómatos de pilha

1. Seja M o AP com alfabeto $\Sigma = \{a, b, c\}$ e alfabeto auxiliar $\Gamma = \{\dagger, a\}$, cuja representação gráfica é a seguinte:



   (a) Verifique que as palavras *aabc* e *aaaabbc* são aceites por M.
   (b) Verifique informalmente que a linguagem reconhecida por M é o conjunto das palavras sobre Σ do tipo $a^{2n}b^nc$ com $n \in \mathbb{N}$.

2. Especifique um AP que reconheça as seguintes linguagens sobre o alfabeto $\{0, 1\}$:

   (a) a linguagem das palavras do tipo 0^n1^m com $m, n \in \mathbb{N}_0$ e $n > m$;

(b) a linguagem das palavras do tipo 0^n1^m com $m, n \in \mathbb{N}_0$ e $n \leq m$;

(c) a linguagem das palavras que são palíndromos;

(d) a linguagem das palavras nas quais o número de 0s é igual ao número de 1s;

(e) a linguagem das palavras nas quais o número de 0s é maior do que o número de 1s;

(f) a linguagem das palavras nas quais o número de 0s é o dobro do número de 1s.

3. Mostre que são independentes do contexto as seguintes linguagens ($n \div 2$ é o quociente da divisão inteira por 2):

(a) $L = \{a^m + a^n = a^{m+n} : \ n, m \in \mathbb{N}_0\}$;

(b) $L = \{a^n \times 2 = a^{2n} : \ n \in \mathbb{N}_0\}$;

(c) $L = \{a^n \div 2 = a^{n \div 2} : \ n \in \mathbb{N}_0\}$;

(d) $L = \{a^i b^j c^k : \ i = j$ ou $i = k$ com $i, j, k \in \mathbb{N}_0\}$.

7 Propriedades das linguagens independentes do contexto

1. Demonstre a Proposição 2.64.

2. Use o lema da bombagem para linguagens independentes do contexto para demonstrar que não são independentes do contexto as linguagens seguintes:

(a) $L = \{0^n1^n0^n : n \in \mathbb{N}_0\}$;

(b) $L = \{ww : w \in \{0, 1\}^*\}$;

(c) $L = \{a^i b^j c^k : \ i, j, k \in \mathbb{N}$ e $i < j < k\}$;

(d) $L = \{a^{2^n} : n \in \mathbb{N}_0\}$.

8 Gramáticas

1. Considere a gramática $G = (\Sigma, V, S, P)$ com $\Sigma = \{a, b\}$, $V = \{S, X, Y\}$, e P constituído pelas produções $S \longrightarrow aX \,|\, bY \quad X \longrightarrow \epsilon \,|\, bY \quad Y \longrightarrow \epsilon \,|\, aX$

(a) Verifique que as palavras *bab* e *abab* são geradas por G, apresentando uma derivação para cada uma delas.

(b) Verifique informalmente que $L(G)$ é o conjunto das palavras sobre Σ que não têm símbolos consecutivos iguais. Classifique a gramática.

2. Considere a gramática $G = (\Sigma, V, S, P)$ com $\Sigma = \{a, b\}$, $V = \{S\}$, e P constituído pelas produções $S \longrightarrow aSbS \mid bSaS \mid \epsilon$

(a) Verifique que as palavras *aabb* e *bababaab* são geradas por G, apresentando uma derivação para cada uma delas.

(b) Verifique informalmente que $L(G)$ é o conjunto das palavras sobre Σ com igual número de as e bs. Classifique a gramática.

3. Considere a gramática $G = (\Sigma, V, S, P)$ com $\Sigma = \{a, b, c\}$, $V = \{S, X\}$, e P constituído pelas produções

$$S \longrightarrow aXSccc \mid aXccc \quad Xa \longrightarrow aX \quad Xb \longrightarrow bbb \quad Xc \longrightarrow bbc$$

(a) Verifique que as palavras $abbccc$ e $aabbbbcccccc$ são geradas por G, apresentando uma derivação para cada uma delas.

(b) Verifique informalmente que $L(G)$ é o conjunto das palavras sobre Σ do tipo $a^n b^{2n} c^{3n}$ com $n \in \mathbb{N}$. Classifique a gramática.

4. Especifique uma gramática regular que reconheça as seguintes linguagens sobre o alfabeto $\{a, b, c\}$:

(a) a linguagem das palavras que têm comprimento par;

(b) a linguagem das palavras que começam em a e terminam em bc;

(c) a linguagem das palavras que têm exatamente dois as;

(d) a linguagem das palavras que têm pelo menos um a e pelo menos um b;

(e) a linguagem das palavras que têm um número ímpar de as;

(f) a linguagem das palavras que têm dois bs consecutivos;

(g) a linguagem das palavras que não têm símbolos consecutivos iguais;

(h) a linguagem das palavras nas quais o primeiro e o último símbolo são iguais;

(i) a linguagem das palavras nas quais os dois últimos símbolos são iguais;

(j) a linguagem das palavras que não têm dois bs consecutivos.

5. Considere a linguagem descrita no Exercício 1.5.

(a) Especifique uma gramática regular que gere essa linguagem.

(b) Apresente uma derivação para a palavra -341.06.

6. Especifique uma gramática independente do contexto que gere as seguintes linguagens:

(a) $L = \{0^{2n}1^n : n \in \mathbb{N}_0\}$;

(b) $L = \{0^n 1^m : m, n \in \mathbb{N}_0 \text{ e } n > m\}$;

(c) $L = \{w \in \{0,1\}^* : w \text{ é um palíndromo}\}$
(recorde que uma palavra w é um palíndromo se $w = w^R$);

(d) $L = \{w \in \{0,1\}^* : \text{ em } w \text{ o número de 0s é maior do que o número de 1s}\}$;

(e) $L = \{a^m + a^n = a^{m+n} : \ n, m \in \mathbb{N}_0\}$;

(f) $L = \{a^n \times 2 = a^{2n} : \ n \in \mathbb{N}_0\}$;

(g) $L = \{a^n \div 2 = a^{n \div 2} : \ n \in \mathbb{N}_0\}$.

7. Considere a linguagem L das palavras sobre o alfabeto $\Sigma = \{x, y, z, \wedge, \vee, \neg, (,)\}$ que representam expressões booleanas com variáveis x, y e z.

   (a) Especifique uma gramática independente do contexto que gere L.

   (b) Apresente uma derivação para a palavra $(x \vee z) \wedge (\neg y)$.

8. Especifique uma gramática G que gere a linguagem sobre o alfabeto $\{0, 1\}$ das palavras do tipo $0^n 1^n 0^n$ com $n \in \mathbb{N}$.

9 Exercícios complementares

1. Demonstre a Proposição 2.17.

2. Demonstre a Proposição 2.23.

3. Demonstre a Proposição 2.27.

4. Demonstre a Proposição 2.30.

5. Demonstre a Proposição 2.54.

6. Mostre como transformar um AFD D numa gramática regular G tal que $L(D) = L(G)$.

7. Mostre como transformar uma gramática regular G num AFND$^\epsilon$ A^ϵ tal que $L(G) = L(A^\epsilon)$.

8. Mostre como transformar diretamente duas gramáticas regulares G_1 e G_2, com um mesmo alfabeto Σ, numa gramática regular G tal que

   (a) $L(G) = L(G_1) \cup L(G_2)$;

   (b) $L(G) = L(G_1).L(G_2)$.

9. Mostre como transformar diretamente uma gramática regular G numa gramática regular G^* tal que $L(G^*) = L(G)^*$.

3

Máquinas de Turing

"Welcome my son, welcome to the machine."
Roger Waters, *Pink Floyd – Wish you were here*, 1975

3.1 A máquina de Turing

Podemos imaginar uma máquina de Turing como um dispositivo, abstracto, que tem uma fita dividida em células de memória e uma cabeça de leitura/escrita que permite ler e escrever símbolos na célula de memória em que está colocada. A fita é infinita para a direita, e a cabeça de leitura/escrita pode deslocar-se por movimentos L e R para a esquerda ou a direita, respectivamente.

Definição 3.1. Máquina de Turing
Uma *máquina de Turing* é formada por:

- um *alfabeto de trabalho* Γ, tal que $\square \in \Gamma$;
- um *alfabeto de entrada/saída* $\Sigma \subseteq \Gamma \setminus \{\square\}$;
- um conjunto finito de *estados de controlo* Q;
- um estado designado $q_{\text{in}} \in Q$, dito *estado inicial*;
- dois estados distintos $q_{\text{ac}}, q_{\text{rj}} \notin Q$, ditos *estado de aceitação* e *estado de rejeição*, respectivamente, que estabelecem $\hat{Q} = Q \cup \{q_{\text{ac}}, q_{\text{rj}}\}$;
- uma *função de transição* $\delta : Q \times \Gamma \to \hat{Q} \times \Gamma \times \{L, R\}$.

Os estados $q_{\text{ac}}, q_{\text{rj}}$ dizem-se *estados de terminação*. △

Para que serve uma máquina de Turing? Serve para desenvolver tarefas computacionais: reconhecer uma linguagem (aceitar exactamente as suas palavras), decidir uma linguagem (aceitar as suas palavras e rejeitar as restantes), calcular uma função (aceitar os *inputs* no domínio da função e calcular o respectivo resultado como *output*).

Exemplo 3.2.
Considere-se a linguagem formada pelas palavras sobre o alfabeto $\{0,1\}$ que são alternantes, isto é, em que cada símbolo é distinto do anterior, como por exemplo 01010. A máquina de Turing representada na Figura 3.1, aceita precisamente as palavras desta linguagem.

O alfabeto de trabalho da máquina é $\Gamma = \{0,1,\square\}$. A convenção de representação gráfica da máquina de Turing é simples, tendo-se $Q = \{q_{\text{in}}, q_1, q_2\}$, e sendo a função de transição δ facilmente intuída a partir das etiquetas das setas. Assim, ter-se-á $\delta(q_{\text{in}}, 0) = (q_1, 0, R)$, por exemplo, o que significa que quando no estado q_{in}, se a cabeça de leitura/escrita está posicionada numa célula de memória onde está o símbolo 0, se dá uma transição para o estado q_1, sendo escrito o símbolo 0 (o que no caso apenas mantém o símbolo preexistente, movimentando-se a cabeça de leitura/escrita para a direita). As transições não representadas consideram-se indefinidas. Por não ser utilizado, também não se representa o estado q_{rj}.

A intuição por detrás da máquina é simples. A partir do estado inicial, lendo o *input* da esquerda para a direita, a máquina transita para q_1 quando lê 0, memorizando dessa forma que o próximo símbolo a ler terá de ser 1. Simetricamente, a máquina transita para q_2 quando lê 1, memorizando dessa forma que o próximo símbolo a ler terá de ser 0. A máquina vai assim alternando entre os estados q_1 e q_2 até processar todo o *input*, o que acontece quando é lido um espaço. Caso dois símbolos consecutivos sejam iguais, a computação fica indefinida, já que não está representada nenhum transição a partir de q_1 quando o símbolo lido é 0, nem a partir de q_2 quando o símbolo lido é 1. △

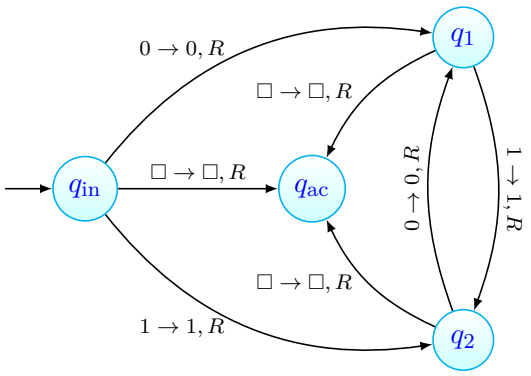
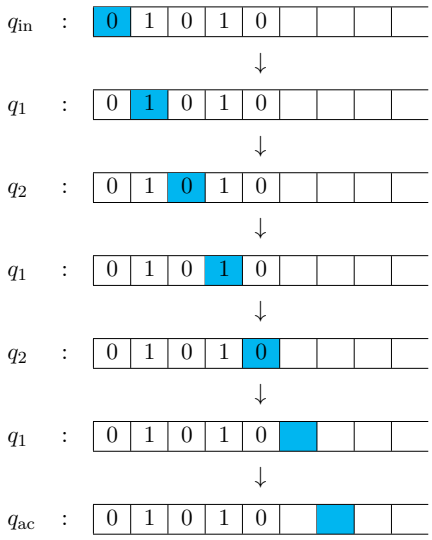


Figura 3.1: Máquina que reconhece as palavras alternantes sobre $\{0,1\}$.

Uma máquina de Turing está, em cada momento, numa certa *configuração*. A configuração define completamente a fita da máquina nesse momento, a posição da cabeça de leitura/escrita, e o estado em que a máquina se encontra.

Dada uma palavra como *input*, uma máquina de Turing inicia a sua computação numa configuração, dita *configuração inicial*, em que o *input* surge na fita nas posições mais à esquerda, seguido de infinitas células vazias, a cabeça de leitura/escrita está posicionada na célula mais à esquerda (pronta a ler o primeiro símbolo do *input*, caso exista), e o estado é q_{in}. A partir daí, a máquina

progride de configuração em configuração, de acordo com a sua função de transição, até atingir um dos estados de terminação, como se ilustra de seguida para a máquina do Exemplo 3.2.



Definição 3.3. CONFIGURAÇÃO
Seja $M = (\Sigma, \Gamma, Q, q_{\text{in}}, q_{\text{ac}}, q_{\text{rj}}, \delta)$ uma máquina de Turing. Uma *configuração* é um triplo denotado por $(u|q|v)$, onde $u \in \Gamma^*$ é a palavra à esquerda da cabeça de leitura/escrita, $q \in \hat{Q}$ o estado corrente, e $v \in \Gamma^*$ a palavra que se inicia na cabeça de leitura/escrita e a que se segue na fita uma sequência infinita de células vazias. Uma *configuração de aceitação* é uma configuração na qual o estado corrente é q_{ac}, e uma *configuração de rejeição* é uma configuração na qual o estado corrente é q_{rj}. △

Claramente, quando numa configuração $(u|q|v)$, a cabeça de leitura/escrita de leitura está pronta a ler o primeiro símbolo de v, tendo à sua esquerda tantas células quantos os símbolos de u. Caso $v = \epsilon$, a cabeça de leitura/escrita está posicionada numa célula vazia. Como é fácil de compreender, dado um *input* $w \in \Sigma^*$, a configuração inicial da máquina corresponde a $(\epsilon|q_{\text{in}}|w)$. Uma *configuração de terminação* é uma configuração de aceitação ou de rejeição.

Definição 3.4. TRANSIÇÃO
Seja $M = (\Sigma, \Gamma, Q, q_{\text{in}}, q_{\text{ac}}, q_{\text{rj}}, \delta)$ uma máquina de Turing. A relação de *transição* $\to_M$ entre configurações da máquina é definida pelas regras

$$\begin{array}{llll}
(u|q|a.v) & \to_M & (u.b|s|v) & \text{se } \delta(q,a) = (s,b,R) \\
(u|q|\epsilon) & \to_M & (u.b|s|\epsilon) & \text{se } \delta(q,\square) = (s,b,R) \\
(u.c|q|a.v) & \to_M & (u|s|cb.v) & \text{se } \delta(q,a) = (s,b,L) \\
(u.c|q|\epsilon) & \to_M & (u|s|cb) & \text{se } \delta(q,\square) = (s,b,L)
\end{array}$$

onde $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, $q \in Q$ e $s \in \hat{Q}$. △

As regras de transição são absolutamente intuitivas. Note-se que, ao contrário dos movimentos à direita, um movimento à esquerda exige que a cabeça de leitura/escrita não esteja colocada na primeira célula da fita, caso em que não fica definida a transição.

Como ilustração, a computação descrita acima, para a máquina do Exemplo 3.2, corresponde a

$$(\epsilon|q_{\text{in}}|01010) \to_M (0|q_1|1010) \to_M (01|q_2|010) \to_M (010|q_1|10) \to_M$$
$$\to_M (0101|q_2|0) \to_M (01010|q_1|\epsilon) \to_M (01010\square|q_{\text{ac}}|\epsilon)$$

Definição 3.5. COMPUTAÇÃO
Seja $M = (\Sigma, \Gamma, Q, q_{\text{in}}, q_{\text{ac}}, q_{\text{rj}}, \delta)$ uma máquina de Turing. A *computação* de M dado um *input* $w \in \Sigma^*$ é a sequência (finita ou infinita) de configurações obtida a partir da configuração inicial $(\epsilon|q_{\text{in}}|w)$ usando $\to_M$. △

A computação *termina* quando a sequência é finita, por ser atingido um estado de terminação, caso em que se designa de *bem sucedida*, ou por *abortar* ao atingir uma configuração a partir da qual não está definida nenhuma regra (por a função de transição estar indefinida, ou por se exigir um movimento à esquerda quando a cabeça de leitura/escrita está na primeira célula). No caso de a sequência ser infinita, a computação diz-se *divergente*. Uma máquina de Turing cuja computação é bem sucedida para todos os *inputs* diz-se um *classificador* ou *máquina classificadora*.

Definição 3.6. ACEITAÇÃO, REJEIÇÃO
Seja $M = (\Sigma, \Gamma, Q, q_{\text{in}}, q_{\text{ac}}, q_{\text{rj}}, \delta)$ uma máquina de Turing.

Uma palavra $w \in \Sigma^*$ diz-se *aceite* por M se a computação da máquina dado o *input* w é bem sucedida e termina no estado de aceitação. Usamos $L_{\text{ac}}(M) \subseteq \Sigma^*$ para denotar a linguagem formada por todas as palavras aceites pela máquina M.

Uma palavra $w \in \Sigma^*$ diz-se *rejeitada* por M se a computação da máquina dado o *input* w é bem sucedida e termina no estado de rejeição. Usamos $L_{\text{rj}}(M) \subseteq \Sigma^*$ para denotar a linguagem formada por todas as palavras rejeitadas pela máquina M.

Caso w seja aceite por M sendo $(u|q_{\text{ac}}|v)$ a configuração atingida pela computação, se $v = x.y$ com $x \in \Sigma^*$ e $y \in \{\square\}^*$ então x diz-se o *output* da computação, o que denotamos por $\phi_M(w) = x$. Em caso contrário, $\phi_M(w)$ fica *indefinido*. Usamos $\phi_M : \Sigma^* \to \Sigma^*$ para denotar a *função calculada por* M. △

Dada uma máquina de Turing M, é usual dizer que M *reconhece* a linguagem $L_{\text{ac}}(M)$, ou que $L_{\text{ac}}(M)$ é a *linguagem reconhecida* por M. Caso M seja classificadora, dizemos que M *decide* $L_{\text{ac}}(M)$, ou que $L_{\text{ac}}(M)$ é a *linguagem decidida* por M. Dizemos ainda que M *calcula* ϕ_M.

Obviamente, $\phi_M(w)$ pode estar indefinido por várias razões: a computação de M dado o *input* w não é bem sucedida, por abortar ou ser infinita, ou mesmo quando a computação é bem sucedida, no caso de w não ser aceite ou, sendo aceite, no caso em que ficam à direita da cabeça de leitura/escrita símbolos que não pertencem a Σ^* (para além dos $\square$s inevitáveis).

Exemplo 3.7.
Recorde-se a linguagem das palavras alternantes apresentada no Exemplo 3.2. Considere-se a sua função característica, ou seja, a função $\chi_{\text{alt}} : \{0,1\}^* \to \{0,1\}$, que quando aplicada a uma qualquer palavra sobre o alfabeto $\{0,1\}$ devolve 1 se a palavra pertencer à linguagem e devolve 0 em caso contrário. Observe-se que a função χ_{alt} está definida para qualquer palavra do alfabeto pelo que a correspondente máquina de Turing terá de aceitar todas as palavras. Uma máquina de Turing que calcula esta função está representada na Figura 3.2.

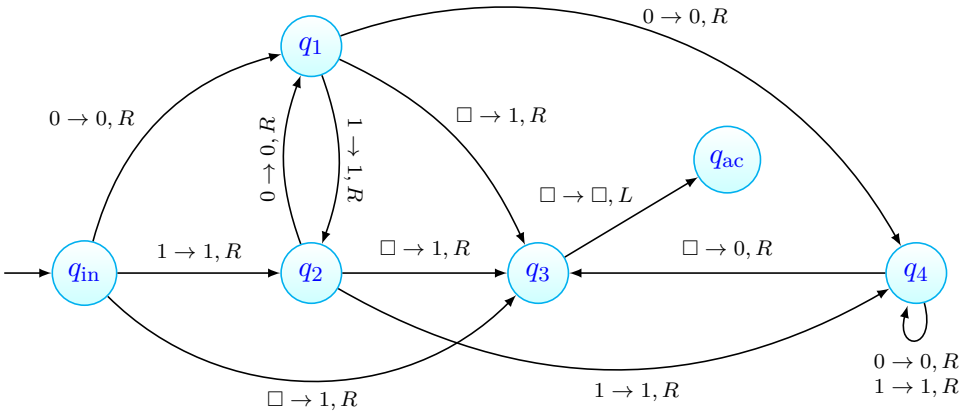


Figura 3.2: Máquina que calcula a função característica da linguagem das palavras alternantes sobre $\{0,1\}$.

A principal diferença em relação à máquina apresentada no Exemplo 3.2 é que, para o *input* w, a máquina deverá terminar numa configuração $(u|q_{\text{ac}}|1)$ caso a palavra w pertença à linguagem, e deverá terminar numa configuração $(u|q_{\text{ac}}|0)$ em caso contrário. △

Exemplo 3.8.
Considere-se a função *sucessor* que a cada número natural faz corresponder o seu sucessor. Assume-se que os números naturais estão representados em notação binária. Uma máquina de Turing que calcula esta função está representada na Figura 3.3. Assume-se a convenção de que $\delta(q_5, *) = (q_{\text{ac}}, *, L)$ representa de facto transições $\delta(q_5, a) = (q_{\text{ac}}, a, L)$ para qualquer símbolo a do alfabeto de trabalho. △

Definição 3.9. EQUIVALÊNCIA
Máquinas de Turing M_1 e M_2, ambas com alfabeto de entrada/saída Σ, dizem-se *equivalentes*, $M_1 \equiv M_2$, se $L_{\text{ac}}(M_1) = L_{\text{ac}}(M_2)$, $L_{\text{rj}}(M_1) = L_{\text{rj}}(M_2)$ e $\phi_{M_1} = \phi_{M_2}$. △

3.2 Variantes

Há na literatura bastantes variações sobre a definição de máquina de Turing. Analisamos de seguida algumas, que nos serão úteis. Apesar de permitirem

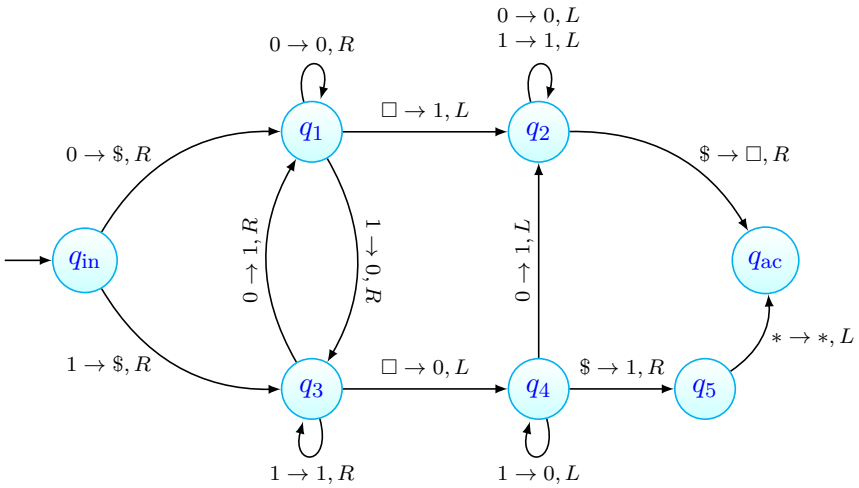


Figura 3.3: Máquina que calcula a função sucessor (notação binária).

maior flexibilidade aparente, na verdade os modelos que analisaremos são essencialmente equivalentes ao modelo original, do ponto de vista da teoria da computabilidade (como veremos mais tarde, há algumas *nuances* importantes do ponto de vista da teoria da complexidade).

3.2.1 Máquinas com transições-S

Por vezes, é útil definir transições em que a cabeça de leitura/escrita não se movimenta, nem para a esquerda nem para a direita.

Definição 3.10. MÁQUINA DE TURING COM TRANSIÇÕES-S
Uma *máquina de Turing com transições-S* é precisamente como uma máquina de Turing, excepto que

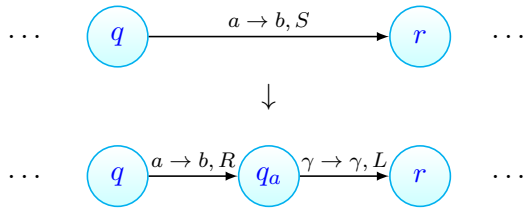
- a *função de transição* é do tipo $\delta : Q \times \Gamma \to \hat{Q} \times \Gamma \times \{L, R, S\}$. △

As noções acima introduzidas nas Definições 3.3–4.1 são facilmente extensíveis para máquinas de Turing com transições-S, o que se deixa como exercício ao leitor. A única diferença substantiva diz respeito às transições-S, para os quais se introduz a seguinte regra adicional:

$$(u|q|a.v) \quad \to_M \quad (u|s|b.v) \quad \text{se } \delta(q,a) \quad = \quad (s,b,S).$$

Proposição 3.11.
Toda a máquina de Turing com transições-S é equivalente a uma máquina de Turing sem transições-S.

Dem. (esboço): Dada uma máquina M com transições-S é muito simples construir uma máquina T (sem transições-S) que tem essencialmente o mesmo comportamento. Para tal, basta substituir em M cada transição-S por duas transições, como se indica abaixo.



onde q_a denota um *novo* estado de T, e γ *cada um dos símbolos* do alfabeto de trabalho Γ. É trivial verificar que $(u|q|a.v) \to_T (u.b|q_a|v) \to_T (u|r|b.v)$, quando na máquina original se tinha $(u|q|a.v) \to_M (u|r|b.v)$. □

Daqui em diante, trabalharemos com máquinas com transições-S, a que chamaremos simplesmente máquinas de Turing.

3.2.2 Máquinas bidireccionais

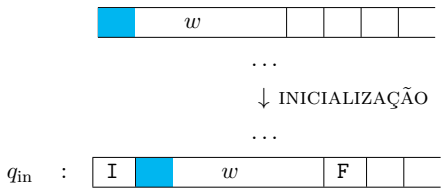
Definição 3.12. MÁQUINA DE TURING BIDIRECCIONAL
Uma *máquina de Turing bidireccional* é precisamente como uma máquina de Turing, onde se assume que a fita de leitura/escrita é infinita em ambas as direcções. As noções acima introduzidas nas Definições 3.3–4.1 são facilmente extensíveis para máquinas de Turing bidireccionais, o que se deixa como exercício ao leitor. A única diferença substantiva diz respeito aos movimentos à esquerda. Introduzem-se as seguintes regras adicionais:

$$\begin{array}{llll} (\epsilon|q|a.v) & \to_M & (\epsilon|s|\Box b.v) & \text{se } \delta(q,a) = (s,b,L) \\ (\epsilon|q|\epsilon) & \to_M & (\epsilon|s|\Box b) & \text{se } \delta(q,\Box) = (s,b,L) \end{array}$$ △

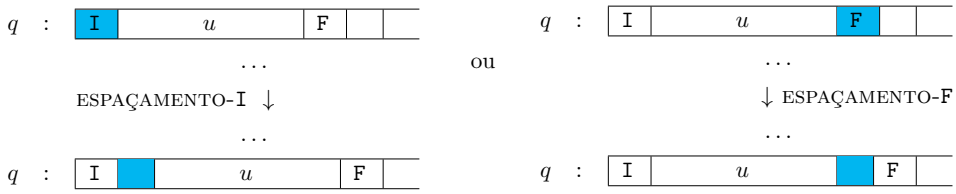
Por oposição, as máquinas de Turing originais, em que a fita de memória é infinita apenas para a direita dizem-se *unidireccionais*.

Proposição 3.13.
Toda a máquina de Turing bidireccional é equivalente a uma máquina de Turing unidireccional.

Dem. (esboço): Dada uma máquina bidireccional M é possível construir uma máquina T (unidireccional) cujas computações são equivalentes. Para tal, é útil demarcar a área de trabalho na fita com símbolos `I`, `F`. Ao receber o *input* w, a máquina T inicializa a área de trabalho

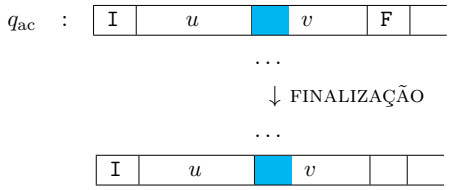


após o que transita para o estado inicial de M, prosseguindo como M faria, excepto quando a computação levar a cabeça de leitura/escrita às células marcadas com os símbolos `I`,`F`. Nessas situações a computação de T deve prosseguir como indicado de seguida



após o que se retomará a computação de M, no estado em que se encontrava.

Isto é suficiente para reconhecer/decidir a mesma linguagem. Para que a máquina calcule a mesma função, há que completar a computação eliminando o marcador `F`, como indicado abaixo, após a computação atingir o estado de aceitação de M.



Os troços da máquina T correspondentes a INICIALIZAÇÃO, ESPAÇAMENTO-I, ESPAÇAMENTO-F e FINALIZAÇÃO são relativamente simples, e deixam-se como exercícios. □

3.2.3 Máquinas multifita

Definição 3.14. MÁQUINA DE TURING MULTIFITA
Uma *máquina de Turing multifita (com $k \in \mathbb{N}$ fitas)* é precisamente como uma máquina de Turing, excepto que

- a *função de transição* é do tipo $\delta : Q \times \Gamma^k \to \hat{Q} \times \Gamma^k \times \{L, R, S\}^k$. △

As noções acima introduzidas nas Definições 3.3–4.1 são facilmente extensíveis para máquinas de Turing multifita, o que se deixa como exercício ao leitor. Aliás, o caso $k = 1$ coincide precisamente com a noção já conhecida. A diferença substantiva está no facto de a máquina trabalhar, em simultâneo, com k fitas de memória, cada uma com a sua cabeça de leitura/escrita. Convenciona-se que o *input* é dado na primeira fita, e o *output*, caso exista, é calculado na última fita.

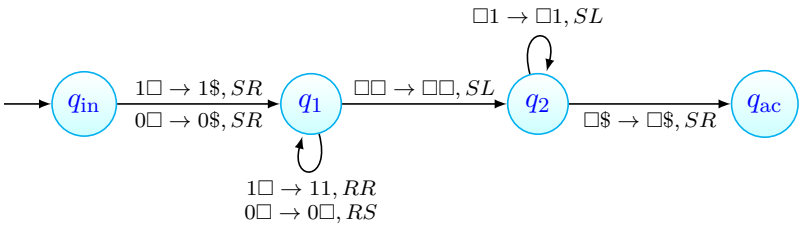
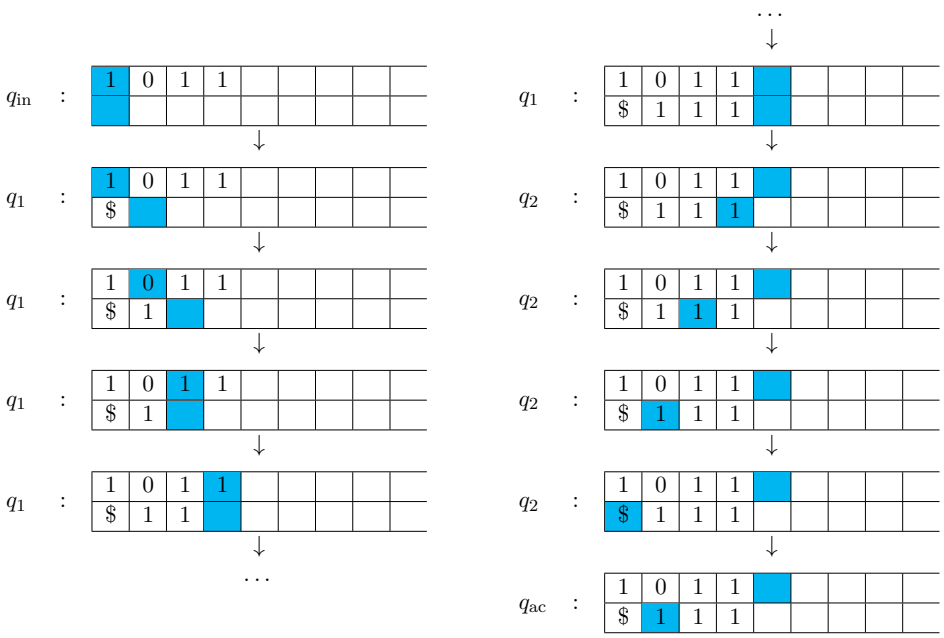


Figura 3.4: Máquina que calcula a função adição (notação unária).

Exemplo 3.15.
Atente-se na máquina com 2 fitas descrita na Figura 3.4. É fácil de verificar que a máquina calcula a função adição de naturais, em notação unária. Por exemplo, para somar 1+2 a máquina evolui da seguinte forma:



△

Não é difícil de perceber que cada máquina multifita pode transformar-se (com algum esforço, é certo) numa máquina equivalente só com uma fita. A ideia passa por codificar numa só fita toda a informação contida nas várias fitas, incluindo a posição das diversas cabeças de leitura/escrita. Por exemplo, com $k = 3$, a configuração correspondente, na máquina multifita, a

0	1	0	1	0				
a	b		c					
x	y	x	y	z	z			

deve fazer-se corresponder na máquina só com uma fita à configuração

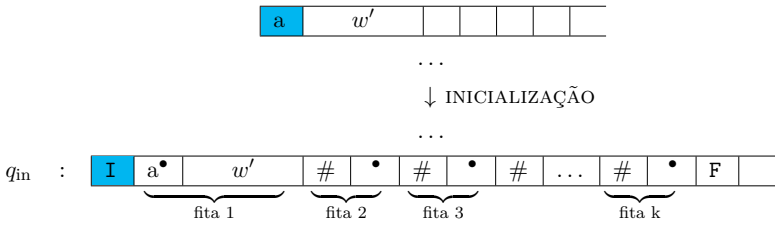
I	$0^\bullet$	1	0	1	0	#	a	b	$\bullet$	c	#	x	y	x	y	z	$z^\bullet$	F	

Proposição 3.16.
Toda a máquina de Turing multifita é equivalente a uma máquina de Turing com apenas uma fita.

Dem. (esboço): Dada uma máquina M com $k > 1$ fitas é possível construir uma máquina T com apenas uma fita, cujas computações são equivalentes. A ideia passa por trabalhar com um alfabeto mais rico $\Gamma \cup \Gamma^\bullet \cup \{\#, \mathtt{I}, \mathtt{F}\}$, usando-se o símbolo # como separador, os símbolos $\mathtt{I}, \mathtt{F}$ como delimitadores, e símbolos

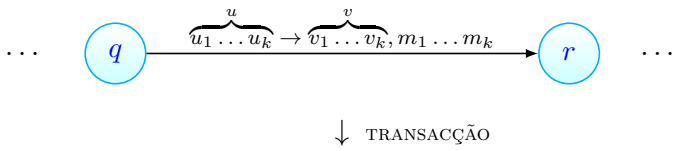
da forma $\gamma^\bullet$ com $\gamma \in \Gamma$ para anotar a posição de cada uma das k cabeças de leitura/escrita.

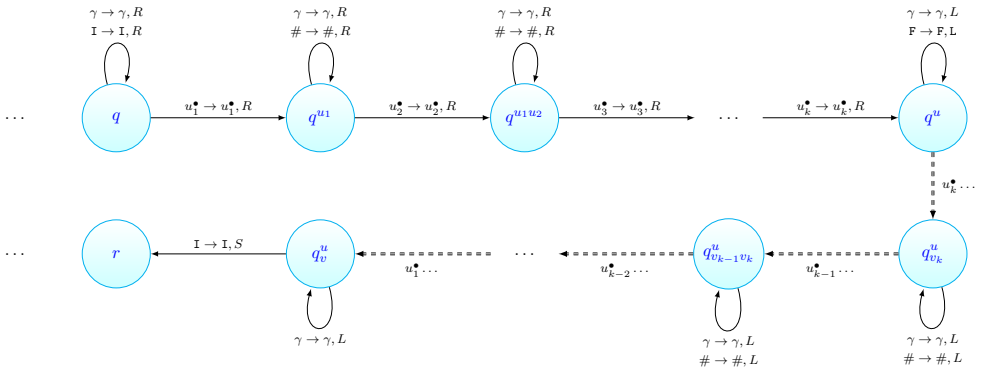
A máquina T começa por inicializar a fita como se segue



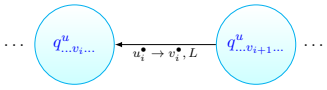
onde assumimos que o *input* é $w = aw'$, ou que $a = \square$ e $w' = \epsilon$ se $w = \epsilon$.

Cada transição de M é simulada em T do seguinte modo: lendo a fita da esquerda para a direita, obter os k símbolos marcados, e de volta a `I` na célula inicial ajustar as marcações da fita de acordo com os símbolos a escrever e os movimentos a efectuar (estes últimos podem obrigar, no troço correspondente a cada uma das fitas da máquina original, a abrir espaços à direita ou à esquerda, com o correspondente arrastamento dos restantes símbolos). Esta ideia resulta na simulação em T, de cada transição de M, por meio de uma sequência de transições, como se indica de seguida.

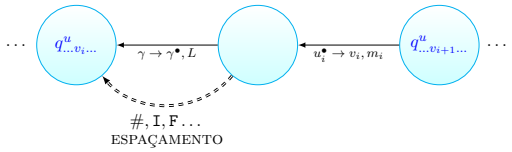




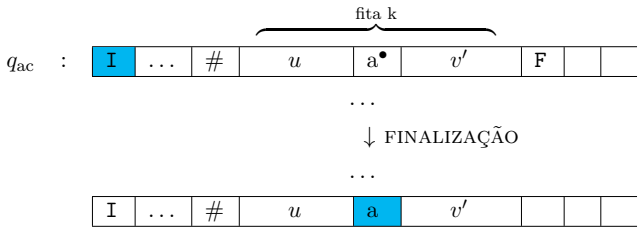
Cada um dos troços assinalados com a seta dupla e tracejada, percorre a fita para a esquerda até encontrar $u_i^\bullet$. Depois, a actividade da máquina depende do movimento m_i. Se $m_i = S$ basta substituir $u_i^\bullet$ por $v_i^\bullet$ e prosseguir para a esquerda.



Se $m_i = R, L$, substitui-se $u_i^\bullet$ por v_i, substituindo-se também o símbolo γ imediatamente à direita/esquerda por $\gamma^\bullet$, o que pode exigir a introdução de um espaço e respectivo arrastamento dos restantes símbolos.



Para devolver o *output* correcto, após a aceitação por M, a máquina T deve substituir o símbolo marcado $\gamma^\bullet$ por γ, bem como eliminar o `F` final, no troço correspondente à última fita, antes de terminar a computação, como ilustrado abaixo



onde assumimos que o *output* é $v = av'$, ou que $a = \square$ caso $v = \epsilon$.

Os troços da máquina T correspondentes a INICIALIZAÇÃO e FINALIZAÇÃO, bem como o detalhe das TRANSACÇÕES de simulação de cada uma das transições de M, incluindo o troço de ESPAÇAMENTO, são relativamente simples, e deixam-se como exercícios. □

3.2.4 Máquinas não-deterministas

Definição 3.17. MÁQUINA DE TURING NÃO-DETERMINISTA
Uma *máquina de Turing não-determinista* é essencialmente como uma máquina de Turing, excepto que

- a *função de transição* é do tipo $\delta : Q \times \Gamma \to \wp(\hat{Q} \times \Gamma \times \{L, R, S\})$. △

As computações de uma máquina de Turing não-determinista organizam-se, em geral, como uma árvore cuja raiz é a configuração inicial e que se ramifica sempre que haja possíveis transições alternativas. Cada árvore de computação pode ser finita ou infinita, apesar de haver sempre um número máximo de ramificações em cada nó. Caso a árvore seja infinita, é claro que terá de haver um ramo infinito (uma escolha de alternativas que leva a uma computação infinita). De facto, cada ramo da árvore corresponde à escolha de uma transição entre as possíveis alternativas. Se $\delta(q, a)$ nunca tiver mais de um elemento, é óbvio que não há não-determinismo, nem escolhas a fazer, pelo que o modelo coincide com a máquina de Turing usual.

Exemplo 3.18.
Considere a máquina não-determinista representada na Figura 3.5. A máquina aceita o *input* (no alfabeto 0,1) se a palavra contém três 1s consecutivos, e rejeita-a em caso contrário. △

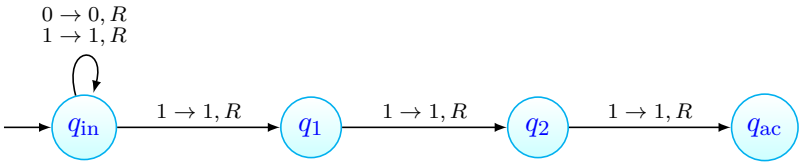


Figura 3.5: Máquina que decide a linguagem das palavras sobre $\{0,1\}$ que têm três 1s consecutivos.

Definição 3.19. ACEITAÇÃO, REJEIÇÃO EM MÁQUINA NÃO-DETERMINISTA
Seja $M = (\Sigma, \Gamma, Q, q_{\text{in}}, q_{\text{ac}}, q_{\text{rj}}, \delta)$ uma máquina de Turing não-determinista.

Uma palavra $w \in \Sigma^*$ diz-se *aceite* por M se alguma computação da máquina dado o *input* w é bem sucedida e termina no estado de aceitação.

Uma palavra $w \in \Sigma^*$ diz-se *rejeitada* por M se todas as computações da máquina dado o *input* w são finitas mas nenhuma termina no estado de aceitação.

Caso uma palavra w seja aceite por M e o *output* em todas as configurações de terminação seja $v \in \Sigma^*$, então definimos $\phi_M(w) = v$. Em caso contrário, $\phi_M(w)$ fica *indefinido*. △

Usamos $L_{\text{ac}}(M), L_{\text{rj}}(M)$ como antes, para denotar as linguagens sobre Σ formadas pelas palavras aceites, resp. rejeitadas, por M.

Diz-se que uma máquina de Turing não-determinista é um *classificador* se todas as computações da máquina são finitas.

Proposição 3.20.
Toda a máquina de Turing não-determinista é equivalente a uma máquina de Turing determinista.

Dem. (esboço): Dada uma máquina não-determinista N, define-se uma máquina determinista D, com 3 fitas, que lhe é equivalente. Tal como em demonstrações anteriores, enriquece-se o alfabeto Γ de N, e trabalha-se com o alfabeto $\Gamma \cup \{\$, \#, \mathtt{I}, \mathtt{F}\} \cup C$ onde $C = \{c_1, \ldots, c_k\}$ é um conjunto finito de símbolos usados para codificar os caminhos da máquina não-determinista (k é o número máximo de opções não-deterministas nalgum estado da máquina, e é sempre limitado pelo número de transições da máquina).

Na fita 1, mantém-se uma cópia do *input* que permanece inalterada durante toda a computação. Na fita 2, codifica-se um caminho que caracterize uma possível computação de N para o *input* dado. A fita 3 emula a fita da máquina N, durante a computação para o *input* dado, de acordo com o caminho descrito na fita 2.

A máquina D começa por inicializar as fitas com ilustrado Figura 3.6. Na fita 2 são reservadas d células para descrever um caminho, espaço esse que depende do *input*. A partir desta configuração, a máquina copia o *input* w para a fita 3. Em seguida, D executa N na fita 3 sobre o caminho na fita 2 até que se verifique uma de duas situações: N atinge um estado de aceitação, ou então o conteúdo da fita 2 não permite continuar a simulação de N (porque

não há mais símbolos na fita 2 ou porque o próximo símbolo dá origem a uma escolha não-determinista inválida). Note-se que na primeira iteração o caminho é constituído exclusivamente por □s.

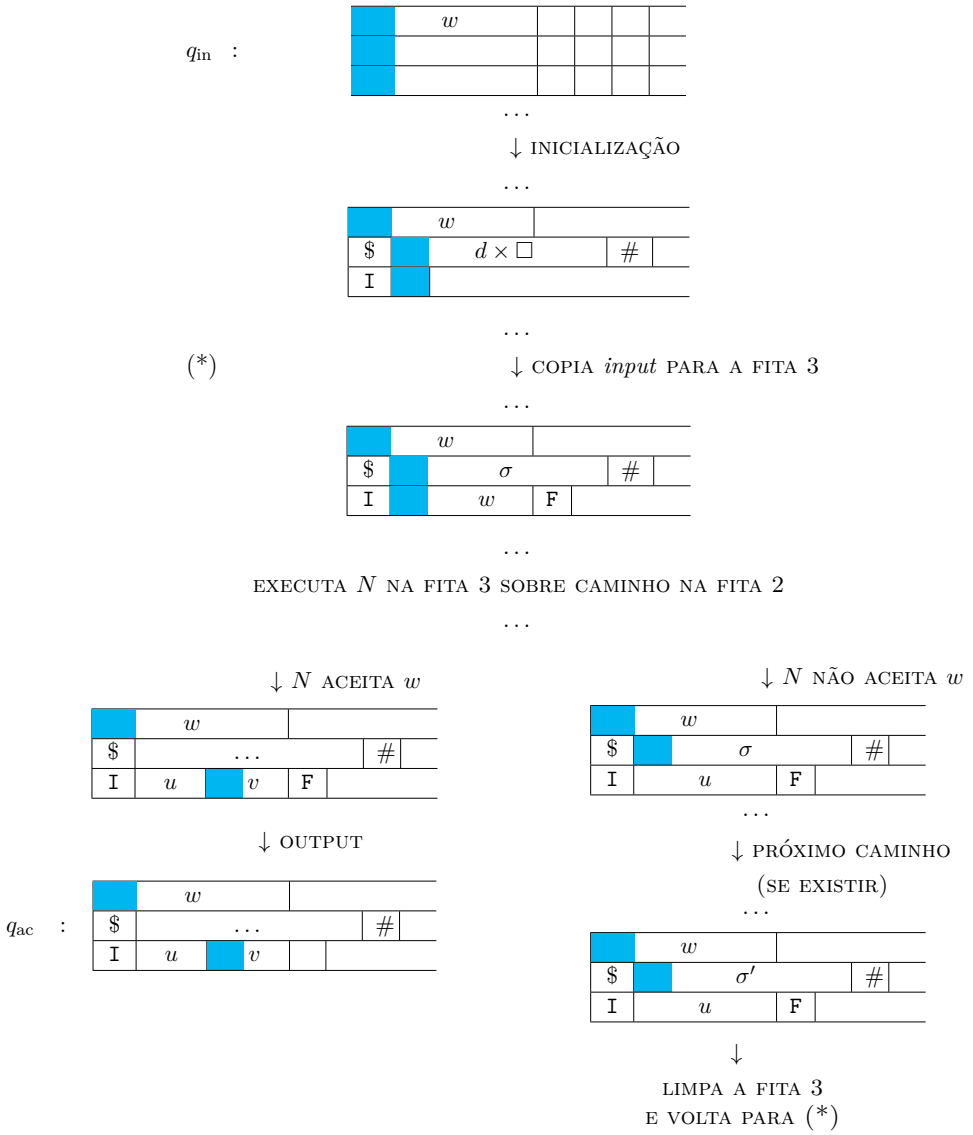


Figura 3.6: Esboço de máquina determinística equivalente a N.

No caso de N aceitar a palavra, D termina no estado de aceitação e devolve a palavra v calculada por N, na fita 3. Para tal, basta apagar o símbolo F do final da fita 3 e colocar a cabeça de leitura/escrita sobre o primeiro símbolo da palavra. Em caso contrário, D escreve na fita 2 o próximo caminho, σ', se existir, e repete o processo anterior, começando por voltar a copiar o conteúdo da fita 1 para a fita 3.

Os caminhos são escritos na fita 2 como se descreve na Figura 3.7. Quando

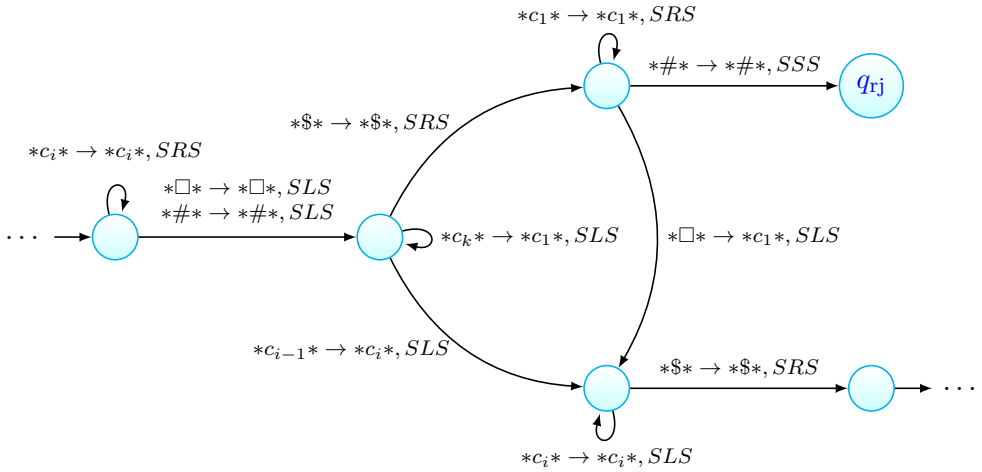


Figura 3.7: Construção dos caminhos de N na máquina da Figura 3.6.

não existe mais nenhum caminho, a máquina D rejeita. Recorde-se que o alfabeto para descrever os caminhos é $C = \{c_1, \ldots, c_k\}$. □

3.3 Máquina universal

Já vimos que o alfabeto $\{0, 1\}$ é suficiente para representarmos qualquer problema. Recorde-se por exemplo a representação canónica de grafos, desenvolvida no final do Capítulo 1. Não será portanto surpreendente que possamos representar sobre o alfabeto $\{0, 1\}$ todas as máquinas de Turing.

Seja $M = (\Sigma, \Gamma, Q, q_{\text{in}}, q_{\text{ac}}, q_{\text{rj}}, \delta)$ uma máquina de Turing, que assumimos, para simplificar mas sem perda de generalidade, estar de acordo com a definição original.

A máquina M tem $n = \#\hat{Q} = \#(Q \cup \{q_{\text{ac}}, q_{\text{rj}}\}) = \#Q + \#\{q_{\text{ac}}, q_{\text{rj}}\} = \#Q + 2$ estados. Denotamos os elementos de $\hat{Q}$ por $q_1, \ldots, q_n$ e assumimos que $q_1 = q_{\text{in}}$, $q_{n-1} = q_{\text{ac}}$ e $q_n = q_{\text{rj}}$. Representamos cada um dos n estados usando palavras de comprimento n com exactamente um 1, nomeadamente

$$\underbrace{100\ldots00}_{q_1 = q_{\text{in}}}, \underbrace{010\ldots00}_{q_2}, \underbrace{001\ldots00}_{q_3}, \ldots, \underbrace{000\ldots10}_{q_{n-1} = q_{\text{ac}}}, \underbrace{000\ldots01}_{q_n = q_{\text{rj}}}.$$

Por outro lado, a máquina M trabalha com $k + 1 = \#\Gamma = \#((\Gamma \setminus \{\square\}) \cup \{\square\}) = \#(\Gamma \setminus \{\square\}) + \#\{\square\} = \#(\Gamma \setminus \{\square\}) + 1$ símbolos distintos. Denotamos os elementos de Γ por $a_0, a_1, \ldots, a_k$ e assumimos que $a_0 = \square$.

Representamos os símbolos de Γ usando palavras de comprimento k. Nomeadamente, representamos $a_0 = \square$ usando apenas 0s, e os símbolos $a_1, \ldots, a_k$ usando palavras com exactamente um 1,

$$\underbrace{000\ldots00}_{a_0 = \square}, \underbrace{100\ldots00}_{a_1}, \underbrace{010\ldots00}_{a_2}, \underbrace{001\ldots00}_{a_3}, \ldots, \underbrace{000\ldots10}_{a_{k-1}}, \underbrace{000\ldots01}_{a_k}.$$

Usamos um símbolo para representar os movimentos da máquina,

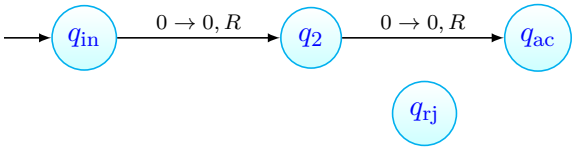


Figura 3.8: Máquina que decide a linguagem das palavras sobre $\{0,1\}$ que começam com dois 0s consecutivos.

$$\underbrace{0}_{L}, \underbrace{1}_{R}.$$

Uma representação canónica da máquina M é então a palavra

$$\underbrace{11\ldots1}_{n \text{ vezes}}0\underbrace{11\ldots1}_{k \text{ vezes}}0\ \text{trans}_1 \ldots \text{trans}_t$$

onde cada $\text{trans}_i \in \{0,1\}^*$, uma palavra de comprimento $2n+2k+1$, representa uma transição,

$$\underbrace{\cdots}_{q_i}\underbrace{\cdots}_{a_j}\underbrace{\cdots}_{q_r}\underbrace{\cdots}_{a_s}\underbrace{\cdots}_{m}$$

desde que $\delta(q_i, a_j) = (q_r, a_s, m)$.

Em vez do alfabeto $\{0,1\}$ qualquer outro alfabeto com pelo menos dois símbolos pode ser usado para representar máquinas de Turing de modo semelhante. Na verdade, pode até mesmo usar-se um alfabeto singular, como $\{1\}$, por exemplo: basta começar por fazer a representação com 0s e 1s atrás descrita e, depois, converter para notação unária o natural (em notação binária) obtido.

A partir do momento em que se fixa um alfabeto para a representação, pode confundir-se cada máquina de Turing com uma sua representação canónica, quando apropriado.

Exemplo 3.21.
A máquina de Turing apresentada na Figura 3.8 reconhece as palavras sobre $\{0,1\}$ que começam com dois 0s consecutivos. Ao contrário de outros exemplos, representamos explicitamente o estado de rejeição, apesar de inacessível, para tornar mais clara a representação. Uma sua representação canónica é, tomando $a_1 = 0$ e $a_2 = 1$, a palavra

$$\overbrace{1111}^{4 \text{ estados}}\ 0\ \overbrace{11}^{2 \text{ símbolos}}\ 0\overbrace{\underbrace{1000}_{q_1=q_{\text{in}}}\underbrace{10}_{a_1=0}\underbrace{0100}_{q_2}\underbrace{10}_{a_1=0}\underbrace{1}_{R}}^{\text{transição}}\overbrace{\underbrace{0100}_{q_2}\underbrace{10}_{a_1=0}\underbrace{0010}_{q_3=q_{\text{in}}}\underbrace{10}_{a_1=0}\underbrace{1}_{R}}^{\text{transição}}.$$

Esta palavra é o natural 16544737557 em notação binária. Assim, a palavra $1^{16544737557} \in \{1\}^*$ é uma representação desta máquina de Turing sobre o alfabeto $\{1\}$. △

Dado um alfabeto Σ, denotamos por $\mathcal{M}^\Sigma$ o conjunto das representações canónicas de máquinas de Turing com alfabeto de entrada/saída Σ.[1] A representação canónica de uma palavra $w \in \Sigma^*$ é denotada por $\text{rep}(w)$.

Proposição 3.22.
Existe uma máquina de Turing U, dita universal*, que para qualquer $M \in \mathcal{M}^\Sigma$ e $w \in \Sigma^*$, onde Σ é um alfabeto e $\$ \notin \Sigma$, satisfaz as seguintes propriedades:*

- *U aceita (resp. rejeita) $M\$\text{rep}(w)$ se e só se M aceita (resp. rejeita) w;*
- $\phi_U(M\$\text{rep}(w)) = \text{rep}(\phi_M(w))$.

Dem.: Seja U a máquina de Turing com seis fitas ilustrada na Figura 3.9 que, ao receber uma palavra x como *input*, começa por verificar se x é da forma $M\$\text{rep}(w)$ para alguma máquina de Turing $M \in \mathcal{M}^\Sigma$ e $w \in \Sigma^*$. Se x não for dessa forma então U rejeita x. Caso contrário, U entra na fase de inicialização:

(1) fase de inicialização. Nesta fase U copia a sequência de 1s correspondente ao número de estados de M para a fita 2 (assumimos que as máquinas são representadas sobre o alfabeto $\{0,1\}$), a sequência de 1s correspondentes ao número de símbolos de M para a fita 3, as transições de M para a fita 4, coloca o estado inicial de M na fita 5 e copia $\text{rep}(w)$ para a fita 6. As cabeças de leitura/escrita destas fitas são colocados no início destas palavras. De seguida U passa para a fase de simulação.

(2) fase de simulação. Nesta fase U faz os seguintes passos repetidamente até ou M abortar ou ficar num estado de aceitação ou rejeição: lê o símbolo onde está posicionado a cabeça de leitura/escrita da fita 6 e o estado atual de M na fita 5 e percorre a fita 3 procurando a transição de M para este símbolo e estado. Se não encontrar transição, U aborta. Se encontrar transição, U altera o estado de M na fita 5 de acordo com o que essa transição indica, U escreve na fita 6 o símbolo indicado por essa transição, e U desloca a cabeça de leitura/escrita da fita 6 de acordo com o indicado por essa transição. Se após isto o estado na fita 5 for de aceitação então U termina aceitando, se o estado for de rejeição então U termina rejeitando.

Se o alfabeto usado para representar as máquinas de Turing for singular, as representações envolvidas (que podem ser entendidas como naturais em notação unária), terão de ser previamente convertidas em representações com pelo menos dois símbolos (naturais em representação binária, por exemplo), o que se deixa como exercício. □

Uma máquina universal deve ser entendida como uma máquina programável, tal como os computadores modernos. Este é um resultado muito importante que decorre do trabalho de Turing.

[1] Se $\{0,1\}$ for o alfabeto escolhido para a representação das máquinas e $\#\Sigma = n$, então $\mathcal{M}^\Sigma$ inclui as máquinas 1^x01^y0w com $y \geq n$, assumindo que existem $y-n$ símbolos auxiliares (para além de $\square$).

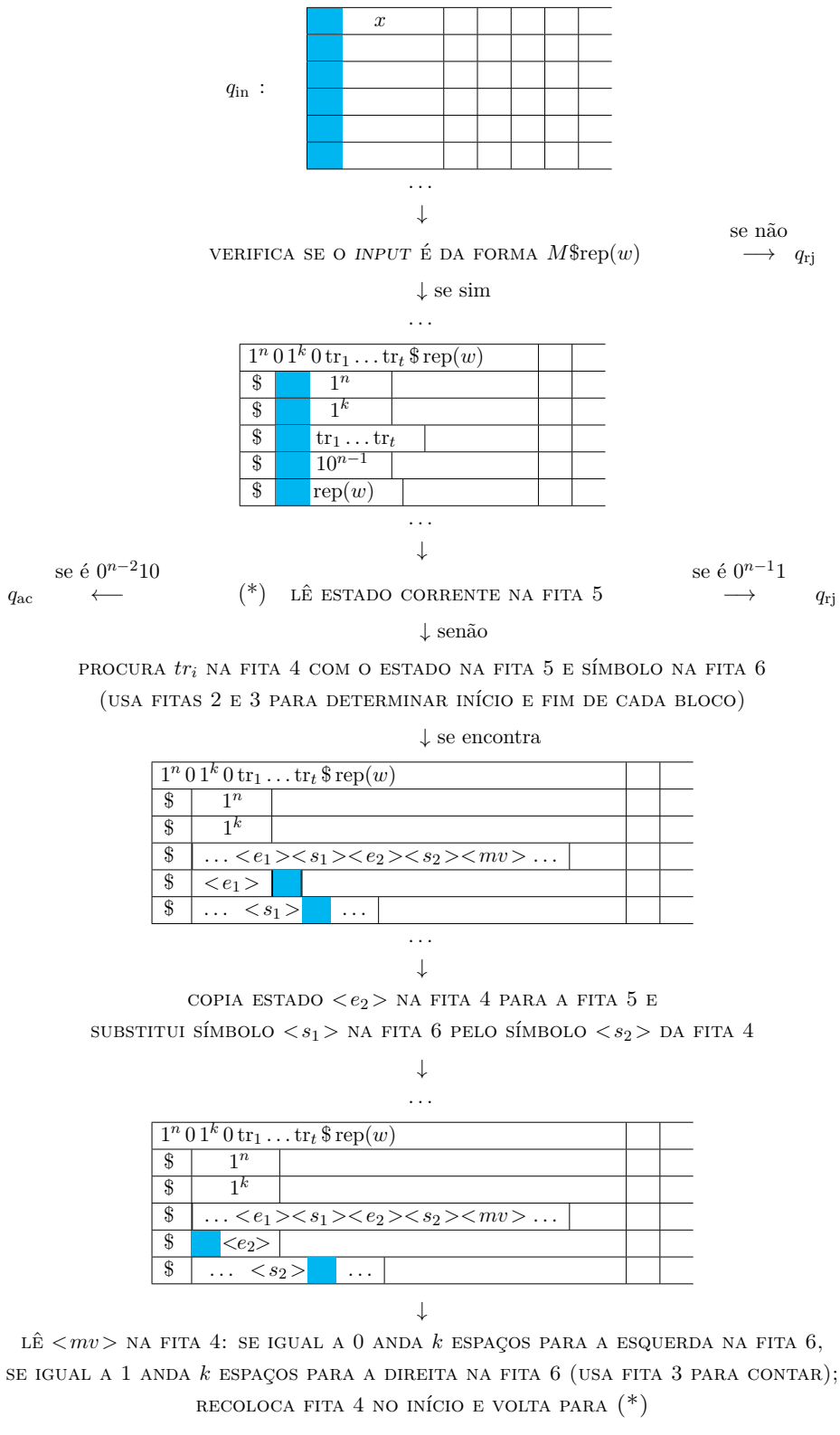


Figura 3.9: Esboço da máquina de Turing universal.

3.4 Modelos de computação e postulado de Church-Turing

Temos visto até agora várias variantes de máquinas de Turing equivalentes em poder computacional à máquina de Turing proposta inicialmente. Mas muitos outros modelos de computação têm sido propostos, muitos deles bastante diferentes de máquinas de Turing. No entanto todos têm a característica fundamental que caracteriza uma máquina de Turing: acesso sem restrições a memória ilimitada, execução em cada passo de uma quantidade finita de trabalho, e são por isso equivalentes a máquinas de Turing, como tem sido mostrado ao longo do tempo.

Mas o que é um algoritmo? Intuitivamente, um algoritmo é uma colecção finita de instruções simples para realizar alguma tarefa. Mas, como podemos definir rigorosamente um algoritmo? Por exemplo se quisermos mostrar que não existe um algoritmo para resolver um problema, como o podemos fazer se não tivermos uma definição rigorosa de algoritmo?

A definição rigorosa de algoritmo foi proposta em 1936 independentemente por Alonzo Church e Alan Turing. Church propôs que um algoritmo é um processo que pode ser descrito pelo que ele designou de *cálculo-λ*. Turing propôs que um algoritmo é um processo que pode ser descrito por uma máquina de Turing. Demonstrou-se que estas duas definições são equivalentes e foi proposto o seguinte postulado que até agora tem resistido ao passar do tempo:

Definição 3.23. Postulado de Church-Turing
A noção intuitiva de algoritmo coincide com a noção de algoritmo sobre máquinas de Turing. Ou mais rigorosamente: uma tarefa é intuitivamente computável (i.e., computável por uma pessoa usando caneta e papel) se e somente se é computável por uma máquina de Turing. △

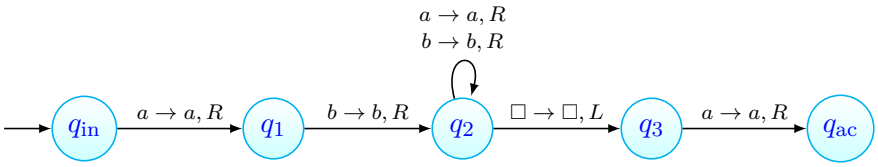
Se fizermos uma descrição suficientemente rigorosa de um algoritmo, como faremos algumas vezes na sequência, o Postulado de Church-Turing garante-nos que existe, por exemplo, uma máquina de Turing ou um programa em *Python* que realiza a mesma tarefa.

De notar que há vários outros modelos formais de computação, alguns até historicamente tão importantes como os modelos de Turing e Church. As *funções recursivas* de Gödel são um exemplo. Apesar da sua diversidade, demonstra-se que todos estes modelos são equivalentes.

Exercícios

1 Emulação de máquinas

1. Seja M a máquina de Turing com alfabeto de entrada/saída $\Sigma = \{a, b\}$ e alfabeto de trabalho $\Gamma = \{a, b, \square\}$, cuja representação gráfica é a seguinte:

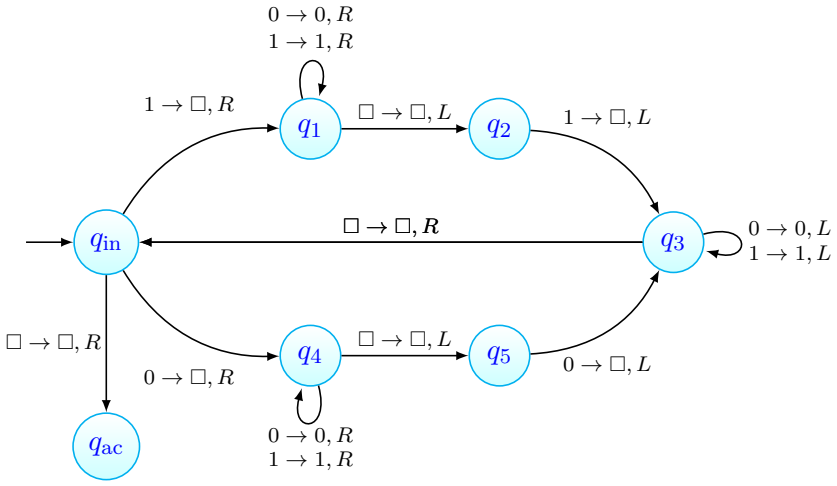


(a) Descreva a evolução de M a partir da configuração inicial com *input*:

i. aab; ii. abb; iii. aba; iv. ab.

(b) Verifique informalmente que a linguagem reconhecida por M é o conjunto das palavras sobre Σ que começam em ab e terminam em a.

2. Seja M a máquina de Turing com alfabeto de entrada/saída $\Sigma = \{a, b, c\}$ e alfabeto de trabalho $\Gamma = \{a, b, c, \square\}$, cuja representação gráfica é a seguinte:

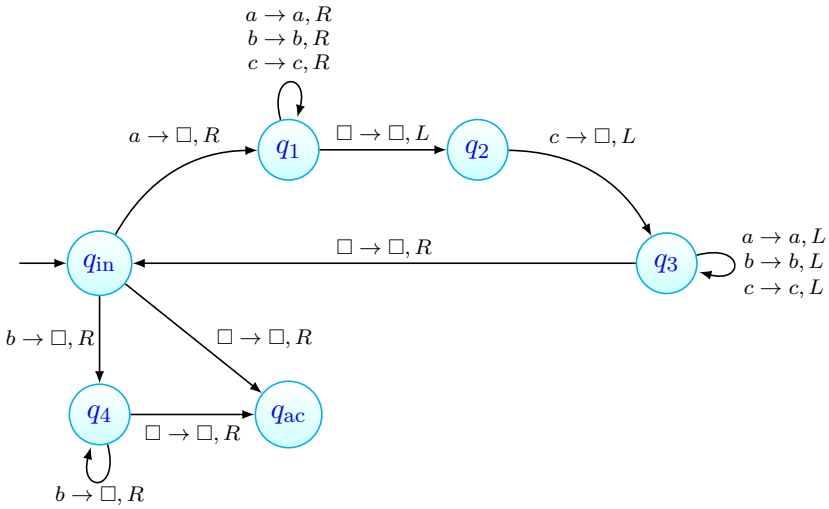


(a) Descreva a evolução de M a partir da configuração inicial com *input*:

i. 001; ii. 0110; iii. 010; iv. 100001.

(b) Verifique informalmente que a linguagem reconhecida por M é o conjunto das palavras sobre Σ do tipo ww^R (recorde que w^R é a palavra w reflectida).

3. Seja M a máquina de Turing com alfabeto de entrada/saída $\Sigma = \{a, b, c\}$ e alfabeto de trabalho $\Gamma = \{a, b, c, \square\}$, cuja representação gráfica é a seguinte:



(a) Descreva a evolução de M a partir da configuração inicial com *input*:

i. *ac*; ii. *aabc*; iii. *abbc*; iv. *aabcc*;

(b) Verifique informalmente que a linguagem reconhecida por M é o conjunto das palavras sobre Σ do tipo $a^n b^m c^n$ com $m, n \in \mathbb{N}_0$.

2 Especificação de máquinas

1. Especifique uma máquina de Turing que reconheça as seguintes linguagens sobre o alfabeto $\{0, 1\}$:

   (a) a linguagem das palavras que têm pelo menos um 0;

   (b) a linguagem das palavras que começam e terminam em 1;

   (c) a linguagem das palavras cujo comprimento é par;

   (d) a linguagem das palavras que têm um número ímpar de 0s;

   (e) a linguagem das palavras do tipo $0^n 1^n$ com $n \in \mathbb{N}_0$;

   (f) a linguagem das palavras do tipo $0^n 1^{n+1}$ com $n \in \mathbb{N}_0$;

   (g) a linguagem das palavras do tipo $0^{2n} 1^n$ com $n \in \mathbb{N}_0$;

   (h) a linguagem das palavras do tipo $0^n 1^m$ com $m, n \in \mathbb{N}_0$ e $n > m$;

   (i) a linguagem das palavras do tipo $0^n 1^m$ com $m, n \in \mathbb{N}_0$ e $n < m$;

   (j) a linguagem das palavras que são palíndromos.

2. Defina uma máquina de Turing que desloque o seu *input* uma célula para a direita. O que seria diferente se o deslocamento fosse para a esquerda?

3. Especifique uma máquina de Turing que reconheça as seguintes linguagens:

   (a) $L = \{a^n b^m c^{n+m} : \ n, m \in \mathbb{N}_0\}$;

   (b) $L = \{a^m + a^n = a^{m+n} : \ n, m \in \mathbb{N}_0\}$;

(c) $L = \{(0^n, 0^m, 0^{n+m}) : \ n, m \in \mathbb{N}_0\}$.

4. Especifique uma máquina de Turing que reconheça as seguintes linguagens:

   (a) $L = \{a^n b^{2n} : \ n \in \mathbb{N}_0\}$;
   (b) $L = \{a^n \times 2 = a^{2n} : \ n \in \mathbb{N}_0\}$;
   (c) $L = \{(0^n, 0^{2n}) : \ n \in \mathbb{N}_0\}$.

5. Especifique uma máquina de Turing que reconheça as seguintes linguagens:

   (a) $L = \{a^n b^{3n} : \ n \in \mathbb{N}_0\}$;
   (b) $L = \{a^n \times 3 = a^{3n} : \ n \in \mathbb{N}_0\}$;
   (c) $L = \{(0^n, 0^{3n}) : \ n \in \mathbb{N}_0\}$.

6. Especifique uma máquina de Turing que reconheça as seguintes linguagens ($n \div 2$ é o quociente da divisão inteira por 2):

   (a) $L = \{a^n b^{n \div 2} : \ n \in \mathbb{N}_0\}$;
   (b) $L = \{a^n \div 2 = a^{n \div 2} : \ n \in \mathbb{N}_0\}$;
   (c) $L = \{(0^n, 0^{n \div 2}) : \ n \in \mathbb{N}_0\}$.

7. Especifique um classificador para as seguintes linguagens sobre $\{0, 1, 2\}$:

   (a) a linguagem das palavras do tipo $w2w$ com $w \in \{0, 1\}^*$;
   (b) a linguagem das palavras do tipo $0^n 1^n 2^n$ com $n \in \mathbb{N}_0$.

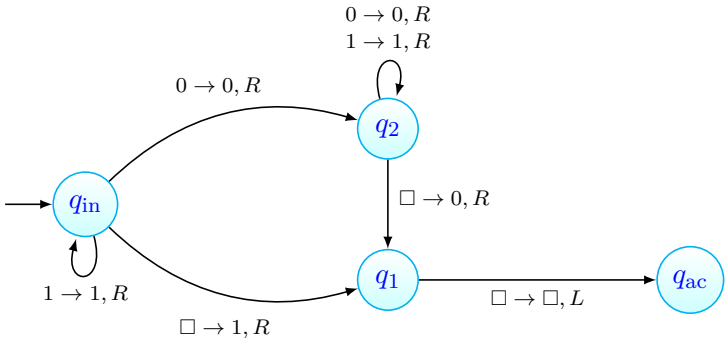
8. Mostre que são decidíveis as seguintes linguagens sobre $\{0, 1\}$:

   (a) a linguagem das palavras nas quais o número de 0s é igual ao número de 1s.
   (b) a linguagem das palavras nas quais o número de 0s é o dobro do número de 1s.

9. Mostre que é decidível a linguagem $L = \{1^n : \ n$ é uma potência de $2\}$ sobre o alfabeto $\{1\}$.

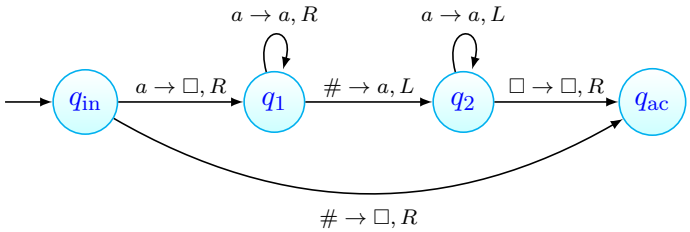
3 Cálculo de funções

1. Considere a máquina de Turing $M = (\Sigma, \Gamma, Q, q_{\text{in}}, q_{\text{ac}}, q_{\text{rj}}, \delta)$ com alfabeto de entrada/saída $\Sigma = \{0, 1\}$, alfabeto de trabalho $\Gamma = \{0, 1, \square\}$ e cuja representação gráfica é a seguinte:



Verifique informalmente que M calcula a função $\text{AND}: \{0,1\}^* \to \{0,1\}$ dada por $\text{AND}(w) = 0$ se em w existe pelo menos um 0 e $\text{AND}(w) = 1$ em caso contrário.

2. Considere a máquina de Turing $M = (\Sigma, \Gamma, Q, q_{\text{in}}, q_{\text{ac}}, q_{\text{rj}}, \delta)$ com alfabeto de entrada/saída $\Sigma = \{a, \#\}$, alfabeto de trabalho $\Gamma = \{a, \#, \square\}$ e cuja representação gráfica é a seguinte:



Verifique informalmente que M calcula a função $f: \{a, \#\}^* \to \{a\}^*$ dada por $f(a^n \# a^m) = a^{n+m}$, para $n, m \in \mathbb{N}_0$.

3. Especifique uma máquina de Turing que calcule $f : \{a, b\}^* \to \{0, 1\}$ tal que $f(v) = 1$ se v tem comprimento par e $f(v) = 0$ em caso contrário.

4. Considere a função $\text{OR} : \{0,1\}^* \to \{0,1\}$ tal que $\text{OR}(w) = 1$ se em w existe pelo menos um 1 e $\text{OR}(w) = 0$ em caso contrário. Especifique uma máquina de Turing que calcule a função OR.

5. Considere a função $\text{XOR} : \{0,1\}^* \to \{0,1\}$ tal que $\text{XOR}(w) = 1$ se em w existe um e um só 1 e $\text{XOR}(w) = 0$ em caso contrário. Especifique uma máquina de Turing que calcule a função XOR.

6. Especifique uma máquina de Turing que calcule a função *sucessor*, isto é, a função que a cada número natural n faz corresponder o seu sucessor, assumindo que:

   (a) n está representado em notação unária;

   (b) n está representado em notação binária;

   (c) n está representado em notação decimal.

7. Especifique uma máquina de Turing que calcule a função *predecessor*, isto é, a função que a cada número natural n faz corresponder o seu predecessor, caso exista, assumindo que:

   (a) n está representado em notação unária;

   (b) n está representado em notação binária.

8. Especifique uma máquina de Turing que calcule a função *dobro*, isto é, a função que a cada número natural n faz corresponder o seu dobro, assumindo que:

   (a) n está representado em notação unária;

   (b) n está representado em notação binária.

9. Especifique uma máquina de Turing que calcule a função *triplo*, isto é, a função que a cada número natural faz corresponder o seu triplo, assumindo que:

   (a) n está representado em notação unária;

   (b) n está representado em notação binária.

10. Especifique uma máquina de Turing que calcule a função que a cada número natural n faz corresponder 1 se ele é par e 0 em caso contrário, assumindo que:

    (a) n está representado em notação unária;

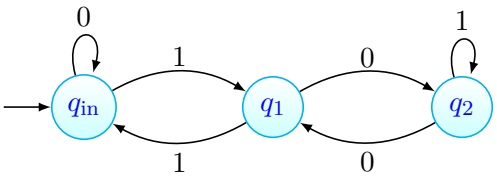
    (b) n está representado em notação binária.

11. Especifique uma máquina de Turing que calcule a função que a cada número natural n faz corresponder 1 se ele é múltiplo de 3 e 0 em caso contrário, assumindo que:

    (a) n está representado em notação unária;

    (b) n está representado em notação binária.

    Sugestão: note que, para cada palavra w sobre $\{0,1\}$ correspondente à representação em notação binária do número n, o autómato finito determinista



    termina no estado q_0 quando $n \bmod 3 = 0$, termina no estado q_1 quando $n \bmod 3 = 1$, e termina no estado q_2 quando $n \bmod 3 = 2$.

12. Especifique uma máquina de Turing que calcule a função que a cada número natural n faz corresponder o resto da sua divisão inteira por 2, assumindo que:

(a) n está representado em notação unária;

(b) n está representado em notação binária.

13. Especifique uma máquina de Turing que calcule a função que a cada número natural faz corresponder o resto da sua divisão inteira por 3, assumindo que:

    (a) n está representado em notação unária;

    (b) n está representado em notação binária.
    Sugestão: ver Exercício 3.11.b.

14. Especifique uma máquina de Turing que calcule $f : \{a, b, c\}^* \to \{0, 1\}^*$ dada por $f(w) = 0$ se w começa por c ou é a palavra vazia, e $f(w) = 1^{n+m}$ onde n é o número de as que ocorrem em w e m é o número de bs que ocorrem em w, em caso contrário.

15. Especifique uma máquina de Turing que calcule $f : \{a, b\}^* \to \{a, b, \#\}^*$ tal que $f(w) = w\#w$ (isto é, a máquina faz uma cópia do conteúdo da fita na configuração inicial, usando o símbolo # para separar a cópia da palavra original).

16. Especifique uma máquina de Turing que calcule $f : \{0, 1\}^* \to \{0, 1\}^*$ que converte a representação em notação binária de um número natural na correspondente representação em notação unária. Por exemplo, $f(110) = 111111$. Sugestão: pode calcular o predecessor do *input* binário e o sucessor do *output* unário.

17. Especifique uma máquina de Turing que calcule $f : \{0, 1\}^* \to \{0, 1\}^*$ que converte a representação em notação unária de um número natural na correspondente representação em notação binária. Por exemplo, $f(11111) = 101$. Sugestão: pode calcular o predecessor do *input* unário e o sucessor do *output* binário.

18. Especifique uma máquina de Turing que calcule a função que a cada par de números naturais n e m faz corresponder o natural $n - m$ se $n > m$ e 0 em caso contrário, assumindo que n e m se encontram representados em notação unária.

4 Especificação de máquinas multifita

1. Especifique uma máquina de Turing de 2 fitas que decida a linguagem $\{a^n b^n : \ n \in \mathbb{N}_0\}$ sobre o alfabeto $\{a, b\}$.

2. Especifique uma máquina de Turing de 2 fitas que decida a linguagem das palavras sobre $\{a, b\}$ que são palíndromos.

3. Especifique uma máquina de Turing de 2 fitas que decida a linguagem das palavras sobre o alfabeto $\{a, b, \#\}$ do tipo $w\#w$ com $w \in \{a, b\}^*$.

4. Especifique uma máquina de Turing de 2 fitas que decida a linguagem $\{a^n b^n c^n : n \in \mathbb{N}_0\}$ sobre o alfabeto $\{a, b, c\}$. Repita o exercício, mas agora com uma máquina de 3 fitas.

5. Especifique uma máquina de Turing de 2 fitas que decida a linguagem das palavras sobre o alfabeto $\{a, b\}$ nas quais o números de as é igual ao número de bs. Repita o exercício, mas agora com uma máquina de 3 fitas.

6. Especifique uma máquina de Turing de 3 fitas que decida a linguagem das palavras sobre o alfabeto $\{0, 1, 2\}$ cujo comprimento é maior ou igual que a soma dos dígitos que as constituem.

7. Especifique uma máquina de Turing de 2 fitas que decida a linguagem $\{ww : w \in \{0, 1\}^*\}$ sobre o alfabeto $\{0, 1\}$.

8. Especifique uma máquina de Turing com 2 fitas que converta a representação em notação binária de um natural na correspondente representação em notação unária.

9. Especifique uma máquina de Turing com 2 fitas que converta a representação em notação unária de um natural na correspondente representação em notação binária.

10. Especifique uma máquina de Turing com 2 fitas que calcule a função que a cada palavra w sobre o alfabeto $\{a, b, c\}$ faz corresponder a palavra $a^{n_a+n_c} b^{n_b+n_c}$ onde n_a é o número de as que ocorre em w, n_b é o número de bs que ocorre em w, e n_c é o número de cs que ocorre em w.

11. Especifique uma máquina de Turing com 2 fitas que calcule a função referida no Exercício 3.18. Repita o exercício, mas agora com uma máquina de 3 fitas.

12. Especifique uma máquina de Turing que calcule a função que a cada palavra w sobre o alfabeto $\{0, 1\}$ faz corresponder a palavra 1^n onde n é o tamanho do maior bloco de símbolos consecutivos iguais que ocorre em w. Nomeadamente, para a palavra $w = 00100001110$ o resultado deverá ser 1111.

13. Especifique uma máquina de Turing que calcule a função que a cada natural n faz corresponder n^2. Use notação unária para os naturais. Sugestão: note que $(n + 1)^2 = n^2 + 2n + 1$.

14. Especifique uma máquina de Turing que calcule a função que a cada natural n faz corresponder 2^n. Use notação unária para os naturais.

15. Especifique uma máquina de Turing que calcule a função f que a cada número natural $n \in \mathbb{N}$ faz corresponder o natural $f(n)$ que melhor aproxima $\log_2(n)$ por excesso, onde os valores são tomados em representação unária. Os valores de f evoluem como se ilustra na tabela seguinte:

n	1	2	3	4	5	6	7	8	9	...	16	17	...
$f(n)$	0	1	2	2	3	3	3	3	4	...	4	5	...

16. Revisite os exercícios das Secções 2 e 3 e veja quais são os que poderiam ter vantagem em serem resolvidos com máquinas de Turing com mais de uma fita.

5 Especificação de máquinas não-deterministas

1. Especifique uma máquina de Turing não-determinista que decida a linguagem das palavras sobre o alfabeto $\{0,1\}$ que terminam em 0.

2. Especifique uma máquina de Turing não-determinista que decida a linguagem $\{ww : w \in \{a,b\}^*\}$ sobre o alfabeto $\{a,b\}$.

3. (Uma versão d)O problema SUBSETSUM consiste em saber se é verdade ou não que um dado valor y pode ser obtido como a soma de valores numa dada lista de naturais $x_1, \ldots, x_n$. Ou seja, trata-se de saber se existe $S \subseteq \{1, \ldots, n\}$ tal que $\sum_{i \in S} x_i = y$. Por exemplo, para a lista $1, 2, 3, 4, 5$ e $y = 11$, é verdadeiro pois $2 + 4 + 5 = 11$. Especifique uma máquina de Turing não-determinista que aceite palavras da forma $x_1 \# \ldots \# x_n \$ y$ se e somente se y pode ser obtido como a soma de elementos da lista $x_1, \ldots, x_n$, assumindo que:

   (a) y e x_i, para $i = 1, \ldots, n$, estão representados em notação unária;

   (b) y e x_i, para $i = 1, \ldots, n$, estão representados em notação binária.

4. Considere a linguagem K sobre o alfabeto $\{1, \$, \&, \#\}$ constituída por todas as palavras da forma

$$v_1 \$ p_1 \& v_2 \$ p_2 \& \ldots \& v_n \$ p_n \# v \$ p$$

   onde $v_1, p_1, \ldots, v_n, p_n, v, p \in \{1\}^*$, para as quais existe um subconjunto I de $\{1, \ldots, n\}$ tal que $v \geq \sum_{i \in I} v_i$ e $p \leq \sum_{i \in I} p_i$, com as somas tomadas sobre a representação em unário dos números naturais. Por exemplo, $11\$11\&111\$11\&1\$1\#1111\$111 \in K$ (com $I = \{2, 3\}$) porque $v \geq v_2 + v_3$, isto é, $4 \geq 3 + 1$, e $p \leq p_2 + p_3$, isto é, $3 \leq 2 + 1$. Por outro lado, $11\$11\&11\$1\#111\$111 \notin K$. Especifique uma máquina de Turing não-determinista que decida S.

5. O problema SAT consiste em saber se uma dada fórmula proposicional na forma normal conjuntiva é ou não satisfatível. Uma versão de SAT pode ser representada considerando uma linguagem S sobre o alfabeto $\{0, 1, 2, \$\}$ constituída por todas as palavras da forma

$$\overbrace{x_{01} \ldots x_{0n}}^{x_0} \$ \overbrace{x_{11} \ldots x_{1n}}^{x_1} \$ \ldots \$ \overbrace{x_{k1} \ldots x_{kn}}^{x_k}$$

   onde $n, k \in \mathbb{N}$ e cada $x_{ij} \in \{0, 1, 2\}$, para as quais exista uma palavra $y = y_1 \ldots y_n \in \{1, 2\}^*$ tal que para cada $i \in \{0, \ldots, k\}$ se tenha $x_{ij} = y_j$

para algum $j \in \{1, \ldots, n\}$. Por exemplo, $010\,\$\,201\,\$\,220\,\$\,112 \in S$ pois $y = 212$ coincide com $x_0 = 010$ na segunda posição, coincide com $x_1 = 201$ na primeira posição, coincide com $x_2 = 220$ na primeira posição, e coincide com $x_3 = 112$ na segunda (e também na terceira) posição. Por outro lado, a palavra $22\,\$\,21\,\$\,10 \notin S$. Especifique uma máquina de Turing não-determinista que decida S.

6. Um número natural n maior do que 1 diz-se composto se tiver um divisor d com $1 < d < n$. Defina uma máquina de Turing não-determinista que aceite números naturais (em notação unária) se e somente se forem números compostos. Sugestão: use notação unária e construa uma máquina de Turing que, dado n, construa d não-deterministicamente.

7. Considere a linguagem L sobre o alfabeto $\{0, 1, \$\}$ constituída por todas as palavras da forma

$$x_{11} \ldots x_{1n}\,\$\,x_{21} \ldots x_{2n}\,\$\,\ldots\,\$\,x_{n1} \ldots x_{nn}$$

onde $n \geq 0$ e cada $x_{ij} \in \{0, 1\}$, para as quais exista $u = u_1 \ldots u_n \in \{a, b, c\}^*$ tal que para quaisquer $i, j \in \{1, \ldots, n\}$ se tenha $u_i \neq u_j$ sempre que $x_{ij} = 1$. Por exemplo, $0110\,\$\,0011\,\$\,0001\,\$\,0000 \in L$ pois $u = abca$ é tal que $x_{12} = 1$ mas $u_1 = a \neq b = u_2$, $x_{13} = 1$ mas $u_1 = a \neq c = u_3$, $x_{23} = 1$ mas $u_2 = b \neq c = u_3$, $x_{24} = 1$ mas $u_2 = b \neq a = u_4$, e $x_{34} = 1$ mas $u_3 = c \neq a = u_4$. A palavra $0111\,\$\,0010\,\$\,0000\,\$\,0011 \notin L$. Especifique uma máquina de Turing não-determinista que decida K.

8. Recorde a representação (canónica) de grafos orientados que se inicia pelo número de nós (em notação unária) seguidos por um 0 e depois pela representação das várias arestas. Cada aresta é representada simplesmente pela sequência dos seus nós origem e destino. Cada nó é identificado com um número natural n e a representação de um nó é uma sequência de comprimento igual ao número de nós, com um 1 na posição n contada da esquerda para a direita e 0s nas restantes posições. Por exemplo, se houver 4 nós, a representação do grafo inicia-se com 1111 (quatro nós) seguida de um 0. O nó 1 é representado por 1000, o nó 2 por 0100, o nó 3 por 0010 e o nó 4 por 0001. Uma aresta do nó 1 para o nó 3 representa-se por 10000010. Por exemplo, 111101000001000100010 representa o grafo com 4 nós e duas arestas, uma do nó 1 (1000) para o nó 3 (0010) e outra do nó 3(0010) para si próprio.

   (a) Defina uma máquina de Turing não-determinista que aceite as palavras sobre alfabeto $\{0, 1\}$ se e somente se estas correspondem à representação canónica de um grafo.

   (b) Defina uma máquina de Turing não-determinista que, dada a representação canónica de um grafo, verifica se este tem duas arestas repetidas.

   (c) Defina uma máquina de Turing não-determinista que, dada a representação canónica de um grafo e um vértice, verifica se existe alguma aresta do grafo com origem no vértice dado.

6 Exercícios complementares

1. Demonstre que qualquer máquina de Turing é equivalente a uma máquina cuja computação nunca aborta. Sugestão: transforme a função de transição da máquina numa função total, introduzindo um estado espúrio.

2. Defina uma noção de *composição sequencial* de máquinas de Turing, de modo a que a máquina composta $M_1; M_2$ satisfaça $\phi_{M_1;M_2} = \phi_{M_2} \circ \phi_{M_1}$. Sugestão: a máquina composta deverá ter o estado inicial de M_1, os estados de terminação de M_2, e identificar o estado de aceitação de M_1 com o estado inicial de M_2.

3. Defina uma noção de *composição paralela* de máquinas de Turing, de modo a que a máquina composta $M_1||M_2$ simule a computação de ambas as máquinas. Sugestão: considere uma máquina com 2 fitas, e estados que correspondem a pares de estados das máquinas M_1 e M_2, e em que cada transição com um par de símbolos corresponde ao respectivo par de transições em cada uma das máquinas (note que, dependendo do propósito da computação paralela, a construção indicada pode ser completada de diferentes formas, nomeadamente com uma fase de pré-processamento em que as 2 fitas sejam inicializadas da forma desejada a partir de um dado *input*, bem como uma fase de finalização em que se tiram conclusões a partir da eventual aceitação/rejeição de cada uma das máquinas).

4. Recorde a noção de *linguagem regular*. Demonstre que para toda a linguagem regular L sobre um alfabeto Σ existe uma máquina de Turing que aceita as palavras de L e rejeita as de $\overline{L}$. Sugestão: construa uma máquina de Turing a partir de um *autómato finito* para a linguagem.

5. Seja Σ um alfabeto. Valide a representação proposta no Exercício 2.2 do Capítulo 1, mostrando que se $L \in \mathcal{L}^{\Sigma}$ é representada por $L^{01} \in \mathcal{L}^{\{0,1\}}$ e $f \in \mathcal{F}^{\Sigma}$ é representada por $f^{01} \in \mathcal{F}^{\{0,1\}}$, então L é reconhecível (resp. decidível) se e só se L^{01} é reconhecível (resp. decidível), e que existe uma máquina de Turing que calcula f se e só se existe uma máquina de Turing que calcula f^{01}.

6. Demonstre que toda a máquina de Turing com $\Sigma = \{0,1\}$ e $\Gamma \neq \{0,1,\square\}$ é equivalente a uma máquina de Turing com $\Sigma = \{0,1\}$ e $\Gamma = \{0,1,\square\}$.

7. Seja Σ um alfabeto. Demonstre que existe uma máquina de Turing M tal que $L_{\text{ac}}(M) = \mathcal{M}^{\Sigma} \subseteq \{0,1\}^*$ é a linguagem formada pelas palavras que denotam máquinas de Turing com alfabeto de entrada/saída Σ na representação canónica. Altere a máquina construída por forma a que também rejeite todas as palavras em $\{0,1\}^*$ que não representem tais máquinas de Turing.

8. Seja Σ um alfabeto. Especifique uma máquina de Turing que, ao receber como *input* $M \in \mathcal{M}^{\Sigma}$, modifica M de modo a que todas as suas computações bem sucedidas passem a ser computações que terminam no

estado de aceitação, ou seja, especifique uma máquina de Turing que calcule $f : \{0,1\}^* \to \{0,1\}^*$ tal que $f(M) \in \mathcal{M}^\Sigma$, para cada $M \in \mathcal{M}^\Sigma$, é a máquina que se obtém quando em M se substitui o estado de rejeição pelo de aceitação.

9. Seja Σ um alfabeto. Especifique uma máquina de Turing que, ao receber como *input* $M \in \mathcal{M}^\Sigma$, transforma M numa máquina de Turing que aceita precisamente as palavras que M rejeita e rejeita as que M aceita, ou seja, especifique uma máquina de Turing que calcule $f : \{0,1\}^* \to \{0,1\}^*$ tal que $f(M) \in \mathcal{M}^\Sigma$, para cada $M \in \mathcal{M}^\Sigma$, é a máquina que se obtém quando em M se troca o estado de aceitação pelo de rejeição, e vice-versa.

10. Seja Σ um alfabeto. Especifique uma máquina de Turing que, ao receber como *input* $M \in \mathcal{M}^\Sigma$, modifica M de modo a que todas as computações que abortam passem a ser computações que terminam no estado de rejeição, ou seja, especifique uma máquina de Turing que calcule $f : \{0,1\}^* \to \{0,1\}^*$ tal que $f(M) \in \mathcal{M}^\Sigma$, para cada $M \in \mathcal{M}^\Sigma$, é a máquina que resulta de acrescentar a M, como transições para o estado de rejeição, todas as transições em falta.

11. Seja Σ um alfabeto. Implemente a construção que é descrita no Exercício 6.2, isto é, especifique uma máquina de Turing que calcule a função $f : \{0,1,\$\}^* \to \{0,1\}^*$ tal que $f(M_1\$M_2) \in \mathcal{M}^\Sigma$, com $M_1, M_2 \in \mathcal{M}^\Sigma$, é a máquina composta $M_1; M_2$.

4

Teoria da Computabilidade

"... the Ghost in the Machine ..."
Gilbert Ryle, *The Concept of Mind*, 1949

4.1 Computabilidade e decidibilidade

Os problemas relevantes que estudamos são problemas de decisão e reconhecimento de linguagens, ou cálculo de funções.

Definição 4.1. LINGUAGEM RECONHECÍVEL/DECIDÍVEL, FUNÇÃO COMPUTÁVEL
Sejam Σ um alfabeto, $L \subseteq \Sigma^*$ uma linguagem e $f : \Sigma^* \to \Sigma^*$ uma função.
A linguagem L diz-se *reconhecível* se existe uma máquina de Turing M com alfabeto de entrada/saída Σ tal que $L_{\text{ac}}(M) = L$.
Denotamos por $\mathcal{R}^\Sigma$ o conjunto de todas as linguagens reconhecíveis sobre o alfabeto Σ, isto é, $\mathcal{R}^\Sigma = \{L_{\text{ac}}(M) : M \in \mathcal{M}^\Sigma\}$.
A linguagem L diz-se *decidível* se existe uma máquina de Turing M com alfabeto de entrada/saída Σ tal que $L_{\text{ac}}(M) = L$ e $L_{\text{rj}}(M) = \overline{L}$.
Denotamos por $\mathcal{D}^\Sigma$ o conjunto de todas as linguagens decidíveis sobre o alfabeto Σ, isto é, $\mathcal{D}^\Sigma = \{L_{\text{ac}}(M) : M \in \mathcal{M}^\Sigma \text{ e } \overline{L_{\text{ac}}(M)} = L_{\text{rj}}(M)\}$.
A função f diz-se *computável* se existe uma máquina de Turing M com alfabeto de entrada/saída Σ tal que $f = \phi_M$.
Denotamos por $\mathcal{C}^\Sigma$ o conjunto de todas as funções computáveis sobre o alfabeto Σ, isto é, $\mathcal{C}^\Sigma = \{\phi_M : M \in \mathcal{M}^\Sigma\}$. △

É também usual designar as linguagens reconhecíveis como *semi-decidíveis*. A razão é simples, exige-se que exista uma máquina que aceite precisamente as palavras de L, mas não se exige que as palavras de $\overline{L}$ sejam rejeitadas. Obviamente, $\mathcal{D}^\Sigma \subseteq \mathcal{R}^\Sigma$.

O seguinte resultado mostra que podemos concentrar-nos nos problemas de decisão ou reconhecimento de linguagens, já que o cálculo de funções se reduz a eles.

Proposição 4.2.
Seja Σ um alfabeto. Sejam $f : \Sigma^* \to \Sigma^*$ uma função e $G_f = \{x\$y : f(x) = y\}$, com $\$ \notin \Sigma$. Então:

1. *f é computável se e só se G_f é reconhecível;*

2. *se f é total, f é computável se e só se G_f é decidível.*

Dem.: (1) Suponha-se que f é computável e seja M_f uma máquina de Turing que calcula f. Considere-se a máquina de Turing T com duas fitas que, ao receber como *input* uma palavra w, começa por verificar se w é da forma $x\$y$, com $x, y \in \Sigma^*$. Se não for, rejeita w. Em caso caso contrário, copia x para a fita 2, e executa M_f com *input* x na fita 2. Se essa execução terminar aceitando, então compara o resultado dessa execução com y. Se forem iguais, aceita. Em caso contrário, rejeita. Se a execução de M_f terminar rejeitando ou abortando, então T termina rejeitando também. É fácil concluir que a linguagem reconhecida por T é G_f.

Suponha-se agora que a linguagem G_f é reconhecível. Seja R uma máquina de Turing que reconhece G_f. Considere-se a máquina de Turing não-determinista N com 3 fitas que, ao receber como *input* uma palavra $x \in \Sigma^*$, começa por escrever $x\$$ na fita 2. Depois, escolhe não-deterministicamente $y \in \Sigma^*$ para escrever na fita 2 e na fita 3, após o que posiciona a cabeça de leitura/escrita da fita 3 no início de y. Executa então R com *input* $x\$y$ na fita 2. A máquina N calcula f. Observe-se que, para cada $x \in \Sigma^*$, existe no máximo uma palavra do tipo $x\$y$ que é reconhecida por R. Logo, a árvore de computações de N para o *input* x tem no máximo uma computação de aceitação com o resultado da função (que está na fita 3).

(2) Demonstração semelhante à anterior. □

Já vimos vários exemplos de linguagens decidíveis e linguagens reconhecíveis, e os conhecimentos de programação e algoritmia do leitor serão certamente esclarecedores a este respeito, à luz do Postulado de Church, mas é útil percebermos como se relacionam e organizam estes conceitos.

4.2 Propriedades de fecho e redução computável

Proposição 4.3.
Seja Σ um alfabeto e $L_1, L_2 \subseteq \Sigma^*$ linguagens decidíveis. Então, $\emptyset$, Σ^*, $L_1 \cap L_2$, $L_1 \cup L_2$ e $L_1 \setminus L_2$ são linguagens decidíveis.

Dem.: Sejam D_1 e D_2 máquinas que decidem L_1 e L_2 respetivamente.

(1) $\emptyset$ é decidível. Considere-se a máquina de Turing que ao receber qualquer *input* termina imediatamente rejeitando. Dado que a linguagem reconhecida pela máquina é a linguagem vazia e a máquina termina sempre, então $\emptyset$ é decidível.

(2) Σ^* é decidível. Considere-se a máquina de Turing que ao receber qualquer *input* $w \in \Sigma^*$ termina imediatamente aceitando. Dado que a linguagem reconhecida pela máquina é Σ^* e a máquina termina sempre, então Σ^* é decidível.

(3) $L_1 \cup L_2$ é decidível. Considere-se a máquina de Turing D com duas fitas que se ilustra na Figura 4.1. A máquina D, ao receber $w \in \Sigma^*$ como *input*, copia w para a fita 2 e executa depois D_1 na fita 1 até D_1 terminar. Se D_1

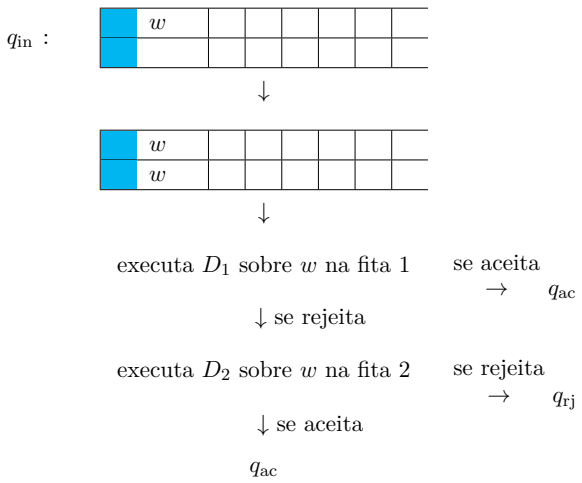


Figura 4.1: Máquina que decide $L_1 \cup L_2$ construída a partir de máquinas D_1 e D_2 que decidem L_1 e L_2, respectivamente.

termina aceitando, então D aceita. Se D_1 termina rejeitando, então D executa D_2 na fita 2 até D_2 terminar. Se D_2 termina aceitando, então D aceita. Em caso contrário, D rejeita. É fácil concluir que D decide $L_1 \cup L_2$.

(4) $L_1 \cap L_2$ é decidível. Deixa-se como exercício.

(5) $L_1 \backslash L_2$ é decidível. Considere-se a máquina de Turing D com duas fitas que se ilustra na Figura 4.2. A máquina D, ao receber $w \in \Sigma^*$ como *input*, copia w para a fita 2 e executa depois D_1 na fita 1 até D_1 terminar. Se D_1 termina rejeitando, então D rejeita. Se D_1 termina aceitando, D executa D_2 na fita 2 até D_2 terminar. Se D_2 termina aceitando, então D rejeita. Em caso contrário, D aceita. É fácil concluir que D decide $L_1 \backslash L_2$.

Como $\overline{L_2} = \Sigma^* \backslash L_2$, tem-se ainda que $\overline{L_2}$ é decidível. □

Proposição 4.4.
Sejam Σ um alfabeto e $L_1, L_2 \subseteq \Sigma^*$ linguagens reconhecíveis. Então, $\emptyset$, Σ^*, $L_1 \cap L_2$ e $L_1 \cup L_2$ são linguagens reconhecíveis. Se L_2 for decidível então $L_1 \setminus L_2$ é reconhecível.

Dem.: Sejam R_1 e R_2 máquinas que reconhecem L_1 e L_2 respetivamente.

(1) $\emptyset$ é reconhecível. A linguagem $\emptyset$ é reconhecível pois é decidível, e todas as linguagens decidíveis são reconhecíveis.

(2) Σ^* é reconhecível. A linguagem Σ^* é reconhecível, pois é decidível, e todas as linguagens decidíveis são reconhecíveis.

(3) $L_1 \cup L_2$ é reconhecível. Considere-se a máquina de Turing R com uma fita que se ilustra na Figura 4.3. A máquina R, ao receber $w \in \Sigma^*$ como *input*, escolhe não-deterministicamente entre executar R_1 ou R_2, e aceita se a máquina da escolha aceitar w. A máquina R reconhece $L_1 \cup L_2$.

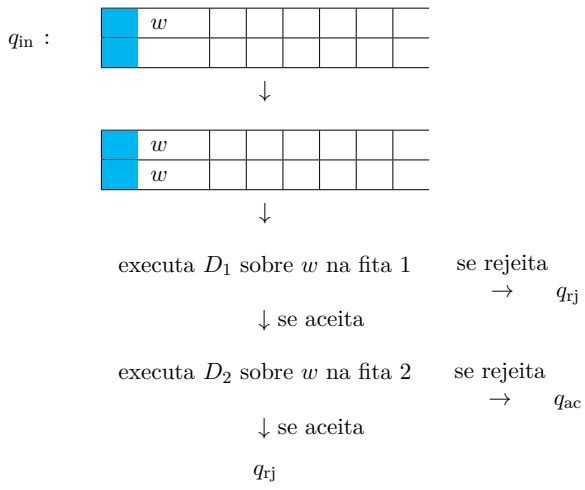


Figura 4.2: Máquina que decide $L_1 \backslash L_2$ construída a partir de máquinas D_1 e D_2 que decidem L_1 e L_2, respectivamente.

(4) $L_1 \cap L_2$ é reconhecível. Considere-se a máquina de Turing R com duas fitas que, ao receber $w \in \Sigma^*$ como *input*, copia w para a fita 2 e, de seguida, executa alternadamente um passo da execução de R_1 na fita 1 e um passo da execução de R_2 na fita 2. Se uma das execuções terminar, então R prossegue a execução da outra máquina no caso de ter terminado aceitando, e R rejeita em caso contrário. Se a execução da outra máquina terminar, então R aceita no caso de ter terminado aceitando, e rejeita em caso contrário. A máquina R reconhece $L_1 \cap L_2$. Note-se que R só termina se as execuções de R_1 e R_2 terminarem.

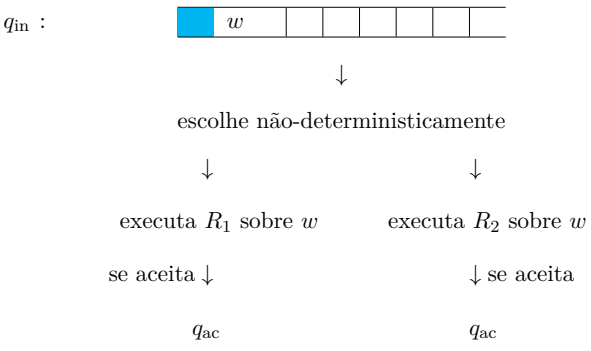


Figura 4.3: Máquina que reconhece $L_1 \cup L_2$ construída a partir de máquinas R_1 e R_2 que reconhecem L_1 e L_2, respectivamente.

(5) $L_1 \setminus L_2$ é reconhecível se L_2 é decidível. Suponha-se que L_2 é decidível e que D_2 é uma máquina que decide L_2. Considere-se a máquina de Turing R com duas fitas que se ilustra na Figura 4.4. A máquina R, ao receber $w \in \Sigma^*$ como

input, copia w para a fita 2, e executa a seguir D_2 na fita 2 até D_2 terminar. Se terminar aceitando, então R rejeita. Em caso contrário, R executa R_1 na fita 1. Se a execução de R_1 terminar, então R aceita se ela terminar aceitando, e rejeita em caso contrário. A máquina R reconhece $L_1 \backslash L_2$. Note-se que a execução de D_2 termina sempre, e que R só termina se a execução de R_1 terminar. □

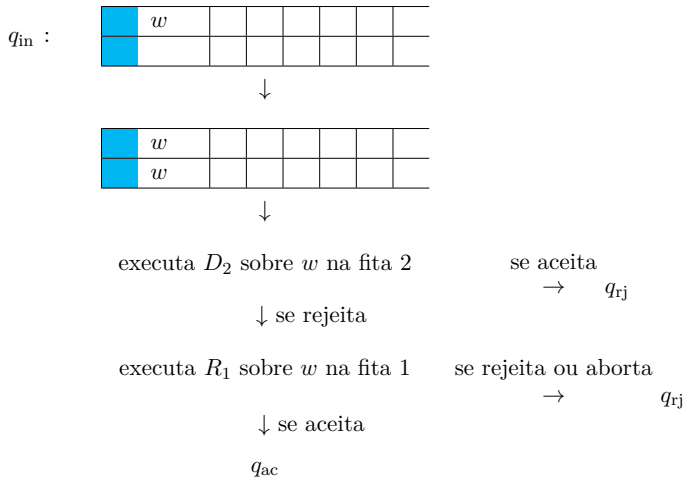


Figura 4.4: Máquina que reconhece $L_1 \backslash L_2$ construída a partir de R_1 que reconhece L_1 e de D_2 que decide L_2.

Na verdade, a diferença essencial entre decidibilidade e semi-decidibilidade está precisamente na complementação. Como veremos, em geral, o complementar de uma linguagem reconhecível pode não ser reconhecível.

Proposição 4.5.
Sejam Σ um alfabeto e L uma linguagem sobre Σ. Então, L é decidível se e só se L e $\overline{L}$ são ambas linguagens reconhecíveis.

Dem.: Sejam Σ um alfabeto e $L \subseteq \Sigma^*$.

Suponha-se que L é decidível. Então, L e $\overline{L}$ são decidíveis e, portanto, são reconhecíveis.

Suponha-se agora que L e $\overline{L}$ são reconhecíveis e sejam R_1 e R_2 máquinas de Turing que reconhecem L e $\overline{L}$, respectivamente. Considere-se a máquina de Turing R com duas fitas que, ao receber $w \in \Sigma^*$ como *input*, copia w para a fita 2 e, de seguida, executa alternadamente um passo da execução de R_1 na fita 1 e um passo da execução de R_2 na fita 2. A máquina T termina quando uma das execuções termina aceitando. Então, T aceita se essa execução for a de R_1, e rejeita se for a de R_2. É fácil concluir T decide L, pois, dada uma palavra $w \in \Sigma^*$, ou w pertence a L ou w pertence a $\overline{L}$, e, portanto, ou R_1 aceita w ou R_2 aceita w . □

A ideia usada acima, de construir novas máquinas recorrendo a máquinas obti-

das previamente, leva-nos à seguinte noção de *redução* entre linguagens, que será particularmente útil adiante.

Definição 4.6. REDUÇÃO COMPUTÁVEL
Sejam Σ_1, Σ_2 alfabetos, e L_1, L_2 linguagens sobre Σ_1 e Σ_2, respectivamente. Dizemos que há uma *redução computável* de L_1 a L_2, ou simplesmente que L_1 *se reduz* a L_2, o que denotamos por $L_1 \leq L_2$ se existe uma função total computável $f : \Sigma_1^* \to \Sigma_2^*$ tal que se tem, para cada $w \in \Sigma_1^*$,

$$w \in L_1 \text{ se e só se } f(w) \in L_2.$$

△

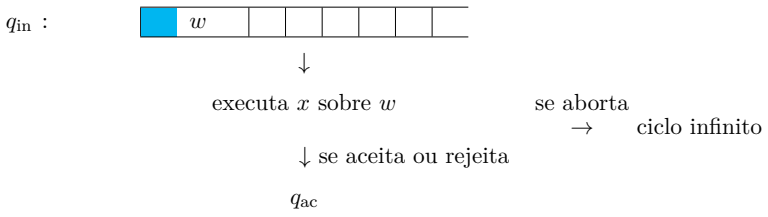
Exemplo 4.7.
Seja Σ um alfabeto. Considerem-se as seguintes linguagens sobre $\{0,1\}$:

$$L_1 = \{M \in \mathcal{M}^\Sigma : M \text{ é máquina classificadora}\}$$
$$L_2 = \{M \in \mathcal{M}^\Sigma : L_{\text{ac}}(M) = \Sigma^*\}.$$

Há uma redução computável de L_1 a L_2. Basta considerar $f : \{0,1\}^* \to \{0,1\}^*$ definida por

$$f(x) = \begin{cases} x' & \text{se } x \in \mathcal{M}^\Sigma \\ x & \text{caso contrário} \end{cases}$$

onde $x' \in \mathcal{M}^\Sigma$ é a seguinte máquina definida a partir de x:



isto é, a máquina que, dado o *input* w, executa x sobre w, e aceita se a computação é bem sucedida, e tem uma computação infinita em caso contrário.

A função f é total, e calculada por uma máquina que verifica se o *input* x é uma máquina, devolvendo x ou x' como *output* consoante o resultado. A máquina x' facilmente se obtém a partir de x substituindo o estado de rejeição pelo de aceitação, e acrescentando as transições em falta de modo a gerarem um ciclo infinito.

Se $x \notin \mathcal{M}^\Sigma$ então $x \notin L_1$ e $f(x) = x \notin L_2$. Se $x \in \mathcal{M}^\Sigma$, então $x \in L_1$ se e só se x é máquina classificadora se e só se x aceita ou rejeita todos os *inputs* se e só se x' aceita todos os *inputs* se e só se $x' = f(x) \in L_2$.

Conclui-se então que $L_1 \leq L_2$. △

Proposição 4.8.
Sejam Σ_1, Σ_2 alfabetos e L_1, L_2 linguagens sobre Σ_1 e Σ_2, respectivamente. Se $L_1 \leq L_2$ e L_2 é decidível (resp. reconhecível) então L_1 é decidível (resp. reconhecível).

Dem.: Assuma-se que $L_1 \leq L_2$ e que D_2 é uma máquina de Turing que decide L_2. Existe então uma função total computável $f : \Sigma_1^* \to \Sigma_2^*$ tal que, para cada $w \in \Sigma_1^*$, $w \in L_1$ se e só se $f(w) \in L_2$. Seja F uma máquina de Turing que calcula f. Considere-se a máquina D_1 que se ilustra na Figura 4.5. A máquina D_1, ao receber w como *input*, executa F sobre w para obter $f(w)$, e

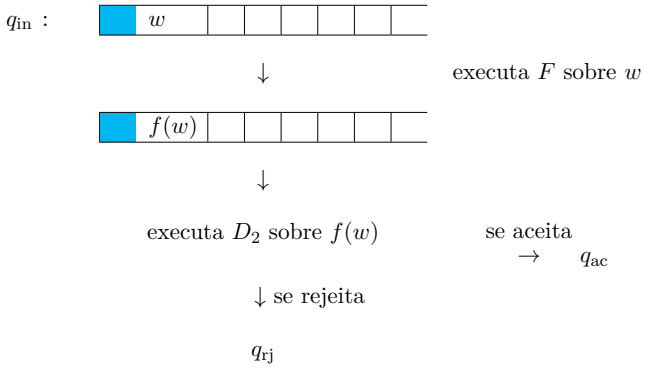


Figura 4.5: Máquina D_1 que decide L_1 construída a partir de F que calcula f e de D_2 que decide L_2.

de seguida executa D_2 sobre $f(w)$. É fácil concluir que D_1 é um classificador e que D_1 aceita(resp. rejeita) w se e só se D_2 aceita (resp. rejeita) $f(w)$. Como, D_2 decide L_2 e $f(w) \in L_2$ se e só se $w \in L_1$, conclui-se que D_1 decide L_1.

A demonstração é semelhante no caso em que se supõe L_2 reconhecível (recorrendo-se agora a uma máquina R_2 que reconheça L_2), e deixa-se como exercício. □

4.3 Indecidibilidade

O argumento mais simples que podemos usar para justificar a existência de problemas não computáveis, envolve cardinalidades. Os resultados seguintes são demonstrados para cada alfabeto Σ, apesar de que, à luz do que já vimos, e nomeadamente dos resultados dos Exercício 2.2 do Capítulo 1 e dos Exercícios 6.5 e 6.6 do Capítulo 3, poderíamos considerar essencialmente os problemas postos sobre o alfabeto $\{0,1\}$.

Proposição 4.9.
Seja Σ um alfabeto e seja $\mathcal{M}^\Sigma \subseteq \{0,1\}^*$ o conjunto de todas as representações canónicas de máquinas de Turing sobre o alfabeto Σ. Tem-se que $\#\mathcal{M}^\Sigma = \#\mathbb{N}$.

Dem.: Considere-se a função $f : \mathcal{M}^\Sigma \to \{0,1\}^*$ dada por $f(M) = M$. É imediato concluir que f é injectiva. Como $\{0,1\}^*$ é numerável, existe uma função $g : \{0,1\}^* \to \mathbb{N}$ injetiva. Logo, $g \circ f : \mathcal{M}^\Sigma \to \mathbb{N}$ é uma função injectiva. Considere-se agora a função $h : \mathbb{N} \to \mathcal{M}^\Sigma$ tal que $h(n) = 1^{n+1}01^k0$ (a representação canónica de uma máquina com $n+1$ estados, k símbolos de entrada/saída,

e sem transições). É também imediato que esta função é injectiva. Conclui-se assim que $\mathcal{M}^\Sigma$ é numerável. □

Observe-se que este resultado também se verifica se, em vez do alfabeto $\{0,1\}$, for escolhido qualquer outro alfabeto para a representação das máquinas de Turing porque, nomeadamente, o conjunto das palavras sobre um alfabeto é sempre numerável. Deixa-se como exercício essa demonstração.

Corolário 4.10.
Seja Σ um alfabeto. Então, $\#\mathcal{D}^\Sigma = \#\mathcal{R}^\Sigma = \#\mathcal{C}^\Sigma = \#\mathbb{N}$.

Dem.: É fácil mostrar que $\mathcal{D}^\Sigma$ é infinito (o que se deixa como exercício). Para mostrar que $\mathcal{D}^\Sigma$ é numerável defina-se $f : \mathcal{D}^\Sigma \to \mathbb{N}$ por $f(L) = h(M_L)$ onde $h : \mathcal{M}^\Sigma \to \mathbb{N}$ é injectiva (vide Proposição 4.9) e M_L é uma máquina de Turing que decide L, isto é, $L_{\text{ac}}(M_L) = L$ e $L_{\text{rj}}(M_L) = \overline{L}$, para cada $L \in \mathcal{D}^\Sigma$. A função f fica bem definida para cada escolha da máquina M_L, e é injectiva porque $f(L_1) = f(L_2)$ implica $h(M_{L_1}) = h(M_{L_2})$ e, portanto, $M_{L_1} = M_{L_2}$. Consequentemente, tem-se $L_1 = L_{\text{ac}}(M_{L_1}) = L_{\text{ac}}(M_{L_2}) = L_2$, pelo que f é injetiva.

A demonstração para $\mathcal{R}^\Sigma$ e $\mathcal{C}^\Sigma$ é análoga. □

No entanto, temos que $\mathcal{L}^\Sigma$ e $\mathcal{F}^\Sigma$ não são numeráveis, como já sabemos das Proposições 1.4 e 1.5, pelo que podemos concluir que terão de existir linguagens não-decidíveis, mesmo não-reconhecíveis, bem como funções não-computáveis.

Corolário 4.11.
Seja Σ um alfabeto. Então, $\#\mathcal{D}^\Sigma < \#\mathcal{L}^\Sigma$, $\#\mathcal{R}^\Sigma < \#\mathcal{L}^\Sigma$ e $\#\mathcal{C}^\Sigma < \#\mathcal{F}^\Sigma$.

A pergunta que podemos colocar-nos é: que linguagens e funções são essas que estão fora do nosso alcance. Haverá problemas interessantes que não são computáveis? Na verdade há muitos. Dado o alfabeto Σ, considerem-se as seguintes linguagens:

- $\mathcal{L}^\Sigma_{\text{ac}} = \{M\$w : M \in \mathcal{M}^\Sigma, w \in L_{\text{ac}}(M)\}$ (o problema da *aceitação*);
- $\mathcal{L}^\Sigma_{\text{rj}} = \{M\$w : M \in \mathcal{M}^\Sigma, w \in L_{\text{rj}}(M)\}$ (o problema da *rejeição*);
- $\mathcal{L}^\Sigma_{\text{su}} = \mathcal{L}^\Sigma_{\text{ac}} \cup \mathcal{L}^\Sigma_{\text{rj}}$ (o problema da *computação bem sucedida*);
- $\mathcal{L}^\Sigma_{\text{ab}} = \{M\$w : M \in \mathcal{M}^\Sigma$ e a computação de M sobre w aborta$\}$ (o problema do *abortamento*);
- $\mathcal{L}^\Sigma_{\text{te}} = \mathcal{L}^\Sigma_{\text{su}} \cup \mathcal{L}^\Sigma_{\text{ab}}$ (o problema da *terminação*).

As diferentes linguagens refletem propriedades das computações das máquinas para o *input* recebido. Cada palavra é constituída pela representação canónica de uma máquina de Turing e uma palavra de *input* $w \in \Sigma^*$, separadas por \$ (o alfabeto destas linguagens é assim constituído pelos símbolos em Σ, pelos símbolos do alfabeto usado para a representação das máquinas, e por um símbolo \$ distinto de todos os anteriores).

Na literatura, a linguagem $\mathcal{L}^\Sigma_{\text{ac}}$ é também designada por A_{TM}, e a linguagem $\mathcal{L}^\Sigma_{\text{su}}$ é também designada por $HALT_{\text{TM}}$.

Proposição 4.12.
Seja Σ um alfabeto. As linguagens $\mathcal{L}_{ac}^{\Sigma}$, $\mathcal{L}_{rj}^{\Sigma}$, $\mathcal{L}_{su}^{\Sigma}$, $\mathcal{L}_{ab}^{\Sigma}$ e $\mathcal{L}_{te}^{\Sigma}$ são reconhecíveis mas não são decidíveis.

Dem.: As demonstrações são semelhantes. Centramo-nos no problema da terminação.

Para mostrar que $\mathcal{L}_{\mathrm{te}}^{\Sigma}$ é reconhecível basta considerar a máquina que, sobre cada *input* $M\$w$, começa por calcular $\mathrm{rep}(w)$ e, depois, simula M sobre $\mathrm{rep}(w)$ como na máquina universal, aceitando assim que a computação de M sobre $\mathrm{rep}(w)$ termine (aceitando, rejeitando ou abortando).

Assuma-se agora, por absurdo, que $\mathcal{L}_{\mathrm{te}}^{\Sigma}$ fosse decidível. Nesse caso, existiria uma máquina de Turing D que decidiria $\mathcal{L}_{\mathrm{te}}^{\Sigma}$. Construa-se então a seguinte máquina T. Para cada *input* M, a máquina T começa por simular D sobre $M\$M$. Se D aceitar, então T deve entrar num ciclo infinito. Se D rejeitar, então T aceita.

É simples verificar que M é aceite por T se e só se a computação da máquina M dado o *input* M não termina. Então, pode concluir-se, para a própria máquina T, que T é aceite por T se e só se a computação da máquina T dado o *input* T não termina. Ora isto é uma contradição. Conclui-se, portanto, que $\mathcal{L}_{\mathrm{te}}$ não é decidível. □

Em particular, a indecidibilidade do problema da terminação (o famoso *halting problem*) é um dos resultados mais emblemáticos de Turing.

Corolário 4.13.
Seja Σ um alfabeto. As linguagens $\overline{\mathcal{L}_{ac}^{\Sigma}}$, $\overline{\mathcal{L}_{rj}^{\Sigma}}$, $\overline{\mathcal{L}_{su}^{\Sigma}}$, $\overline{\mathcal{L}_{ab}^{\Sigma}}$ e $\overline{\mathcal{L}_{te}^{\Sigma}}$ não são reconhecíveis.

4.4 Teorema de Rice

O seguinte teorema é uma ferramenta bastante genérica para demonstrar a indecidibilidade de linguagens constituídas por máquinas de Turing cujas linguagens satisfaçam alguma propriedade não-trivial.

Teorema 4.14.
Sejam Σ um alfabeto e $L \subseteq \mathcal{M}^{\Sigma}$ tal que se $M_1 \in L$ e $M_1 \equiv M_2$ então $M_2 \in L$. Se $\emptyset \neq L \neq \mathcal{M}^{\Sigma}$ então L é indecidível.

Dem.: Seja $\emptyset \neq L \neq \mathcal{M}^{\Sigma}$. Por absurdo, admitamos que L é decidível. Isto implica que também $\overline{L}$ é decidível. Começamos por considerar uma máquina $M_{\emptyset} \in \mathcal{M}^{\Sigma}$ que aborta para todos os *inputs*. Obviamente, tem-se $L_{\mathrm{ac}}(M_{\emptyset}) = L_{\mathrm{rj}}(M_{\emptyset}) = \emptyset$ e $\phi_{M_{\emptyset}}$ está sempre indefinida.

(i) Suponha-se que $M_{\emptyset} \notin L$. Vamos mostrar que $\mathcal{L}_{\mathrm{ac}} \leq L$. Para tal, considere-se $M_1 \in L \neq \emptyset$, e uma função total f tal que

$$f(x) = \begin{cases} M_w & \text{se } x = M\$w \\ M_{\emptyset} & \text{caso contrário} \end{cases}$$

onde $M \in \mathcal{M}^{\Sigma}$, $w \in \Sigma^*$, \$ é um símbolo separador, e $M_w \in \mathcal{M}^{\Sigma}$ é a máquina seguinte: ao receber $u \in \Sigma^*$ como *input*, M_w começa por executar M sobre w;

se M rejeita ou aborta, então M_w aborta e, se M aceita, então M_w executa M_1 sobre u. É fácil concluir que, se M aceita w, então $L_{\text{ac}}(M_w) = L_{\text{ac}}(M_1)$, $L_{\text{rj}}(M_w) = L_{\text{rj}}(M_1)$ e $\phi_{M_w} = \phi_{M_1}$, e, portanto, $M_w \equiv M_1$. Por outro lado, se M não aceita w, então $L_{\text{ac}}(M_w) = \emptyset = L_{\text{ac}}(M_\emptyset)$, $L_{\text{rj}}(M_w) = \emptyset = L_{\text{rj}}(M_\emptyset)$ e $\phi_{M_w} = \phi_{M_\emptyset}$ (pois, ϕ_{M_w} está sempre indefinida), e, portanto, $M_w \equiv M_\emptyset$.

Vejamos que $x \in \mathcal{L}_{\text{ac}}$ se e só se $f(x) \in L$. Se $x \in \mathcal{L}_{\text{ac}}$ então $x = M\$w$ com $M \in \mathcal{M}^\Sigma$ que aceita $w \in \Sigma^*$, e, portanto, $M_w \equiv M_1$. Como $M_1 \in L$, as hipóteses do teorema garantem que $M_w = f(x) \in L$. Reciprocamente, se $f(x) \in L$, então $f(x) \neq M_\emptyset \notin L$ e, portanto, $f(x) = M_w \in L$ e $x = M\$w$ com $M \in \mathcal{M}^\Sigma$ e $w \in \Sigma^*$. Como $M_w \in L$, de novo as hipóteses do teorema garantem que $M_w \not\equiv M_\emptyset \notin L$, o que significa que M aceita w, ou seja, $x = M\$w \in \mathcal{L}_{\text{ac}}$. Deixa-se como exercício concluir que a função f é computável.

(ii) Suponha-se que $M_\emptyset \in L$. Vamos mostrar que $\mathcal{L}_{\text{ac}} \leq \overline{L}$. Para tal, considere-se $M_2 \in \mathcal{M}^\Sigma$ tal que $M_2 \notin L \neq \mathcal{M}^\Sigma$ e uma função total f semelhante à definida em (i), em que M_w usa M_2 em vez de M_1. Raciocinando como em (i) conclui-se que se M aceita w, então $M_w \equiv M_2$ e, se M não aceita w, então $M_w \equiv M_\emptyset$.

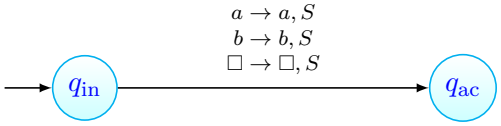
Vejamos que $x \in \mathcal{L}_{\text{ac}}$ se e só se $f(x) \in \overline{L}$. Se $x \in \mathcal{L}_{\text{ac}}$ então $x = M\$w$ com $M \in \mathcal{M}^\Sigma$ que aceita $w \in \Sigma^*$, e, portanto, $M_w \equiv M_2$. Como $M_2 \notin L$, as hipóteses do teorema garantem que $M_w = f(x) \notin L$, isto é, $f(x) \in \overline{L}$. Reciprocamente, se $f(x) \in \overline{L}$, então $f(x) \neq M_\emptyset \in L$ e, portanto, $f(x) = M_w \in \overline{L}$ e $x = M\$w$ com $M \in \mathcal{M}^\Sigma$ e $w \in \Sigma^*$. Assim, $M_w \notin L$, e as hipóteses do teorema garantem que $M_w \not\equiv M_\emptyset \in L$, o que significa que M aceita w, ou seja, $x = M\$w \in \mathcal{L}_{\text{ac}}$. Deixa-se como exercício concluir que a função f é computável.

(iii) Das alíneas anteriores decorre que ou $\mathcal{L}_{\text{ac}} \leq L$ ou $\mathcal{L}_{\text{ac}} \leq \overline{L}$. Em ambos os casos, poder-se-ia concluir que $\mathcal{L}_{\text{ac}}$ é decidível, o que é uma contradição. A linguagem L é, pois, indecidível. □

Este teorema tem inúmeras aplicações na demonstração de resultados de indecidibilidade.

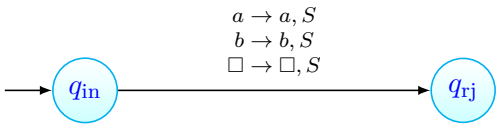
Exemplo 4.15.
Considere-se a linguagem $L = \{M \in \mathcal{M}^{\{a,b\}} : L_{\text{ac}}(M) \text{ é um conjunto infinito}\}$. Pode recorrer-se ao Teorema de Rice para demonstrar que esta linguagem é indecidível. Como $L \subseteq \mathcal{M}^{\{a,b\}}$, então o teorema garante que L é indecidível desde que satisfaça três requisitos: (1) $L \neq \emptyset$, (2) $L \neq \mathcal{M}^{\{a,b\}}$, e (3) se $M \in L$ e $M \equiv M'$, então $M' \in L$. Mostra-se de seguida que todos se verificam.

(1) Considere-se a máquina A com alfabeto de entrada/saída $\{a, b\}$ representada graficamente por:



A máquina A aceita todos os *inputs*, pelo que $L_{\text{ac}}(A) = \{a, b\}^*$. Como $\{a, b\}^*$ é um conjunto infinito, $A \in L$ e, portanto, $L \neq \emptyset$.

(2) Considere-se a máquina B com alfabeto de entrada/saída $\{a, b\}$ representada graficamente por:



A máquina B rejeita todos os *inputs*, pelo que $L_{\text{ac}}(B) = \emptyset$. Como $\emptyset$ não é um conjunto infinito, $B \notin L$ e, portanto, $L \neq \mathcal{M}^{\{a,b\}}$.

(3) Suponha-se que $M \in L$ e $M \equiv M'$. Dado que $M \in L$, M tem alfabeto de entrada/saída $\{a, b\}$ e $L_{\text{ac}}(M)$ é um conjunto infinito. Como $M \equiv M'$, o alfabeto de entrada/saída de M' é também $\{a, b\}$, e $L_{\text{ac}}(M) = L_{\text{ac}}(M')$, pelo que $L_{\text{ac}}(M')$ é também infinito. Conclui-se então que $M' \in L$.

Pelo Teorema de Rice concluímos que L_2 é indecidível. △

Note-se, no entanto, que no caso de uma linguagem ser indecidível nem sempre essa propriedade pode ser demonstrada com recurso ao Teorema de Rice. Um exemplo simples é a linguagem $\mathcal{L}^{\Sigma}_{\text{te}}$: esta linguagem é indecidível (Proposição 4.12), mas não se pode aplicar o Teorema de Rice a $\mathcal{L}^{\Sigma}_{\text{te}}$, uma vez que $\mathcal{L}^{\Sigma}_{\text{te}} \not\subseteq \mathcal{M}^{\Sigma}$.

4.5 Teorema da recursão

O próximo resultado é uma versão simplificada do resultado original, devido a Kleene, que constitui o fundamento daquilo a que hoje chamamos de *programação recursiva*, e que explora o fenómeno da *auto-referência* em computação. Como vimos anteriormente, existe uma máquina de Turing universal que simula o comportamento de qualquer outra máquina. Mais surpreendentemente, no entanto, podemos mostrar que qualquer máquina simula o comportamento de alguma máquina que receba como *input*.

Teorema 4.16.
Qualquer que seja a máquina de Turing $M \in \mathcal{M}$ existe $F \in \mathcal{M}$ tal que, qualquer que seja $x \in \{0,1\}^*$, a computação de F sobre x simula a computação de M sobre $F\$x$.

Dem.: Considere-se a máquina de Turing M. Dado $u \in \{0,1\}^*$, seja M_u a máquina que quando recebe como *input* x simula M sobre $u\$x$.

Considere-se a máquina W que, quando recebe como *input* uma máquina T calcula $t = \phi_T(T)$ e devolve como *output* M_t. Claramente, $\phi_W(T) = M_{\phi_T(T)}$.

Tome-se $F = M_w$ onde $w = \phi_W(W)$.

A computação de F sobre x, isto é, a computação de M_w sobre x, simula a computação de M sobre $w\$x$. Mas $w = \phi_W(W) = M_{\phi_W(W)} = M_w = F$, como queríamos demonstrar. □

O resultado tem imensas aplicações práticas, desde logo em demonstrações de indecidibilidade.

Exemplo 4.17.
Considere-se $\mathcal{L}_{\text{ac}}$ e use-se o Teorema da Recursão para obter uma demonstração alternativa da sua indecidibilidade. Suponha-se, por absurdo, que $\mathcal{L}_{\text{ac}}$ fosse decidível. Nesse caso, também se teria $\overline{\mathcal{L}_{\text{ac}}}$ decidível e portanto existiria uma máquina $\overline{D_{\text{ac}}}$ que decidiria $\overline{\mathcal{L}_{\text{ac}}}$. Pelo Teorema da Recursão, ter-se-ia também uma máquina F tal que a computação de $\overline{D_{\text{ac}}}$ sobre $F\$x$ simula a computação de F sobre x, para cada $x \in \{0,1\}^*$. Nesse caso, facilmente, x é aceite por F se e só se $F\$x$ é aceite por $\overline{D_{\text{ac}}}$ se e só se $F\$x \in \overline{\mathcal{L}_{\text{ac}}}$ se e só se $F\$x \notin \mathcal{L}_{\text{ac}}$ se e só se x não é aceite por F, o que é obviamente uma contradição. △

Uma das aplicações mais interessantes do Teorema da Recursão é a demonstração de existência de certos tipos, muito simples e inócuos, de *vírus auto-replicáveis*: $V \in \mathcal{M}$ diz-se *auto-replicável* se $\phi_V(w) = V$ qualquer que seja $w \in \{0,1\}^*$. Uma máquina auto-replicável é também conhecida por *Quine*.

Proposição 4.18.
Existe uma máquina de Turing auto-replicável. As máquinas de Turing auto-replicáveis formam uma linguagem indecidível.

Dem.: Tome-se uma máquina de Turing M tal que $\phi_M(u\$x) = u$ quaisquer que sejam $u, x \in \{0,1\}^*$. Pelo Teorema da Recursão, existe $F \in \mathcal{M}$ tal que a computação F sobre x simula a computação de M sobre $F\$x$, para qualquer $x \in \{0,1\}^*$. Então, $\phi_F(x) = \phi_M(F\$x) = F$.

Suponha-se, por absurdo, que existe uma máquina de Turing D_{vir} que decide a linguagem $\mathcal{L}_{\text{vir}} = \{M \in \mathcal{M} : M \text{ é auto-replicável}\}$. Considere-se a máquina T que quando recebe como *input* $M\$x$ simula D_{vir} sobre M, devolvendo M como *output* caso D_{vir} rejeite M, e entrando em ciclo infinito caso D_{vir} aceite M.

Facilmente, $\phi_T(M\$x) = M$ se M não é auto-replicável, e $\phi_T(M\$x)$ indefinido se M é auto-replicável.

Pelo Teorema da Recursão, existe uma máquina F tal que a computação de T sobre $F\$x$ simula a computação de F sobre x, para qualquer x. Então, temos que F é auto-replicável se e só se $\phi_F(w) = F$ se e só se $\phi_T(F\$w) = F$ se e só se F não é auto-replicável, o que é uma contradição. □

Exercícios

1 Computabilidade e decidibilidade

1. Seja L uma linguagem sobre um alfabeto Σ. Mostre que:

   (a) L é decidível se e só se é computável a sua função característica, isto é, a função $\chi : \Sigma^* \to \{0,1\}$ tal que $\chi(w) = 1$ se $w \in L$, e $\chi(w) = 0$ em caso contrário;

   (b) L é reconhecível se e só se é computável a sua função semi-característica, isto é, a função $\chi : \Sigma^* \to \{0,1\}$ tal que $\chi(w) = 1$ se $w \in L$, e $\chi_L(w)$ não está definida em caso contrário.

2. Seja Σ um alfabeto. Demonstre que se uma função $f : \Sigma^* \to \Sigma^*$ é computável então é reconhecível a linguagem $\text{dom}(f) = \{w \in \Sigma^* : f(w) \text{ definido}\}$.

3. Seja Σ um alfabeto. Demonstre que se $f, g \in \mathcal{C}^{\Sigma}$ então $(g \circ f) \in \mathcal{C}^{\Sigma}$.

2 Propriedades de fecho e redução computável

1. Sejam L_1, L_2, L_3 linguagens sobre um alfabeto Σ. Mostre que se L_1, L_2 e L_3 são linguagens decidíveis então são também decidíveis as seguintes linguagens:

   (a) $L_1 \cap L_2$;
   (b) $L_1 \cup L_2$;
   (c) $\overline{L_1}$;
   (d) $L_1 \backslash L_2$;
   (e) $L_1 \times L_2$;
   (f) $L_1.L_2$;
   (g) L_1^*;
   (h) $L = \{w \in \Sigma^* : w \in L_1 \text{ ou } w \in L_2, \text{ mas } w \notin L_1 \cap L_2\}$;
   (i) $L = \{w \in \Sigma^* : w \in L_1 \cap L_3 \text{ e } w \notin L_2\}$.

2. Sejam L_1, L_2, L_3 linguagens sobre um alfabeto Σ. Assumindo que L_1 e L_2 são reconhecíveis e L_3 é decidível, mostre que:

   (a) $L_1 \cap L_2$ é reconhecível;
   (b) $L_3 \cup L_2$ é reconhecível;
   (c) $L_1 \cup L_2$ é reconhecível;
   (d) $L_1 \backslash L_3$ é reconhecível;
   (e) $L_1 \times L_2$ é reconhecível;
   (f) $L_1.L_2$ é reconhecível;
   (g) L_1^* é reconhecível;
   (h) $(\overline{L_3} \cap L_2) \cup L_1$ é reconhecível.

3. Seja L uma linguagem sobre um alfabeto Σ. Mostre que se L e $\overline{L}$ são reconhecíveis então L é decidível.

4. Sejam L_1, L_2 e L_3 linguagens sobre um alfabeto Σ. Mostre que:

   (a) $L_1 \leq L_2$ se e só se $\overline{L_1} \leq \overline{L_2}$;
   (b) se $L_1 \leq L_2$ e $L_2 \leq L_3$ então $L_1 \leq L_3$.

5. Sejam L_1 e L_2 linguagens sobre um alfabeto Σ tais que $L_1 \leq L_2$. Mostre que:

   (a) se L_2 é decidível então L_1 é decidível;
   (b) se L_1 não é decidível então L_2 não é decidível;
   (c) se L_2 é reconhecível então L_1 é reconhecível;

(d) se L_1 não é reconhecível então L_2 não é reconhecível.

6. Seja Σ um alfabeto. Mostre que se $L \subseteq \Sigma^*$ é reconhecível e $L \leq \overline{L}$ então L é decidível.

7. Mostre que $L \subseteq \{0,1\}^*$ é decidível se e só se $L \leq \{0^n 1^n : \text{ com } n \in \mathbb{N}_0\}$.

8. Mostre que $L \subseteq \{0,1\}^*$ é reconhecível se e só se $L \leq \mathcal{L}_{\text{ac}}^{\{0,1\}}$.

9. Seja Σ um alfabeto. Mostre que $\mathcal{L}_{\text{ac}}^{\Sigma} \not\leq \overline{\mathcal{L}_{\text{ac}}^{\Sigma}}$, onde $\overline{\mathcal{L}_{\text{ac}}^{\Sigma}}$ é a linguagem complementar de $\mathcal{L}_{\text{ac}}^{\Sigma}$.

10. Recorde as demonstrações das Proposições 4.4 e 4.5.

    (a) A máquina R construída na demonstração de (3) da Proposição 4.4 poderia começar por executar a máquina R_1, e depois, se necessário, a máquina R_2? E poderia recorrer à execução alternada de um passo de R_1 e de R_2?

    (b) A máquina R construída na demonstração de (4) da Proposição 4.4 poderia começar por executar a máquina R_1, e depois a máquina R_2? E poderia recorrer à escolha não-determinística entre a execução da máquina R_1 ou a execução da máquina R_2?

    (c) A máquina R construída na demonstração da Proposição 4.5 poderia recorrer à escolha não-determinística entre a execução da máquina R_1 ou a execução da máquina R_2?

3 Incomputabilidade

1. Seja $\mathcal{F}^t_{\{1\}^*\to\{1\}^*}$ o conjunto das funções totais de $\{1\}^*$ para $\{1\}^*$.

    (a) Demonstre que $\#\mathbb{N} < \#\mathcal{F}^t_{\{1\}^*\to\{1\}^*}$.

    (b) Demonstre que existem funções totais de $\{1\}^*$ para $\{1\}^*$ que não são calculadas por nenhuma máquina de Turing.

2. Repita o Exercício 3.1 para os seguintes conjuntos de funções:

    (a) $\mathcal{F}^t_{\{0,1\}^*\to\{0,1\}}$ (funções totais de $\{0,1\}^*$ para $\{0,1\}$);

    (b) $\mathcal{F}^t_{\{0,1\}^*\to\{0,1\}^*}$ (funções totais de $\{0,1\}^*$ para $\{0,1\}^*$);

    (c) $\mathcal{F}^t_{\{a,b,c\}^*\to\{a,b,c\}^*}$ (funções totais de $\{a,b,c\}^*$ para $\{a,b,c\}^*$);

    (d) $\mathcal{F}_{\{1\}^*\to\{1\}^*}$ (funções de $\{1\}^*$ para $\{1^*\}$);

    (e) $\mathcal{F}_{\{0,1\}^*\to\{0,1\}}$ (funções de $\{0,1\}^*$ para $\{0,1\}$);

    (f) $\mathcal{F}_{\{0,1\}^*\to\{1\}}$ (funções de $\{0,1\}^*$ para $\{1\}$).

3. Seja Σ um alfabeto.

    (a) Mostre que $\#\mathbb{N} < \#\mathcal{L}^{\Sigma}$.

    (b) Mostre que existem linguagens em $\mathcal{L}^{\Sigma}$ que não são reconhecíveis.

4 Indecidibilidade

1. Sejam L_1 e L_2 linguagens sobre um alfabeto Σ tais que $L_1 \leq L_2$. Mostre que se L_1 é indecidível então L_2 também é indecidível.

2. Seja Σ um alfabeto. Para cada $L \in \{\mathcal{L}_{\text{ac}}^{\Sigma}, \mathcal{L}_{\text{te}}^{\Sigma}, \mathcal{L}_{\text{su}}^{\Sigma}\}$ mostre que:

   (a) L é reconhecível;

   (b) L não é decidível;

   (c) a linguagem complementar de L não é reconhecível.

3. Seja Σ um alfabeto. Considere as linguagens $\mathcal{L}_{\text{ac}}^{\Sigma}$ e $\mathcal{L}_{\text{su}}^{\Sigma}$.

   (a) Mostre que $\mathcal{L}_{\text{ac}}^{\Sigma} \leq \mathcal{L}_{\text{su}}^{\Sigma}$.
   Sugestão: considere uma função f dada por

   $$f(\alpha) = \begin{cases} M'\$w & \text{se } \alpha = M\$w \\ \epsilon & \text{caso contrário} \end{cases}$$

   onde M' é uma máquina que, dado o *input* x, simula o comportamento de M com *input* x, e chega a uma configuração de aceitação se M chega a uma configuração de aceitação, e tem uma evolução infinita em caso contrário.

   (b) Use a alínea anterior para demonstrar que $\mathcal{L}_{\text{su}}^{\Sigma}$ não é decidível.

4. Seja Σ um alfabeto. Considere as linguagens

   $$L_1 = \{M \in \mathcal{M}^{\Sigma} : M \text{ é máquina classificadora}\}$$
   $$L_2 = \{M \in \mathcal{M}^{\Sigma} : L_{\text{ac}}(M) = \Sigma^*\}.$$

   (a) Mostre que L_2 não é decidível. Sugestão: conclua que $\mathcal{L}_{\text{ac}}^{\Sigma} \leq L_2$.

   (b) Mostre que $L_2 \leq L_1$ e conclua que L_1 não é decidível.

5. Seja Σ um alfabeto. Mostre que o *problema da palavra vazia* relativo a máquinas de Turing não é decidível, ou seja, mostre não é decidível a linguagem

   $$\mathcal{L}_{\epsilon}^{\Sigma} = \{M \in \mathcal{M}^{\Sigma} : \ \epsilon \in L_{\text{ac}}(M)\}.$$

6. Seja Σ um alfabeto. Mostre que o *problema da linguagem vazia* relativo a máquinas de Turing não é decidível, ou seja, mostre não é decidível a linguagem

   $$V_{\text{TM}}^{\Sigma} = \{M \in \mathcal{M}^{\Sigma} : L_{\text{ac}}(M) = \emptyset\}.$$

7. Sejam Σ um alfabeto e p uma palavra sobre Σ. Mostre que não é decidível a linguagem

   $$DOM_{\text{TM}}^{\Sigma,p} = \{M \in \mathcal{M}^{\Sigma} : M \text{ aceita } p\}.$$

   A não-decidibilidade desta linguagem é também conhecida como a não-decidibilidade do *"input problem"*.

8. Sejam Σ um alfabeto e p uma palavra sobre Σ. Mostre que não é decidível a linguagem

$$CDOM_{\mathrm{TM}}^{\Sigma,p} = \{M \in \mathcal{M}^{\Sigma} : M \text{ escreve } p\}.$$

A expressão "M escreve p" significa aqui que existe pelo menos uma configuração inicial a partir da qual M chega a uma configuração de aceitação na qual o conteúdo da fita é p. A não decidibilidade desta linguagem é também conhecida como a não decidibilidade do *"printing problem"*.

9. Seja Σ um alfabeto. Mostre que não é decidível a linguagem

$$FIN_{\mathrm{TM}}^{\Sigma} = \{M \in \mathcal{M}^{\Sigma} : L_{\mathrm{ac}}(M) \text{ é linguagem finita}\}.$$

10. Seja Σ um alfabeto. Mostre que não é decidível a linguagem

$$PAR_{\mathrm{TM}}^{\Sigma} = \{M \in \mathcal{M}^{\Sigma} : \text{as palavras aceites por } M \text{ têm comprimento par}\}.$$

11. Seja Σ um alfabeto. Mostre que não é decidível a linguagem

$$IMP_{\mathrm{TM}}^{\Sigma} = \{M \in \mathcal{M}^{\Sigma} : \text{as palavras aceites por } M \text{ têm comprimento ímpar}\}.$$

12. Seja Σ um alfabeto. Considere a linguagem

$$IG_{\mathrm{TM}}^{\Sigma} = \{M_1\$M_2 : M_1, M_2 \in \mathcal{M}^{\Sigma} \text{ e } L_{\mathrm{ac}}(M_1) = L_{\mathrm{ac}}(M_2)\}$$

e seja $\overline{IG_{\mathrm{TM}}^{\Sigma}}$ a linguagem complementar de $IG_{\mathrm{TM}}^{\Sigma}$.

   (a) Mostre que o *problema da igualdade de linguagens* relativo a máquinas de Turing não é decidível, ou seja, mostre que $IG_{\mathrm{TM}}^{\Sigma}$ não é decidível.

   (b) Mostre que $\mathcal{L}_{\mathrm{ac}}^{\Sigma} \leq \overline{IG_{\mathrm{TM}}^{\Sigma}}$.

   (c) Use a alínea (b) para demonstrar que o *problema da igualdade de linguagens* não é reconhecível, isto é, $IG_{\mathrm{TM}}^{\Sigma}$ não é reconhecível.

13. Mostre que as seguintes linguagens não são decidíveis:

   (a) $L = \{M \in \mathcal{M}^{\{0,1\}} : L_{\mathrm{ac}}(M) \text{ é o conjunto das palavras que começam e terminam em } 0\}$;

   (b) $L = \{M \in \mathcal{M}^{\{0,1\}} : L_{\mathrm{ac}}(M) \text{ é o conjunto das palavras que têm o mesmo número de 0s e de 1s}\}$;

   (c) $L = \{M \in \mathcal{M}^{\{0,1\}} : L_{\mathrm{ac}}(M) \text{ é o conjunto dos palíndromos em } \{0,1\}^*\}$.

14. Uma *gramática* é um tuplo $G = (V, \Sigma, P, S)$ em que

   - V é um conjunto finito (conjunto dos símbolos auxiliares);
   - Σ é um alfabeto;

- P é um conjunto finito de regras de substituição (ou produções); uma regra de substituição é uma expressão $\alpha \to \beta$ em que $\alpha, \beta \in (\Sigma \cup V)^*$ e α tem pelo menos um símbolo em V;
- $S \in V$ (símbolo inicial).

Uma palavra $w \in \Sigma^*$ diz-se *gerada* por uma gramática $G = (V, \Sigma, P, S)$ se existe uma sequência finita $w_1, \ldots, w_n$ de palavras em $(\Sigma \cup V)^*$ tal que $w_1 = S$, $w_n = w$ e, para cada $1 < i \leq n$, a palavra w_i obtém-se a partir w_{i-1} por aplicação de uma regra de substituição (w_i obtém-se a partir w_{i-1} por aplicação de uma regra de substituição $\alpha \to \beta$ se α é uma subpalavra de w_{i-1}, e w_i se obtém de w_{i-1} substituindo uma ocorrência de α em w_{i-1} por β). A *linguagem gerada* por G é o conjunto das palavras $w \in \Sigma^*$ que são geradas por G.

(a) Uma linguagem diz-se *regular* se é gerada por uma gramática na qual todas as regras de substituição são do tipo $X \to \beta$, onde $X \in V$ e $\beta \in \Sigma \cup \{\epsilon\}$ ou $\beta = sY$ com $s \in \Sigma$ e $Y \in V$. Mostre que não é decidível a linguagem

$$REG_{\mathrm{TM}}^{\Sigma} = \{M \in \mathcal{M}^{\Sigma} : L_{\mathrm{ac}}(M) \text{ é linguagem regular}\}.$$

Pode assumir como provado que a linguagem $\{1^n : n \in \mathbb{N} \text{ é primo}\}$ não é regular.

(b) Uma linguagem diz-se *independente do contexto* se é gerada por uma gramática na qual todas as regras de substituição são do tipo $X \to \beta$ com $X \in V$. Mostre que não é decidível a linguagem

$$IC_{\mathrm{TM}}^{\Sigma} = \{M \in \mathcal{M}^{\Sigma} : L_{\mathrm{ac}}(M) \text{ é linguagem independente do contexto}\}.$$

Pode assumir como provado que a linguagem $\{1^n : n \in \mathbb{N} \text{ é primo}\}$ não é independente do contexto.

15. Complete a demonstração da Proposição 4.12.

16. A função total *Busy Beaver* $BB : \mathbb{N} \to \mathbb{N}$ foi apresentada pela primeira vez em 1962 no artigo científico "*On non-computable functions*", de Tibor Radó. Para cada $n \in \mathbb{N}$, $BB(n)$ é definido por:

$BB(n) =$ número de transições da maior computação que termina, de uma máquina de Turing com n estados de controlo, com alfabeto de trabalho $\{1, \square\}$ e sem transições-S, para o *input* ϵ.

Os valores exactos de $BB(n)$ só são conhecidos valores de n inferiores a 5: $BB(1) = 1$, $BB(2) = 6$, $BB(3) = 21$ e $BB(4) = 107$. Relativamente a $n = 5$ e $n = 6$, os maiores limites inferiores conhecidos são 47 176 870 e 2.5×10^{2879}, respectivamente. Encontram-se na Figura 4.6 máquinas de Turing que testemunham os valores de $BB(3)$ e $BB(4)$, bem como o referido limite inferior para $BB(5)$.

(a) Mostre que se BB for uma função computável então a linguagem $L_\epsilon = \{M \in \mathcal{M}^{\{1\}} : M \text{ termina com } \textit{input } \epsilon\}$ é decidível.

(b) Usando a alínea anterior, conclua que a função BB não é computável. Sugestão: recorde que $\mathcal{L}_{\text{te}}^{\Sigma}$ não é decidível.

(c) Considere a seguinte linguagem:

$$\mathcal{BB} = \{M \in \mathcal{M}^{\{1\}} : M \text{ termina com } \textit{input } \epsilon \text{ em } BB(\text{n}^o \text{ de estados de controlo de } M) \text{ passos}\}.$$

Mostre que se $\mathcal{BB}$ é decidível então a função BB é computável.

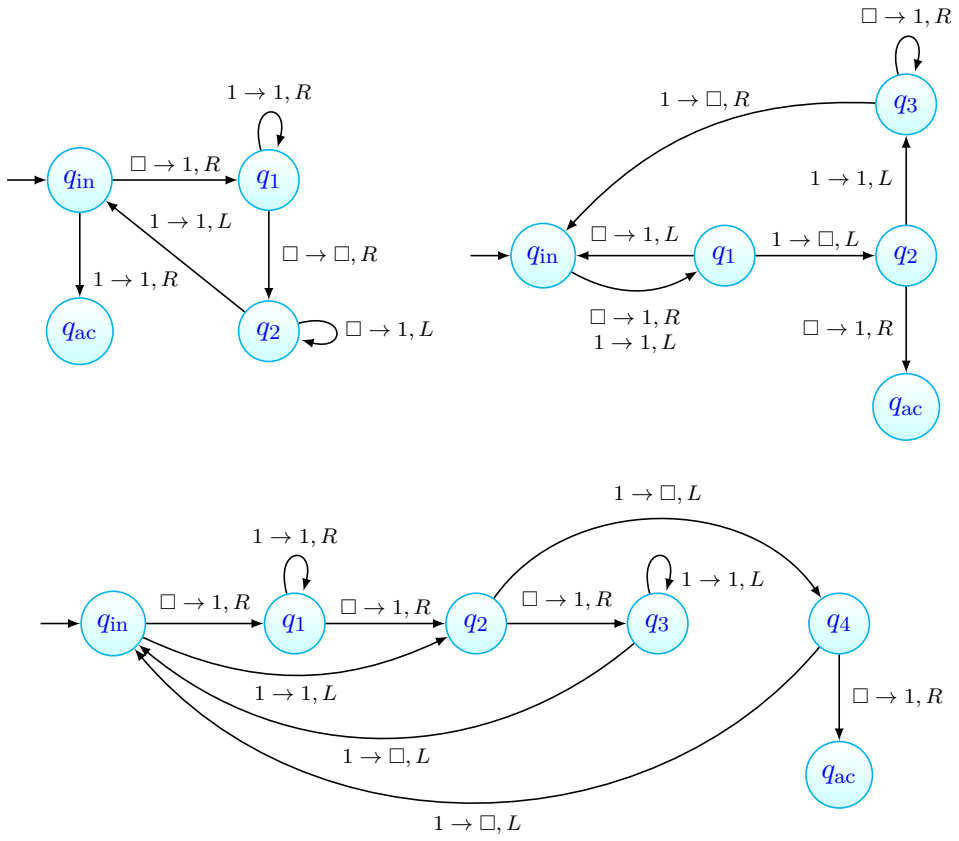


Figura 4.6: Exemplo de máquina de Turing para $BB(3)$ (em cima à esquerda), para $BB(4)$ (em cima à direita) e para o maior limite inferior conhecido para $BB(5)$ (em baixo).

Para consultar mais informação e contexto histórico sobre a função *Busy Beaver* aconselhamos um artigo de divulgação escrito pelo cientista da computação Scott Aaronson:

`http://www.scottaaronson.com/writings/bignumbers.html`.

5 Teorema de Rice

1. Seja Σ um alfabeto, $a \in \Sigma$ e $p \in \Sigma^*$. Use o Teorema de Rice para demonstrar que as seguintes linguagens não são decidíveis:

   (a) $L = \{M \in \mathcal{M}^\Sigma : \epsilon \notin L_{\text{ac}}(M)\}$;

   (b) $L = \{M \in \mathcal{M}^\Sigma : L_{\text{ac}}(M)$ tem pelo menos duas palavras$\}$;

   (c) $L = \{M \in \mathcal{M}^\Sigma : L_{\text{ac}}(M)$ é decidível$\}$;

   (d) $L = \{M \in \mathcal{M}^\Sigma : L_{\text{ac}}(M) = \emptyset\}$;

   (e) $L = \{M \in \mathcal{M}^\Sigma : M$ aceita $p\}$;

   (f) $L = \{M \in \mathcal{M}^\Sigma : L_{\text{ac}}(M)$ é uma linguagem finita$\}$;

   (g) $L = \{M \in \mathcal{M}^\Sigma :$ as palavras aceites por M têm comprimento par$\}$;

   (h) $L = \{M \in \mathcal{M}^\Sigma :$ as palavras aceites por M têm comprimento ímpar$\}$;

   (i) $L = \{M \in \mathcal{M}^\Sigma : L_{\text{ac}}(M)$ é o conjunto das palavras que começam e terminam em $a\}$.

2. Pode usar o Teorema de Rice para demonstrar que as seguintes linguagens não são decidíveis? Em caso afirmativo, use-o para demonstrar que a linguagem em causa não é decidível.

   (a) $L = \{M \in \mathcal{M}^{\{0,1\}} : M$ escreve 111$\}$
   (a expressão "M escreve 111" significa aqui que existe pelo menos uma configuração inicial a partir da qual M chega a uma configuração de aceitação na qual o conteúdo da fita é 111).

   (b) $L = \{M \in \mathcal{M}^{\{0,1\}} : M$ aceita 111$\}$.

   (c) $L = \{M \in \mathcal{M}^{\{0,1\}} : M$ tem mais de 10 estados$\}$.

   (d) $L = \{M \in \mathcal{M}^{\{0,1\}} : L_{\text{ac}}(M)$ é reconhecível$\}$.

   (e) $L = \{M \in \mathcal{M}^{\{0,1\}} : M$ nunca atinge uma configuração de rejeição$\}$.

   (f) $L = \{M \in \mathcal{M}^{\{0,1\}} : M$ é classificador$\}$.

   (g) $L = \{M \in \mathcal{M}^{\{0,1\}} : L_{\text{ac}}(M)$ e $\overline{L_{\text{ac}}(M)}$ são linguagens infinitas$\}$.

3. Seja Σ um alfabeto. Considere as linguagens

$$L_1 = \{M \in \mathcal{M}^\Sigma : L_{\text{ac}}(M) \text{ é finita}\}$$
$$L_2 = \{M_1\$M_2 : M_1, M_2 \in \mathcal{M}^\Sigma \text{ são tais que } L_{\text{ac}}(M_1) \cap L_{\text{ac}}(M_2) \text{ é finita}\}$$

   (a) Use o Teorema de Rice para demonstrar que L_1 é indecidível.

   (b) É possível usar o Teorema de Rice para demonstrar que L_2 é indecidível?

   (c) Mostre que $L_1 \leq L_2$ e conclua que L_2 é indecidível.

4. Seja Σ um alfabeto que inclui 0 e 1. Considere as linguagens

$$P = \{w \in \{0,1\}^* : |w| \text{ é ímpar}\}$$
$$L = \{M_1\$M_2 : M_1, M_2 \in \mathcal{M}^\Sigma \text{ são tais que } P \subseteq L_{\text{ac}}(M_1) \cap L_{\text{ac}}(M_2)\}.$$

(a) É possível usar o Teorema de Rice para demonstrar que L é indecidível?

(b) Defina adequadamente uma linguagem indecidível S tal que $S \leq L$.

(c) Conclua, a partir da alínea (b), que L é indecidível.

5

Complexidade computacional

"Hardness ever of hardiness is mother"
William Shakespeare, *Cymbeline*, 1610

5.1 Eficiência de máquinas

A computação é uma tarefa que consome recursos. Os recursos que mais usualmente são considerados são o tempo (a duração da computação) e o espaço (uma medida da informação que é armazenada/consumida durante a computação), mas também é possível considerar outras quantidades, como a energia (necessária para executar a computação). É intuitivo que com mais recursos, mais tempo, mais espaço de armazenamento, deveremos poder realizar mais tarefas e por conseguinte resolver mais problemas.

A complexidade computacional consiste no estudo dos recursos necessários para resolver um problema (que possa ser resolvido, claro). Como ponto de partida precisamos de medir a eficiência de cada algoritmo que resolva o problema.

Podemos calcular a eficiência de uma máquina de Turing (que seja um classificador) relativamente ao tempo e ao espaço, de forma relativamente simples, se tomarmos o número de passos de cada computação como uma medida do tempo, e o número de células de memória lidas/escritas ao longo de cada computação como uma medida de espaço.

Definição 5.1. Eficiência no tempo/espaço
Seja M uma máquina classificadora. Definem-se as funções $\text{time}_M, \text{space}_M : \mathbb{N} \to \mathbb{N}$ da seguinte forma:

- $\text{time}_M(n)$ é o comprimento máximo de uma computação de M sobre um *input* w com $|w| \leq n$;

- $\text{space}_M(n)$ é o número máximo de células de memória lidas/escritas durante uma computação de M sobre um *input* w com $|w| \leq n$. △

Para cada $n \in \mathbb{N}$, $\text{time}_M(n)$ e $\text{space}_M(n)$ dão-nos uma avaliação do pior caso, em termos de duração da computação ou da quantidade de memória necessária, respectivamente, para processar *inputs* de tamanho limitado por n.

Mais do que a expressão exacta das funções time_M e space_M associadas a um classificador M, estamos interessados em avaliar o seu crescimento. Por essa razão é usual usar notação assintótica, nomeadamente a notação $\mathcal{O}$ (ver Secção 1.4).

Exemplo 5.2.
Para a máquina classificadora M da Figura 5.1, que decide a linguagem L das palavras alternantes sobre o alfabeto $\{0,1\}$, tem-se $\text{time}_M(n) = n+2$ e $\text{space}_M(n) = n+1$. △

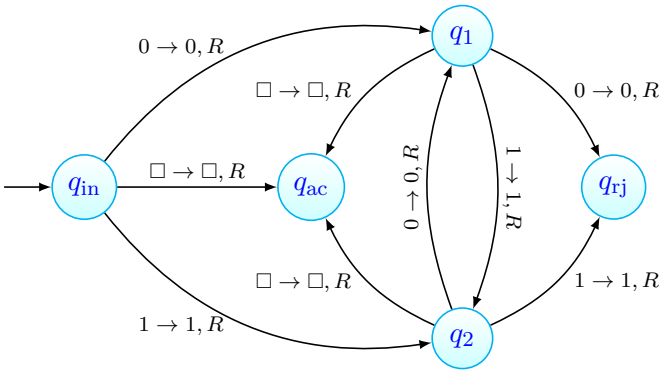


Figura 5.1: Máquina classificadora M que decide a linguagem das palavras alternantes sobre $\{0,1\}$.

5.2 Classes de complexidade

Agrupamos problemas, de acordo com a sua dificuldade relativa, em classes de complexidade.

Definição 5.3. Classes de tempo/espaço determinista
Seja $f : \mathbb{N} \to \mathbb{R}_0^+$ uma função. Definem-se as seguintes classes de linguagens:

- $\mathbf{TIME}(f(n)) = \{L : \text{existe máquina } M \text{ que decide } L \text{ com } \text{time}_M(n) = \mathcal{O}(f(n))\}$;
- $\mathbf{SPACE}(f(n)) = \{L : \text{existe máquina } M \text{ que decide } L \text{ com } \text{space}_M(n) = \mathcal{O}(f(n))\}$. △

Nomeadamente, $\mathbf{TIME}(f(n))$ agrupa todas as linguagens que podem ser decididas por alguma máquina cujas computações têm comprimento máximo limitado por algum múltiplo de $f(n)$. Passa-se algo análogo para $\mathbf{SPACE}(f(n))$.

Observe-se que em todas as situações nas quais o que é relevante é determinar se $\text{time}_M(n) = \mathcal{O}(f(n))$ para alguma função f, no cálculo de $\text{time}_M(n)$ pode usar-se indistintamente o comprimento das computações de M, ou o número

de passos (transições) dessas computações. O mesmo acontece com o cálculo de $\text{space}_M(n)$, no qual pode ser usado indistintamente o número de células lidas/escritas, ou o número de células visitadas.

É usual, em complexidade, considerar como eficientes as máquinas cujo comportamento é limitado por um polinómio. Por outro lado, o protótipo das máquinas ineficientes é precisamente o crescimento exponencial. Estas considerações levam à definição das seguintes classes de complexidade.

Definição 5.4. Classes de complexidade deterministas
Definem-se as seguintes classes de linguagens, relativamente ao tempo:[1]

- $\mathbf{P} = \bigcup_{k\in\mathbb{N}} \mathbf{TIME}(n^k)$;
- $\mathbf{EXPTIME} = \bigcup_{k\in\mathbb{N}} \mathbf{TIME}(2^{n^k})$;

e também, relativamente ao espaço, as classes:

- $\mathbf{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{SPACE}(n^k)$;
- $\mathbf{EXPSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{SPACE}(2^{n^k})$. △

P contém as linguagens que podem ser decididas em tempo polinomial (limitado por algum polinómio), tal como **PSPACE** contém as linguagens que podem ser decididas em espaço polinomial. Analogamente, **EXPTIME** contém as linguagens que podem ser decididas em tempo exponencial, tal como **EXPSPACE** contém as linguagens que podem ser decididas em espaço exponencial.

Exemplo 5.5.
Existe uma máquina M que decide a linguagem L das palavras alternantes sobre $\{0,1\}$ com $\text{time}_M(n) = n+2$ e $\text{space}_M(n) = n+1$ (ver Exemplo 5.2). Como $\text{time}_M(n) = \mathcal{O}(n)$, tem-se que $L \in \mathbf{TIME}(n)$ e, consequentemente, $L \in \mathbf{P}$. De igual modo, $\text{space}_M(n) = \mathcal{O}(n)$, logo $L \in \mathbf{SPACE}(n)$ e, portanto, $L \in \mathbf{PSPACE}$. △

Estando associadas a quantidades físicas, é natural que haja algum tipo de relação em espaço e tempo que nos possa ser útil.

Proposição 5.6.
Seja M uma máquina classificadora. Tem-se que time_M e space_M são funções monótonas e, para qualquer $n \in \mathbb{N}$:

1. $\text{space}_M(n) \leq \text{time}_M(n)$;
2. $\text{time}_M(n) \leq 2^{\mathcal{O}(\text{space}_M(n))}$.

Dem.: O facto de time_M e space_M serem monótonas é uma consequência imediata da Definição 5.1. Vejamos as outras afirmações.
(1) Por cada célula lida/escrita para além da inicial, M tem de fazer pelo menos

[1] Definem-se com uma estrutura análoga classes de funções (tipicamente com o sufixo **F**). Por exemplo, $\mathbf{PF} = \{f : \text{existe uma máquina } M \text{ que calcula } f \text{ com } \text{time}_M(n) = \mathcal{O}(n^k), n \in \mathbb{N}\}$.

um movimento para se deslocar para essa célula. Assim, o número de movimentos que M faz é pelo menos o número de células lidas/escritas para além da inicial. Como $\text{time}_M(n)$ é superior a esse número de movimentos tem-se que $\text{space}_M(n) \leq \text{time}_M(n)$
(2) Considere-se a computação da máquina de Turing M com alfabeto de trabalho Γ e conjunto de estados Q sobre um *input* de comprimento menor ou igual a n, com o maior número de configurações. Então r não tem configurações repetidas. Portanto o número de configurações em r tem de ser inferior ou igual ao número de configurações que podem existir no espaço que r utiliza. Suponha-se que em r são visitadas m células. Observe-se que o número de configurações possíveis utilizando m células é

$$|\Gamma|^m \times |Q| \times m.$$

Então:

$$\begin{aligned} \text{time}_M(n) &\leq |\Gamma|^m \times |Q| \times m \\ &\leq 2^{\log_2(|\Gamma|)\times m} 2^{\log_2(|Q|)} 2^m \\ &= 2^{m(\log_2(|\Gamma|)+1)+\log_2(|Q|)} \\ &= 2^{\mathcal{O}(\text{space}_M(n))}. \end{aligned}$$

□

Fazer nota de que $\text{space}_M(n) \leq \text{time}_M(n)$ implica precisamente que

$$\mathbf{TIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$$

para qualquer função f.

Este resultado permite-nos começar a relacionar as classes de complexidade definidas.

Corolário 5.7.
$\mathbf{P} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{EXPSPACE}$.

5.3 Variantes

É útil neste ponto discutir a eficiência de variantes da máquina de Turing, e avaliar o impacto que as diferenças que encontrarmos possam ter na definição de classes de complexidade.

5.3.1 Transições-S

Vimos na Proposição 3.11 que qualquer máquina de Turing com transições-S é equivalente a uma máquina de Turing sem transições-S, no sentido em que as máquinas reconhecem (decidem, se for o caso) a mesma linguagem e calculam a mesma função. É simples perceber que há pequenas diferenças, assintoticamente negligenciáveis, no que diz respeito à eficiência destas máquinas quer no espaço quer no tempo.

Proposição 5.8.
Toda a máquina de Turing M com transições-S é equivalente a uma máquina de Turing T sem transições-S tal que $\text{time}_T(n) = \mathcal{O}(\text{time}_M(n))$ *e* $\text{space}_T(n) = \mathcal{O}(\text{space}_M(n))$.

Dem. (esboço): Atente-se na máquina T construída a partir de M tal como na demonstração da Proposição 3.11.

É fácil de verificar que cada passo de computação de M corresponde no máximo a dois passos de computação de T, já que as transições-S passam a realizar-se com uma transição à direita seguida de outra à esquerda. Assim temos:

$$\text{time}_T(n) \quad \leq \quad 2.\text{time}_M(n) \quad = \quad \mathcal{O}(\text{time}_M(n)).$$

Relativamente ao espaço, pela razão acima apontada, a máquina T poderá usar no máximo uma célula de memória adicional quando simula uma transição-S de M no extremo direito do seu espaço de trabalho. Assim temos:

$$\text{space}_T(n) \quad \leq \quad 1 + \text{space}_M(n) \quad = \quad \mathcal{O}(\text{space}_M(n)). \qquad \square$$

5.3.2 Memória bidireccional

Vejamos se haverá diferenças mais sensíveis no que diz respeito à eficiência de máquinas bidireccionais.

Proposição 5.9.
Toda a máquina bidireccional M é equivalente a uma máquina de Turing T (unidireccional) tal que $\text{time}_T(n) = \mathcal{O}(n+\text{time}_M(n)^2)$ *e* $\text{space}_T(n) = \mathcal{O}(\text{space}_M(n))$.

Dem. (esboço): Atente-se na máquina T construída a partir de M tal como na demonstração da Proposição 3.13.

A computação de T começa por balizar o *input* com os símbolos `I` e `F`, num total de $2n + 3$ passos, o que corresponde a uma computação com 2n+4 configurações, onde n é o tamanho do *input*. De seguida prossegue exactamente como M excepto nos extremos do espaço de trabalho, em que necessita de introduzir espaçamento. No pior caso, cada transição de M poderá dar azo a um novo espaçamento. O espaçamento à direita, sobre `F`, é muito simples e totaliza 3 passos. O espaçamento à esquerda, sobre `I`, implica copiar todo o espaço de trabalho, o que totalizará no máximo $\mathcal{O}(\text{space}_M(n))$ passos. No final, há que apagar o delimitador `F`, o que demora $\mathcal{O}(\text{space}_M(n))$ passos. Assim temos:

$$\begin{aligned}
\text{time}_T(n) &\leq (2n+4) + \text{time}_M(n) \times \max(1, 3, \mathcal{O}(\text{space}_M(n))) + \\
&\qquad\qquad \mathcal{O}(\text{space}_M(n)) \\
&\leq \mathcal{O}(n) + \text{time}_M(n) \times \mathcal{O}(\text{space}_M(n)) + \mathcal{O}(\text{space}_M(n)) \\
&\leq \mathcal{O}(n) + \text{time}_M(n) \times \mathcal{O}(\text{time}_M(n)) + \mathcal{O}(\text{time}_M(n)) \\
&= \mathcal{O}(n) + \mathcal{O}(\text{time}_M(n)^2) + \mathcal{O}(\text{time}_M(n)) \\
&= \mathcal{O}(n + \text{time}_M(n) + \text{time}_M(n)^2) \\
&= \mathcal{O}(n + \text{time}_M(n)^2).
\end{aligned}$$

Com o espaço delimitado, a máquina T usará precisamente duas células de memória adicionais, ou seja

$$\text{space}_T(n) \leq 2 + \text{space}_M(n) = \mathcal{O}(\text{space}_M(n)).$$

□

Neste ponto fica claro que apesar da vantagem, potencialmente quadrática, das máquinas bidireccionais, o impacto desta variante nas classes de complexidade temporal mais relevantes é negligenciável, ou seja, se M for limitada no tempo por um polinómio então também T o será (o grau do polinómio é que será dobrado).

5.3.3 Memória multifita

Analisemos agora o caso, ainda mais interessante, das máquinas multifita.

Proposição 5.10.
Toda a máquina de Turing multifita M é equivalente a uma máquina de Turing T (com apenas uma fita) tal que $\text{time}_T(n) = \mathcal{O}(n + \text{time}_M(n)^2)$ *e* $\text{space}_T(n) = \mathcal{O}(\text{space}_M(n))$.

Dem. (esboço): Atente-se na máquina T construída a partir de M tal como na demonstração da Proposição 3.16.

A computação de T começa por inicializar a fita de memória, balizando o *input* com os símbolos `I` e `F`, e demarcando o espaço de cada uma das k fitas da máquina T, num número de passos da ordem de $\mathcal{O}(n+k)$, onde n é o tamanho do *input*. De seguida simula cada uma das transições de M, percorrendo a fita da esquerda para a direita de forma a ler os símbolos marcados, e depois da direita para a esquerda actualizando as marcações, visitando um número de células da ordem de $\mathcal{O}(\text{space}_M(n))$. Pode ter de efectuar um máximo de k espaçamentos, 1 em cada fita, visitando assim um número máximo de células da ordem de $k.\mathcal{O}(\text{space}_M(n))$. Note-se que numa máquina com k fitas cada passo de computação envolve também k células de memória. Finalmente, após a aceitação por M, os símbolos marcados são adequadamente substituídos, e o símbolo `F` é removido, o que de novo implica que um número de células da ordem de $\mathcal{O}(\text{space}_M(n))$ seja visitado. Assim, temos:

$$\begin{aligned}
\text{time}_T(n) &\leq \mathcal{O}(n+k) + \text{time}_M(n) \times (\mathcal{O}(\text{space}_M(n)) + \\
&\qquad k \times \mathcal{O}(\text{space}_M(n))) + \mathcal{O}(\text{space}_M(n)) \\
&\leq \mathcal{O}(n) + \text{time}_M(n) \times (\mathcal{O}(k \times \text{time}_M(n)) + \\
&\qquad \mathcal{O}(k^2 \times \text{time}_M(n))) + \mathcal{O}(k \times \text{time}_M(n)) \\
&= \mathcal{O}(n) + \text{time}_M(n) \times \mathcal{O}(\text{time}_M(n)) + \mathcal{O}(\text{time}_M(n)) \\
&= \mathcal{O}(n) + \mathcal{O}(\text{time}_M(n)^2) + \mathcal{O}(\text{time}_M(n)) \\
&= \mathcal{O}(n + \text{time}_M(n) + \text{time}_M(n)^2) \\
&= \mathcal{O}(n + \text{time}_M(n)^2).
\end{aligned}$$

Com o espaço delimitado, a máquina T usará precisamente $k+1$ células de memória adicionais, para marcação, ou seja

$$\text{space}_T(n) \leq k + 1 + \text{space}_M(n) = \mathcal{O}(\text{space}_M(n)).$$ □

De novo, apesar da vantagem quadrática no tempo, as máquinas multifita não trazem diferenças fundamentais às classes de complexidade mais relevantes.

5.3.4 Não-determinismo

No caso das máquinas não-deterministas vai surgir a primeira diferença substantiva relativamente à teoria da computabilidade. Do ponto de vista da complexidade computacional, o não-determinismo parece de facto ser um mecanismo demasiado poderoso. Note-se que, desde logo, necessitamos de redefinir rigorosamente a eficiência no espaço e no tempo para máquinas não-deterministas (não foi necessário fazê-lo nas variantes anteriores pois a Definição 5.1 continuava a fazer pleno sentido).

Definição 5.11. Eficiência e não-determinismo
Seja M uma máquina não-determinista classificadora. Definem-se as funções $\text{ntime}_M, \text{nspace}_M : \mathbb{N} \to \mathbb{N}$ da seguinte forma:

- $\text{ntime}_M(n)$ é o comprimento do maior ramo de computação de M sobre um *input* w com $|w| \leq n$;
- $\text{nspace}_M(n)$ é o número máximo de células de memória lidas/escritas durante algum dos ramos de computação de M sobre um *input* w com $|w| \leq n$. △

Definem-se por analogia as classes de complexidade relativas a máquinas não-deterministas.

Definição 5.12. Classes de tempo/espaço não-determinista
Seja $f : \mathbb{N} \to \mathbb{R}_0^+$ uma função. Definem-se as seguintes classes de linguagens:

- $\mathbf{NTIME}(f(n)) = \{L : \text{existe máquina não-determinista } M$ que decide L com $\text{ntime}_M(n) = \mathcal{O}(f(n))\}$;
- $\mathbf{NSPACE}(f(n)) = \{L : \text{existe máquina não-determinista } M$ que decide L com $\text{nspace}_M(n) = \mathcal{O}(f(n))\}$.

Definem-se as seguintes classes de linguagens, relativamente ao tempo:

- $\mathbf{NP} = \bigcup_{k\in\mathbb{N}} \mathbf{NTIME}(n^k)$;
- $\mathbf{NEXPTIME} = \bigcup_{k\in\mathbb{N}} \mathbf{NTIME}(2^{n^k})$;

e também, relativamente ao espaço, as classes:

- $\mathbf{NPSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{NSPACE}(n^k)$;
- $\mathbf{NEXPSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{NSPACE}(2^{n^k})$. △

Como bem sabemos, máquinas deterministas são casos particulares de máquinas não-deterministas, o que justifica o resultado seguinte.

Proposição 5.13.
Verificam-se as seguintes inclusões:

$$\mathbf{P} \subseteq \mathbf{NP}$$

$$\mathbf{PSPACE} \subseteq \mathbf{NPSPACE}$$

$$\mathbf{EXPTIME} \subseteq \mathbf{NEXPTIME}$$

$$\mathbf{EXPSPACE} \subseteq \mathbf{NEXPSPACE}.$$

As propriedades abaixo continuam a valer para máquinas não-deterministas.

Proposição 5.14.
Seja M uma máquina não-determinista classificadora. Tem-se que ntime_M e nspace_M são funções monótonas e, para qualquer $n \in \mathbb{N}$:

1. $\text{nspace}_M(n) \leq \text{ntime}_M(n)$;
2. $\text{ntime}_M(n) \leq 2^{\mathcal{O}(\text{nspace}_M(n))}$.

Dem.: O facto de ntime_M e nspace_M serem monótonas é uma consequência imediata da Definição 5.11. As outras afirmações têm justificação semelhante à apresentada na demonstração da Proposição 5.6, agora para um ramo de uma árvore de computação da máquina de Turing M sobre um *input* de comprimento menor ou igual a n em que M lê/escreve o maior número de células (no primeiro caso), ou em que M faz o maior número de transições (no segundo caso). □

Este resultado permite-nos começar a relacionar também as classes de complexidade não-deterministas.

Corolário 5.15.
$\mathbf{NP} \subseteq \mathbf{NPSPACE} \subseteq \mathbf{NEXPTIME} \subseteq \mathbf{NEXPSPACE}$.

Vejamos então qual a eficiência relativa de uma máquina de Turing determinista que simule uma máquina não-determinista.

Proposição 5.16.
Toda a máquina de Turing não-determinista N é equivalente a uma máquina de Turing determinista T tal que $\text{time}_T(n) = \mathcal{O}(n + \text{ntime}_N(n)) \times 2^{\mathcal{O}(\text{ntime}_N(n))}$ *e* $\text{space}_T(n) = \mathcal{O}(n + \text{ntime}_N(n))$.

Dem. (esboço): Atente-se na máquina T construída a partir de N tal como na demonstração da Proposição 3.20.

A computação de T começa por inicializar 3 fitas de memória, a segunda das quais com o comprimento máximo das computações possíveis, num número de passos da ordem de $\mathcal{O}(\text{ntime}_N(n))$, onde n é o tamanho do *input*. De seguida, copia o *input* da primeira para a terceira fita, balizando-o com os símbolos `I` e `F`, e executando um número de passos da ordem de $\mathcal{O}(n)$. Na terceira fita é então simulado o caminho de computação de N descrito na fita 2, num número de passos inferior ou igual a $\text{ntime}_N(n)$, após o que volta a limpar a fita 3, visitando um número de células da ordem de $\mathcal{O}(\text{space}_N(n))$. Em caso de aceitação por N, há ainda que, na terceira fita, remover o símbolo `F` e colocar a cabeça de leitura/escrita no início da palavra, visitando um número de células da ordem de $\mathcal{O}(\text{nspace}_N(n))$. Seja b o número máximo de escolhas não-deterministas na máquina N. Temos assim:

$$\begin{aligned}
\text{time}_T(n) &\leq \mathcal{O}(\text{ntime}_N(n)) + \\
&\qquad b^{\text{ntime}_N(n)} \times (\mathcal{O}(n) + \text{ntime}_N(n) + \mathcal{O}(\text{nspace}_N(n))) + \\
&\qquad \mathcal{O}(\text{nspace}_N(n)) \\
&\leq \mathcal{O}(\text{ntime}_N(n)) + b^{\text{ntime}_N(n)} \times (\mathcal{O}(n) + \mathcal{O}(\text{ntime}_N(n))) + \\
&\qquad \mathcal{O}(\text{ntime}_N(n)) \\
&= \mathcal{O}(\text{ntime}_N(n)) + b^{\text{ntime}_N(n)} \times (\mathcal{O}(n) + \mathcal{O}(\text{ntime}_N(n))) \\
&= \mathcal{O}(n + \text{ntime}_N(n)) \times 2^{\mathcal{O}(\text{ntime}_N(n))}.
\end{aligned}$$

Relativamente ao espaço, a máquina T terá os contributos das 3 fitas de memória, ou seja

$$\begin{aligned}
\text{space}_T(n) &\leq \mathcal{O}(n) + \mathcal{O}(\text{ntime}_N(n)) + \mathcal{O}(\text{nspace}_N(n)) \\
&\leq \mathcal{O}(n) + \mathcal{O}(\text{ntime}_N(n)) + \mathcal{O}(\text{ntime}_N(n)) \\
&= \mathcal{O}(n + \text{ntime}_N(n)).
\end{aligned}$$

□

Espacialmente, há uma diferença sensível pois o espaço da máquina determinista não depende apenas do espaço da máquina não-determinista, como nos casos anteriores. Ainda assim esta distinção não tem implicações profundas ao nível das classes de complexidade espacial mais relevantes, o que confirmaremos mais adiante. Relativamente ao tempo, no entanto, a existência de uma máquina não-determinista de tempo polinomial para resolver um problema parece não nos poder garantir mais do que uma máquina determinista de tempo exponencial para resolver o mesmo problema. Intuitivamente isto não é inesperado, mas trata-se de um problema em aberto, com inúmeras repercussões importantes e interessantes, demonstrar que de facto não é possível fazer esta simulação de forma mais eficiente. Se não conseguimos demonstrá-lo, será que existe mesmo tal algoritmo, por muito estranho que possa ser?

5.4 Propriedades de fecho e redução polinomial

Proposição 5.17.
Seja $\mathcal{C}$ uma das classes de complexidade **P**, **NP**, **PSPACE**, **NPSPACE**,

EXPTIME, **NEXPTIME**, **EXPSPACE**, **NEXPSPACE**, *e sejam* Σ *um alfabeto, e* L_1, L_2 *linguagens sobre* Σ *tais que* $L_1, L_2 \in \mathcal{C}$.

Então, $\emptyset, \Sigma^*, L_1 \cap L_2, L_1 \cup L_2 \in \mathcal{C}$.

Se $\mathcal{C} \neq$ **NP** *e* $\mathcal{C} \neq$ **NEXPTIME** *então também* $L_1 \setminus L_2 \in \mathcal{C}$.

Dem.: Exercício. □

Stephen Cook e Leonid Levin mostraram que existem algumas linguagens em **NP** às quais todas as outras linguagens dessa classe se reduzem. Isto é, o problema de decidir uma qualquer linguagem de **NP** pode ser reduzido ao problema de decidir uma dessas linguagens. Mais ainda, mostraram que essa redução pode ser feita em tempo polinomial! Por isso se se encontrar uma solução eficiente para um desses problemas de **NP** ao qual todos os outros se reduzem, também se consegue decidir eficientemente qualquer outro problema de **NP**. Vamos então começar por definir o que é uma redução polinomial de uma linguagem a outra: trata-se simplesmente de uma redução computável, no sentido da Definição 4.6, eficiente.

Definição 5.18. Redução polinomial
Sejam Σ_1, Σ_2 alfabetos, L_1 e L_2 linguagens sobre Σ_1 e Σ_2, respectivamente. Dizemos que há uma *redução polinomial* de L_1 a L_2, ou simplesmente que L_1 *se reduz polinomialmente* a L_2, o que denotamos por $L_1 \leq_P L_2$ se existe uma função total $f : \Sigma_1^* \to \Sigma_2^*$ calculada por uma máquina determinista em tempo polinomial tal que se tem, para cada $w \in \Sigma_1^*$,

$$w \in L_1 \text{ se e só se } f(w) \in L_2.$$

△

Obviamente, $L_1 \leq_P L_2$ implica que $L_1 \leq L_2$.

Proposição 5.19.
Sejam Σ_1, Σ_2 *alfabetos,* L_1 *e* L_2 *linguagens sobre* Σ_1 *e* Σ_2*, respectivamente. Seja* $\mathcal{C}$ *uma das classes de complexidade* **P**, **NP**, **PSPACE**, **NPSPACE**, **EXPTIME**, **NEXPTIME**, **EXPSPACE**, **NEXPSPACE**. *Se* $L_1 \leq_P L_2$ *e* $L_2 \in \mathcal{C}$ *então* $L_1 \in \mathcal{C}$.

Dem.: As demonstrações são semelhantes para todas as classes. Ilustra-se a demonstração para a classe **NP**.

Assuma-se que $L_1 \leq_P L_2$ e que $L_2 \in$ **NP**. Sejam N uma máquina de Turing não-determinista que decide L_2 e $k_1 \geq 1$ tal que $\text{ntime}_N(n) = \mathcal{O}(n^{k_1})$, e sejam f a redução polinomial de L_1 a L_2, M uma máquina de Turing que calcula f e $k_2 \geq 1$ tal que $\text{time}_M(n) = \mathcal{O}(n^{k_2})$.

Considere-se a máquina de Turing não-determinista T que ao receber um *input* w passa o controlo a M até M terminar e de seguida passa o controlo a N até N terminar. Após N terminar então T termina da mesma forma que N terminou.

Demonstra-se primeiro que $L_{\text{ac}}(T) = L_1$.

(i) $L_{\text{ac}}(T) \subseteq L_1$. Seja $w \in L_{\text{ac}}(T)$. Então há um ramo da computação de T que aceita w e portanto há um ramo da computação de N que aceita $f(w)$. Logo $f(w) \in L_2$. Como f é uma redução polinomial de L_1 a L_2, então $w \in L_1$.

(ii) $L_1 \subseteq L_{\text{ac}}(T)$. Seja $w \in L_1$. Então $f(w) \in L_2$, pois f é uma redução polinomial de L_1 a L_2. Logo, há um ramo da computação de N que aceita $f(w)$, isto é, há um ramo da computação de N que aceita o resultado da computação de M sobre w. Consequentemente, há um ramo da computação de T que aceita w, pelo que $w \in L_{\text{ac}}(T)$.

Demonstra-se agora que $L_{\text{rj}}(T) = \overline{L_1}$.

(i) $L_{\text{rj}}(T) \subseteq \overline{L_1}$. Seja $w \in L_{\text{rj}}(T)$. Então T rejeita w e, portanto, nenhum ramo da computação de N aceita $f(w)$. Logo, $f(w) \notin L_2$. Como f é uma redução polinomial de L_1 a L_2, então $w \notin L_1$, pelo que $w \in \overline{L_1}$.

(ii) $\overline{L_1} \subseteq L_{\text{rj}}(T)$. Seja $w \in \overline{L_1}$, isto é, $w \notin L_1$. Então $f(w) \notin L_2$, pois f é uma redução polinomial de L_1 a L_2. Logo, nenhum ramo da computação de N aceita $f(w)$, isto é, nenhum ramo da computação de N aceita o resultado da computação de M sobre w. Consequentemente, $w \in L_{\text{rj}}(T)$.

Finalmente, mostra-se que $\text{ntime}_T(n) = \mathcal{O}(n^k)$ para algum $k \geq 1$. Tem-se

$$\begin{aligned}
\text{ntime}_T(n) &= \text{time}_M(n) + \text{ntime}_N(|f(x)|) \\
&\leq \text{time}_M(n) + \text{ntime}_N(|x| + \text{time}_M(|x|)) \\
&= \text{time}_M(n) + \text{ntime}_N(n + \text{time}_M(n)) \\
&= \mathcal{O}(n^{k_2}) + \mathcal{O}((n + n^{k_2})^{k_1}) \\
&= \mathcal{O}(n^{k_2} + (n + n^{k_2})^{k_1}) \\
&= \mathcal{O}(n^{k_1 k_2}).
\end{aligned}$$

Conclui-se então que $L_1 \in \mathbf{NP}$. □

5.5 Teorema de Savitch

Nas secções anteriores vimos que:

- **PSPACE** $\subseteq$ **NPSPACE**;
- **EXPSPACE** $\subseteq$ **NEXPSPACE**;

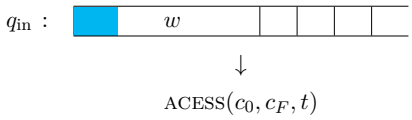
mas não estabelecemos nenhuma majoração das classes não-deterministas de espaço por classes deterministas, que é o que vamos fazer agora com o chamado *Teorema de Savitch.*

Teorema 5.20.
Seja $f : \mathbb{N} \to \mathbb{R}^+$ *tal que* $f(n) \geq n$. *Então*

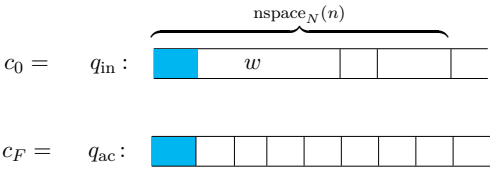
$$\mathbf{NSPACE}(f(n)) \subseteq \mathbf{SPACE}(f(n)^2).$$

Dem.: Assuma-se que $L \in \mathbf{NSPACE}(f(n))$. Então existe uma máquina de Turing não-determinista N que decide L e que é tal que $\text{nspace}_N(n) = \mathcal{O}(f(n))$. Sem perda de generalidade, assume-se que N, imediatamente antes de transitar para o estado de aceitação, limpa a fita e posiciona a cabeça de leitura/escrita na célula mais à esquerda.

Seja R a máquina de Turing que, ao receber como *input* uma palavra w de comprimento n, testa se numa árvore de computação de N há um ramo que evolui da configuração c_0 até à configuração c_F, em 2^t ou menos passos com $2^t > \text{ntime}_N(n)$:



onde c_0 e c_F são as configurações de N



e, para $i \in \mathbb{N}_0$ e c_1, c_2 configurações de N, ACESS(c_1, c_2, i) determina, como se segue, se numa árvore de computação de N há um ramo que evolui da configuração c_1 até à configuração c_2, em 2^i ou menos passos:

ACESS(c_1, c_2, i):
  se $i = 0$ então
    verifica se $c_1 = c_2$ ou se $c_1 \to_{N,1} c_2$ e aceita se sim, e rejeita se não [2]
  caso contrário
    para cada configuração possível c de N
    testa ACESS$(c_1, c, i-1)$ e ACESS$(c, c_2, i-1)$
    e aceita se ambos aceitam, e rejeita em caso contrário

Note-se que o Teorema da Recursão (Teorema 4.16) assegura a existência de uma máquina de Turing que funciona como se descreve acima.

Relativamente à função space$_R(n)$, e recordando que nspace$_N(n) = \mathcal{O}(f(n))$ e $f(n) \geq n$, tem-se

$$\begin{array}{rcl} \text{space}_R(n) & \leq & \mathcal{O}(n) + \mathcal{O}(\log(\text{ntime}_N(n))) \times \mathcal{O}(\text{nspace}_N(n)) \\ & \leq & \mathcal{O}(n) + \mathcal{O}(\log(2^{\text{nspace}_N(n)})) \times \mathcal{O}(\text{nspace}_N(n)) \\ & = & \mathcal{O}(\text{nspace}_N(n)^2) \\ & = & \mathcal{O}(f(n)^2). \end{array}$$

Na primeira linha, a expressão $\mathcal{O}(n)$ corresponde ao espaço necessário para construir as configurações c_0 e c_F, e a expressão $\mathcal{O}(\log(\text{ntime}_N(n))) \times \mathcal{O}(\text{nspace}_N(n))$ ao espaço necessário para memorizar (reutilizando espaço) as $\mathcal{O}(t)$ configurações possíveis em casa passo de recursão do algoritmo.

Tem-se $L_{\text{ac}}(R) = L$, pois $w \in L_{\text{ac}}(R)$ se e só se R aceita w se e só se existe uma evolução não-determinista de N desde a sua configuração inicial para w até à sua configuração de aceitação se e só se N aceita w se e só se $w \in L$.

Tem-se também que $L_{\text{rj}}(R) = \overline{L}$, pois $w \in L_{\text{rj}}(R)$ se e só se R rejeita w se e só se não existe uma evolução não-determinista de N desde a sua configuração inicial para w até à sua configuração de aceitação se e só se N rejeita w se e só se $w \notin L$ se e só se $w \in \overline{L}$.

Pode então concluir-se que $L \in \mathbf{SPACE}(f(n)^2)$. □

Este resultado permite-nos identificar classes de complexidade espaciais deterministas e não-deterministas.

[2] $c_1 \to_{N,1} c_2$ denota que numa árvore de computação de N há um ramo que evolui da configuração c_1 até à configuração c_2 em um passo.

Corolário 5.21.
PSPACE = **NPSPACE** *e* **EXPSPACE** = **NEXPSPACE**.

5.6 Teoremas de hierarquia

Até este momento, e relativamente às classes de complexidade mais importantes, sabemos que **P** ⊆ **NP** ⊆ **PSPACE** ⊆ **EXPTIME**, onde temos entre **P** (a classe dos problemas de decisão com soluções eficientes) e **EXPTIME** (a classe dos problemas de decisão com soluções exponenciais) uma estrutura interessante. Gostaríamos de poder, de facto, mostrar que as inclusões acima são estritas, significando que estaríamos a falar de classes de problemas de facto cada vez mais difíceis, de acordo com a intuição. Mas não vimos ainda como demonstrar a separação de classes. É esse o interesse dos seguintes teoremas de hierarquia.

Definição 5.22. Função construtível no espaço
Uma função $f : \mathbb{N} \to \mathbb{N}$ diz-se *construtível no espaço* se $\log(n) = \mathcal{O}(f(n))$ e se a função que mapeia palavras 1^n para a representação binária de $f(n)$ é computável em espaço $\mathcal{O}(f(n))$. △

Todas as funções mais comuns que são pelo menos $\mathcal{O}(\log(n))$, como, por exemplo, $\log(n)$, $n\log(n)$, n, n^2 e 2^n, são construtíveis no espaço.

Teorema 5.23.
Seja $f : \mathbb{N} \to \mathbb{N}$ uma função construtível no espaço. Então existe uma linguagem decidível em espaço $\mathcal{O}(f(n))$ mas não em espaço $o(f(n))$.

Dem.: Considere-se a linguagem $L = \{M\$1^k : M \in \mathcal{M}^\Sigma$ não aceita $M\$1^k$ em espaço inferior ou igual a $f(|M\$1^k|)\}$ e a máquina de Turing D que, ao receber o *input* w de comprimento n, evolui como se segue:

calcula $f(|w|)$ $(= s)$
delimita s células de memória
verifica se $w = M\$1^k$ e
  se não, rejeita
  se sim,
    obtém $q = |Q|$ e $\gamma = |\Gamma|$ a partir de M
    inicializa relógio binário com $\log_2(q) + (\log_2 \gamma + 1) \times s$ *bits* [3]
    simula M sobre $M\$1^k$, incrementando o relógio em cada passo, e
      se a simulação termina aceitando, rejeita
      se a simulação termina rejeitando ou abortando,
      ou se o espaço delimitado ou o tempo estipulado é ultrapassado, aceita

A máquina D é uma máquina classificadora. Para concluir que decide L, demonstra-se que $L_{\text{ac}}(D) = L$.

(i) $L_{\text{ac}}(D) \subseteq L$. Seja $w \in L_{\text{ac}}(D)$. Então, $w = M\$1^k$ e a máquina M, quando recebe $M\$1^k$ como *input*, ou rejeita ou aborta em espaço inferior ou igual a $s\,(= f(|M\$1^k|))$ e tempo inferior ou igual a $2^{\log_2(q)+(\log_2\gamma+1)\times s}$, ou tem

[3] isto é, delimita este número de células de memória e escreve 0 em cada uma (para poder contar até $2^{\log_2(q)+(\log_2\gamma+1)\times s}$).

uma computação que usa um número de células ou tem comprimento maior que os valores indicados correspondentes. No primeiro caso, M não aceita $M\$1^k$ em espaço inferior ou igual a $f(|M\$1^k|)$, logo $w \in L$. No segundo caso, se o espaço delimitado é ultrapassado, é imediato concluir que $w \in L$; o mesmo se pode concluir quando o comprimento da computação de M excede $2^{\log_2(q)+(\log_2\gamma+1)\times s}$, pois neste caso M também não aceita $M\$1^k$ em espaço inferior ou igual a $f(|M\$1^k|)$, uma vez que $\gamma^s \times q \times s$ é precisamente o número das distintas configurações possíveis de M quando utiliza apenas s células de memória.

(ii) $L \subseteq L_{\text{ac}}(D)$. Seja $w \in L$. Suponha-se, por absurdo, que $w \in L_{\text{rj}}(D)$. Então, como $w = M\$1^k$, tem-se que a simulação de M com *input* $M\$1^k$ termina e, em particular, em espaço inferior ou igual ao estipulado, o que contradiz a hipótese $w \in L$. Logo, $w \notin L_{\text{rj}}(D)$. Consequentemente, $w \in L_{\text{ac}}(D)$, uma vez que D é classificadora.

Estuda-se de seguida a função $\text{space}_D(n)$. O cálculo de $f(|w|)$ pode ser efectuado em espaço da ordem de $\mathcal{O}(f(n))$, dado que f é construtível no espaço. A simulação é, por construção, efetuada em espaço da ordem de $\mathcal{O}(f(n))$. O espaço necessário ao relógio binário é também da ordem de $\mathcal{O}(f(n))$. Assim,

$$\text{space}_D(n) = \mathcal{O}(f(n)) + \mathcal{O}(f(n)) + \mathcal{O}(f(n)) = \mathcal{O}(f(n)).$$

Consequentemente, $L \in \mathbf{SPACE}(f(n))$.

Para terminar, mostra-se que se M é uma máquina de Turing classificadora tal que $\text{space}_M(n) = o(f(n))$, então $L_{\text{ac}}(M) \neq L$. Com efeito, suponha-se, por absurdo, que existe uma máquina M tal que $\text{space}_M(n) = o(f(n))$ que decide L, e tome-se k suficientemente grande tal que $\text{space}_M(k) \leq f(k)$. Então, com $t = k - |M| - 1$ (ou seja, $k = |M\$1^t|$) tem-se que $M\$1^t \in L$ se e só se M aceita $M\$1^t$ (em espaço menor ou igual a $f(k)$) se e só se $M\$1^t \notin L$, o que é uma contradição. □

Este resultado permite-nos separar classes de complexidade temporais.

Corolário 5.24.
Sejam $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ tais que $f_2(n)$ é construtível no espaço e $f_1(n) = o(f_2(n))$. Então

$$\mathbf{SPACE}(f_1(n)) \subsetneq \mathbf{SPACE}(f_2(n)).$$

Corolário 5.25.
PSPACE $\neq$ EXPSPACE.

Dem.: Deixa-se como exercício verificar que $\log(n) = \mathcal{O}(2^n)$. Por outro lado, para o *input* 1^n o *output* pretendido é 10^n, e a função que a 1^n faz corresponder 10^n é computável em tempo linear, logo também em tempo $\mathcal{O}(2^n)$. Consequentemente, a função $f(n) = 2^n$ é construtível no espaço. Pelo Teorema 5.23, existe $L \in \mathbf{SPACE}(2^n) \subseteq \mathbf{EXPSPACE}$ que não pode ser decidida por nenhuma máquina cujo espaço seja $o(2^n)$. Resta verificar que $n^c = o(2^n)$ para qualquer $c \in \mathbb{N}$: usando sucessivamente a regra de l'Hôpital, tem-se

$$\lim_{n\to\infty} \frac{n^c}{2^n} = \lim_{n\to\infty} \frac{c.n^{c-1}}{\log(2).2^n} = \lim_{n\to\infty} \frac{c(c-1).n^{c-2}}{\log(2)^2.2^n} = \ldots$$

$$= \lim_{n\to\infty} \frac{c!}{\log(2)^c.2^n} = \frac{c!}{\log(2)^c}. \lim_{n\to\infty} \frac{1}{2^n} = 0.$$

Conclui-se que **PSPACE** $\neq$ **EXPSPACE**, pois L está em **EXPSPACE**, mas não em **PSPACE**. □

Definição 5.26. FUNÇÃO CONSTRUTÍVEL NO TEMPO
Uma função $t : \mathbb{N} \to \mathbb{N}$ diz-se *construtível no tempo* se $n\log(n) = \mathcal{O}(t(n))$ e se a função que mapeia palavras 1^n para a representação binária de $t(n)$ é computável em tempo $\mathcal{O}(t(n))$. △

Todas as funções mais comuns que são pelo menos $\mathcal{O}(n\log(n))$, como, por exemplo, $n\log(n)$, n^2 e 2^n, são construtíveis no tempo.

Teorema 5.27.
Seja $t : \mathbb{N} \to \mathbb{N}$ uma função construtível no tempo. Então existe uma linguagem decidível em tempo $\mathcal{O}(t(n))$ mas não em tempo $o(\frac{t(n)}{\log(t(n))})$.

Este resultado permite-nos separar classes de complexidade temporais.

Corolário 5.28.
Sejam $t_1, t_2 : \mathbb{N} \to \mathbb{N}$ tais que $t_2(n)$ é construtível no tempo e $t_1(n) = o(\frac{t_2(n)}{\log(t_2(n))})$. Então

$$\mathbf{TIME}(t_1(n)) \subsetneq \mathbf{TIME}(t_2(n)).$$

Corolário 5.29.
P $\neq$ **EXPTIME**.

Dem.: Deixa-se como exercício verificar que $n\log(n) = \mathcal{O}(2^n)$. Por outro lado, para o *input* 1^n o *output* pretendido é 10^n, e a função que a 1^n faz corresponder 10^n é computável em tempo linear, logo também em tempo $\mathcal{O}(2^n)$. Logo, a função $f(n) = 2^n$ é construtível no tempo. Pelo Teorema 5.27, existe $L \in \mathbf{TIME}(2^n) \subseteq \mathbf{EXPTIME}$ que não pode ser decidida por nenhuma máquina cujo tempo seja $o(2^n/\log(2^n))$. Resta verificar que $n^c = o(2^n/\log(2^n))$ para qualquer $c \in \mathbb{N}$: usando sucessivamente a regra de l'Hôpital, tem-se

$$\lim_{n\to\infty} \frac{n^c\log(2^n)}{2^n} = \lim_{n\to\infty} \frac{n^c\log(2)\log_2(2^n)}{2^n} = \lim_{n\to\infty} \frac{n^{c+1}\log(2)}{2^n} = \lim_{n\to\infty} \frac{(c+1)n^c}{2^n}$$

$$= \lim_{n\to\infty} \frac{(c+1)cn^{c-1}}{\log(2)2^n} = \cdots = \lim_{n\to\infty} \frac{(c+1)!}{\log(2)^c.2^n} = \frac{(c+1)!}{\log(2)^c}. \lim_{n\to\infty} \frac{1}{2^n} = 0.$$

Assim, **P** $\neq$ **EXPTIME**, pois L está em **EXPTIME** mas não em **P**. □

5.7 P versus NP

Sabemos então que **P** $\subseteq$ **NP** $\subseteq$ **PSPACE** $\subseteq$ **EXPTIME** e também que, de facto, **P** $\neq$ **EXPTIME**. Isto implica que pelo menos uma das inclusões acima é também estrita, isto é, que **P** $\neq$ **NP** ou que **NP** $\neq$ **PSPACE** ou que **PSPACE** $\neq$ **EXPTIME**, mas os resultados da secção anterior não são aplicáveis na separação destas classes. Todas estas questões são problemas em

aberto, mas o problema **P** versus **NP** é certamente o mais famoso e importante, por estar relacionado com problemas que são extraordinariamente importantes na prática, em todas as áreas de aplicação em ciência e tecnologia, para os quais não se conhecem algoritmos eficientes. É claro que se um dia se demonstrar que $\mathbf{P} \neq \mathbf{NP}$, como é usual conjecturar, nada de substancial mudará. Mas um prova de $\mathbf{P} = \mathbf{NP}$ traria potencialmente uma enorme revolução na forma como nos relacionamos com o mundo. Para compreender isto, vale a pena compreender melhor a estrutura dos problemas **NP**.

Antes de mais, é útil compreender que é possível, e certamente mais simples e intuitivo, definir a classe **NP** sem recorrer a máquinas não-deterministas.

Proposição 5.30.
Sejam Σ um alfabeto e L uma linguagem sobre Σ. Então, $L \in \mathbf{NP}$ se e só se existe uma linguagem $R \in \mathbf{P}$ e um polinómio p tal que, qualquer que seja $x \in \Sigma^*$, $x \in L$ precisamente se $x\$y \in R$ para algum y com $|y| \leq p(|x|)$.

Esta caracterização é particularmente interessante e informativa: o problema de decidir se um problema x tem solução está em **NP** precisamente quando, dada uma possível solução y (usualmente denominada de *testemunha* ou *certificado*), existe um algoritmo polinomial para verificar se a solução y para x está ou não correcta.

Com esta caracterização é fácil de perceber que há uma miríade de problemas, muito relevantes na prática em todas as áreas da ciência e tecnologia, que têm esta estrutura. A solução do problema passa por procurar um espaço exponencial de possíveis soluções e testá-las (eficientemente) uma a uma. Isto é essencialmente o que conseguimos na Proposição 5.16 onde simulámos deterministicamente uma máquina não-determinista, e a que é usual chamar de *força bruta* ou *perebor* (como ficou conhecida a técnica em trabalhos científicos na antiga União Soviética precursores da moderna teoria da complexidade).

Exemplos de problemas em **NP**:

- dados $n, k \in \mathbb{N}$, determinar se n tem (ou não) um divisor entre 2 e k (*factorização*);
- dados $n_1, n_2, \ldots, n_k, s \in \mathbb{N}$, determinar se existe $I \subseteq \{1, \ldots, k\}$ tal que $\sum_{i \in I} n_i = s$ (SUBSETSUM);
- dada uma fórmula proposicional na forma normal conjuntiva, determinar se é ou não satisfatível (SAT);
- dado um grafo G e $k \in \mathbb{N}$, determinar se G admite uma coloração dos nós com k cores de modo a que não existam vértices adjacentes com a mesma cor (*coloração de grafos*).

O problema da factorização é fundamental na criptografia moderna, e o facto de não ser conhecido nenhum algoritmo eficiente para o resolver é o pilar fundamental da segurança de informação e comunicações. Os problemas da coloração de grafos, SUBSETSUM e SAT têm aplicações em diversos problemas de optimização. Há inúmeros outros problemas relevantes na classe **NP**. No

entanto, nem todos têm o mesmo estatuto. Desde logo porque obviamente $\mathbf{P} \subseteq \mathbf{NP}$ e portanto há algum problemas "fáceis" em **NP**. Mas, além disso, há problemas mais difíceis que outros. O problema da factorização, apesar da sua importância, não é dos mais difíceis. Já os problemas da coloração de grafos, SUBSETSUM e SAT são dos mais difíceis, e na verdade igualmente difíceis. Ainda assim, SAT é um problema especial, por razões históricas, já que foi o primeiro destes problemas a ser identificado e estudado, e continua até hoje a ser objecto de intensa investigação. Tornemos agora estas ideias mais rigorosas.

Definição 5.31. DIFICULDADE, COMPLETUDE
Seja $\mathcal{C}$ uma classe de complexidade. Uma linguagem A diz-se:

- $\mathcal{C}$*-difícil* se qualquer que seja $L \in \mathcal{C}$ se tem $L \leq_P A$;
- $\mathcal{C}$*-completa* se é $\mathcal{C}$-difícil e $A \in \mathcal{C}$. △

À luz da Proposição 5.19, uma linguagem é $\mathcal{C}$-difícil quando a existência de um algoritmo eficiente para decidir essa linguagem implica a existência de algoritmos eficientes para decidir todas as linguagens da classe $\mathcal{C}$. Neste sentido, uma linguagem $\mathcal{C}$-completa é o protótipo da dificuldade inerente à resolução de todos os problemas da classe. Os problemas da coloração de grafos, SUBSETSUM e SAT são todos **NP**-completos. O resultado fundador da moderna teoria da complexidade é precisamente o aclamado Teorema de Cook e Levin (1971) publicado em [7], onde se estabelece que SAT é **NP**-completo. A partir deste resultado, é relativamente simples obter a **NP**-completude de outros problemas.

Proposição 5.32.
Sejam $\mathcal{C}$ uma classe de complexidade e A uma linguagem $\mathcal{C}$-completa. Se $L \in \mathcal{C}$ então L é $\mathcal{C}$-completa se e só se $A \leq_P L$.

Dem.: Exercício. □

Formalizamos o problema SUBSETSUM (em binário[4]) da seguinte forma:

$$\{w_1\$w_2\$\ldots\$w_k\$u : w_1, \ldots, w_k, u \in \{0,1\}^* \text{ e}$$
$$\text{existe } I \subseteq \{1,\ldots,k\} \text{ tal que } \textstyle\sum_{i\in I} w_i = u\}.$$

Proposição 5.33.
O problema SUBSETSUM é **NP***-completo.*

Dem.: Recorde o problema SAT (ver Exercício 5.5 do Capítulo 3). A máquina não-determinista óbvia para decidir SUBSETSUM (ver Exercício 5.3 do Capítulo 3) tem tempo polinomial, pelo que o problema está em **NP**. Basta demonstrar que SAT $\leq_P$ SUBSETSUM. Vamos apenas esboçar a ideia da transformação.

[4]O problema análogo em unário (ver Exercício 5.3 do Capítulo 3) também está em **NP** mas não é **NP**-completo. Isto deve-se ao facto de a representação de um número n em unário ser exponencialmente maior que a sua representação em binário (ou noutra base $b \geq 2$).

	$p \vee q \vee r$	$\overline{p} \vee \overline{q} \vee s$	$\overline{p} \vee s \vee \overline{r}$	p	q	r	s
p	01	00	00	1	0	0	0
$\overline{p}$	00	01	01	1	0	0	0
q	01	00	00	0	1	0	0
$\overline{q}$	00	01	00	0	1	0	0
r	01	00	00	0	0	1	0
$\overline{r}$	00	00	01	0	0	1	0
s	00	01	01	0	0	0	1
$\overline{s}$	00	00	00	0	0	0	1
$1 : p \vee q \vee r$	01	00	00	0	0	0	0
$2 : p \vee q \vee r$	01	00	00	0	0	0	0
$1 : \overline{p} \vee \overline{q} \vee s$	00	01	00	0	0	0	0
$2 : \overline{p} \vee \overline{q} \vee s$	00	01	00	0	0	0	0
$1 : \overline{p} \vee s \vee \overline{r}$	00	00	01	0	0	0	0
$2 : \overline{p} \vee s \vee \overline{r}$	00	00	01	0	0	0	0
objectivo	11	11	11	1	1	1	1

Figura 5.2

Considere-se a fórmula (em forma normal conjuntiva)

$$(p \vee q \vee r) \wedge (\overline{p} \vee \overline{q} \vee s) \wedge (\overline{p} \vee s \vee \overline{r})$$

com 3 cláusulas (a cláusula $p \vee q \vee r$, a cláusula $\overline{p} \vee \overline{q} \vee s$ e a cláusula $\overline{p} \vee s \vee \overline{r}$) e 4 variáveis proposicionais (p, q, r e s). Criamos o problema SUBSETSUM dado pelos números em binário representados pelas linhas da tabela da Figura 5.2, que é construída como se segue: considera-se uma coluna por cada cláusula, uma coluna por cada variável proposicional, uma linha por cada literal (variável proposicional ou sua negação), e duas linhas por cada cláusula; nas posições da tabela que têm uma cláusula na coluna e um literal na linha escreve-se 01 se o literal ocorre na cláusula, e 00 em caso contrário; nas posições que têm cláusulas na coluna e na linha escreve-se 01 se as cláusulas são iguais, e 00 em caso contrário; nas posições que têm uma variável proposicional na coluna escreve-se 1 se na linha está essa variável ou a sua negação, e 0 em caso contrário; por fim, considera-se uma última linha, e escreve-se 3 (em binário) nas colunas correspondentes às cláusulas, e 1 na restantes.

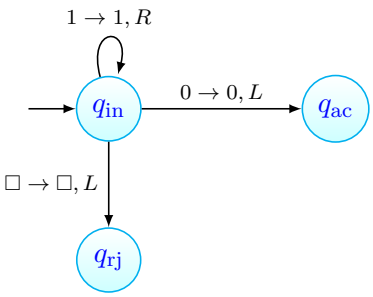
Este problema SUBSETSUM tem solução se e só se a fórmula original é satisfatível. Mais ainda, cada solução para a satisfatibilidade da fórmula dá origem a uma solução do problema SUBSETSUM também. A fórmula em causa é de facto satisfatível. Deixa-se como exercício a verificação destas afirmações. Em particular, note-se que, quando a fórmula é satisfatível, o subconjunto na solução do problema SUBSETSUM vai inclui as linhas relativas aos literais que são satisfeitos e, eventualmente, linhas relativas a cláiusulas (para garantir que se obtém 3 nas colunas correspondentes): se apenas um literal da cláusula é satisfeito, ambas as linhas relativas a essa cláusula são incluídas; se são dois, só é incluída uma linha; se são três nenhuma é incluída.

Note-se ainda que a tabela tem tamanho polinomial: são $2v + 2c$ números com $c + v$ bits onde c é o número de cláusulas e v o número de variáveis da fórmula (no caso $c = 3$ e $v = 4$). □

Exercícios

1 Eficiência de máquinas

1. Considere a seguinte máquina de Turing M com alfabeto de entrada/saída $\{0,1\}$ que decide a linguagem $L = \{w \in \{0,1\}^* : w \text{ tem pelo menos um } 0\}$:



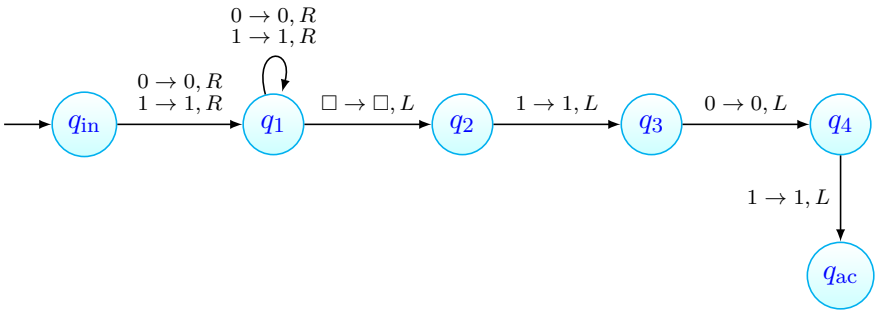
   (a) Calcule o tempo (ou complexidade temporal) time_M de M, e caracterize o seu comportamento assintótico indicando uma função g tal que $\text{time}_M(n) = \mathcal{O}(g(n))$.

   (b) Calcule o espaço (ou complexidade espacial) space_M de M, e caracterize o seu comportamento assintótico indicando uma função g tal que $\text{space}_M(n) = \mathcal{O}(g(n))$.

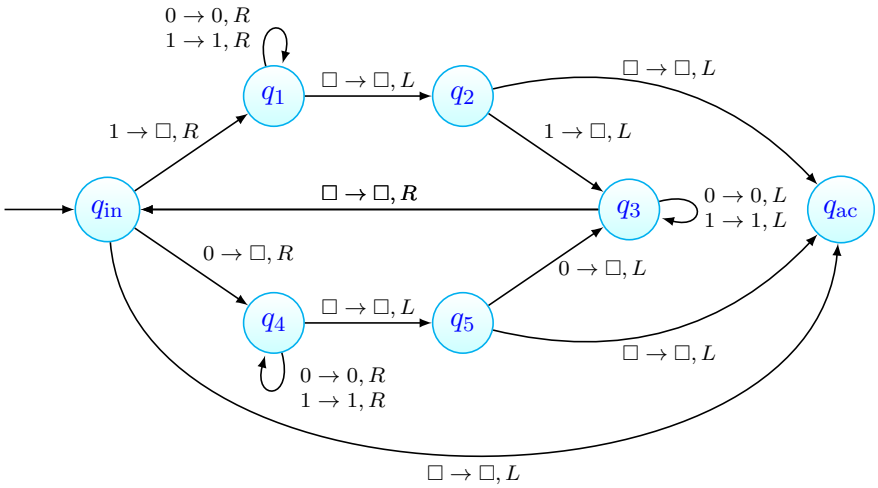
   (c) Tendo em conta as alíneas anteriores indique classes de linguagens $\mathbf{TIME}(t(n))$ e $\mathbf{SPACE}(s(n))$ apropriadas tais que $L \in \mathbf{TIME}(t(n))$ e $L \in \mathbf{SPACE}(s(n))$.

   (d) Verifica-se $L \in \mathbf{P}$? E $L \in \mathbf{PSPACE}$?

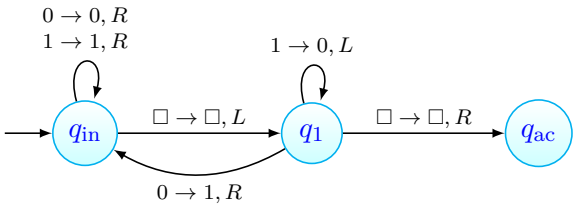
2. Repita o Exercício 1.1 considerando agora a seguinte máquina de Turing classificadora M com alfabeto de entrada/saída $\{0,1\}$ que decide a linguagem $L = \{w \in \{0,1\}^* : w \text{ termina em } 101\}$ (omitem-se as transições para q_{rj}):



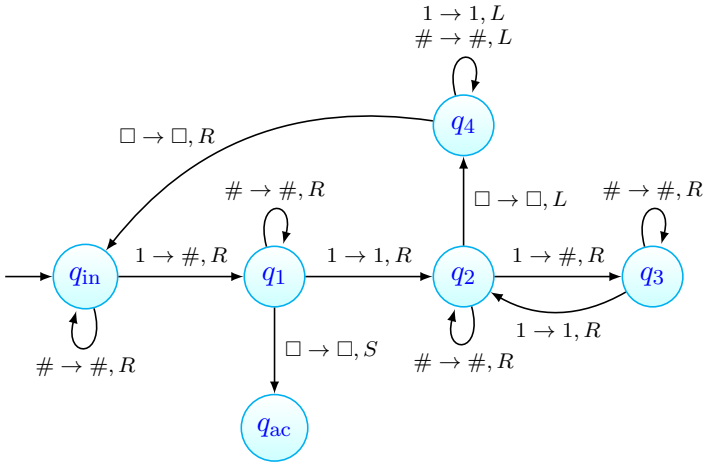
3. Repita o Exercício 1.1 considerando agora a seguinte máquina de Turing classificadora M com alfabeto de entrada/saída $\{0,1\}$ que decide a linguagem L dos palíndromos em $\{0,1\}^*$ (omitem-se as transições para q_{rj}):



4. Repita o Exercício 1.1 considerando agora a seguinte máquina de Turing classificadora M com alfabeto de entrada/saída $\{0, 1\}$:

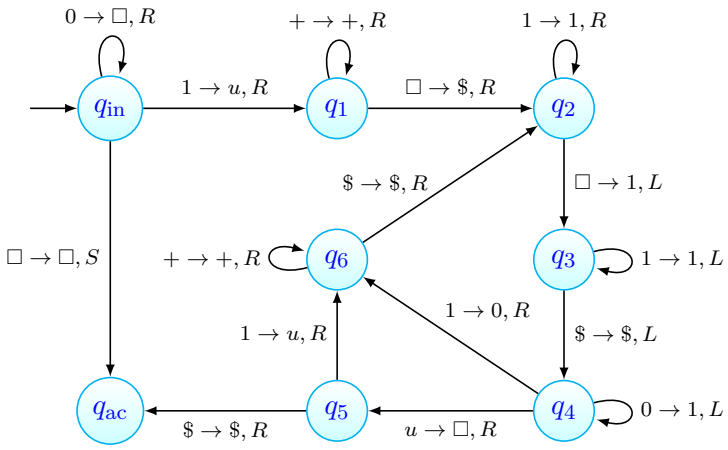


5. A seguinte máquina de Turing classificadora M, com alfabeto de entrada/saída $\{1\}$, decide a linguagem do Exercício 2.9 do Capítulo 2, isto é, a linguagem $L = \{1^n : n \text{ é uma potência de } 2\}$ (omitem-se as transições para q_{rj}):



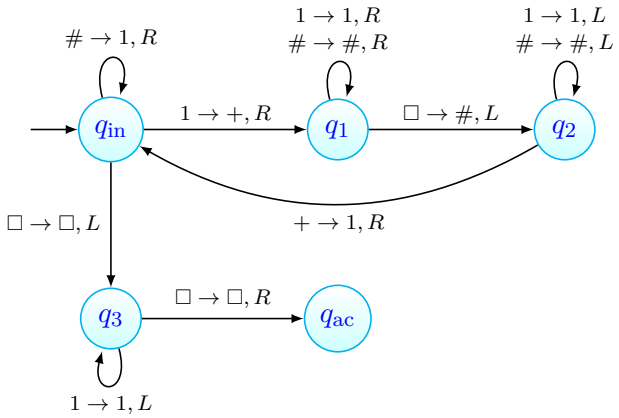
Estude a complexidade computacional de M no tempo e no espaço, isto é, estude o comportamento assintótico de time_M e space_M.

6. Considere a seguinte máquina de Turing classificadora M que calcula a função do Exercício 3.16 do Capítulo 2, isto é, a função que converte a representação em notação binária de um número natural na correspondente representação em notação unária (omitem-se as transições para q_{rj}):



Estude a complexidade computacional de M no tempo e no espaço, isto é, estude o comportamento assintótico de time_M e space_M.

7. Considere a seguinte máquina de Turing classificadora M que calcula a função *dobro*, isto é, a função que a cada número natural em notação unária faz corresponder o seu dobro (omitem-se as transições para q_{rj}):



Mostre que a função *dobro* pertence a **PF** e a **PSPACEF**.

8. Considere a linguagem $L = \{0^n 1^n : n \in \mathbb{N}_0\}$ sobre o alfabeto $\{0, 1\}$ (Exercício 2.1e do Capítulo 2). Mostre que $L \in \mathbf{TIME}(n \log(n))$.

9. Considere a linguagem $L = \{w\#w : w \in \{1\}^*\}$ sobre o alfabeto $\{1, \#\}$. Mostre que $L \in \mathbf{TIME}(n \log(n))$.

10. Considere a linguagem $L=\{1^n : n \text{ é uma potência de } 2\}$ sobre o alfabeto $\{1\}$ (Exercício 2.9 do Capítulo 2). Mostre que $L \in \mathbf{TIME}(n \log(n))$.

11. Recorde a linguagem S descrita no Exercício 5.5 do Capítulo 2. Mostre que $S \in \mathbf{NTIME}(n^2)$.

12. Recorde a função que converte a representação em notação binária de um número natural na correspondente representação em notação unária (Exercício 3.16 do Capítulo 2). Mostre que existe uma máquina de Turing M que calcula esta função tal que $\text{time}_M(n) = \mathcal{O}(2^n)$.

2 Classes de complexidade

1. Mostre que:

   (a) $\mathbf{P} \subseteq \mathbf{PSPACE}$;

   (b) $\mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$;

   (c) $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE} \subseteq \mathbf{EXPTIME}$.

2. Seja $f : \mathbb{N}_0 \to [1, +\infty[$. Mostre que $\mathbf{TIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$.

3. Seja Σ um alfabeto. Sejam $f : \Sigma^* \to \Sigma^*$ uma função total em **PF** e x um elemento do contradomíno de f. Demonstre que a linguagem $A = \{y \in \Sigma^* : f(y) = x\}$ pertence à classe **P**.

4. Seja Σ um alfabeto. Mostre que a linguagem

   $$L = \{N\$w : N \in \mathcal{M}^\Sigma \text{ aceita } w \in \Sigma^* \text{ em, no máximo, } 2^{|w|} \text{ transições}\}$$

   não está em **P**. Sugestão: faça uma demonstração por absurdo, começando por concluir que se $L \in \mathbf{P}$ então $L' \in \mathbf{P}$ onde $L' = \{N : N \notin \mathcal{M}^\Sigma \text{ ou } N \in \mathcal{M}^\Sigma \text{ não aceita } N \text{ em, no máximo, } 2^{|N|} \text{ transições}\}$, e notando depois que se obtém uma contradição quando se analisa a evolução de D' com *input* D', onde D' é um classificador para L' com tempo polinomial.

5. Considere a classe de complexidade $\mathbf{coNP} = \{L : \overline{L} \in \mathbf{NP}\}$.

   (a) Mostre que $\mathbf{P} \subseteq \mathbf{coNP} \subseteq \mathbf{PSPACE}$.

   (b) Mostre que se $\mathbf{P} = \mathbf{NP}$ então $\mathbf{coNP} = \mathbf{NP}$.

3 Propriedades de fecho e redução polinomial

1. Mostre que a classe **P** é fechada para as seguintes operações sobre linguagens:

   (a) intersecção; (b) reunião; (c) complementação; (d) diferença.

   Uma classe $\mathcal{C}$ de linguagens diz-se *fechada para a intersecção* se dadas duas linguagens $L_1, L_2 \in \mathcal{C}$ se tem que $L_1 \cap L_2 \in \mathcal{C}$. As noções de classe fechada para a reunião, diferença e para a complementação são semelhantes.

2. Mostre que a classe **PSPACE** é fechada para as seguintes operações sobre linguagens:

   (a) intersecção; (b) reunião; (c) complementação; (d) diferença.

3. Sejam L_1 e L_2 linguagens sobre um alfabeto Σ. Mostre que se $L_1 \leq_P L_2$ então $\overline{L_1} \leq_P \overline{L_2}$.

4. Sejam L_1, L_2 e L_3 linguagens sobre um alfabeto Σ. Mostre que se $L_1 \leq_P L_2$ e $L_2 \leq_P L_3$ então $L_1 \leq_P L_3$.

5. Sejam A e B linguagens sobre um alfabeto Σ.

   (a) Mostre que se $A \leq_P B$ e $B \in \mathbf{P}$ então $A \in \mathbf{P}$.
   (b) Mostre que se $A \leq_P B$ e $A \notin \mathbf{P}$ então $B \notin \mathbf{P}$.
   (c) Repita as alíneas (a) e (b) mas agora para a classe **PSPACE**.
   (d) Repita as alíneas (a) e (b) mas agora para a classe **NP**.

6. Considere a linguagem S sobre o alfabeto $\{0, 1, 2\}$ constituída por todas as palavras da forma $0^n 1^n 2^n$ com $n \in \mathbb{N}_0$.

   (a) Demonstre que $S \in \mathbf{SPACE}(n) \cap \mathbf{TIME}(n \log(n))$.
   (b) Seja L uma linguagem sobre o alfabeto $\{0, 1, 2\}$. Use o resultado da alínea (a) para demonstrar que se $L \leq_P S$ então $\overline{L} \in \mathbf{P}$.

7. Considere a linguagem S sobre o alfabeto $\{1, \$\}$ formada por todas as palavras da forma

   $$w_1 \$ w_2 \$ \ldots \$ w_k$$

   onde $k \in \mathbb{N}$ e $w_1, \ldots, w_k \in \{1\}^*$, para as quais existe $I \subseteq \{1, \ldots, k\}$ tal que $\sum_{i \in I} w_i = \sum_{i \notin I} w_i$, sendo as somas tomadas sobre a representação em unário de naturais. Por exemplo, tem-se $111\$1\$11111\$1\$1111 \in S$ (com $I = \{1, 5\}$) pois verifica-se $w_1 + w_5 = w_2 + w_3 + w_4$, isto é $3+4 = 1+5+1$. Por outro lado, $111\$1111\$11 \notin S$.

   (a) Demonstre que $S \in \mathbf{NP}$.
   (b) Seja L uma linguagem sobre $\{1, \$\}$. Use a alínea (a) para demonstrar que se $L \leq_P S$ então $L \cup S \in \mathbf{NP}$.

8. Considere a linguagem S sobre o alfabeto $\{1, \$, \#\}$ formada por todas as palavras da forma

   $$u_1 \$ v_1 \# u_2 \$ v_2 \# \ldots \# u_k \$ v_k$$

   onde $k \in \mathbb{N}$ e $u_1, v_1, \ldots, u_k, v_k \in \{1\}^*$, para as quais existe $U \subseteq \{1, \ldots, k\}$ tal que $\sum_{i \in U} u_i + \sum_{i \notin U} v_i = \sum_{i \notin U} u_i + \sum_{i \in U} v_i$, sendo as somas tomadas sobre a representação em unário de naturais. Tem-se $111\$1\#11\$1\#1\$11 \in S$, por exemplo, pois verifica-se $u_1 + v_2 + u_3 = v_1 + u_2 + v_3$, isto é $3+1+1 = 1+2+2$. Por outro lado, $111\$1111\#11\$11 \notin S$.

   (a) Demonstre que $S \in \mathbf{NTIME}(n \log(n))$.
   (b) Seja L uma linguagem sobre $\{1, \$, \#\}$. Use a alínea (a) para demonstrar que se $L \leq_P S$ então $L \cap S \in \mathbf{NP}$.

4 Teorema de Savitch e teoremas de hierarquia

1. Mostre que as funções $\log(n)$, $n\log(n)$, n, n^2 e 2^n são construtíveis no espaço.
2. Mostre que as funções $n\log(n)$, n^2 e 2^n são construtíveis no tempo.
3. Sejam $c_1, c_2 \in \mathbb{N}$ tais que $c_2 > c_1$. Mostre que:
   (a) $\mathbf{TIME}(n^{c_1}) \subsetneq \mathbf{TIME}(n^{c_2})$;
   (b) $\mathbf{SPACE}(n^{c_1}) \subsetneq \mathbf{SPACE}(n^{c_2})$.
4. Mostre que:
   (a) $\mathbf{NPSPACE} \subseteq \bigcup_{k\in\mathbb{N}} \mathbf{SPACE}(n^{2k})$;
   (b) $\mathbf{NTIME}(n^2) \subsetneq \mathbf{SPACE}(n^8)$.
5. Mostre que:
   (a) $\mathbf{TIME}(2^n) = \mathbf{TIME}(2^{n+1})$;
   (b) $\mathbf{TIME}(2^n) \subsetneq \mathbf{TIME}(2^{2n})$;
   (c) $\mathbf{NTIME}(n) \subsetneq \mathbf{PSPACE}$.
6. Mostre que:
   (a) $\mathbf{P} \subsetneq \mathbf{EXPTIME}$;
   (b) $\mathbf{PSPACE} \subsetneq \mathbf{EXPSPACE}$.
7. **NL** é a classe das linguagens que são decidíveis por uma máquina de Turing não-determinista em espaço logarítmico, i.e., $\mathbf{NL} = \mathbf{NSPACE}(\log n)$. Mostre que:
   (a) $\mathbf{NL} \subseteq \mathbf{P}$;
   (b) $\mathbf{NL} \subsetneq \mathbf{PSPACE}$.

5 P versus NP

1. Seja $\mathcal{C}$ uma classe de linguagens sobre um alfabeto Σ e L uma linguagem sobre Σ. Mostre que:
   (a) se L é uma linguagem *$\mathcal{C}$-difícil* e $L \in \mathbf{P}$ então todas as linguagens da classe $\mathcal{C}$ pertencem a **P**;
   (b) se L é uma linguagem *$\mathcal{C}$-difícil* e $L \in \mathbf{PSPACE}$ então todas as linguagens da classe $\mathcal{C}$ pertencem a **PSPACE**.
2. Seja $\mathcal{C}$ uma classe de linguagens sobre um alfabeto Σ e L uma linguagem sobre Σ. Mostre que:
   (a) se L é uma linguagem *$\mathcal{C}$-difícil* e $L \leq_P A$ para alguma linguagem A então A é também $\mathcal{C}$-difícil;

(b) se L é uma linguagem $\mathcal{C}$-*completa*, $A \in \mathcal{C}$ e $L \leq_P A$ então A é também $\mathcal{C}$-completa.

3. Seja $\mathcal{C}$ uma classe de linguagens sobre o alfabeto Σ, fechada para a complementação (isto é, se $A \in \mathcal{C}$ então $\overline{A} \in \mathcal{C}$), e seja $L \in \mathcal{L}^{\Sigma}$. Mostre que se L é $\mathcal{C}$-difícil então $\overline{L}$ é $\mathcal{C}$-difícil.

4. Recorde a linguagem K sobre o alfabeto $\Sigma = \{1, \$, \&\}$ descrita no Exercício 5.4 do Capítulo 2.

   (a) Demonstre que $K \in \mathbf{NTIME}(n^2)$.

   (b) Seja $\mathcal{C} \subseteq \mathcal{L}^{\Sigma}$ e $A \in \mathcal{L}^{\Sigma}$. Use a alínea (a) para demonstrar que se $A \leq_P K$ e $A \cap K$ é $\mathcal{C}$-completa então $\mathcal{C} \subseteq \mathbf{NP}$.

5. O problema $3SAT$ é semelhante ao problema SAT mas em que o conjunto de fórmulas está restringido a fórmulas na forma normal conjuntiva em que cada cláusula tem, no máximo, 3 literais. Mostre que $3SAT$ é **NP**-completo. Sugestão: comece por mostrar que $SAT \leq_P 3SAT$.

Bibliografia

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[2] M. Atallah and M. Blanton, editors. *Algorithms and Theory of Computation Handbook: General Concepts and Techniques*. Chapman & Hall/CRC, 2nd edition, 2010.

[3] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984.

[4] E. Blum and A. Aho. *Computer Science: The Hardware, Software and Heart of It*. Springer-Verlag, 2011.

[5] D. Bridges. *Computability: A Mathematical Sketchbook*. Springer-Verlag, 1994.

[6] J. Brookshear. *Theory of Computation: Formal Languages, Automata, and Complexity*. Benjamin-Cummings, 1989.

[7] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC'71, pages 151–158. ACM, 1971.

[8] R. Epstein and W. Carnielli. *Computability: Computable Functions, Logic, and the Foundations of Mathematics*. Wadsworth, 2nd edition, 2000.

[9] J. Campos Ferreira. *Introdução à Análise Matemática*. Fundação Calouste Gulbenkian, 1987.

[10] L. Fortnow, editor. *The Golden Ticket: P, NP, and the Search for the Impossible*. Princeton University Press, 2013.

[11] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[12] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[13] J. Hein. *Theory of Computation: An Introduction*. Jones and Bartlett, 1996.

[14] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages and Computability.* Addison-Wesley Longman, 2nd edition, 2000.

[15] D. Johnson. A catalog of complexity classes. In van Leeuwen [27], pages 67–161.

[16] D. Kozen. *Theory of Computation.* Texts in Computer Science. Springer-Verlag, 2006.

[17] L. Levin. Universal search problems (in Russian). *Problems of Information Transmission*, 9(3):115–116, 1973. Translated into English by B. Trakhtenbrot in [24].

[18] H. Lewis and C. Papadimitriou. *Elements of the Theory of Computation.* Prentice Hall, 2nd edition, 1997.

[19] A. Maheshwari and M. Smid. *Introduction to the Theory of Computation.* School of Computer Science, Carleton University, Ottawa, Canada, 2014.

[20] Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill, Inc., 1974.

[21] J. Martin. *Introduction to Languages and the Theory of Computation.* McGraw-Hill, 2nd edition, 1997.

[22] H. Rogers. *Theory of Recursive Functions and Effective Computability.* MIT Press, 1987.

[23] M. Sipser. *Introduction to the Theory of Computation.* Thomson Course Technology, 2nd edition, 2006.

[24] B. Trakhtenbrot. A survey of russian approaches to perebor (brute-force search) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

[25] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1937.

[26] P. van Emde Boas. Machine models and simulations. In van Leeuwen [27], pages 1–66.

[27] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity.* MIT Press, 1990.