

# Técnicas Criptográficas

José Manuel E. Valença \*

8 de Junho de 2009

---

\*Departamento de Informática, Universidade do Minho, Campus de Gualtar Braga

# 1.Fundamentos Matemáticos

Neste capítulo introdutório faz-se uma revisão dos fundamentos matemáticos essenciais à generalidade das técnicas criptográficas.

Vamos focar algumas noções de computabilidade, falaremos dos domínios relevantes ao processo computacional e algumas noções de Teoria das Probabilidades. Falaremos de algoritmos e da sua relação com a formalização da segurança das técnicas criptográficas. Iniciamos porém, com alguma notação relevante ao estudo da dimensão dos objectos computacionais com que iremos lidar.

Neste curso  $\mathbb{N}$  designa o domínio dos números naturais.  $\mathbb{B} = \{0, 1\}$  representa o domínio finito dos bits e  $\mathbb{B}^n$  o domínio finito das palavras de  $n$  bits. O único elemento de  $\mathbb{B}^0$  designa-se por **string vazia** e é representado por  $\varepsilon$ .  $\mathbb{B}^*$  e  $\mathbb{B}^\infty$  representam domínios de bit-strings, finitas e infinitas, respectivamente.

Dentro dos reais vamos resignar por  $\mathbb{I}$  o intervalo unitário real  $[0, 1]$  e por  $\mathbb{R}_+ = [0, +\infty]$  a “compactificação” dos reais positivos; isto é, os reais  $\geq 0$  adicionados de um ponto especial  $+\infty$ .

A cada natural  $x$  associamos um **comprimento**  $|x|$  que é o menor número de bits necessários para a representação binária de  $x$ ; isto é  $|x| = \lfloor \log_2(x + 1) \rfloor$ .

## 1.1 Notação Assintótica

Frequentemente lidamos com funções reais de argumento inteiro positivo  $f : \mathbb{N} \rightarrow \mathbb{R}_+$  cujo comportamento não é conhecido com rigor mas que, ainda assim, pode-se ver contido dentro de determinados limites.

Muitas vezes o argumento  $n$  da função é o tamanho de um determinado objecto finito (tipicamente um número natural ou uma string finita) e basta conhecer a “ordem de grandeza” dos resultados  $f(|x|)$  quando  $x$  percorre um qualquer domínio “computável”.

Como estas funções podem ser ilimitadas, o estudo do comportamento destas funções exige que acrescentemos o símbolo  $\infty$  a  $\mathbb{N}$ . Como símbolo  $\infty$  deve verificar

$$\infty \leq \infty \quad \text{e} \quad n < \infty$$

O símbolo é incluído em  $\mathbb{Q}$  e  $\mathbb{R}$  com a imersão de  $\mathbb{N}$  nestes domínios.  $f(\infty) = \infty$  denota uma função ilimitada.

Usaremos o quantificador  $(\exists_{\infty} x)$  para representar a asserção *existe um número infinito de valores  $x$* ;  $(\exists_{\infty})$  também se escreve como **i.o.** (“infinitely often”).

Usaremos  $(\forall_{\infty} x)$  como o quantificador que indica *para todos os valores de  $x$  com um número finito de excepções*; a sua denominação alternativa é **a.e.** (“almost everywhere”).

1 NOÇÃO (NOTAÇÃO  $O$ )

- $f(n) = O(g(n))$  sse existe  $C > 0$  tal que  $\forall_{\infty} n \cdot f(n) \leq C g(n)$
- $f(n) = o(g(n))$  sse  $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$ .
- $f(n) = \Theta(g(n))$  sse  $f(n) = O(g(n)) \wedge g(n) = O(f(n))$
- $f(n) = O^{\log}(g(n))$  sse  $\forall_{\infty} n \cdot f(n) \leq g(n) + O(1)$ .
- $f = \Omega(g)$  é equivalente a  $g = O(f)$ , e  $f = \Omega^{\log}(g)$  é equivalente a  $g = O^{\log}(f)$ .

## Notas

1. A notação pode ser usada para exprimir vários tipos de propriedades das funções  $n \mapsto f(n)$ . Por exemplo  $f(n) = c + o(1)$  é apenas um modo de escrever  $\lim_{n \rightarrow \infty} f(n) = c$ ; ou,  $f(n) = O(1)$  é apenas outro modo de afirmar que  $f$  é uma constante positiva.
2. A comparação de funções usando  $O(\cdot)$  é a mais frequentemente usada. Diz-nos que, em valor absoluto,  $f(n)$  está limitada superiormente por uma função  $C g(n)$  com a “forma” de  $g$ . Diz também que  $|f(n)/g(n)|$  está limitado superiormente à constante  $C$ .  
A comparação  $o(\cdot)$  é mais exigente; diz que este quociente tem de tender para zero. Por isso  $f(n) = o(g(n))$  implica  $f(n) = O(g(n))$  mas a implicação inversa não se verifica necessariamente.



3. Usando a definição de  $\Omega(\cdot)$  a definição de  $\Theta(\cdot)$  equivale à afirmação de que  $|f(n)/g(n)|$  está limitado superiormente por uma constante  $C > 0$  e inferiormente por uma constante  $c > 0$  para todo  $n$  suficientemente grande

$$c|g(n)| \leq |f(n)| \leq C|g(n)| \quad \text{ou} \quad c \leq |f(n)/g(n)| \leq C$$

A figura 1 representa  $f(n) = \Theta(n g(n))$ ; ou seja,

$$c n \leq |f(n)/g(n)| \leq C n$$

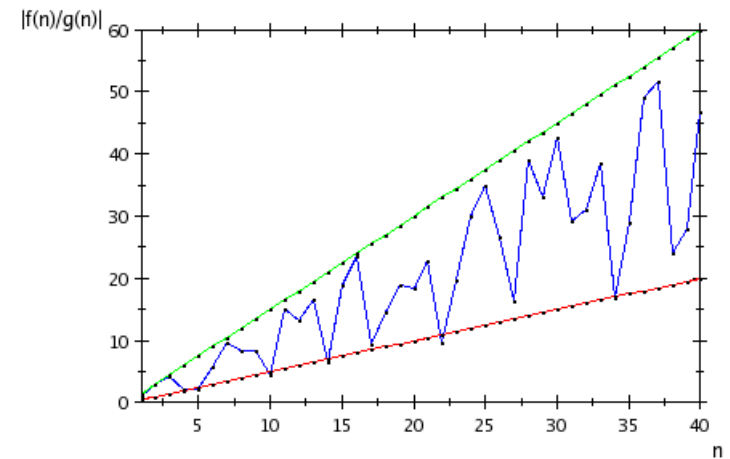


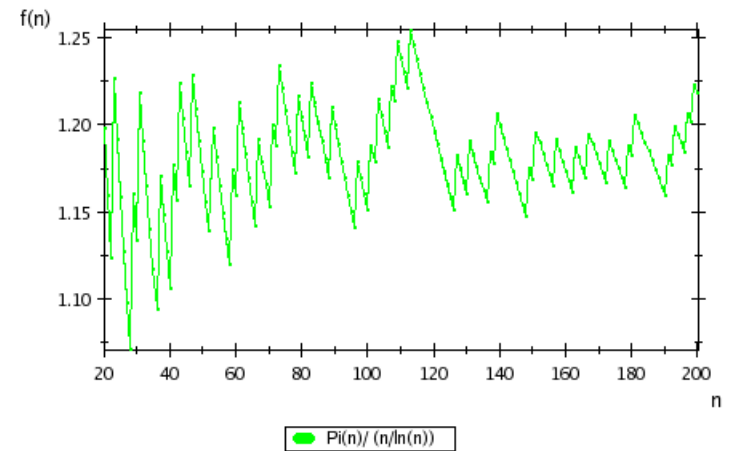
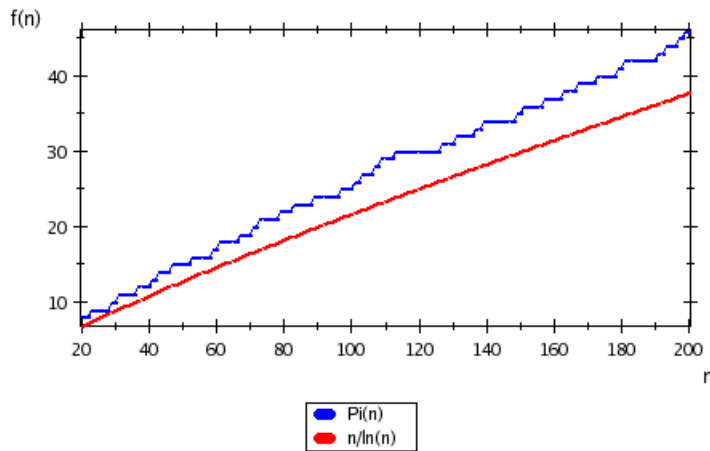
Figura 1:  $f(n) = \Theta(n g(n))$ .

**EXEMPLO 1:** O valor  $\pi(n)$  define-se como *o número de números primos  $\leq n$* . Não é possível construir uma função que calcule  $\pi(n)$  de forma eficiente para todo argumento  $n$  mas é possível aproximar a função  $\pi$  por funções que podem ser calculadas de forma eficiente.

É normal aproximar  $\pi(n)$  pela expressão  $n/\ln n$ . A primeira figura apresenta a variação destas duas funções



( $\pi(\cdot)$  e a sua aproximação) na gama  $n = 20..200$ .



A segunda figura apresenta o quociente  $\frac{\pi(n)}{n/\ln n}$ . Note-se que o quociente está limitado acima e abaixo por duas constantes  $\sim 1$ . Por isso faz sentido afirmar que

$$\pi(n) = \Theta(n/\ln n)$$

No estudo dos algoritmos a notação assintótica é usada, principalmente, para caracterizar a sua **complexidade**.



Uma medida de complexidade, frequentemente usada, é o número de *slots* temporais (por exemplo, ciclos de relógio) que uma máquina de referência usa para correr esse algoritmo. Pode-se também medir um número de células de memória usadas nesse processo. No primeiro caso avalia-se a chamada **complexidade temporal** do algoritmo enquanto que no segundo caso avalia-se a **complexidade espacial** do mesmo.

Em qualquer dos casos temos uma função do tipo  $\tau : \mathbb{N} \rightarrow \mathbb{R}_+$  que mede essa complexidade. O argumento da função representa, quase sempre, uma medida da incerteza nos argumentos do algoritmo. Em Criptografia é normal usar-se o número de *bits* que são necessários para determinar o argumento do algoritmo. Deste modo, no estudo da complexidade,  $\tau(n)$  conta o número médio de *slots* temporais ou o número médio de células de memória para um argumento do algoritmo especificado com  $n$  *bits*.<sup>1</sup>

Alguns casos particulares da ordem de uma função  $\tau : \mathbb{N} \rightarrow \mathbb{R}_+$

## 2 NOÇÃO (ORDEM DE COMPLEXIDADE)

Diz-se que  $\tau$  tem **ordem**

- **polinomial** quando  $\tau(n) = O(p(n))$ , para algum polinómio positivo  $p(n)$ <sup>2</sup>,
- **polinomial inversa** quando  $\tau(n)^{-1} = O(p(n))$ , para um polinómio positivo  $p(n)$ <sup>3</sup>,

<sup>1</sup>Em alternativa, a complexidade pode contar o número *máximo* ou o número *mínimo* de *slots* temporais ou unidades de memória.

<sup>2</sup>Como casos particulares temos as ordens **linear** ( $\tau(n) = O(n)$ ), **quadrática** ( $\tau(n) = O(n^2)$ ), **cúbica** ( $\tau(n) = O(n^3)$ ), etc. . .

<sup>3</sup>Versões “inversas” das ordens anteriores: **inversa linear** ( $\tau(n)^{-1} = O(n)$ ), **inversa quadrática** ( $\tau(n)^{-1} = O(n^2)$ ), etc. . .



- **exponencial** quando  $\tau(n)$  não é polinomial,
- **desprezável** quando  $\tau(n)$  não é polinomial inversa,
- **sub-exponencial** quando  $\tau(n) = O(2^{o(n)})$ .

A ordem sub-exponencial é algo intermédio entre a ordem polinomial e exponencial e é frequentemente escrita numa notação específica.

### 3 NOÇÃO

Define-se  $L[p, c](n) = O(2^{cn^p} (\log_2 n)^{1-p})$ , para  $p \in [0, 1]$  e  $c > 0$ .

É fácil verificar que:

1. A ordem  $L[0, c](n) = O(n^c)$  é a ordem polinomial.
2. A ordem  $L[1, c](n) = O(2^{cn})$  é a ordem completamente exponencial.

Assim a ordem  $L[p, c](n)$  aproxima-se da ordem polinomial quanto mais pequeno for  $p$  e aproxima-se da ordem exponencial quanto maior for  $p$  (limitado, obviamente, a 1).

□

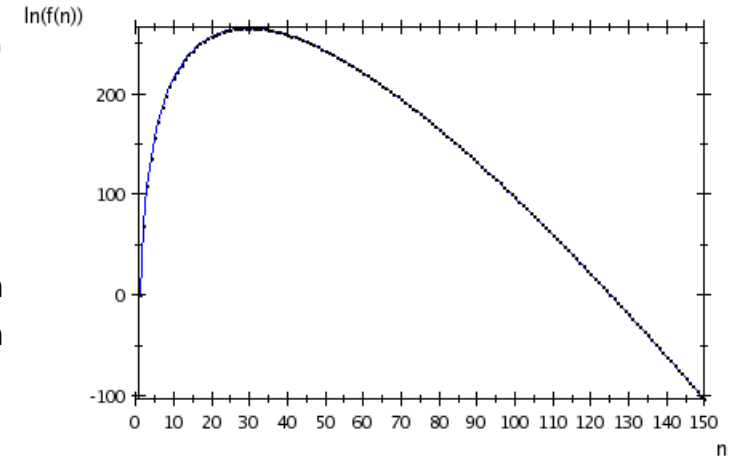


Funções  $p(n)$  da forma  $n^2$ , ou  $1 + n^4 + n^{256}$ , são exemplos de funções **polinomiais positivas**. Funções da forma  $p(n)/q(n)$ , em que  $p(n)$  e  $q(n)$  são polinomiais positivas, dizem-se **racionais positivas**.

Um paradigma de função desprezável é  $\delta(n) = 2^{-q(n)}$ , com  $q(n)$  um p.p. de grau  $> 0$ .

O mesmo se pode dizer da função  $\delta(n) = q(n)/n!$ .

Esta figura ilustra o comportamento de  $n^{100}/n!$  em escala logarítmica; após um máximo perto de  $n = 30$ , o logaritmo da função tende rapidamente para  $-\infty$  o que indica que  $n^{100}/n!$  tende rapidamente para zero.



## 1.2 Funções, Computabilidade e Recursividade<sup>4</sup>

Criptografia lida essencialmente com objectos **computáveis** e, por isso, é muito importante estabelecer, desde o início, o nosso entendimento sobre esses conceitos.

Tradicionalmente associa-se computabilidade de uma função  $f$  à capacidade de uma máquina de Turing simular o seu comportamento. Como vamos lidar, essencialmente, com funções parciais  $f, g: \mathbb{N} \rightarrow \mathbb{N}$  convém introduzir alguma notação prévia.

1.  $f(x) \simeq y$  é um **predicado** que indica que  $f$  é definido em  $x$  e o seu valor é  $y$ .  $f(x) \not\simeq y$  é a sua negação:  $f(x)$  não é definido ou, se for, é diferente de  $y$ .
2.  $f(x)\uparrow$  e  $f(x)\downarrow$  são **predicados** que indicam, respectivamente, que  $f$  não é definido em  $x$  e que  $f$  é definido em  $x$ .
3.  $\text{dom}(f) = \{x \mid f(x)\downarrow\}$  é o **domínio** de  $f$  e  $\text{rng}(f) = \{y \mid \exists x \cdot f(x) \simeq y\}$  é o **contra-domínio** de  $f$ .
4. Se  $\forall x \cdot f(x)\downarrow$  (equivalentemente, quando  $\text{dom}(f) \equiv \mathbb{N}$ ) a função  $f$  diz-se **total**. Funções totais são também designadas por **sequências**.
5.  $Ff$  é equivalente a  $(\forall_{\infty} x)[f(x)\uparrow]$ ; ou seja, é finito o número de argumentos  $x$  onde  $f(x)$  é definido.

<sup>4</sup>Este capítulo e o seguinte (dedicado a conjuntos) deriva essencialmente, e com pequenas adaptações, da obra *Classical Recursion Theory*, P.G. Odifreddi, North-Holland, Volume 1 (1992) e Volume 2 (1999).



6.  $f(x)\Downarrow = (\forall y < x)[f(y)\Downarrow]$  é o predicado que indica que  $f(y)$  é definido para todos os valores  $y < x$ .
7. Funções parciais admitem uma ordem parcial  $\lesssim$  definida por

$$f \lesssim g \quad \text{sse} \quad (\forall x) [f(x)\Downarrow \Rightarrow g(x) \simeq f(x)]$$

A função parcial  $\emptyset$  (também representada por  $\perp$ ) é o mínimo desta ordem parcial; é a função que é indefinida para todo o possível argumento:  $(\forall x \in \mathbb{N}) [\emptyset(x)\uparrow]$ .

8.  $f \simeq g$  compara duas funções; diz-nos que num argumento arbitrário  $x$  ambos os lados ou são ambos indefinidos ou são definidos e têm o mesmo valor. Isto é

$$f \simeq g \quad f \lesssim g \quad \wedge \quad g \lesssim f$$

9. Uma função parcial é uma **função característica** se o seu único valor definido é 0; isto é,

$$f(x)\Downarrow \Rightarrow f(x) \simeq 0$$

No contexto do estudo das funções computáveis identificamos **conjuntos** de  $\mathbb{N}$  com funções características. De facto uma função característica  $f$  determina um conjunto por compreensão  $\{x \mid f(x) \simeq 0\}$ . Dessa forma a classificação que iremos fazer dos conjuntos reflecte a classificação nas funções. Por exemplo diremos que um conjunto pertence a uma classe  $\mathcal{C}$  se e só se a função característica que o determina pertence a essa classe.

10. Uma **relação** é uma função total  $r$  cujos únicos resultados possíveis são 0 e 1. Isto é,

$$(\forall x) [r(x) \simeq 0 \vee r(x) \simeq 1]$$

Cada relação identifica-se também com o conjunto  $\{x \mid r(x) \simeq 1\}$ . A função constante 0 determina o conjunto vazio e, por isso, essa relação é denotada por  $\emptyset$ .

Inversamente cada conjunto  $A$  pode sempre ser visto como uma relação  $r$  tal que  $r(x) \simeq 1 \Leftrightarrow x \in A$ .

11. Um dos problemas essenciais na construção de funções é a representação dos pares de inteiros. Vamos assumir a existência de um operador binário  $\|.\|: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  que codifica pares de inteiros em inteiros e é sobrejectivo. Vamos assumir também que existem duas **funções projecção** sobrejectivas  $\pi_1, \pi_2: \mathbb{N} \rightarrow \mathbb{N}$  que verificam  $\forall x, y, z \in \mathbb{N}$

$$\pi_1(x\|y) \simeq x \quad , \quad \pi_2(x\|y) \simeq y \quad , \quad \pi_1(z)\|\pi_2(z) \simeq z \quad (1)$$

12. O **grafo** de uma função  $f$  é a relação  $r$  que verifica

$$r(x\|y) \simeq 1 \quad \Leftrightarrow \quad f(x) \simeq y \quad (2)$$

Dado que cada relação determina um conjunto por compreensão, cada função  $f$  determina também um conjunto através do seu grafo:  $\{z \mid f(\pi_1(z)) \simeq \pi_2(z)\}$ . Portanto, neste curso, a designação “grafo” refere-se (dependendo do contexto) quer à relação em (126) como o conjunto que lhe está associado.

13. A construção do grafo de uma função  $f$  é uma construção que permite representar qualquer função por uma relação. Inversamente é possível construir uma função recursiva  $f$ , a partir de uma relação  $r$ , que verifique

$$f(x)\downarrow \quad \Leftrightarrow \quad (\exists y) [r(x\|y) \simeq 1] \quad (3)$$

e que tenha  $r$  como grafo; isto é, verifique (126).

Esta construção é designada por **valor mínimo** ou **recursividade**- $\mu$  e a função escreve-se  $f(x) \simeq \mu y \cdot r(x||y)$ . Intuitivamente  $\mu y \cdot r(x||y)$  procura o menor valor de  $y$  que verifica  $r(x||y) \simeq 1$ . Portanto

$$\mu y \cdot r(x||y) \simeq z \Leftrightarrow [r(x||z) \simeq 1] \wedge (\forall y < z) [r(x||y) \simeq 0] \quad (4)$$

□

Diferentes classes de máquinas de Turing cracterizam diferentes noções de computabilidade:

### Máquinas de Turing Determinísticas (DTM's)

Caracterizadas por uma “memória longa” (sequência duplamente infinita de células contendo bits ou marcas que seleccionam “conteúdo”), uma “posição”, um conjunto de “estados” e um conjunto de instruções para, em função do estado e do conteúdo da célula seleccionada pela posição, determina um novo estado e altera memória e posição.

O triplo  $estado \times posição \times conteúdo$  designa-se por **configuração**. A função transição mapeia configurações em configurações. Uma configuração é inicial (ou final) se contém o estado inicial (ou final).

#### 4 NOÇÃO

Uma máquina de Turing  $M$  **para** em  $x$ , e escreve-se  $M(x) \downarrow$ , se partindo de uma configuração inicial com conteúdo  $x$  alcança uma configuração final.

## 5 NOÇÃO

Uma máquina de Turing determinística  $M$  **simula** a função parcial  $f: \mathbb{N} \rightarrow \mathbb{N}$  com complexidade  $T(n), S(n)$  – e escreve-se  $f \lesssim_{T,S} M$  – se, partindo de uma configuração inicial com conteúdo  $x$  onde  $f$  seja definido, se alcança, em não mais de  $T(|x|)$  passos e não ocupando mais do que  $S(|x|)$  células, uma configuração final com conteúdo  $f(x)$ .

A máquina  $M$  **computa**  $f$  – e escreve-se  $M \simeq_{T,S} f$  – se simula  $f$  e, adicionalmente,  $M(x) \downarrow \Rightarrow f(x) \downarrow$ .

Note-se que a definição de simulação nada diz sobre as situações onde  $f(x)$  não é definido; aqui a máquina pode ou não alcançar um estado final. A definição de computação já exige que a máquina pare exactamente para os mesmos argumentos onde  $f$  está definido.

Quando os recursos  $T$  e  $S$  estão implícitos, não são explicitamente colocados na notação e escreve-se simplesmente  $f \lesssim M$  ou  $f \simeq M$ .

### Máquinas de Turing Não-Determinísticas (NDTM)

Com a mesma constituição que as DTM's mas com a opção de, em cada configuração – par  $(estado, posição)$  – haver transições não para uma mas para um número finito de configurações.

A noção de computabilidade é a mesma: partindo de uma configuração com conteúdo  $x$ , a computação termina quando uma das possibilidades de transição atingir o estado final. A máquina é consistente se, nessas circunstâncias, o conteúdo for sempre o mesmo.

### Máquina de Turing Probabilística (PTM)

O resultado de cada transição é uma variável aleatória caracterizada por uma determinada distribuição probabilística. Em cada passo a máquina lê um bit fornecido por uma bit-strings infinita e aleatória; a nova

configuração é função desse bit e da configuração actual.

A noção de computabilidade é agora diferente. A máquina de Turing simula funções “a menos de um erro”.

## 6 NOÇÃO

Uma máquina de Turing probabilística  $M$  **simula** a função  $f: \mathbb{N} \rightarrow \mathbb{N}$  com complexidade  $T(n), S(n), \varepsilon(n)$  se, partindo de uma configuração inicial e com conteúdo  $x$  arbitrário no domínio de  $f$ , se alcança, em não mais de  $T(|x|)$  passos, não ocupando mais do que  $S(|x|)$  células e com uma probabilidade de falha que não excede  $\varepsilon(|x|)$ , uma configuração final de conteúdo  $f(x)$ . A máquina  $M$  **computa**  $f$  se simula  $f$  e pára em  $x$  só se  $f$  converge em  $x$ .

Em função do tipo de máquina seleccionada e dos recursos admissíveis (tempo de processamento  $T(n)$ , espaço ocupado  $S(n)$  e margem de erro  $\varepsilon(n)$ ) podem-se definir várias classes de computabilidade. Por exemplo

- TM** classe das funções computáveis por máquinas de Turing (determinísticas ou não-determinísticas) sem quaisquer limitações de recursos.
- PPTM** funções computáveis por máquinas de Turing probabilísticas com margem de erro desprezável.
- PTIME** funções computáveis por máquinas de Turing determinísticas em tempo  $T(n)$  polinomial.
- PPTIME** funções computáveis por máquinas de Turing probabilísticas em tempo  $T(n)$  polinomial e com margem de erro desprezável.
- NPTIME** funções computáveis por máquinas de Turing não-determinísticas em tempo polinomial.

- . . . classes análogas podem-se definir com restrições no espaço em vez do tempo; várias combinações destas classes são possíveis.

Um dos resultados mais importantes da computabilidade é a enumeração das máquinas de Turing e a existência de uma máquina de Turing universal.

Em primeiro lugar pode-se provar que qualquer uma das classes de computabilidade atrás referidas são enumeráveis. Por isso é possível indexar as diferentes máquinas dentro da classe criando uma sequência  $\mathcal{C} = \{M_n\}$  dos seus elementos.

#### 7 TEOREMA

*Seja  $\mathcal{C} = \{M_n\}$  uma enumeração de máquinas de Turing e sejam  $\{f_n\}$  funções parciais que estas máquinas computam:  $M_n \simeq f_n$ . Seja  $f$  a função que verifica  $f(n||x) = f_n(x)$ . Então existe uma máquina de Turing, dita **universal**,  $\mathcal{U}$  tal que  $\mathcal{U} \simeq f$ .*

*Adicionalmente, se cada  $M_n$  computar  $f_n$  em  $T(n)$  passos, a máquina  $\mathcal{U}$  computa  $f$  em  $O(T(n) \log T(n))$  passos.*



Independentemente da classe de funções que se considere computáveis, e que depende essencialmente da constituição das máquinas que as simulam, é necessário classificar as funções pela forma como elas próprias são construídas. Surge, assim, toda a importância da noção de **recursividade**.





## 8 NOÇÃO (FUNÇÕES PRIMITIVAS E PRIMITIVAS RECURSIVAS)

São **funções primitivas** a constante 0, a função “sucessor”  $x \mapsto x + 1$ , e as projecções  $\pi_1, \pi_2$ .

Uma função  $f$  é **definida por recursividade primitiva** a partir das funções  $g$  e  $h$  quando, para todo  $x, y \in \mathbb{N}$ ,

$$f(x\|0) \simeq g(x) \quad , \quad f(x\|y + 1) \simeq h(x\|y\|f(x\|y)) \quad (5)$$

Uma função  $f$  diz-se **primitiva recursiva** se é uma função primitiva, se é a composição de duas outras funções primitivas recursivas ou se é definida por recursividade primitiva a partir de funções  $g, h$  primitivas recursivas.

Funções primitivas recursivas que têm só dois resultados possíveis, 0 e 1, designam-se por **relações primitivas recursivas**.

Como primeiro resultado das funções primitivas recursivas, tem-se

## 9 FACTO

A função  $f$  é primitiva recursiva primitiva se e só se a sua **função história**  $\hat{f}$ , que verifica para todo  $x \in \varpi$ ,

$$\hat{f}(x\|0) \simeq f(x\|0) \quad , \quad \hat{f}(x\|y + 1) \simeq f(x\|y + 1) \parallel \hat{f}(x\|y) \quad (6)$$

for primitiva recursiva.



É necessário uma construção adicional para ser possível definir completamente a noção de recursividade. Para isso usamos o chamado **princípio do número mínimo** que, essencialmente diz,

*Se uma propriedade  $P$  ocorre em algum  $x$  então existe um único  $y \leq x$  tal que  $P$  ocorre em  $y$  e não ocorre em nenhum  $z < y$ .*

$$P(x) \Rightarrow (\exists_1 y \leq x)[P(y) \wedge (\forall z < y) \neg P(z)] \quad (7)$$

Denota-se por  $\mu P$  (ou, mais explicitamente,  $\mu x \cdot P(x)$ ) o valor  $y$  que verifica (7). Caso não exista nenhum  $x$  onde a propriedade  $P(x)$  ocorra diz-se que  $\mu P$  é indefinido e representa-se esse facto por  $\mu P \uparrow$ . Note-se que a capacidade para determinar esse valor  $\mu P$  depende crucialmente do facto de a propriedade ser sempre definida pelo menos até ao valor  $y$ .

Uma algoritmo possível para determinar  $\mu P$  consistiria em percorrer todos os números naturais (desde o 0) até encontrar o primeiro  $y$  onde  $P$  ocorra. Porém de  $P$  for indefinido num valor  $z < y$ , como é computacionalmente indicível verificar se  $f(z)$  é definido, nunca seria possível ao algoritmo alcançar o valor  $y$ . Portanto o algoritmo só pode funcionar se  $P$  for definido para todos os  $z < y$ .

Como caso particular de propriedades considere-se, para qualquer função parcial  $f$  e qualquer natural  $l$ ,

$$\mathcal{Z}[f](x) = f(x) \Downarrow \wedge f(x) \simeq 0 \quad (8)$$

$$\mathcal{Z}[l, f](x) = x \leq l \wedge \mathcal{Z}[f](x) \quad (9)$$

## Notas

1.  $\mathcal{Z}$  e  $\mathcal{Z}[l, \cdot]$  mapeiam uma função parcial arbitrária  $f$  em predicados. Como predicados são casos especiais de funções parciais,  $\mathcal{Z}$  é uma “função de ordem superior” dado transformar funções em funções.

No contexto da teoria da recursividade, estes objectos de ordem superior designam-se por **funcionais**. Enquanto as funções actuam sobre naturais e dão resultados naturais, as funcionais actuam sobre funções (ou predicados, ou conjuntos, etc.) e dão, como resultado, objectos com esta mesma natureza.

2.  $\mathcal{Z}[l, f]$  é uma “variante limitada” de  $\mathcal{Z}[f]$ . A diferença fundamental entre elas está na decidibilidade de predicados do tipo  $(\exists x)[\mathcal{Z}[f](x)]$ . Normalmente este tipo de fórmulas não são decidíveis mesmo no caso simples em que  $f$  é uma função recursiva primitiva.

Sendo  $f$  definido por recursividade primitiva, será uma função total; o predicado  $(\exists x)[\mathcal{Z}[f](x)]$  é, simplesmente,  $(\exists x)[f(x) = 0]$ . Se for válido é computável verificar esse facto; no entanto, se não for válido, já não é computável verificá-lo.

Já  $(\exists x)[\mathcal{Z}[l, f](x)]$  coincide com  $(\exists x \leq l)[f(x) = 0]$ ; por isso é sempre computável, quer seja válido ou não.

## 10 NOÇÃO (RECURSIVIDADE)

A função parcial  $h$  define-se por **recursividade- $\mu$  não-limitada**, a partir da função parcial  $f$  se

$$h(y) \simeq \mu \mathcal{Z}[f_y] \quad \text{em que} \quad f_y(x) = f(y||x) \quad (10)$$

$h$  define-se por **recursividade limitada- $\mu$**  se, para algum  $l$ , em (117)  $\mathcal{Z}[f]$  é substituído por  $\mathcal{Z}[l, f]$ .

A classe das **funções parciais recursivas**, representada por  $\mathcal{RP}$ , é a menor classe que contém as funções primitivas e é fechada por composição, recursividade primitiva e recursividade- $\mu$  não-limitada.



Dito de outra forma, uma função  $f$  é **parcial recursiva** se e só se cai dentro de uma das quatro possibilidades enunciadas: (1) é uma função primitiva, (2) ou é a composição de duas funções recursivas, (3) ou é obtida por recursividade primitiva a partir de duas funções recursivas, (4) ou é obtida por recursividade- $\mu$  não-limitada a partir de uma função recursiva.

## 11 NOÇÃO

1. Funções parciais recursivas  $f$  cujo domínio é finito (isto é, onde se verifica  $F f$ ) designam-se por **funções finitas**.
2. Funções parciais recursivas cujo único valor definido é 0 designam-se por **funções características**.
3. As funções  $f \in \mathcal{RP}$  que são totais (isto é, verificam  $(\forall x)[f(x) \downarrow]$ ) designam-se simplesmente por **recursivas**. A respectiva classe é representada por  $\mathcal{R}$ .
4. As funções recursivas que têm apenas resultados 0 ou 1 designam-se por **relações recursivas**. A sua classe designa-se por  $\mathcal{RS}$ <sup>5</sup>.

Relativa a uma relação recursiva arbitrária  $\rho$  usa-se a seguinte notação:

- (i) os predicados  $[\rho(x) \simeq 1]$  e  $[\rho(x) \simeq 0]$  escrevem-se, respectivamente, como  $\rho(x)$  e como  $\neg\rho(x)$ .
- (ii)  $\mu\rho$  é uma abreviatura de  $\mu[\rho(x) \simeq 1]$ .

---

<sup>5</sup>De "recursive set".

(iii)  $\rho^+$  e  $\rho^-$  denotam as funções parciais cujo único valor definido é 0 e que verificam

$$\rho^+(x)\downarrow \Leftrightarrow \rho(x) \quad , \quad \rho^-(x)\downarrow \Leftrightarrow \neg\rho(x) \quad (11)$$

É simples provar que ambas são funções parciais recursivas.

Uma propriedade essencial destas sub-classe de funções:

#### 12 FACTO

*A sub-classe das funções características é fechada por composição, recursividade primitiva e recursividade- $\mu$  não limitada. A sub-classe das relações recursivas é fechada por composição e recursividade primitiva.*



A recursividade traduz uma definição de funções pela forma como são construídas. As máquinas de Turing representam funções pela forma como são computadas. Um dos resultados fundamentais da Matemática liga estes dois conceitos.

#### 13 TEOREMA (COMPUTABILIDADE BÁSICA)

*As classe  $\mathcal{R}$  e TM coincidem.*

Essencialmente o teorema diz-nos que as funções parciais recursivas são precisamente as mesmas funções que são

computáveis por máquina de Turing não-probabilísticas sem restrições de recursos. Para vários outros modelos da computação (autómatos,  $\lambda$ -Calculus, etc.) é possível definir equivalências análogas.

No entanto estes resultados dizem pouco sobre a verdadeira natureza das funções computáveis; isto porque as máquinas de Turing sem restrições de recursos são modelos pouco realistas da computação; normalmente “computabilidade” subentende “computabilidade efectiva”, no sentido em que só são interessantes os modelos computacionais de dispositivos realistas que possam computar resultados com recursos realistas. Por isso saber que uma função é recursiva pouco diz sobre a sua aptidão para ser simulada por um tal dispositivo. A afirmação *se a função é recursiva é efectivamente computável* é, desta forma, falsa; existem muitas funções recursivas que não são efectivamente computáveis.

O inverso, dizer que *toda a função efectivamente computável é recursiva* ou, equivalentemente, *toda a função que não é recursiva não pode ser efectivamente computável* é a famosa **tese de Turing**.

Esta é uma questão bastante mais complexa, que não pode ser colocada ao nível exclusivo da ciência matemática porque está fortemente ligada à epistemologia (e mesmo à psicologia): o que é saber calcular, que percepção se tem dos possíveis resultados do cálculo, etc. Por isso está, obviamente, fora do contexto deste curso.

Voltamos, assim, à nossa questão inicial: quais são as funções que devemos considerar “computáveis”? Neste curso vamos seguir a abordagem normalmente assumida em Criptografia e, a menos que seja explicitamente estipulado em contrário, vamos convencionar

### Funções Computáveis

São **efectivamente computáveis** as funções **PPTM**; isto é, as funções parciais computadas, em tempo polinomial e com margem de erro desprezável, por uma máquina de Turing probabilística.

□

### Enumeração e Normalização das Funções Parciais Recursivas

Dado que a classe  $\mathcal{RP}$  das funções parciais recursivas é definida indutivamente a partir de um número finito de constantes e um número finito de regras, é sempre possível codificar uma função num número inteiro seguindo, simplesmente, a forma indutiva da sua construção. Assim, a construção das funções p.r. reflecte-se numa codificação  $\mathcal{I}: \mathcal{R} \rightarrow \mathbb{N}$ ; a aplicação  $\mathcal{I}$  codifica a função  $f$  num inteiro  $p$  a que chamaremos **índice** (ou **programa**) de  $f$ .

O essencial desta codificação é a sua capacidade em recuperar qualquer função, a partir do seu programa, usando um “interpretador universal”. Em primeiro lugar para as funções recursivas

#### 14 PROPOSIÇÃO (AVALIAÇÃO DAS FUNÇÕES RECURSIVAS)

Existe uma avaliação recursiva  $\varphi: \mathbb{N} \rightarrow \mathbb{N}$  tal que, para todo  $f \in \mathcal{R}$  e todo  $p \in \mathbb{N}$

$$\mathcal{I}(f) = p \quad \iff \quad (\forall x) [ \varphi(p||x) \simeq f(x) ] \quad (12)$$

A função recursiva  $\varphi$  é um “interpretador universal”<sup>6</sup> das funções totais recursivas: recebe o programa  $p$  e o argumento  $x$  e produz o mesmo resultado  $f(x)$ .

O conhecimento da função  $f$  permite obter o programa  $p$ . Porém não é possível decidir recursivamente se um inteiro arbitrário  $p$  é um programa para alguma função total; isto é, a fórmula  $(\forall p \in \mathbb{N})(\exists f \in \mathcal{R})[\mathcal{I}(f) \simeq p]$  não é computavelmente decidível.

Esta assimetria é levantada quando se considera funções parciais. Agora qualquer inteiro poderá ser programa para uma função parcial mesma que ela seja a função trivial  $\emptyset$ . Adicionalmente a função avaliação  $\varphi$  (que é recursiva) pode ser normalizada para uma construção que usa duas funções primitivas recursivas.

#### 15 TEOREMA (NORMALIZAÇÃO DAS FUNÇÕES PARCIAIS RECURSIVAS)

Existe uma função primitiva recursiva  $\mathcal{U}$ , e uma relação primitiva recursiva  $\mathcal{S}: \mathbb{N} \rightarrow \mathbb{N}$ , que determinam uma função parcial recursiva  $\varphi$  por

$$\varphi(x) \simeq \mathcal{U}(\mu \mathcal{S}_x) \quad \text{sendo} \quad \mathcal{S}_x(y) = \mathcal{S}(x||y) \quad (13)$$

de tal forma que a sequência  $\{\varphi_n(x) = \varphi(n||x)\}$  enumera  $\mathcal{RP}$ .

Dizer que  $\{\varphi_n\}$  enumera  $\mathcal{RP}$  significa que toda a função parcial recursiva  $f$  está associada a um programa  $n$  tal que  $f \simeq \varphi_n$ . Inversamente, cada programa  $n$  determina a sua função p.r.  $\varphi_n$ .

<sup>6</sup>Os programadores de linguagens funcionais reconhecem em  $\varphi$  a função `eval` do LISP ou o operador `$` do HASKELL.



## 16 NOÇÃO

Dois programas  $p, q$  são **extensionalmente equivalentes**, e escreve-se  $p \cong q$ , quando  $\varphi_p \simeq \varphi_q$ .

Comparando com a proposição 14 nota-se uma restrição importante: a “avaliação”  $\varphi$  tem, aqui, uma forma particular determinada por duas funções que são totais e primitivas recursivas.

Como primeira consequência, a propriedade  $\mathcal{Z}[\mathcal{S}_x](y)$  simplifica-se no simples teste  $\mathcal{S}(x||y) = 0$  e  $\varphi(x)$  será definido se e só se  $(\exists y) [\mathcal{S}(x||y) = 0]$ . A forma genérica de  $\varphi$  diz-nos que a recursividade- $\mu$  só necessita de ser usada uma vez: todas as funções recursivas podem ser calculadas usando apenas composição e recursividade primitiva e uma única utilização (eventual) da recursividade- $\mu$  não-limitada.

O seguinte resultado é uma consequência simples da normalização.

## 17 FACTO

Se  $f, g$  forem duas funções totais recursivas (a.k.a., “sequências recursivas”), e  $f$  é injectiva, então existe uma função parcial recursiva  $\phi$  tal que  $\text{dom}(\phi) = \text{rng}(f)$  e  $\phi(f(n)) \simeq g(n)$ , para todo  $n$ .

**Prova** Define-se  $\phi(x) = g(\mu n \cdot f(n) \simeq x)$ . É trivial verificar que  $\phi$  satisfaz as propriedades requeridas.

A única utilização da recursividade- $\mu$  não-limitada, definida no teorema 15, sugere uma variante limitada da função  $\varphi$  substituindo a recursividade ilimitada por recursividade limitada.



## 18 NOÇÃO

Sejam  $\mathcal{U}, \mathcal{S}$  as funções primitivas recursivas definidas no teorema 15. Para cada natural  $l$ , a **enumeração das aproximações** é a sequência de funções  $\{\varphi_n^l(x) = \varphi^l(n\|x)\}$  com  $\varphi^l$  definida por

$$\varphi^l(x) \simeq \mathcal{U}(\mu \mathcal{Z}[l, \mathcal{S}_x]) \quad \text{sendo} \quad \mathcal{S}_x(y) = \mathcal{S}(x\|y) \quad (14)$$

As propriedades essenciais da aproximação  $\varphi^l$  são

$$\varphi^l \lesssim \varphi \quad , \quad \varphi^l(x) \simeq \varphi(x) \Leftrightarrow (\exists y < l) \mathcal{S}(x\|y) \quad (15)$$

A consequência mais importante é que  $\varphi^l(x)\downarrow \Leftrightarrow (\exists y < l) \mathcal{S}(x\|y)$  é uma relação recursiva primitiva.

□

A noção de indexação das funções parciais recursivas, funções recursivas ou relações recursivas, pode ser estendida da seguinte forma.

## 19 NOÇÃO (ÍNDICES)

Um **sistema aceitável de índices** em  $\mathcal{RP}$  (ou  $\mathcal{R}$ ) é uma enumeração  $\{\psi_e\}$  de elementos de  $\mathcal{RP}$  (ou elementos de  $\mathcal{R}$ ) que verifica as seguintes propriedades:



- **enumeração**: existe um programa particular  $a$  (de “avaliação”) tal que, para todos  $e, x$ ,  $\psi_a(e||x) \simeq \psi_e(x)$ .
- **parametrização**: existe uma função recursiva  $\eta$  tal que, para todo  $x, y$ ,  $\psi_{\eta(x||e)}(y) \simeq \psi_e(x||y)$ .

**Notas** Num sistema aceitável de índices  $\{\psi_e\}$ , a propriedade de “enumeração” diz-se que a avaliação é um programa  $a$  que recebe, como argumento, o emparelhamento  $(e||x)$  do par *programa+argumento* e reproduz o mesmo resultado que o programa  $e$  origina no argumento  $x$ .

$$\psi(a||(e||x)) \simeq \psi(e||x)$$

A propriedade da “parametrização” diz-nos que é possível decompor um argumento da forma  $(x||y)$  da função  $\psi_e$ , de tal forma que, com esse valor  $x$  e o programa original  $e$ , se constrói um novo programa  $i = \eta(x||e)$  que, aplicado ao argumento que falta  $y$ , dá o resultado original.

$$\psi(\eta(x||e)||y) \simeq \psi_e(x||y)$$

Nomeadamente, combinando estas duas observações, vê-se que  $\psi_e(x) \simeq \psi(a||(e||x)) \simeq \psi_{\eta(e||a)}(x)$ , para todo  $x, e$ . Portanto, para todo  $e$ ,  $\eta(e||a) \cong e$ .

As propriedades essenciais dos sistemas aceitáveis de índices resume-se em:

## 20 TEOREMA

1.  $\{\psi_e\}$  é um sistema aceitável de índices em  $\mathcal{RP}$  se e só se existem funções recursivas  $f$  e  $g$  tais que, para todo  $e$ ,  $\psi_e \simeq \varphi_{f(e)}$  e  $\varphi_e \simeq \psi_{g(e)}$ , em que  $\{\varphi_e\}$  é a enumeração standard das funções parciais recursivas.



2. *Todo o sistema aceitável de índices satisfaz a **propriedade do ponto fixo**: dada uma qualquer função recursiva  $f$  existe um índice  $e$  tal que  $\psi_e \simeq \psi_{f(e)}$ .*

Recordemos que a designação **funcional** está reservada para as aplicações que transformam funções (ou relações, ou conjuntos, etc.) em objectos com a mesma natureza. A indexação vai permitir definir funcionais em  $\mathcal{R}$ , como **funções extensionais** nos índices. Formalmente

## 21 NOÇÃO

Seja  $\mathcal{C} = \{\psi_e\}$  um sistema aceitável de índices. Uma função  $f$  é  **$\mathcal{C}$ -extensional** sse

$$\psi_a \simeq \psi_b \implies \psi_{f(a)} \simeq \psi_{f(b)}$$

Uma funcional  $F$  é  **$\mathcal{C}$ -efectiva** em  $\mathcal{RP}$ , quando existe uma função recursiva  $f$  que é  $\mathcal{C}$ -extensional e verifica

$$F[\psi_e] \simeq \psi_{f(e)}$$

A funcional  $F$  é  **$\mathcal{C}$ -efectiva** em  $\mathcal{R}$ , quando é a restrição a  $\mathcal{R}$  de uma funcional efectiva em  $\mathcal{RP}$ .

Sendo  $F$   $\mathcal{C}$ -efectiva e  $g$  uma qualquer função parcial recursiva, o predicado  $g \in F$  determina se a função  $g$  está no seu contradomínio; isto é,

$$g \in F \iff (\exists e) \{ g \simeq F[\psi_e] \} \iff (\exists e) \{ g \simeq \psi_{f(e)} \}$$



Uma sequência de funcionais efectivas  $\{F_n\}_{n \in \varpi}$  é **uniforme** se existe uma função recursiva extensional  $f$  tal que

$$F_n[\psi_e] \simeq \psi_{f(n||e)}$$

Quando o sistema de índices é a enumeração standard de  $\mathcal{RP}$ , designaremos as funções simplesmente por **extensionais** e as funcionais por **efectivas**. A menos que o sistema de índices não seja explicitamente referido, estas serão as funcionais que iremos usar.

Dado que a enumeração standard de  $\mathcal{RP}$  contém um função universal  $\varphi$ , verifica-se  $\varphi_{f(e)}(x) \simeq \varphi(f(e)||x)$ . Assim, aí  $F$  será uma funcional efectiva quando existe uma função recursiva e extensional  $f$  tal que, para todo  $x, e$ ,

$$F[\varphi_e](x) \simeq \varphi(f(e)||x) \quad (16)$$

Se tivermos uma sequência uniforme de funcionais efectivas  $\{F_n\}_{n \in \varpi}$  será, para todo  $x, e, n$ ,

$$F_n[\varphi_e](x) \simeq \varphi(f(n||e)||x) \quad (17)$$

Funcionais efectivas permitem resolver o problema seguinte: *como caracterizar as classes de funções parciais que possam ser construídas de forma computacionalmente relevante.*

Recordemos que uma função parcial  $u$  é finita se o seu domínio é finito. Obviamente, toda a função finita é parcial recursiva. Interessam, particularmente, as funções finitas  $u$  geradas por funcionais efectivas.



## 22 NOÇÃO (CLASSES EFECTIVAMENTE ENUMERÁVEIS DE FUNÇÕES)

Cada funcional efectiva  $F$  determina uma classe de funções, que designaremos por  $F^\sim$ , formado por todas as funções parciais recursivas que são mais definidas que uma função finita  $u \in F$ . Isto é

$$f \in F^\sim \Leftrightarrow (\exists e) [f \simeq \varphi_e] \wedge (\exists u \in F) [Fu \wedge u \lesssim f] \quad (18)$$

Uma classe de funções  $\mathcal{A}$  diz-se **efectivamente enumerável (e.e.)**<sup>7</sup> se é da forma  $F^\sim$  para alguma funcional efectiva  $F$ .

Uma sequência de classes,  $\{\mathcal{A}_n\}$ , é **uniformemente recursiva** (ou só **uniforme**) quando cada  $\mathcal{A}_n$  é da forma  $F_n^\sim$  para uma sequência uniforme  $\{F_n\}$  de funcionais efectivas.

□

O sistema standard de índices permite definir, também, uma noção de aleatoriedade. A intuição está em procurar, dado um  $y$ , o menor programa  $e$  que, sob um "input" fixo (neste caso, o inteiro 0) calcula  $y$ .

## 23 NOÇÃO

O **índice de Kolmogorov** de  $y$  é o inteiro  $\kappa(y)$  tal que

$$\varphi_{\kappa(y)}(0) \simeq y \quad , \quad \varphi_e(0) \not\simeq y \quad \text{par todo } e < \kappa(y) \quad (19)$$

<sup>7</sup>Também designada por **completamente recursivamente enumerável** ou  $\Sigma_1^0$ -classe.



Existe pelo menos uma função que, no argumento 0, produz  $y$ ; nomeadamente a função  $x \mapsto x + y$ ; assim  $\kappa(y)$  é sempre definido. Por outro lado, seria tentador definir  $\kappa(y)$  por recursividade- $\mu$  a partir da função recursiva parcial  $g_y(e) \simeq \varphi(e||0) - y$ . No entanto a recursividade  $\mu$ , apesar de possível, não produz o resultado pretendido já que  $g_y(e)$  pode não ser definida para valores  $e < \kappa(y)$ . Por isso

## 24 FACTO

*A função  $y \mapsto \kappa(y)$  é total mas não é recursiva.*

## 25 NOÇÃO

*Um inteiro  $y$  diz-se **compressível** quando  $y > \kappa(y)$ . Os inteiros não compressíveis dizem-se  **$\kappa$ -aleatórios**.*

Se virmos  $y$  como o código que representa um objecto num determinado domínio (por exemplo, strings de bits), a ideia de compressibilidade tem a haver com a possibilidade de existir um programa menor do que  $y$  que, aplicado a um *input* standard 0, reproduza o mesmo objecto. A noção de aleatoriedade que está associada à noção de compressibilidade, define como aleatórios precisamente os objectos que não são compressíveis.

## Oráculos

Suponhamos que acrescentamos às constantes da definição usual de recursividade (noção 10, página 18) uma relação  $\Phi$  (função total com valores em  $\{0, 1\}$ ). A classe de funções resultante é representada por  $\mathcal{RP}^\Phi$  e designam-se por funções recursivas parciais com **oráculo**  $\Phi$



## 26 NOÇÃO (RECURSIVIDADE COM ORÁCULOS)

$\mathcal{RP}^\Phi$  é a menor classe de funções que contém  $\Phi$ , as funções primitivas e é fechada por composição, recursividade primitiva e recursividade- $\mu$  ilimitada.

A noção de oráculo deriva, originalmente, das máquinas de Turing. Numa máquina de Turing com oráculo  $\Phi$ , cada passo pode, ou não, recorrer ao oráculo  $\Phi$  fornecendo-lhe o conteúdo da configuração actual para ajudar a determinar a próxima configuração.

Algo semelhante se passa numa máquina probabilística onde cada passo recorre a um bit de informação externo para ajudar a determinar a próxima configuração. Numa máquina com oráculo o bit não é externo mas resulta da aplicação de  $\Phi$  ao conteúdo actual e, principalmente, este recurso não é necessariamente usado em todos os passos: é um recurso cujo uso é controlado.

Esta diferença traduz-se na definição de simulação e computação no caso geral de uma M.T. probabilística.

## 27 NOÇÃO

Uma máquina de Turing com oráculo  $M^\Phi$  **simula** a função  $f$  com complexidade  $T(n)$ ,  $S(n)$ ,  $\varepsilon(n)$ ,  $O(n)$  se, partindo de uma configuração inicial e com conteúdo  $x$  arbitrário no domínio de  $f$ , se alcança, em não mais de  $T(|x|)$  passos, não ocupando mais do que  $S(|x|)$  células, com uma probabilidade de falha que não excede  $\varepsilon(|x|)$  e consultando o oráculo não mais que  $O(|x|)$  vezes, uma configuração final de conteúdo  $f(x)$ . A máquina  $M$  **computa**  $f$  se simula  $f$  e pára em  $x$  só se  $f$  converge em  $x$ .





O uso de oráculos tem uma importância muito grande nas classes de complexidade das funções computáveis. Por exemplo, o seguinte resultado ilustra bem essa importância.

## 28 TEOREMA

*Existem oráculos recursivos  $\Phi$  e  $\Psi$  tais que  $P^\Phi = NP^\Phi$  e  $P^\Psi \neq NP^\Psi$ .*

Como sabemos, a asserção  $P \neq NP$  é uma das grandes questões não decididas da Matemática. Este resultado dá respostas parciais recorrendo a oráculos recursivos.

## 1.3 Conjuntos, classes e sua computabilidade

Como foi referido anteriormente vamos identificar os conjuntos de naturais, para os quais queremos associar propriedades de computabilidade, com domínios de funções características. Note-se que, no contexto deste estudo, funções características são sempre recursivas. As propriedades desses conjuntos são, portanto, as propriedades desta classe de funções.

### 29 PROPOSIÇÃO

*A sub-classe de  $\mathcal{RP}$  formada pelas funções características, é um reticulado distributivo com um limite inferior  $\emptyset$  e limite superior  $\mathbf{0}$ . Os elementos  $f$  que têm complemento  $\bar{f}$  são aqueles para os quais existe um relação recursiva  $\rho$  tal que  $f \simeq \rho^+$  e  $\bar{f} \simeq \rho^-$ .*

**Notas** A função trivial  $\emptyset$  é um caso particular de função característica. A constante  $\mathbf{0}$  é uma outra instância da mesma classe de funções. O domínio da primeira é o conjunto vazio  $\emptyset$  enquanto que o domínio da segunda é todo o conjunto  $\omega$ .

Dadas funções características  $f, g$ , é sempre possível definir funções características “união”  $f \cup g$  e “intersecção”  $f \cap g$  que verificam

$$(f \cup g)(x) \downarrow \Leftrightarrow f(x) \downarrow \vee g(x) \downarrow \quad , \quad (f \cap g)(x) \downarrow \Leftrightarrow f(x) \downarrow \wedge g(x) \downarrow$$

Pode-se pensar numa função parcial  $\bar{f}$ , cujo único resultado definido seja 0, e que verifique

$$\bar{f}(x) \downarrow \Leftrightarrow f(x) \uparrow$$

Esta construção, porém, não gera necessariamente uma função recursiva. Obviamente, se  $\bar{f}$  for recursiva, será uma função característica.



Neste contexto, a caracterização das funções primitivas transfere-se para sub-conjuntos dos naturais.

### 30 NOÇÃO (CONJUNTOS RECURSIVAMENTE ENUMERÁVEIS E CONJUNTOS RECURSIVOS)

Um conjunto  $A$  é **recursivamente enumerável (r.e.)** se é o domínio de uma função característica  $f$ ; neste contexto

$$x \in A \Leftrightarrow f(x) \downarrow$$

Um conjunto  $A$  diz-se **recursivo** quando ele e o seu complemento  $\overline{A}$  são ambos recursivamente enumeráveis. Equivalentemente, quando  $A$  é r.e. e existe uma relação recursiva  $\rho$  tal que  $x \in A \Leftrightarrow \rho(x)$ .

O conjunto recursivo  $A$ , que verifica  $(\forall x)[x \in A]$ , designa-se por  $\varpi$ ; extensionalmente  $\varpi$  coincide com  $\mathbb{N}$ . O conjunto recursivo  $A$ , para o qual o predicado  $(\exists x)[x \in A]$  não é válido, designa-se por  $\emptyset$ .

Existe uma assimetria fundamental entre a verificação da asserção  $x \in A$  quando  $x$  pertence ou não ao conjunto. Se  $A$  for só recursivamente enumerável, verificar  $x \in A$  requer uma quantidade finita de informação enquanto que verificar  $x \notin A$  pode necessitar de informação infinita; isto deriva, mais uma vez, da indecidibilidade da verificação de que uma função é definida num argumento particular. No entanto, se  $A$  for um conjunto recursivo, verificar  $x \in A$  ou  $x \notin A$  requer, em ambos os casos, informação finita.

Uma das primeiras consequências da enumeração das funções recursivas parciais é

### 31 TEOREMA (GRAFOS)

Uma função  $f$  é parcial recursiva se e só o seu grafo for recursivamente enumerável. Uma função  $f$  é recursiva se e só se o seu grafo for recursivo.



Como consequência da proposição 29 e do facto 12 tem-se

32 TEOREMA (CONSISTÊNCIA ESTRUTURAL DOS CONJUNTOS R.E.)

1. Os conjuntos recursivamente enumeráveis formam um reticulado distributivo **ReSet** com o limite superior  $\emptyset$  e limite superior  $\varpi$ . Os conjuntos recursivos **RSet** são, neste reticulado, os elementos que têm complemento (equivalentemente, **RSet** é a maior álgebra booleana contida em **ReSet**).
2. Se  $A$  é recursivamente enumerável e  $f$  é uma função parcial recursiva, então tanto  $f(A)$  como  $f^{-1}(A)$  são recursivamente enumeráveis. Se  $A$  é recursivo e  $f$  é recursivo então  $f^{-1}(A)$  é recursivo.
3. Se  $A$  é finito então é recursivo. Se  $A$  é infinito e recursivamente enumerável, então contém um subconjunto infinito que é recursivo.

O item (2) implica que, sendo  $A$  e  $f$  ambos recursivos, o conjunto  $f^{-1}(A)$  é recursivamente enumerável; no entanto não força ele seja necessariamente recursivo.

Conjuntos, que sendo infinitos, não contêm qualquer sub-conjunto infinito que seja recursivo, dizem-se **imunes**. O item (3) diz-nos que nenhum conjunto recursivamente enumerável é imune. Por isso, conjuntos imunes podem não parecer úteis; porém, como veremos adiante, existem conjuntos imunes extremamente relevantes à Criptografia.

□

A classe  $\mathcal{RP}$  pode, no entanto, ser demasiado lata para caracterizar “computabilidade efectiva”. Por isso é frequente seleccionar uma qualquer classe de funções efectivamente computáveis  $\mathcal{C}$ , tomá-la como referência e considerar

apenas as funções características que caem nessa classe.

Nestas circunstâncias pode-se modificar a classificação de conjuntos e dizer

### 33 NOÇÃO (CONJUNTO COMPUTÁVELMENTE ENUMERÁVEIS E COMPUTÁVEIS)

Um conjunto  $A$  é **computavelmente enumerável (c.e.)** se é o domínio de uma função característica efectivamente computável. O conjunto é **computável** se ele e o seu complemento são computavelmente enumeráveis.

Porém a classe  $\mathcal{C}$  das funções consideradas efectivamente computáveis tem de ser suficientemente lata para que se verifique um resultado de consistência estrutural análogo ao teorema 32. Nomeadamente,  $\mathcal{C}$  tem de conter suficientes elementos para que a classe dos conjuntos computavelmente enumeráveis forme um reticulado distributivo com limite inferior  $\emptyset$  e limite superior  $\mathbf{0}$  que contenha a imagem e a pré-imagem, por uma função em  $\mathcal{C}$ , de qualquer conjunto computavelmente enumerável.

### Diagonalização

Considere-se a enumeração standard das funções parciais recursivas  $\{\varphi_n\}$  introduzida no teorema da normalização (teorema 15, página 23). Seja  $\varphi$  a função universal; isto é a função parcial recursiva tal que, para todo  $x$

$$\varphi_n(x) \simeq \varphi(n||x)$$

Seja  $r$  uma qualquer função recursiva: *como caracterizar o conjunto  $\{\varphi_{r(n)}\}$  das funções enumeradas por  $r$ ?*



Nomeadamente  $x \mapsto \varphi_{r(x)}(x)$ , onde o argumento determina simultaneamente o programa e o argumento ao qual a função é aplicada, é uma função parcial recursiva porque, pela normalização,  $\varphi_{r(x)}(x) \simeq \varphi(r(x)||x)$ . Se esta função é ou não recursiva é um problema que iremos discutir.

Uma outra questão fundamental no estudo dos conjuntos recursivamente enumeráveis: *existe algum conjunto recursivamente enumerável que não seja recursivo?* Veremos, em seguida, que a resposta é afirmativa e que ela se baseia nas propriedades de uma função deste tipo.

Este tipo de questões, e as técnicas a elas associadas, designam-se por **diagonalização** e são essenciais ao estudo de problemas essenciais da computabilidade. Exemplos de técnicas de diagonalização aparecem nos seguintes resultados.

### 34 TEOREMA

1. Não existe nenhuma função recursiva  $r$  que enumere o conjunto das relações recursivas.
2. Existe um conjunto  $\mathcal{K}$ , definido por

$$x \in \mathcal{K} \quad \Leftrightarrow \quad \varphi_x(x) \downarrow$$

que é recursivamente enumerável e não é recursivo.

### Esboço de prova

1. Suponhamos que existia  $r$  recursiva tal que  $x \mapsto \varphi_{r(x)}$  enumerava todas as relações recursivas. Então pode-se definir uma nova relação recursiva  $u(x) \simeq 1 - \varphi_{r(x)}(x) = 1 - \varphi(r(x)||x)$  e, por isso, existe algum  $n$  tal que  $u \simeq \varphi_{r(n)}$ . Obtém-se uma contradição porque, pela definição, tem-se  $u(n) = 1 - \varphi_{r(n)}(n)$  e, pela enumeração, tem-se  $u(n) = \varphi_{r(n)}(n)$ .



2. A “função diagonal”  $\delta: x \mapsto \varphi_x(x) = \varphi(x||x)$  é parcial recursiva; como  $x \in \mathcal{K} \Leftrightarrow \delta(x) \downarrow$  o conjunto  $\mathcal{K}$  é recursivamente enumerável. Não é recursivo porque, se fosse, existiria a função característica do seu complemento e, portanto, um programa  $n$  tal que  $x \notin \mathcal{K} \Leftrightarrow \varphi_n(x) \downarrow$  para todo  $x$ . Particularizando para  $x = n$  resultaria uma contradição

$$n \in \mathcal{K} \Leftrightarrow \varphi_n(n) \downarrow \Leftrightarrow n \notin \mathcal{K}$$

O resultado no item (1) é frequentemente apresentada aproveitando a ligação entre as funções recursivas e as máquinas de Turing e associando a parcialidade das funções ao problema da paragem das máquinas de Turing. Nesse contexto é apresentado como

*Não existe nenhuma máquina de Turing que, sob input  $x$ , decida se a máquina de Turing universal pára nesse input  $x$ .*

□

## Representação

Uma propriedade essencial dos conjuntos está na forma como os seus elementos são construídos como resultados de funções; essa “representação” dos conjuntos é uma consequência imediata do teorema da normalização.

### 35 NOÇÃO (CONJUNTOS REPRESENTÁVEIS)

*O conjunto  $A$  é **recursivamente representável** (ou só, **representável**) se é vazio ou então é o contradomínio de uma função recursiva. Tal função designa-se por **representação** de  $A$ .*



Equivalentemente, um conjunto  $A$  é **representável** se for o contradomínio de uma função parcial recursiva.

Para mostrar a equivalência entre as duas formas partimos de um resultado intermédio

- 36 LEMA *Seja  $f$  uma função parcial recursiva. Então, existe uma função recursiva cujo contradomínio coincide com o contradomínio de  $f$ .*

### Esboço de prova

Seja  $a$  um elemento arbitrário do contradomínio de  $f$  e seja  $e$  um programa, na enumeração standard das funções parciais recursivas, tal que  $f \simeq \varphi_e$ . Defina-se

$$f_a(z) \simeq \begin{cases} y & \text{se } \varphi_e^l(x) \simeq y \\ a & \text{em caso contrário} \end{cases} \quad \text{sendo } \pi_1(z) \simeq l, \pi_2(z) \simeq x$$

Como  $x \mapsto \varphi_e^l(e\|x)\downarrow$  é uma relação primitiva recursiva, esta função é recursiva cujo contra-domínio coincide com o de  $f$ .

Existe agora uma relação básica entre as noções de enumeração recursiva e representatividade.

- 37 TEOREMA (REPRESENTAÇÃO)

1. *Um conjunto  $A$  é recursivamente enumerável sse for recursivamente representável.*
2. *Um conjunto  $A \neq \emptyset$  é recursivo sse for representável por uma função recursiva, monótona e não-decrescente.*

### Esboço de prova





1. Se  $A$  for recursivamente enumerável com a função característica  $c$ , então é o contradomínio da função parcial recursiva  $t(x) \simeq x + c(x)$ . A função  $t$  é parcial recursiva com contradomínio coincidente com  $A$ . Usando o lema 36 constrói-se uma função recursiva  $f$  com o mesmo contradomínio.

Inversamente se  $A$  é o contradomínio de uma função recursiva  $r$ , constrói-se uma função característica  $f$  por

$$f(x) \simeq 0 \cdot \mu y. [r(y) = x]$$

2. Neste caso é simples construir uma função recursiva usando a recursividade- $\mu$  para procurar, por ordem crescente, os elementos do conjunto. A seguinte função é recursiva, é monótona não decrescente e enumera os elementos de  $A$ .

$$r(0) = \mu x. [x \in A] \quad , \quad r(n+1) = \begin{cases} \mu P_n & \text{se } (\exists x) P_n(x) \\ r(n) & \text{em caso contrário} \end{cases}$$

$$\text{sendo } P_n(x) = [x \in A \wedge x > r(n) \wedge x \leq n + r(n)]$$

Note-se que, porque  $(x \in A)$  é uma relação recursiva (já que  $A$  é recursivo), também  $P_n(x)$  e  $(\exists x) P_n(x)$  são relações recursivas.



## Indexação

Tal como para as funções parciais recursivas, é possível indexar os conjuntos recursivamente enumeráveis. Note-se que, dada uma qualquer função parcial recursiva  $\varphi_e$  é possível construir uma função característica  $c_e \simeq 0 \cdot \varphi_e$  que



tem precisamente o mesmo domínio. Por isso é possível ver os conjuntos r.e. como domínios de funções parciais recursivas. Representaremos por  $\mathcal{W}_e$  o domínio da função parcial recursiva de índice  $e$ .

Porém podem existir, normalmente, muitas funções parciais diferentes que têm exactamente o mesmo domínio  $A$ ; como cada uma destas funções tem um índice diferente, a cada conjunto r.e.  $A$  está associado um conjunto de **r.e. índices**. Em termos de informação, basta o conhecimento de um desses índices para determinar o conjunto.

No caso particular em que o conjunto  $A$  é recursivo, e para além dos r.e. índices que possui como qualquer conjunto r.e., está associado aos índices das várias relações recursivas  $\rho$  que o determina; estes são os **índices característicos**.

Finalmente quando  $A$  é um conjunto finito tem, também, um único **índice canónico**  $(A)$  que é o inteiro  $a$  cuja representação binária tem o valor 1 precisamente nos elementos de  $A$ ; isto é,  $(A) = \sum_{i \in A} 2^i$ .

Designaremos por  $D_a$  o conjunto finito cujo índice canónico é  $a$ ; por convenção  $D_0 = \emptyset$ . Portanto  $(D_a) = a$  e  $D_{(A)} = A$ . Designaremos por  $\mathcal{S}_a$  a classe dos superconjuntos de  $D_a$  que são recursivamente enumeráveis

$$\mathcal{W}_x \in \mathcal{S}_a \Leftrightarrow D_a \subseteq \mathcal{W}_x \quad (20)$$

É essencial atender ao facto que  $\mathcal{S}_a$  não contém qualquer superconjunto de  $D_a$  que não seja recursivamente enumerado. Esta classe representa todas as computações que “continuam” o conjunto codificado por  $a$ .

Pode-se estender o conceito de classe de conjuntos representando “continuações computáveis”; para tal, em vez de considerarmos um só índice  $a$ , vamos fazer  $a$  percorrer um conjunto  $A$  que seja recursivamente enumerável.

### 38 NOÇÃO (CLASSE EFECTIVAMENTE ENUMERÁVEL DE CONJUNTOS)

Uma classe de conjuntos recursivamente enumeráveis  $\mathcal{A}$  é **efectivamente enumerável**<sup>8</sup>, se existe um conjunto recursivamente enumerável  $A$ , chamado **representação** de  $\mathcal{A}$ , tal que

$$\mathcal{W}_x \in \mathcal{A} \Leftrightarrow (\exists a) [a \in A \wedge D_a \subseteq \mathcal{W}_x] \quad (21)$$

Equivalentemente, se existe uma funcional efectiva  $F$  tal que

$$\mathcal{W}_x \in \mathcal{A} \Leftrightarrow \varphi_x \in F \quad (22)$$

Uma visão alternativa das classes efectivamente enumeráveis é dada pelo seguinte resultado.

### 39 FACTO

Uma classe de conjuntos recursivamente enumeráveis  $\mathcal{A}$  é efectivamente enumerável se e só se  $\{x \mid \mathcal{W}_x \in \mathcal{A}\}$  é um conjunto recursivamente enumerável.

Uma noção mais fraca é

---

<sup>8</sup>Também designada por **completamente recursivamente enumerável** ou por  $\Sigma_1^0$ -classe.

## 40 NOÇÃO

Uma classe  $\mathcal{A}$  de conjuntos recursivamente enumeráveis diz-se **recursivamente enumerável** se existe uma função recursiva  $r$  que enumera um índice de cada um dos seus elementos; isto é,  $\mathcal{A} = \left\{ \mathcal{W}_{r(e)} \right\}_{e \in \omega}$ .

Em (21), sem perda de generalidade, pode-se considerar que os elementos da representação  $A$  codificam conjuntos finitos disjuntos dois a dois. Isto é,  $A$  verifica

$$(a \in A) \wedge (b \in A) \Rightarrow D_a \cap D_b = \emptyset \quad (23)$$

## 41 NOÇÃO (ARRAY)

Um conjunto recursivamente enumerado  $A$  que verifica (23) designa-se por **array**<sup>9</sup>.

## 42 FACTO

Uma classe é efectivamente enumerado sse existe um array  $A$  que a represente.

□

Existe uma relação forte entre conjuntos finitos e naturais que é estabelecida pela correspondência  $a \leftrightarrow D_a$  dada pela indexação canónica. Isto faz com que algumas noções relevantes a objectos de nível 0 (naturais) se extendam para alguns objectos de nível 1 (conjuntos de naturais). Por exemplo,

<sup>9</sup>No Odifreddi (pag. 228) este conceito é designado por “strong disjoint array”.

## 43 NOÇÃO

Se  $A$  é um conjunto finito então o seu **índice de Kolmogorov**  $\kappa(A)$  é o índice de Kolmogorov do seu índice canónico; isto é,  $\kappa(D_a) \doteq \kappa(a)$ .

$A$  é **compressível** quando o seu índice canónico é compressível e é  **$\kappa$ -aleatório** se não for compressível.

## Conjuntos “quase” recursivos

Vimos que a diferença essencial entre um conjunto recursivo (ou computável) e um conjunto r.e (ou c.e.) é que, no primeiro caso, o predicado  $x \in A$  é recursivamente verificável, quer seja ou não válido, enquanto que, no 2º caso, o predicado só é recursivamente verificável se for válido.

Existem conjuntos que são recursivamente enumeráveis e que, não sendo recursivos, são “quase recursivos”. Temos, pelo menos, duas classes intermédias importantes: a dos **conjuntos semi-recursivos** e a dos **conjuntos simples**.

## 44 NOÇÃO (CONJUNTOS SEMI-RECURSIVOS)

Uma função recursiva  $\sigma$  é uma **função de escolha** se, para todos  $x, y$  se verifica sempre  $\sigma(x||y) = x$  ou  $\sigma(x||y) = y$ . Um conjunto  $A$  é **semi-recursivo** se existe uma função de escolha  $\sigma$  tal que, para todos  $x, y$

$$(x \in A) \Rightarrow \sigma(x||y) \in A \quad e \quad (y \in A) \Rightarrow \sigma(x||y) \in A$$



Note-se que se  $A$  for recursivo, com informação finita, é possível decidir se  $x \in A$  e  $y \in A$  e, desta forma, decidir  $f(x||y) \in A$ . Porém, se  $A$  for só recursivamente enumerável, informação finita pode não ser suficiente para decidir  $x \notin A$  e  $y \notin A$ ; portanto pode não haver informação suficiente para decidir  $f(x||y) \in A$ . Desta forma se  $A$  for recursivo é semi-recursivo mas se for só r.e. pode não ser semi-recursivo. Ao invés, se for semi-recursivo é recursivamente representável.



Outra noção “intermédia” é a de **conjunto simples**. Conjuntos simples ligam-se à noção de compressibilidade e, por isso, à de aleatoriedade. Para se ver como, temos de recorrer a uma outra noção: a de **conjunto retráctil**.

Suponhamos que  $A$  é recursivamente enumerável por uma função total (não necessariamente recursiva) crescente  $r$ ; isto é  $A = \{a_x = r(x)\}_{x \in \omega}$ . O conjunto diz-se **retráctil** se existe uma função parcial recursiva  $\sigma$  tal que

$$\sigma(a_{x+1}) \simeq a_x \quad , \quad \sigma(a_0) \simeq a_0$$

#### 45 FACTO

*Se o conjunto  $A$  é recursivo é retráctil. Se  $A$  e o seu complemento  $\overline{A}$  são retracteis, então  $A$  é recursivo. Se  $A$  é retráctil e não é recursivo, é imune<sup>10</sup>.*

#### 46 NOÇÃO (CONJUNTOS SIMPLES)

*Um conjunto é **simples** se é recursivamente enumerável e o seu complemento é imune.*

---

<sup>10</sup>Recordemos que imunes são os conjuntos infinitos que não contém qualquer subconjunto infinito recursivamente enumerável.



O seguinte resultado exprime um conceito fundamental da aleatoriedade de Kolmogorov

47 TEOREMA (KOLMOGOROV)

*O conjunto dos números  $\kappa$ -compressíveis (não-aleatórios) é simples.*

Isto significa que o conjunto dos números aleatórios é infinito e não contém nenhum sub-conjunto infinito que seja recursivamente enumerável; para além disso, os não-aleatórios formam um conjunto recursivamente enumerável.

### Reducibilidade de Turing

48 NOÇÃO

*Para uma função total  $g$ , a classe de todas as funções **recursivas em  $g$**  é a menor classe de funções que contém  $g$  e é fechada por composição, recursividade primitiva e recursividade  $\mu$  limitada.*

*Se  $f$  é recursivo em  $g$  representamos esse facto por  $f \leq_T g$  e diz-se que é **Turing redutível** a  $g$ . Se  $f \leq_T g$  e  $g \leq_T f$  diz-se que  $f$  e  $g$  são **Turing equivalentes** e escreve-se  $f \equiv_T g$ .*

*As classes de equivalência de funções totais definidas pela relação  $\equiv_T$  designam-se por **graus de Turing**.*

Os graus de Turing colocam na mesma classe as funções que são recursivas em relação umas às outras usando recursividade limitada; portanto, duas funções no mesmo grau podem-se considerar redutíveis uma à outra em termos de complexidade computacional.



O resultado seguinte ajuda a perceber a estrutura destes objectos computacionais.

#### 49 PROPOSIÇÃO

*Cada grau de Turing  $\alpha$  contém pelo menos um conjunto e contém qualquer conjunto recursivo.*

#### Esboço de prova

Dada um representante arbitrário  $f \in \alpha$ , seja  $g$  o seu grafo. Claramente, como  $f$  é uma função total, existe uma construção de  $g$  recursiva em  $f$  e, inversamente, existe uma construção de  $f$  que é recursiva em  $g$  (teorema 31, página 34). Como cada grafo é uma relação e, por isso, determina um conjunto. Por outro lado, são Turing equivalentes a  $f$  todas as funções totais recursivas. Nomeadamente todas as relações recursivas e, deste modo, todos os conjuntos recursivos.

Recordemos o conjunto  $\mathcal{K} = \{x \mid \varphi_x(x) \downarrow\}$  que, como vimos no teorema 34, página 37, é recursivamente enumerável mas não é recursivo. Então

#### 50 TEOREMA (POST)

*Se  $A$  é um conjunto recursivamente enumerável então  $A \leq_T \mathcal{K}$ .*

Se tomarmos os graus que contêm, pelo menos, um conjunto recursivamente enumerável (os chamados **graus recursivamente enumeráveis**) existem dois que têm importância especial:

- (i) o grau representado por  $\mathbf{0}$  que é formado exclusivamente pelas funções recursivas (equivalentemente, pelos conjuntos recursivos). A proposição 49 diz-nos que qualquer grau  $\alpha$  contém  $\mathbf{0}$ .





- (ii) o grau representado por  $\mathbf{0}'$  que é o menor grau que contém o conjunto  $\mathcal{K}$ . Pelo teorema 50 qualquer grau que contenha um conjunto recursivo está contido em  $\mathbf{0}'$ .

## 1.4 Domínios

*...God made the integers, all else is the work of man...*<sup>11</sup>

*...God made the bit-strings, all else is the work of programmers...*<sup>12</sup>

Tomemos por referência uma qualquer classe  $\mathcal{C}$  de funções consideradas como efectivamente computáveis. Tipicamente escolhe-se **PPTIME**.

No contexto deste curso designamos por **domínio** qualquer colecção de objectos que podem ser posto em correspondência biunívoca com um sub-conjunto dos números naturais  $\mathbb{N}$  ou uma sub-classe do respectivo “power-set”  $\wp(\mathbb{N})$ .

### 51 NOÇÃO (DOMÍNIO ENUMERÁVEL)

Um domínio  $X$  que pode ser posto em correspondência biunívoca com um sub-conjunto recursivo  $R \subseteq \mathbb{N}$  designa-se por **domínio numerável**. Se existir uma representação computável  $\psi: \mathbb{N} \rightarrow X$ , o domínio  $X$  diz-se **enumerável**.

Um domínio que está em correspondência biunívoca com todo  $\mathbb{N}$  **identifica-se** com  $\varpi$ .

---

<sup>11</sup>LEOPOLD KRONECKER- 1823/1891

<sup>12</sup>senso comum!



Um sub-conjunto  $A \subseteq X$  diz-se **recursivamente enumerável (recursivo)** se está em correspondência biunívoca com um conjunto recursivamente enumerável (recursivo) de  $\omega$ .

Todo o sub-conjunto  $Y \subseteq \mathbb{N}$  é ordenável pela ordem nele induzido pela ordem natural dos inteiros. Todo o domínio enumerável  $X \cong Y$  é ordenável pela ordem  $\preceq$  nele induzido pela sua correspondência com  $Y$ .

### EXEMPLO 2(ENUMERAÇÕES):

Para definir um domínio enumerável basta identificar o “primeiro” elemento do domínio e o “sucessor” de um elemento genérico  $x \in X$ .

Um domínio enumerável, definido deste modo, é o conjunto  $\mathbb{Q}$  dos números racionais. É bem conhecida a enumeração por “diagonalização” que ao racional  $n/d$  faz suceder o racional  $(n+1)/(d-1)$ , caso  $d > 1$ , ou o racional  $1/(n+1)$  caso  $d = 1$ .

A enumeração dos pares de naturais  $\mathbb{N} \times \mathbb{N}$  é análoga à dos racionais: o primeiro elemento do domínio é o par  $(0, 0)$  e a função “sucessor” mapeia o par  $(x, y)$  no par  $(x+1, y-1)$ , caso seja  $y > 0$ , ou no par  $(1, x+1)$  caso seja  $y = 0$ .

## 52 NOÇÃO (REPRESENTAÇÃO E CODIFICAÇÃO)

Se  $X$  é um qualquer domínio enumerável, uma função sobrejectiva  $r: \mathbb{N} \rightarrow X$  é uma **representação** de  $X$ .



A família de conjuntos  $\{r^{-1}(x)\}_{x \in X}$  é uma partição de  $\mathbb{N}$ ; i.e. os conjuntos são mutuamente disjuntos e cobrem todo o domínio. Os inteiros  $n \in r^{-1}(x)$  designam-se por  **$r$ -códigos** de  $x$ .

Uma função  $s: X \rightarrow \mathbb{N}$  que mapeie cada  $x \in X$  num dos seus  $r$ -códigos<sup>13</sup> designa-se por  **$r$ -codificação** de  $X$ .

### bit-strings

O domínio enumerável que usaremos mais frequentemente é, porém, o domínio das bit-strings finitas  $\mathbb{B}^*$ . Importante (mas não enumerável) é, também, o domínio  $\mathbb{B}^\infty$  das bit-strings infinitas. Notações alternativas para estes domínios são, respectivamente,  $2^{<\omega}$  e  $2^\omega$ .

Apesar de, como veremos em seguida,  $2^{<\omega}$  e  $2^\omega$  se identificarem, respectivamente, com os naturais e os conjuntos de naturais, exigem algumas noções e notações que lhe são específicas.

## 53 NOÇÃO

*Se  $x$  é uma string finita, existe um único  $n$  tal que  $x \in \mathbb{B}^n$ ; esse inteiro é o **comprimento** de  $x$  e é representado por  $|x|$ . Se  $u$  é uma string infinita, convencionou-se que  $|u| = +\infty$ . Se  $u$  é uma string finita ou infinita, para todo  $0 \leq i < |u|$ , a componente de ordem  $i$  de  $u$  representa-se por  $u_i$ .*

<sup>13</sup>Note-se que a codificação é uma “inversa-à-direita” de  $r$ ; isto é verifica-se  $r(c(x)) = x$ , para todo  $x \in X$ . Como consequência a cardinalidade de  $X$  é sempre menor ou igual à cardinalidade de  $\mathbb{N}$ ; donde, nenhum domínio não enumerável pode ser representável.

A **concatenação** de uma string finita  $x$  com uma string finita ou infinita  $v$  é um bit-strings representada por  $xv$  tal que  $|xv| = |x| + |v|$  e

$$(xv)_i = \begin{cases} x_i & \text{se } i < |x| \\ v_{i-|x|} & \text{se } i \geq |x| \end{cases}$$

Representa-se por  $x \leq v$  quando existe uma terceira string  $s$  tal que  $xs = v$ . Neste caso diz-se que  $x$  é um **prefixo** de  $v$ ; um prefixo de comprimento  $n$  diz-se um  **$n$ -prefixo**. Inversamente, dada uma string finita ou infinita  $v$ , representa-se por  $\downarrow v$  o conjunto dos prefixos de  $v$ ; isto é,  $\downarrow u = \{x \mid x \leq u\}$ .

Representa-se por  $\uparrow u$  o conjunto de todas as strings infinitas que têm  $u$  como prefixo.

$$\uparrow x = \{u \in \mathbb{B}^\infty \mid x \leq u\} = \{xv \mid v \in \mathbb{B}^\infty\} \quad (24)$$

Um sub-conjunto  $L \subseteq 2^{< \omega}$  designa-se por **linguagem**. A linguagem  $L$  diz-se **livre de prefixos** quando nenhum dos seus elementos é prefixo de um outro elemento da mesma linguagem. Isto é, para todos  $x \neq y \in L$ , nunca ocorre  $x \leq y$ ; equivalentemente  $\uparrow x$  e  $\uparrow y$  são sempre disjuntos.

Por exemplo, a string finita ou infinita  $v$  determina a linguagem  $\downarrow v$ ; neste caso, a linguagem tem cardinalidade igual ao comprimento da string. Da mesma forma, cada linguagem  $L$  gera uma outra linguagem, representada como  $\downarrow L$ , formada pelos prefixos de elementos de  $L$ ; isto é,  $\downarrow L = \{x \mid \exists y \in L: x \leq y\} = \bigcup_{y \in L} \downarrow y$ .

**Nota**

É elementar provar que, dadas duas linguagens  $L$  e  $H$ , se for  $L \subseteq H$ , então terá de ser  $\downarrow L \subseteq \downarrow H$ . Porém, o inverso não é necessariamente válido: pode ocorrer  $\downarrow L \subseteq \downarrow H$  sem que  $L \subseteq H$  ocorra.

Considere-se, por exemplo, as linguagens  $L = (11)^*$  e  $H = 1(11)^*$ . A primeira é formada por todas as sequências de 1's de comprimento par; a segunda é idêntica mas as strings têm comprimento ímpar. As linguagens são distintas mas têm o mesmo conjunto de prefixos;  $\downarrow L = \downarrow H = 1^*$ .

Se  $L$  é uma linguagem livre de prefixos, representamos por  $\uparrow L$  a colecção de strings infinitas que contêm algum elemento de  $L$  como prefixo; isto é

$$v \in \uparrow L \Leftrightarrow (\exists^1 x \in L) [x \leq v] \quad (25)$$

Note-se que, porque  $L$  é livre de prefixos, os diversos  $\uparrow x$ , com  $x \in L$ , são disjuntos. Assim, cada  $u \in \uparrow L$  tem, quanto muito, um prefixo contido em  $L$ .

## 54 FACTO

Se  $L$  é uma linguagem livre de prefixos então  $\sum_{x \in L} 2^{-|x|} \leq 1$ .

**Esboço de Prova**

Seja  $l_k$  o número de elementos de  $L$  de tamanho  $k$ . O número de strings de comprimento  $n$  que têm um prefixo de tamanho  $k < n$  em  $L$  é, portanto,  $2^{n-k} l_k$ . Portanto  $l_n \leq 2^n - \sum_{k < n} l_k 2^{n-k}$ ; o que implica  $\sum_{k=0}^n l_k 2^{-k} \leq 1$ , independentemente do valor de  $n$ . Tomando o limite  $n \rightarrow \infty$ , e notando que  $\sum_{x \in L} 2^{-|x|} = \sum_{k=0}^{\infty} l_k 2^{-k}$ , conclui-se que  $\sum_{x \in L} 2^{-|x|} \leq 1$ .



## Representação das bit-strings finita

Para vermos que  $\mathbb{B}^*$  é enumerável e identificável com  $\mathbb{N}$  basta representar os elementos deste conjunto numa árvore binária enumerando os seus nodos como se indica na figura 2.

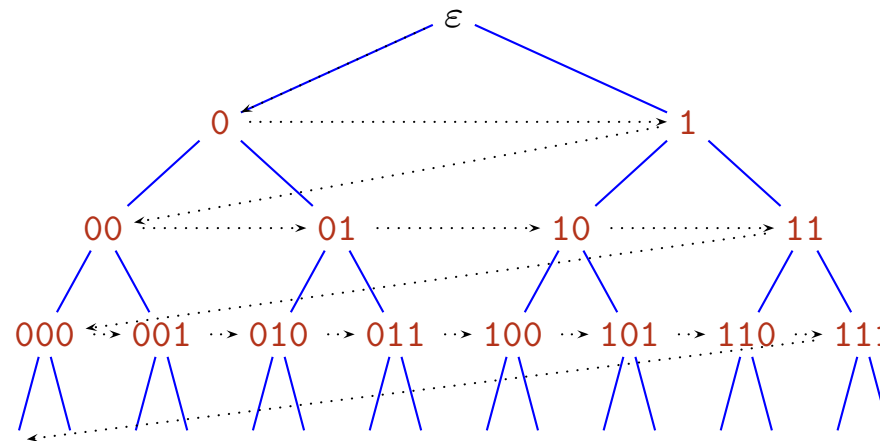


Figura 2: Enumeração das bit-strings finitas.

A função representação  $r: \mathbb{N} \rightarrow \mathbb{B}^*$ , e a respectiva codificação  $s: \mathbb{B}^* \rightarrow \mathbb{N}$ , constroem-se a partir das funções biunívocas  $r_n: [0, 2^n - 1] \rightarrow \mathbb{B}^n$ , que constituem a representação standard em  $n$  bits dos naturais, e as respectivas inversas  $s_n = r_n^{-1}$  que codificam strings finitas em naturais.

Então

$$r(x) = r_{|x|}(x - (2^{|x|} - 1)) \quad , \quad s(u) = (2^{|u|} - 1) + s_{|u|}(u) \quad (26)$$

Considere-se a representação e codificação standard de bit-strings finitas definidas em (26). Dado que  $2^{<\omega}$  e  $\omega$  se identificam, a cada linguagem  $L \subseteq 2^{<\omega}$  corresponde univocamente um conjunto de naturais formado pelos códigos das strings em  $L$ : a cada linguagem  $L$  está associado um único conjunto  $A_L \in 2^\omega$  dada por

$$A_L = \{ s(u) \mid u \in L \}$$

Inversamente, a cada conjunto finito  $A \in 2^\omega$  corresponde uma única linguagem  $L_A \subseteq 2^{<\omega}$  formado pelas bit-strings representadas pelos elementos de  $A$ ; isto é,

$$L_A = \{ r(a) \mid a \in A \}$$

Desta forma o conjunto de todas as linguagens  $2^{2^{<\omega}}$  identifica-se com o conjunto de todos os conjuntos de naturais  $\wp(\mathbb{N})$ , e ambos são designados por  $2^\omega$ .

As caracterizações usuais dos conjuntos de naturais transferem-se directamente para o domínio das linguagens. Por exemplo



## 55 NOÇÃO

A linguagem  $L$  é **recursivamente enumerável** (**recursiva**) se o conjunto  $A_L$  é recursivamente enumerável (recursivo). A linguagem  $L$  é **finita** quando  $A_L$  é finito.  $L$  é **compressível** ( $\kappa$ -aleatório) se é finita e  $A_L$  é compressível ( $\kappa$ -aleatório).

A identificação entre conjuntos de naturais e linguagens pode ser extendida a outras classes de objectos. Nomeadamente

## 56 NOÇÃO

- Cada conjunto  $A$  identifica-se com a relação  $\alpha$  tal que  $\alpha(x) \simeq 1 \Leftrightarrow x \in A$ .
- Cada relação  $\alpha$  identifica-se a bit-strings infinita  $u$  que verifica  $\alpha(x) \simeq 1 \Leftrightarrow u_x = 1$ .
- Cada conjunto  $A$  identifica-se com a string  $u$  que verifica  $x \in A \Leftrightarrow u_x = 1$ .
- Cada linguagem  $L$  identifica-se com a string infinita associada ao conjunto  $A_L$ .

Estes vários domínios representam facetas concretas do domínio abstracto  $2^\omega$ ; para cada uma destas facetas pode-se ver uma noção de “truncatura”.

## 57 NOÇÃO

Dado  $A \in 2^\omega$ , a **truncatura**  $A \upharpoonright n$  denota o prefixo de comprimento  $n$  da string infinita que representa  $A$ .

Para  $x \in \omega$ , o predicado  $(\exists n)[A \upharpoonright n \simeq x]$  é representado por  $x \prec A$ . A classe  $\{A \in 2^\omega \mid x \prec A\}$  é representada por  $\uparrow x$ . O conjunto  $\{x \in \omega \mid x \prec A\}$  é representado por  $\downarrow A$ .



Vendo  $A \in 2^{\omega}$  como string infinita, a truncatura  $A \upharpoonright n \in \omega$  é a string formada pelos primeiros  $n$  bits de  $A$  mas também pode ser vista como o natural  $s(A \upharpoonright n)$  que a codifica. Vendo  $A \in 2^{\omega}$  como um conjunto,  $A \upharpoonright n$  identifica-se também com o sub-conjunto  $\{x \mid x \in A \wedge x \leq n\}$ .

## Reais

Indistintamente  $2^{\omega}$  identifica o domínio das bit-strings infinitas, o domínio dos conjuntos de naturais e o domínio das linguagens. Falta um quarto domínio: o intervalo real unitário  $\mathbb{I} = [0, 1]$ .

Começemos pelas bit-strings finitas e a sua relação com uma determinada classe de números racionais.

### 58 NOÇÃO (RACIONAL DE LEBESGUE)

*Uma string finita  $x \in \mathbb{B}^*$  (equivalentemente, um natural  $x$ ) determina um racional*

$$x^{\sim} = \sum_{k < |x|} x_k 2^{-k-1} \quad (27)$$

*Números racionais desta forma (somas finitas de fracções  $2^{-k}$ ) chamam-se **racionais de Lebesgue**.*

É importante notar que os racionais de Lebesgue são limitados superiormente por 1 e que, se for  $u \leq v$  verifica-se  $u^{\sim} \leq v^{\sim}$ . Portanto a função  $\cdot^{\sim}: \mathbb{B}^* \rightarrow \mathbb{Q}_+$  é monótona crescente e limitada a 1.



A noção de racional de Lebesgue permite construir uma forma “inversa” do facto 54 (página 53). Esse resultado diz-nos que, para uma linguagem livre de prefixos  $L = \{x_n\}$ , a sequência  $\{d_n\}$ , com  $d_n = |x_n|$ , verifica  $\sum_n 2^{-d_n} \leq 1$ . O resultado “inverso” está expresso no seguinte teorema.

59 TEOREMA (LEVIN, SCHNORR, CHAITIN, MILLER)

Seja  $\{d_n\}$  uma sequência computável de naturais tal que  $\sum_n 2^{-d_n} \leq 1$ . Então existe uma sequência computável  $\{x_n\}$ , livre de prefixos, tal que  $|x_n| = d_n$ , para todo  $n$ . Adicionalmente, se  $\{d_n\}$  for recursiva, então  $\{x_n\}$  enumera uma linguagem recursiva livre de prefixos.

### Intuição

A construção da sequência  $\{x_n\}$  é feita via racionais de Lebesgue. Considere-se, para cada  $n$ ,  $z_n \in 2^{<\omega}$  a menor string tal que  $(z_n)^\sim = 1 - \sum_{i \leq n} 2^{-i}$ ; seja  $t_n = |z_n|$ . O último bit de  $z_n$  (i.e.  $(z_n)_{t_n-1}$ ) tem de ser 1 (caso contrário  $z_n$  não seria a menor string que satisfaz esta relação).

Através dos bits dos diversos  $z_n$  que são 1, a prova deste teorema constrói indutivamente uma sequência de linguagens livres de prefixos  $L_0 \subset L_1 \subset \dots \subset L_n \subset \dots$  de tal forma que  $x_i \in L_n$ , para todo  $i \leq n$ .

Tomando agora uma bit-strings infinita  $u$ , pode-se construir uma série de Cauchy no intervalo  $[0, 1]$

$$u^\sim = \sum_{k=0}^{\infty} u_k 2^{-k-1} \quad (28)$$

Note-se que  $u^\sim$  é um real bem definido porque é o limite de uma sequência crescente de racionais limitada por 1. É também o limite dos racionais de Legesgue, determinados pelos prefixos  $u \upharpoonright n$  de  $u$ .

$$u^\sim = \lim_{n \rightarrow \infty} (u \upharpoonright n)^\sim \quad (29)$$

Para simplificar a notação designemos por  $x_n \in 2^{<\omega}$  a string finita  $x_n = u \upharpoonright n$  que se obtém truncando  $u$  a  $n$  bits;  $\downarrow u$  é a linguagem formada por todos estes prefixos. Uma característica da linguagem  $\downarrow u$  é o ser *completamente ordenada*; isto é,  $x, y \in \downarrow u$  implica  $x \leq y$  ou  $y \leq x$ . Se, adicionalmente, a linguagem for recursivamente enumerável, ou recursiva, as diversas truncaturas  $x_n = u \upharpoonright n$  formam “aproximações computáveis” da string  $u$ ; assim,  $u$  e o real de Lebesgue  $u^\sim$  que determina são, de algum modo, “computáveis”.

Por isso faz sentido definir,

#### 60 NOÇÃO (REAL RECURSIVO (R.R.))

Um **real recursivo** é um sub-conjunto recursivo  $R \subseteq \mathbb{Q}_+$  que tem um limite superior. Equivalentemente é o limite de uma função recursiva, limitada e monótona não-decrescente  $r: \mathbb{N} \rightarrow \mathbb{Q}_+$ .

Um **real recursivo de Legesgue (r.L.)** é uma linguagem  $L$  recursiva e totalmente ordenada; isto é,  $x, y \in L$  implica  $x \leq y$  ou  $y \leq x$ .

#### Notas

Na definição de real recursivo, note-se que  $R$  (por ser recursivo) pode ser enumerado por uma função monótona e não decrescente. Esta



função determina uma sequência não-decrescente e limitada de racionais positivos; por definição de real, esta sequência determina um número real.

Claramente um real recursivo de Lebesgue é um real recursivo; como  $L$  é uma linguagem recursiva cujos elementos formam uma ordem total, pode ser enumerada por uma função recursiva, monótona, não-decrescente (teorema 37, página 39). Os elementos de  $L$ , assim enumerado, originam uma sequência não-decrescente de racionais de Lebesgue.

Vimos como associar strings infinitas  $u \in 2^{\omega}$  a reais  $u^{\sim}$  no intervalo unitário  $[0, 1]$ . Como é que esta aplicação  $\cdot^{\sim}: 2^{\omega} \rightarrow [0, 1]$  transforma classes de strings infinitas?

Tomemos, como exemplo, uma string finita  $x \in 2^{<\omega}$  e a classe  $\uparrow x = \{xv \mid v \in 2^{\omega}\}$  formada por todas as strings infinitas  $u = xv$  que têm  $x$  como prefixo. Pela definição (28) tem-se

$$u^{\sim} = x^{\sim} + 2^{-|x|} v^{\sim}$$

Quando  $v$  percorre todo o  $2^{\omega}$ ,  $v^{\sim}$  percorre o intervalo  $[0, 1]$ . Por isso,  $u^{\sim}$  percorre o intervalo  $[x^{\sim}, x^{\sim} + 2^{-|x|}]$ . Isto justifica

#### 61 FACTO

*As classes em  $2^{\omega}$  da forma  $\uparrow x$  estão em correspondência biunívoca com os intervalos  $\mathbb{I}_x = [x^{\sim}, x^{\sim} + 2^{-|x|}]$ .*

Os intervalos reais  $\mathbb{I}_x$  referidos no resultado anterior designam-se por **intervalos de Lebesgue**. É relevante frisar que o intervalo é compacto e que o seu tamanho é  $2^{-|x|}$ , sendo  $|x|$  o comprimento da string  $x$  que o determina.



## Eventos

Vimos que conjuntos  $A$  se identificam com strings infinitas. Falta agora ver a forma como classes de conjuntos se identificam com colecções de strings infinitas.

Para isso é necessário introduzir algumas noções:

### 62 NOÇÃO ( $\sigma$ -ÁLGEBRA)

*Seja  $X$  um domínio. Uma família  $\Sigma$  de sub-conjuntos de  $X$ , fechada por uniões contáveis e complementos, designa-se por  $\sigma$ -álgebra. Os elementos de uma  $\sigma$ -álgebra designam-se por **eventos**.*

**Notas** Porque  $\Sigma$  é fechado por uniões contáveis, contém o conjunto vazio (união vazia); porque é fechado por complementos contém o próprio domínio  $X$  por ser o complemento de  $\emptyset$ . Finalmente, o complemento de uma união contável é uma intersecção contável; por isso  $\Sigma$  é também fechado pela intersecção contável de eventos.

O exemplo imediato de  $\sigma$ -álgebra  $\Sigma_{\mathcal{L}}$  formada pelas uniões ou intersecções contáveis de intervalos de Lebesgue. Designaremos por **intervalos** os elementos desta  $\sigma$ -álgebra específica. Formalmente  $\Sigma_{\mathcal{L}}$  é a menor  $\sigma$ -álgebra que contém todos os intervalos de Lebesgue.



Genericamente eventos generalizam a noção de intervalo. Por isso é importante definir algo que dê a noção de tamanho desses eventos de forma análoga à noção de tamanho nos intervalos.

### 63 NOÇÃO (MEDIDA EM $\sigma$ -ÁLGEBRAS)

Se  $\Sigma$  é uma  $\sigma$ -álgebra, uma função  $\mu: \Sigma \rightarrow \mathbb{R}_+$  é uma **medida** se verifica:

1.  $\mu(\bigcup_i \alpha_i) = \sum_i \mu(\alpha_i)$  para qualquer enumeração  $\{\alpha_i\}$  de eventos mutuamente disjuntos.
2. Se  $\{\alpha_i\}$  é uma sequência decrescente ( $\alpha_{i+1} \subseteq \alpha_i$  para todo  $i$ ), então  $\mu(\bigcap \alpha_i) = \lim_{i \rightarrow \infty} \mu(\alpha_i)$ .

A noção de medida traduz a ideia de que um “intervalo”, que é a união de “intervalos” disjuntos, tem uma medida que é a soma da medida das parcelas; portanto a medida é uma função crescente nos eventos. Como caso particular, o evento vazio tem medida nula.

Quando os eventos formam uma cadeia descendente, as respectivas medidas devem decrescer. Impomos, adicionalmente, que o evento limite desta cadeia tenha uma medida que é o limite das medidas dos elementos da sequência.

Considere-se  $\Sigma_{\mathcal{L}}$ ; isto é, a menor  $\sigma$ -álgebra que contém todos os intervalos de Lebesgue. A medida  $\mu$  é, neste caso, a função que verifica  $\mu(\mathbb{I}_x) = 2^{-|x|}$ . Portanto  $\mu$  associa a cada intervalo o seu tamanho.

Na perspectiva do domínio genérico  $2^{\omega}$ , pode-se construir uma generalização desta  $\sigma$ -álgebra.

64 NOÇÃO ( $\sigma$ -ÁLGEBRA DE LEBESGUE)

No domínio  $2^\omega$  a  **$\sigma$ -álgebra de Lebesgue**, representada por  $\mathcal{L}$ , é a menor  $\sigma$ -álgebra que contém todas as classes da forma  $\uparrow x$  com  $x \in \omega$ .

Nas  $\sigma$ -álgebras de Lebesgue, fixando um inteiro  $n > 0$ , têm importância particular os eventos que são formados por uniões contáveis de eventos elementares  $\uparrow x$  em que  $|x| \geq n$ ; estes eventos designam-se por  **$n$ -eventos**.

Eventos da forma  $\bigcap_{x \in A} \uparrow x$ , quando  $A$  é um conjunto recursivamente enumerável, designam-se por **recursivamente enumeráveis**. Igualmente, se  $A$  for recursivo, o evento designa-se por **recursivo**.

## 65 NOÇÃO (MEDIDAS E COMPRIMENTOS DE LEBESGUE)

Dado um real  $1 \geq \epsilon > 0$ , uma  **$\epsilon$ -medida de Lebesgue** é uma medida em  $\mathcal{L}$  tal que  $0 < \mu(\uparrow x) \leq 2^{-\epsilon|x|}$  para todo  $x \in \omega$ . Quando, para todo  $x \in \omega$ , se tem  $\mu(\uparrow x) = 2^{-|x|}$ , a medida diz-se **standard**.

Fixando uma medida  $\mu$ , um evento  $\alpha$  diz-se **nulo** se  $\mu(\alpha) = 0$ . O **comprimento de Lebesgue** do evento  $\alpha$ , representado por  $\lambda(\alpha)$ , é  $+\infty$  se  $\alpha$  for nulo ou, caso não seja, é  $-\log_2 \mu(\alpha)$ .

Se  $L$  é uma linguagem livre de prefixos, designa-se por **medida de  $L$**  à medida de Lebesgue do evento  $\uparrow L$ .

É importante nota-se que esta definição de  $\sigma$ -álgebra de Lebesgue é feita para o domínio abstracto  $2^\omega$  e serve para qualquer um dos domínios concretos a ele associados: bit-strings infinitas, conjuntos, relações e linguagens.





As classes elementares  $\uparrow x$  são também definidas de forma abstracta;  $x$  pode ser um string de bits finita ou então ser um natural. A classe  $\uparrow x$  representa, indistintamente, todas as strings infinitas com prefixo  $x$  (vendo  $x$  como string) ou todos os conjuntos que contém  $D_x$  (vendo  $x$  como natural), etc.

**Notas** Fixando um qualquer tamanho  $n$ , tem-se  $2^\varpi = \bigcup_{|x|=n} \uparrow x$ ; donde  $\mu(2^\varpi) \leq 2^n \cdot 2^{-\epsilon n}$ , para todo  $n$ . Se admitissemos a hipótese de ser  $\epsilon > 1$ , a medida seria reduzida à situação trivial em que todos os eventos têm medida zero.

A medida standard de um evento  $\uparrow x$  é sempre  $2^{-|x|}$ ; por isso, nessa medida, tem-se sempre  $\lambda(\uparrow x) = |x|$ ; isto é, o comprimento do evento elementar  $\uparrow x$  coincide com o tamanho do objecto  $x \in \varpi$  que o determina.

Se a medida não for standard, apenas se pode afirmar que  $\mu(\uparrow x) \leq 2^{-\epsilon|x|}$ , para um  $\epsilon > 0$  que depende apenas de  $\mu$  (e não de  $x$ ). Neste caso pode-se afirmar que  $\lambda(\uparrow x) \geq \epsilon|x|$ .

**EXEMPLO 3:** Interpretando  $2^\varpi$  como o domínio  $\mathbb{B}^\infty$  das bit-strings infinitas, vamos ver que o conjunto singular  $\Omega = \{u\}$  (com  $u \in \mathbb{B}^\infty$ ) é um evento e, além disso, é um evento nulo.

Seja  $x_n = u \upharpoonright n$  a truncatura de  $u$  a  $n$  bits, e considere-se os eventos elementares  $U_n = \uparrow x_n$ . O evento  $U_n$  contém todas as strings que, até ao  $n$ -ésimo bit, coincidem com  $u$ . Note-se que  $\Omega = \bigcap_n U_n$  e, porque é a intersecção contável de eventos, é também um evento.

Em qualquer medida de Lebesgue, tem-se  $\mu(U_n) \leq 2^{-\epsilon|x_n|} = 2^{-\epsilon n}$ . Pela definição de medida, notando que a sequência  $\{U_n\}$  é decrescente e  $\epsilon > 0$ , tem-se

$$\mu(\Omega) = \lim_{n \rightarrow \infty} \mu(U_n) \leq \lim_{n \rightarrow \infty} 2^{-\epsilon n} = 0$$

Portanto  $\Omega$  é nulo e o respectivo comprimento é  $\lambda(\Omega) = +\infty$ .



Recuperando a visão abstracta do domínio  $2^{\omega}$ , convém ver algumas formas particulares de eventos:

Eventos da forma  $\Omega = \{A\}$ , com  $A \in 2^{\omega}$ , designam-se por **eventos singulares**. O exemplo anterior mostra que esses eventos, para uma qualquer medida de Lebesgue, são nulos.

Unões contáveis de eventos singulares  $\mathcal{A} = \bigcup_i \Omega_i$ , com  $\Omega_i = \{A_i\}$  e  $A_i \in 2^{\omega}$ , designam-se por **eventos discretos**. Obviamente, para qualquer medida,  $\mu(\mathcal{A}) = \sum_i \mu(\Omega_i) = 0$ . Por isso os eventos discretos são também nulos.

**EXEMPLO 4 (CLASSE DOS CONJUNTOS FINITOS):** Considere-se, por exemplo, a classe  $\mathcal{F}$  formada por todos os conjuntos finitos. Como os conjuntos finitos estão em correspondência biunívoca com os seus índices canónicos,  $\mathcal{F}$  é enumerável. Obviamente  $\mathcal{F} = \bigcup_{u \in \mathcal{F}} \{u\}$  é um evento discreto e, por isso, é nulo.

### Aproximações às classes efectivamente enumeráveis

No estudo da computabilidade estamos particularmente interessados nas classes de conjuntos efectivamente enumeráveis na perspectiva que representam possíveis “continuações computáveis” de uma determinada computação.

Estamos também interessados em avaliar, de alguma forma, o “tamanho” dessas classes usando o mecanismo dos eventos e das medidas a eles associados.



## 66 NOÇÃO

Seja  $\mathcal{A} \subseteq 2^{\omega}$  uma classe de conjuntos. Um evento  $\alpha$  que verifique  $\mathcal{A} \subseteq \alpha$  designa-se por **cobertura** de  $\mathcal{A}$ . Se  $\alpha$  for um  $n$ -evento designa-se por  **$n$ -cobertura**.

## 67 NOÇÃO

Fixe-se uma medida de Lebesgue  $\mu$  e uma classe  $\mathcal{A} \subseteq 2^{\omega}$ .  $\mathcal{A}$  é **nula**, e escreve-se  $\lambda(\mathcal{A}) = +\infty$ , se tem uma cobertura nula.  $\mathcal{A}$  não-nulo tem **comprimento** não-inferior a um real  $r \in \mathbb{R}_+$ , e escreve-se  $\lambda(\mathcal{A}) \geq r$ , se tem uma cobertura de comprimento  $r$ .

É importante focar o seguinte ponto: os eventos, normalmente, não traduzem computações; são apenas usados para “cobrir” as classes efectivamente enumeráveis (essas sim, representam computações) e, deste modo, aferir as suas dimensões.

Para ilustrar este ponto, tome-se um qualquer natural  $x$  e compare-se a classe efectivamente enumerável  $\mathcal{S}_x$  (ver (20) na página 41) e o evento elementar  $\uparrow x$ .

- Os conjuntos  $A \in \mathcal{S}_x$  são os conjuntos recursivamente enumerados que contêm o conjunto  $D_x$ .
- A classe  $\uparrow x$  é formada por todos os conjuntos  $A$  em que os elementos de  $A$  de comprimento  $\leq |x|$  foram um sub-conjunto que coincide com  $D_x$ .

Para além de ambas as classes conterem  $D_x$ , existe muito pouco de comum entre elas. Alguns pontos a reter:



Na classe  $\mathcal{S}_x$  estão todos os conjuntos finitos que contêm  $D_x$ ; porém, qualquer classe de conjuntos finitos é (como vimos atrás) enumerável; portanto, é um evento discreto e, conseqüentemente, nulo. Por isso, numa medida do “tamanho” de  $\mathcal{S}_x$ , só são relevantes os seus conjuntos infinitos. O mesmo se pode dizer quando se toma uma classe efectivamente enumerável  $\mathcal{A}$  que, como sabemos, é uma união contável de classes da forma  $\mathcal{S}_x$ . Para avaliar o seu “tamanho” só são relevantes os conjuntos infinitos.

Dado que  $x$  não é necessariamente um prefixo dos conjuntos em  $\mathcal{S}_x$ , surge a questão de saber quais são esses prefixos. Nomeadamente só são relevantes os prefixos dos  $\mathcal{W}_e$  que sejam infinitos. Define-se, assim,  $\downarrow\mathcal{S}_x$  como o conjunto de todos os prefixos de conjuntos recursivamente enumeráveis infinitos que contêm  $D_x$ .

$$z \in (\downarrow\mathcal{S}_x) \Leftrightarrow (\exists e) [z \prec \mathcal{W}_e \wedge \mathcal{W}_e \text{ não finito} \wedge \mathcal{W}_e \supseteq D_x] \quad (30)$$

## 68 NOÇÃO

*Sendo  $\mathcal{A}$  a classe efectivamente enumerável representada pelo “array”  $A$ , os **prefixos** de  $\mathcal{A}$  formam o conjunto, representado por  $\downarrow\mathcal{A}$ , que verifica*

$$z \in (\downarrow\mathcal{A}) \Leftrightarrow (\exists x) [x \in A \wedge z \in (\downarrow\mathcal{S}_x)] \quad (31)$$

Recordemos que uma  $n$ -cobertura  $\alpha$  de  $\mathcal{A}$  é um  $n$ -evento que cobre  $\mathcal{A}$ ; isto é, tem a forma  $\bigcup \uparrow z$ , com  $|z| \geq n$ .



## 69 NOÇÃO

A ***n*-cobertura mínima** de  $\mathcal{A}$ , representado por  $\mathcal{A}^{(n)}$ , é o  $n$ -evento gerado pelos prefixos de  $\mathcal{A}$

$$\mathcal{A}^{(n)} = \bigcup_{|z| \geq n \wedge z \in (\downarrow \mathcal{A})} \uparrow z \quad (32)$$

É elementar verificar que sequência de eventos  $\{\mathcal{A}^{(n)}\}$  é decrescente e todos os elementos da sequência contêm  $\mathcal{A}$ ; isto é, para todo  $n$ ,  $\mathcal{A} \subseteq \mathcal{A}^{(n+1)} \subseteq \mathcal{A}^{(n)}$ . Por isso, faz sentido

## 70 NOÇÃO

A ***cobertura mínima***<sup>14</sup> de  $\mathcal{A}$  é o evento  $\hat{\mathcal{A}} = \bigcap_n \mathcal{A}^{(n)}$ . Fixando uma medida de Lebesgue  $\mu$ , o real  $\mu(\hat{\mathcal{A}})$  representa-se por  $H^\mu(\mathcal{A})$  e designa-se por  ***$\mu$ -medida exterior de Hausdorff*** de  $\mathcal{A}$ .

## 71 FACTO

Cada evento  $\mathcal{A}^{(n)}$  é recursivamente enumerável. No entanto  $\hat{\mathcal{A}}$  pode não ser recursivamente enumerável.

Note-se que

$$H^\mu(\mathcal{A}) = \mu(\hat{\mathcal{A}}) = \lim_{n \rightarrow +\infty} \mu(\mathcal{A}^{(n)}) = \lim_{n \rightarrow +\infty} \sum_{|z| \geq n \wedge z \in (\downarrow \mathcal{A})} \mu(\uparrow z) \quad (33)$$

<sup>14</sup>Também designada por ***cobertura de Hausdorff***.



É importante ter em atenção que  $H^\mu$  não é, normalmente, uma medida. É uma avaliação do “tamanho” de uma classe através da medida dos eventos que a cobrem.



### Domínios algébricos e co-algébricos

Os domínios enumeráveis são, frequentemente, **algébricos** (também designados por **indutivos**) no sentido em que os elementos do domínio podem ser construídos indutivamente a partir de outros elementos do mesmo domínio.

#### EXEMPLO 5:

Pode-se construir os elementos de  $\mathbb{N}$  pela construção de Peano, partindo da constante 0 e da função “sucessor”  $\text{suc}: x \mapsto x + 1$ . Por indução são gerados, com estes dois construtores, todos os elementos de  $\mathbb{N}$ .

Do mesmo modo as bit-strings finitas são geradas por três construtores: a constante string vazia  $\varepsilon$  e duas funções  $\text{inc1}: u \mapsto u1$  e  $\text{inc0}: u \mapsto u0$  (a primeira acrescenta o bit 1 e a segunda acrescenta o bit 0 ao argumento).

Domínios indutivos incluem as listas  $X^*$  de elementos de um domínio enumerável  $X$ , e as árvores com nodos em  $X$ . Domínios enumeráveis formados por conjuntos de  $X$ , ou “bags” já não são, normalmente, algébricos.

É fácil verificar que todo o domínio indutivo é primitivamente representável.

Enquanto que  $\mathbb{B}^*$  e  $\mathbb{N}$  são paradigmas de domínios enumeráveis, os domínios dos seus sub-conjuntos (respectivamente  $\wp(\mathbb{B}^*)$  e  $\wp(\mathbb{N})$ ) são paradimas de domínios não-enumeráveis.



**EXEMPLO 6:**

A prova de que  $\wp(\mathbb{N})$  não é enumerável é um exemplo típico de prova por redução ao absurdo através de um argumento de diagonalização: assume-se que existe uma enumeração por conjuntos  $\{A_k\}$  que cobre todo  $\wp(\mathbb{N})$ ; define-se um conjunto “diagonal”  $A$  que contém o inteiro  $k$  se e só se o conjunto  $A_k$  não contém  $k$ ; claramente a “diagonal”  $A$  não pode pertencer à enumeração e, portanto, esta não pode cobrir  $\wp(\mathbb{N})$ .

Os domínios não enumeráveis (os anteriores e ainda as bit-strings infinitas  $\mathbb{B}^\infty$ , o intervalo real  $[0, 1]$ , os domínios de funções  $\mathbb{B}^\mathbb{N}$  e  $\mathbb{N}^\mathbb{N}$ , etc.) são, quanto muito, **co-algébricos** (ou **co-indutivos**) no sentido em que os elementos do domínio podem ser indutivamente observados; isto é, indutivamente o resultado de uma observação é construído a partir de resultados de observações análogas. No entanto não existe uma função computável única capaz de construir um elemento do domínio a partir de outros elementos do mesmo domínio ou de outros domínios.

O seguinte exemplo procura ilustrar estes pontos.

**EXEMPLO 7:**

Considere-se a “função”  $r: \omega \mapsto \omega^\sim$ , “definida” em (28) que relaciona bit-strings infinitas com reais.

Não se trata de uma função computável já que constrói apenas os elementos de uma sequência de Cauchy. É um “processo”, e não um “algoritmo”, que vai construindo sequencialmente os racionais  $r_k$  que convergem para o “resultado”. Sequencialmente, no passo  $k$ , o processo lê o bit de  $\omega_k$  e calcula o racional  $r_k = r_{k-1} + \omega_k 2^{-k-1}$ . Nunca se chega a atingir o “resultado” limite; a sequência de racionais  $\{r_k\}$  é o verdadeiro “output” deste processo.

Suponhamos que se quer construir uma “função inversa” que tome um número real no intervalo  $\mathbb{I} = [0, 1]$  e gere a bit-strings que lhe corresponde. Mais uma vez não existe nenhuma função computável mas um eventual processo. Considere-se a função  $h: \mathbb{I} \rightarrow \mathbb{B} \times \mathbb{I}$



definida do seguinte modo

$$h(x) = \begin{cases} (0, 2x) & \text{se } x < 1/2 \\ (1, 2x - 1) & \text{se } x \geq 1/2 \end{cases} \quad (34)$$

Esta função gera, a partir de um valor inicial  $x$ , uma sequência de bits  $b_0, b_1, b_2, \dots$  através da função de recorrência

$$x_0 = x, \quad (b_i, x_{i+1}) = f(x_i)$$

De facto não existe um processo computável de gerar esses bits uma vez que a função  $h$  não é computável. De facto não existe sequer uma forma computável para determinar se um real arbitrário é ou não inferior a  $1/2$ .

Como os reais não são enumeráveis, não é possível, genericamente, definir funções computáveis que construam reais. No entanto é possível aproximar um determinado real por uma sequência de racionais que “cresça” para um determinado limite<sup>15</sup>.

Resta saber se os elementos dessa sequência podem ser computados. Surge assim a noção de “real recursivo” que essencialmente traduz o facto de a representação  $r(\cdot)$  ser uma função recursiva que, para determinar um real  $x \geq 0$ , calcula uma sequência crescente de aproximações racionais  $\{r(n)\}$  convergindo para  $x$ ; i.e.  $x = \lim \uparrow r(n)$ .

A noção de real r.L. é, no entanto, mais geral e pode assumir várias facetas identificando objectos “computacionais” de natureza diversa: linguagens, testes, etc. Caso não exista ambiguidade usaremos a mesma designação (real

<sup>15</sup>Aliás esta é a definição formal de real.



recursivo de Lebesgue) para designar a linguagem  $L$  da definição, a função recursiva  $r$  que a enumera e o real  $\lim \uparrow r(n)^\sim$ .

## 1.5 Aleatoriedade

A aleatoriedade é um dos conceitos fundamentais em Criptografia uma vez que a noção de “segredo” está, intuitivamente, ligada à incapacidade de um atacante distinguir informação protegida de não-informação; isto é, informação aleatória.

### Aleatoriedade de Kolmogorov

A primeira abordagem à aleatoriedade foi apresentada através do “índice de Kolmogorov”. Recordemos a noção de índice de Kolmogorov (noção 23, página 29) de um inteiro  $y$  como o inteiro  $\kappa(y)$  tal que

$$\varphi_{\kappa(y)}(0) \simeq y \quad , \quad \varphi_e(0) \not\simeq y \quad \text{par todo } e < \kappa(y)$$

Vimos que  $\kappa$  era uma função total mas não recursiva.

Pode-se estender esta noção usando funções parciais que recorrem a um oráculo  $O$  e definir  $\kappa(y|O)$  como o inteiro que verifica

$$\varphi_{\kappa(y)}^O(0) \simeq y \quad , \quad \varphi_e^O(0) \not\simeq y \quad \text{par todo } e < \kappa(y)$$

Como sabemos  $\varphi_e^O$  é a o elemento de ordem  $e$  na indexação das funções parciais recursivas que recorrem à relação  $O$  como função primitiva.

O inteiro  $y \in \omega$  é  **$\kappa$ -compressível** quando  $\kappa(y) < y$ . Os inteiros que não são  $\kappa$ -compressíveis são  **$\kappa$ -aleatórios**.



Estas noções estendem-se para strings finitas, definindo  $\kappa(r(y)) = \kappa(y)$  ( $r: \mathbb{N} \rightarrow \mathbb{B}^*$  é a representação standard de strings finitas), e para conjuntos finitos, definindo  $\kappa(D_y) \doteq \kappa(y)$ .

O teorema de Kolmogorov (teorema 47, página 46) permite dar uma primeira caracterização dos objectos (inteiros, ou strings finitas, ou conjuntos finitos)  $\kappa$ -compressíveis e, por complemento, os objectos  $\kappa$ -aleatórios. O teorema diz-nos os inteiros  $\kappa$ -compressíveis formam um conjunto simples; isto é, um conjunto recursivamente enumerável cujo complemento é imune (infinito e nenhum dos seus sub-conjuntos infinitos é recursivamente enumerável).

□

Note-se que todos estes resultados são independentes do sistema de índice usado mas o índice de Kolmogorov vai depender, em princípio do sistema de índices usado. Por isso, para definir uma noção de aleatoriedade que seja universal, convém restringir o sistema de índices usado.

A enumeração das funções parciais recursivas tem uma variante quando se

## 72 TEOREMA

*Existe um sistema de índices  $\{\varphi_e\}$  que enumera todas as funções parciais recursivas com domínio livre de prefixos e uma função universal  $\varphi$  com domínio livre de prefixos que verifica  $\varphi_e(x) \simeq \varphi(e||x)$ .*

**Esboço de prova** A prova deste teorema é uma consequência da normalização das funções parciais recursivas (teorema 15, página 23) e da representação dos sistemas aceitáveis de índices (teorema 20, página 26).



Neste resultado estamos a ver  $\varphi$  como uma função parcial recursiva de strings finitas em strings finitas; a condição de o seu domínio ser livre de prefixos pode-se escrever

$$\varphi(x)\downarrow \wedge \varphi(y)\downarrow \wedge x \neq y \Rightarrow \uparrow x \cap \uparrow y = \emptyset \quad (35)$$

As funções  $\varphi$  que verificam as condições do resultado anterior, dizem-se **universais livres de prefixos (u.l.p.)**; as indexações que determinam dizem-se **livres de prefixos**.

Por o domínio de  $\varphi$  ser livre de prefixos, tem-se  $\sum_{\varphi(z)\downarrow} 2^{-|z|} \leq 1$  (ver facto 54, página 53). Esta observação tem uma forma “inverso”

### 73 FACTO

*Dada uma qualquer sequência recursiva de  $\{d_n\}_n$  que verifique  $d = \sum_n 2^{-d_n} \leq 1$ , existe uma função parcial recursiva  $\varphi_e$  cujo domínio é uma linguagem livre de prefixos com medida  $d$ .*

**Prova** Seja  $\{x_n\}$  a sequência recursiva livre de prefixos gerada pela sequência  $\{d_n\}$  (teorema 59, página 58). A função  $\varphi_e(z) = \mu_n \cdot [z = x_n]$  é parcial recursiva e o seu domínio é a linguagem livre de prefixos enumerada por  $\{x_n\}$ . Como  $|x_n| = d_n$ , para todo  $n$ , tem-se  $\sum_n 2^{-|x_n|} = \sum_n 2^{-d_n} = d$ .

□

### 74 NOÇÃO (COMPLEXIDADE DE KOLMOGOROV)

*Seja  $\varphi$  uma função universal livre de prefixos. Define-se  $\varphi$ -complexidade de Kolmogorov<sup>16</sup> de um  $y \in \varpi$ ,*

<sup>16</sup>Na generalidade dos textos esta complexidade é normalmente designada “prefix free Kolmogorov complexity”. No entanto, devido



representado por  $K_\varphi(y)$ , como  $\min\{|x| \mid \varphi(x) \simeq y\}$ . Dado um conjunto  $O$  (um oráculo) define-se  $K_\varphi(y|O) = \min\{|x| \mid \varphi^O(x) \simeq y\}$ .

Algumas propriedades de  $K$

#### 75 TEOREMA

1. A menos de uma constante  $O(1)$  independente de  $y$ , tem-se  $K_\varphi(y) = \min\{|e| \mid \varphi_e(0) \simeq y\} + O(1)$ .
2. Para qualquer linguagem  $L$  verifica-se  $\sum_{y \in L} 2^{-K_\varphi(y)} \leq 1$ .

**Esboço de prova** O resultado (1) deriva da indexação de funções. Este é um resultado que estabelece a relação entre a complexidade de Kolmogorov e o índice de Kolmogorova para indexações livres de prefixos.

O resultado (2) deriva do facto de que o conjunto  $\{x \mid \varphi(x) \simeq y \wedge y \in L\}$  ser livre de prefixos já que o domínio de  $\varphi$  é livre de prefixos (ver facto 54, página 53).

Por outro lado temos uma consequência imediata do teorema de Kolmogorov particularizando a indexação para uma que seja livre de prefixos.

#### 76 TEOREMA

Para toda a constante  $c \geq 0$ , o conjunto  $\{y \mid K_\varphi(y) \leq |y| - c\}$  é simples.

ao modo como é aqui definida a indexação das funções parciais recursivas, esta complexidade é sempre “prefix free” e não tem qualquer interesse introduzir outra que o não seja.



**Esboço de prova**

O conjunto  $\{y \mid K_\varphi(y) \leq |y| - c\}$  é um sub-conjunto do conjunto dos objectos compressíveis que, pelo teorema de Kolmogorov, é simples. É fácil verificar que o sub-conjunto também tem de ser simples.

Por outro lado, usando as relações entre os sistemas aceitáveis de índices (ver teorema 20, página 26) prova-se

## 77 TEOREMA

*A menos de uma constante,  $K_\varphi(y)$  (ou  $K_\varphi(y|O)$ ) não depende da função universal livre de prefixos  $\varphi$ .*

Nestas circunstâncias, entendendo que  $K(y)$  e  $K(y|O)$  são sempre definidos a menos de uma constante independente de  $y$ , podemos prescindir do índice  $\varphi$  em  $K_\varphi$ .

**Intuição**

A indexação das funções parciais recursivas usa intensivamente a construção de pares  $(x||y)$ . Uma questão importante é o de saber quais são os limites ao tamanho do par  $x||y$  em função dos tamanhos de  $x$  e  $y$ .

Intuitivamente sabemos que, para ser possível recuperar  $x$  e  $y$  a partir de  $x||y$ , tem de existir neste valor informação sobre o tamanho do primeiro elemento do par. A codificação óptima de  $|x|$  exige alguma informação constante  $C$  (que depende apenas do algoritmo de emparelhamento) e  $||x||$  bits. Por isso,

## 78 FACTO

*Existe uma codificação óptima de pares  $x, y$  tal que*

$$|x||y| \leq C + ||x|| + |x| + |y|$$



Considere-se agora o índice  $e$  da projecção  $\pi_1$ . Temos  $\varphi(e|(y|0)) \simeq \pi_1(y|0) \simeq y$ . Pela propriedade de parametrização dos índices, existe uma função recursiva  $\lambda$  tal que  $\varphi(\lambda(y|e)|0) \simeq \varphi(e|(y|0)) \simeq y$ . Por isso  $\kappa(y) \leq \lambda(y|e)$ .

Basta agora ver que  $|\lambda(z)| \leq |z| + O(1)$  para concluirmos que  $|\kappa(y)| \leq C + |y|e \leq C + ||y|| + |y| + |e|$ . Logo  $|\kappa(y)| \leq O(1) + ||y|| + |y|$ , absorvendo na constante  $O(1)$  quer a constante  $C$  como  $|e|$ .

Baseado neste tipo de intuição, e nas propriedades de enumeração e parametrização dos índices, prova-se que

### 79 FACTO

A complexidade de Kolmogorov  $K$  é “sub-aditiva”; isto é,

$$K(x|y) \leq O(1) + K(x) + K(y)$$

Para exprimir propriedades genéricas da complexidade de Kolmogorov é conveniente alguma notação prévia:

### 80 NOÇÃO

Vamos designar por **défice de informação** de  $y$  a quantidade

$$k_d(y) = K(y) - |y| - K(|y|) \quad (36)$$

e por **défice inicial de informação** a quantidade

$$K_d(y) = K(y) - |y| = k_d(y) + K(|y|) \quad (37)$$



A informação própria de  $y$  (aquela que a mensagem  $y$  transmite) está nos seus bits e no seu tamanho. Por isso entendemos que ela é  $|y| + K(|y|)$ .

Desta forma  $k_d(y)$  representa um “défice” de informação: a diferença entre a informação  $K(y)$  no “input”, indispensável para que uma máquina de Turing produza  $y$ , e a informação própria do “output”  $y$ . A quantidade  $K_d$  exprime o mesmo tipo de défice mas tomando como referência a informação nos bits de  $y$ .

Ao longo deste curso vai ser muito importante caracterizar a **incerteza** com que podemos, dado um conjunto de possibilidades, seleccionar uma dessas possibilidades. A próxima secção será dedicada ao estudo desta noção mas, para já, é necessário avançar com algumas noções prévias.

#### 81 NOÇÃO (INCERTEZA DE HARTLEY E UNIFORME)

Designamos por **incerteza de Hartley** (ou, caso não exista ambiguidade, simplesmente **incerteza**<sup>17</sup>) de um conjunto finito  $A$ , e representamos por  $\vartheta(A)$ , e definida por  $\vartheta(A) = \lceil \log_2 \llbracket A \rrbracket \rceil$ , sendo  $\llbracket \cdot \rrbracket$  a função recursiva que conta o número de elementos de um conjunto finito. Define-se  $\vartheta(\emptyset)$  como  $+\infty$ .

A **incerteza uniforme** à funcional  $\vartheta[\cdot]$  que a cada conjunto  $A$  (finito ou não) associa a função  $\vartheta[A](n) \simeq \vartheta(A \upharpoonright n)$ .

Propriedades genéricas da complexidade de Kolmogorov exprimem a sua relação com a noção de incerteza uniforme.

<sup>17</sup>A próxima secção, dedicada ao conceito de incerteza, vai introduzir outras formas para exprimir esta noção.



## 82 TEOREMA (CONTAGEM DE CHAITIN)

Para todo  $n$ , todo  $k$ , e para uma constante  $O(1)$  independente de  $n$  e  $k$ ,

$$\max_{|y|=n} k_d(y) = \pm O(1) \quad (38)$$

$$n - \vartheta[\{y \mid k_d(y) \leq -k\}](n) \geq k - O(1) \quad (39)$$

O teorema da contagem presuppõe que se restringe os  $y$ 's a objectos de tamanho  $n$  e diz-nos duas coisas:

- (i) Para todo  $n$ , o máximo do déficite  $k_d(y)$  dos  $y$ 's de tamanho  $n$ , é uma constante  $C$  independente de  $n$ .
- (ii) O predicado  $P(y) = k_d(y) \leq -k$  avalia se o défice de informação de  $y$  é inferior a  $-k$  bits; isto é equivalente a dizer que o “supro informação” é igual ou superior a  $k$  bits.

Restringindo os valores  $y$ 's ao comprimento  $n$ , o conjunto de possibilidades tem incerteza  $n$ . Assim a diferença  $n - \vartheta_n[y.k_d(y) \leq -k]$  mede a quantidade de informação que é preciso fornecer para ter a certeza que a propriedade  $[k_d(y) \leq -k]$  se verifica nesse conjunto de possibilidades. Essencialmente (ii) diz-nos que, a menos de uma constante positiva, é necessário fornecer (de novo) os mesmos  $k$  bits de informação.

□

A complexidade de Kolmogorov  $K(y)$  é uma medida que pode ser usada em conjuntos finitos mas não em conjuntos infinitos. Se  $A$  é um conjunto infinito, temos de atender à complexidade  $K(A \upharpoonright n)$  das diferentes truncaturas de  $A$ .



Esta ideia e o teorema da contagem está na base da seguinte definição:

### 83 NOÇÃO (ALEATORIEDADE DE KOLMOGOROV-LEVIN-CHAITIN)

Um conjunto  $A$  é **KLC-aleatório** se existe um  $c \in \varpi$  tal que, para todo  $n$ ,  $K(A \upharpoonright n) > n - c$ . O conjunto  $A$  é **KLC-aleatório relativo a  $O$**  se, para algum  $c$ , se verifica  $K(A \upharpoonright n | O) > n - c$  para todo  $n$ .

Em alternativa pode-se apresentar a definição de aleatoriedade, introduzindo a família de conjuntos  $U_c$  definidos como

$$U_c = \{ y \mid K(y) \leq |y| - c \} \quad (40)$$

Note-se que, pelo teorema 76, todos estes conjuntos são simples. Então a noção 83 reescreve-se facilmente.

### 84 FACTO

$A$  é KLC-aleatório sse existe uma constante  $c \in \varpi$  tal que, para todo  $n$ ,  $A \upharpoonright n \notin U_c$ . Equivalentemente,  $A$  não é KLC-aleatório sse está contido em  $\bigcap_c \uparrow U_c$ .

#### Prova

A prova da primeira parte é uma simples reescrita da definição 83 e da definição dos conjuntos  $U_c$ . Para a segunda parte, temos que  $A$  não é KLC-aleatório sse, para todo  $c$ , existe  $n$  tal que  $(\exists n)[K(A \upharpoonright n) \leq n - c]$ ; ou seja,  $(\exists n)[A \upharpoonright n \in U_c]$  ou ainda, equivalentemente,  $A \in \uparrow U_c$ . Como  $A$  tem de pertencer a todos  $\uparrow U_c$  concluímos que tem de pertencer à sua intersecção.

Os conjuntos  $U_c$  determinam eventos  $\uparrow U_c$  que são “raros”. Esse facto deriva do seguinte resultado.



85 FACTO

Seja  $U_c$  definido em (40); então  $\sum_{y \in U_c} 2^{-|y|} \leq 2^{-c}$ .

**Esboço de prova**

Para  $y \in U_c$  tem-se  $c - |y| \leq -K(y)$ ; logo  $2^{c-|y|} \leq 2^{-K(y)} \leq 1$ . Consequentemente  $\sum_{y \in U_c} 2^{-|y|} \leq 2^{-c}$ .

Combinando o teorema da contagem com a esta definição de aleatoriedade, para um conjunto  $A$  ser KLC-aleatório tem existir constante  $O(1)$  tal que, para todo  $n$ ,

$$0 \leq K(A \upharpoonright n) - n \pm O(1) \leq K(n)$$

Note-se que só se define complexidade de Kolmogorov a menos de uma constante; por isso, para medir a informação necessária para computar o prefixo  $A \upharpoonright n$ , temos de usar  $K(A \upharpoonright n) \pm O(1)$ .

A quantidade  $K(A \upharpoonright n) - n \pm O(1)$  mede, a menos de uma constante, o “défice” inicial de informação que é preciso colocar na computação relativa à informação nos bits do prefixo. Este défice é sempre positivo e está limitado superiormente pela complexidade do comprimento  $n$ .

Essa é a intuição associada à complexidade de Kolmogorov: para gerar qualquer prefixo dum conjunto aleatório é sempre preciso por mais informação do que a que se consegue retirar. No entanto aquilo que é importante, em termos de aleatoriedade, é a “velocidade” com que esta défice cresce com  $n$ .

Esta relação entre a noção de KLC-aleatoriedade razão de crescimento de  $K(A \upharpoonright n) - n$ , exprime-se em termos das chamadas funções  $\varpi$ -similares.

## 86 NOÇÃO

Uma função total  $f$  é  $\varpi$ -similar quando verifica  $\sum_n 2^{-f(n)} < \infty$ . Em caso contrário diz-se  $\varpi$ -limitada.

Note-se que adicionar ou subtrair uma constante a  $f$  não altera o facto de  $f$  ser  $\varpi$ -similar ou não. Adicionalmente é simples provar que, se  $f$  é  $\varpi$ -similar e  $g$  é  $\varpi$ -limitada, então  $(\exists_{\infty} n) [f(n) > g(n)]$ .

## 87 TEOREMA (MILLER &amp; YU)

Se  $A$  é um conjunto KLC-aleatório, então a função de  $n$ ,  $K(A \upharpoonright n) - |A \upharpoonright n|$  é  $\varpi$ -similar.

Como complemento a este teorema temos algo que nos ajuda a responder a uma dúvida básica: *será que existem conjuntos aleatórios?* O seguinte teorema dá uma resposta a esta questão.

## 88 TEOREMA (MILLER)

Para todo  $\varpi$ -similar  $f$  existe um conjunto KLC-aleatório  $A$  tal que, para todo  $n$ ,  $K(A \upharpoonright n) - n \leq f(n) + O(1)$ .

Uma outra propriedade importante da complexidade de Kolmogorov deriva da existência dos chamados **conjuntos de Kraft-Chaitin** que, de uma forma que iremos esclarecer, “aproximam” conjuntos aleatórios.



## 89 NOÇÃO (CONJUNTO DE KRAFT-CHAITIN)

Uma sequência recursiva de pares  $R = \{d_n \| y_n\}$  que verifiquem  $\sum_n 2^{-d_n} \leq 1$  designa-se por **conjunto de Kraft-Chaitin**.

Os pares  $r_n = d_n \| y_n$  são designados por **requests**; o elemento  $d_n$  designa-se por **incerteza** de  $y_n$  e o elemento  $y_n$  por **resposta** a  $r(n)$ .

Pode-se ver  $R = \{d_n \| y_n\}_n$  como uma enumeração de respostas à selecção de um  $n$  condicionado por incertezas  $d_n$ . É como uma função recursiva  $n \mapsto y_n$  mas associando incertezas  $d_n$  à selecção do argumento e com a restrição adicional de os pares incerteza+resposta serem recursivamente enumeráveis. Desta forma os  $w_n = 2^{-d_n}$  são uma **probabilidade** de um acto de selecção do índice/argumento  $n$ .

**EXEMPLO 8:** Alguns exemplos particularmente simples de conjuntos de Kraft-Chaitin

1. O domínio abstract  $\varpi$  determina um conjunto de Kraft-Chaitin  $\varpi' = \{d_n \| n\}_{n \in \varpi}$  com  $d_n = n + k$ , para um qualquer  $k > 0$  independente de  $n$ . De facto  $\sum_{y \in \varpi} 2^{-d_n} = 2^{-k} \sum_{n \in \varpi} 2^{-n} = 2^{-k+1} \leq 1$ .
2. Seja  $A$  um conjunto recursivo; então o conjunto dos seus prefixos  $\downarrow A = \{A \upharpoonright n\}_{n \in \varpi}$  é também recursivo. A sequência  $n \mapsto d_n = n + k$  é recursiva e a sequência de pares  $A' = \{d_n \| A \upharpoonright n\}_{n \in \varpi}$  é recursiva. Como (tal como no caso anterior),  $\sum_n 2^{-d_n} \leq 1$ , conclui-se que  $A'$  é um KC-conjunto.



3. Genericamente seja  $Y$  um qualquer conjunto recursivamente enumerável e seja  $\alpha: \varpi \rightarrow Y$  uma enumeração recursiva dos seus elementos. Suponhamos que é possível encontrar uma função recursiva  $d$  tal que  $K(\alpha(n)) \leq d(n)$ , para todo  $n$ . Então  $R = \{d(n) \parallel \alpha(n)\}_n$  é um conjunto de Kraft-Chaitin. De facto, porque  $\alpha$  e  $d$  são ambas recursivas, o conjunto  $\{d(n) \parallel \alpha(n)\}_{n \in \varpi}$  é recursivamente enumerável. Por outro lado, usando o teorema 75,  $\sum_n 2^{-d(n)} \leq \sum_n 2^{-K(\alpha(n))} \leq \sum_{y \in R} 2^{-K(y)} \leq 1$ .
4. Em particular, considere-se, para  $c \in \varpi$ , o conjunto  $U_c = \{y \mid K(y) \leq |y| - c\}$ . Vimos (teorema 76) que  $U_c$  é um conjunto recursivamente enumerável. Considere-se a função recursiva  $\tau(y) = |y| - c$ ; seja  $d(n) = \tau(\alpha(n))$  sendo  $\alpha$  uma enumeração de  $U_c$ . Como, para todo  $y \in U_c$ , se verifica  $K(y) \leq \tau(y)$  o exemplo anterior diz-nos que  $\{\tau(y) \parallel y\}_{y \in U_c}$  é um conjunto de Kraft-Chaitin.

O exemplo 8 diz-nos que a existência de uma função recursiva  $d$  e um conjunto recursivamente enumerável  $Y$  tal que  $K(y) \leq d(y)$ , para todo  $y \in Y$ , origina um conjunto de Kraft-Chaitin da forma  $\{d(y) \parallel y\}_{y \in Y}$ . O resultado inverso é um teorema importante.

#### 90 TEOREMA (LEVIN, SCHNORR, CHAITIN)

*Se  $R = \{d_n \parallel y_n\}$  for um conjunto de Kraft-Chaitin, então existe uma sequência recursiva, livre de prefixos  $\{x_n\}_n$  e uma função parcial recursiva  $\phi$  tais que, para todo  $n$ ,  $|x_n| = d_n$  e  $\phi(x_n) \simeq y_n$ .*

*Como consequência, para todo  $n$  e para uma constante  $O(1)$  independente de  $n$ , verifica-se  $K(y_n) \leq d_n + O(1)$ .*



Essencialmente o resultado diz-nos que, para cada “request”  $r_n = d_n || y_n$  existe um programa  $e_n$  com o comprimento da incerteza  $d_n$ , que é capaz de gerar a resposta  $y_n$ ; ou seja,  $\varphi(e_n || 0) \simeq y_n$  e  $|e_n| \geq d(n)$ . Note-se que a construção de Kraft-Chaitin impõe que o conjunto de pares  $\{d_n || y_n\}_n$  seja computacionalmente enumerável o que é algo que não precisa de acontecer, por exemplo, no conjunto  $\{K(y_n) || y_n\}_n$ .

### Esboço de prova

Pelo teorema 59 (página 58), existe uma sequência recursiva, livre de prefixos  $\{x_n\}$  tal que  $|x_n| = d_n$ ; portanto a função  $h(x) \simeq \mu n \cdot [x = x_n]$  é parcial recursiva. Como  $\{y_n\}$  é uma sequência recursiva, a função  $\phi(x) = y_{h(x)}$  é parcial recursiva e verifica  $\phi(x_n) \simeq y_n$ , para todo  $n$ .

Seja  $\varphi$  uma qualquer função universal cujo domínio é livre de prefixos. Seja  $e$  o índice de  $\phi$  em  $\varphi$ . Então  $y_n \simeq \phi(x_n) = \varphi(e || x_n)$ ; logo  $K_\varphi(y_n) \leq |e| + |x_n| = |e| + d_n$ .

## 91 NOÇÃO

Um conjunto de Kraft-Chaitin  $R = \{d_n || y_n\}$  diz-se **suficientemente aleatório** quando  $\sum_n 2^{-\sigma_n} < \infty$  sendo  $\{\sigma_n\}$  a sequência definida por  $\sigma_n = K(y_n) + d_n - |y_n|$ .

Um conjunto-KC  $R$  suficientemente aleatório, distribui a aleatoriedade por duas componentes: nas incertezas  $d_n$  e nas respostas  $y_n$ . A incerteza na geração de  $y_n$  é dada por  $K(y_n)$ ; a incerteza na “escolha” de  $y_n$  é representada por  $d_n$ ; somando as duas incertezas e subtraindo a informação específica de  $y_n$  (dada por  $|y_n|$ ) temos a diferença  $\sigma_n$ . A “aleatoriedade suficiente” impõe que a sequência  $\{\sigma_n\}$  seja  $\varpi$ -similar.



Esta noção é sugerida pelo teorema de Miller & Yu (teorema 87, na página 83) onde se toma um conjunto  $A$  e a sequência das suas truncaturas,  $y_n = A \upharpoonright n$ ; fazendo  $\sigma_n = K(y_n) - |y_n|$  e sendo  $A$  aleatório, o teorema diz-nos que  $\{\sigma_n\}$  é uma sequência  $\varpi$ -similar.

Note-se que, neste caso, a sequência  $\{y_n\}$  não é recursiva e portanto não se trata, formalmente, de um conjunto de Kraft-Chaitin. Por outro lado não são necessárias incertezas  $d_n$ : a aleatoriedade de  $A$  é já suficiente para garantir que  $\{\sigma_n\}$  seja  $\varpi$ -similar. Se  $A$  não for aleatório as incertezas  $d_n$  que tornam  $\{\sigma_n\}$   $\varpi$ -similar medem a “falta de aleatoriedade” de  $A$ .

Essa abordagem é ilustrada nos exemplos seguintes.

**EXEMPLO 9:** Considere-se os conjuntos de Kraft-Chaitin apresentados no exemplo 8 e procuremos averiguar quais os que são suficientemente aleatórios:

1. Considere-se o KC-conjunto  $\mathcal{N} = \{n + k \mid n\}_n$  com  $k > 0$ . Então  $\sigma_n = K(n) + (n + k) - |n| = K(n) + (n + k) - \lfloor \log_2(n + 1) \rfloor \geq K(n)$ , para todo  $n \in \varpi$ . Consequentemente  $\sum_n 2^{-\sigma_n} \leq \sum_n 2^{-K(n)} < \infty$  e, portanto,  $\mathcal{N}$  é suficientemente aleatório.

É possível definir incertezas  $d_n$  menores que  $n + k$  e que ainda geram conjuntos  $\mathcal{N}$  suficientemente aleatórios. Por exemplo, sabe-se que, para  $s > 1$ , a série  $\zeta(s) = \sum_{n>0} n^{-s}$  converge. De facto, quando  $s$  percorre os complexos esta é a famosa zeta-função de Riemann que converge<sup>18</sup> para todos  $\Re(s) > 1$ . Defina-se uma

<sup>18</sup>Um dos resultados mais surpreendentes da zeta-função de Riemann é **fórmula de produtos de Euler** que diz que, para  $\Re(s) > 1$ ,



variante da função comprimento da forma seguinte:  $|n|_s = \lfloor s \log_2(n+1) \rfloor$ . Então, para um real  $s > 1$ , a série  $\sum_n 2^{-|n|_s}$  converge; de facto tem-se  $2^{-|n|_s} \leq (n+1)^{-s}$  e, portanto,  $\sum_n 2^{-|n|_s} \leq \zeta(s)$ .

Se  $s > 1$  for um real recursivo, a função  $d_s(n) = |n|_s + k$  é recursiva e é possível escolher  $k > 0$  de tal forma que se verifique  $\sum_n 2^{-d_s(n)} \leq 1$ : basta escolher  $k \geq \log_2(\zeta(s))$ .

Designemos por  $\mathcal{N}_s$  o conjunto de Kraft-Chaitin  $\{d_s(n) \parallel n\}_{n \in \omega}$  com  $d_s$  assim definida.

Com tais incertezas tem-se  $\sigma_n = K(n) + d_s(n) - |n| = K(n) + k + |n|_s - |n|$ . Para  $s > 1$ , tem-se sempre  $|n|_s - |n| \geq 0$ ; logo  $\sigma_n \geq K(n)$  para todo  $n$  e, conseqüentemente,  $\sum_n 2^{-\sigma_n} < \infty$ . Com estas incertezas  $\mathcal{N}_s$  continua a ser suficientemente aleatório.

2. Seja  $A$  um conjunto recursivo e  $A'_k = \{n+k \parallel A \upharpoonright n\}_n$ ; no exemplo 8 viu-se que, para  $k > 0$ ,  $A'_k$  é um conjunto de Kraft-Chaitin. Temos  $\sigma_n = K(A \upharpoonright n) + (n+k) - n = K(A \upharpoonright n) + k$ ; logo  $\sum_n 2^{-\sigma_n} = 2^{-k} \sum_n 2^{-K(A \upharpoonright n)} < \infty$ . Conseqüentemente  $A'_k$  é suficientemente aleatório.

3. Seja  $Y = \{y_n\}$  um conjunto recursivamente enumerável; vimos que a existência de uma função recursiva  $d$  tal que, para todo  $n$ ,  $K(y_n) \leq d(n) + k$  assegura, com constantes  $k$  apropriadas, que  $Y' = \{d_n \parallel y_n\}$  é um

---

verifica-se  $\zeta(s) = \prod_p \frac{1}{1-p^{-s}}$  em que o produto percorre todos os *números primos*  $p$  e converge uniformemente numa vizinhança de  $s$ .

Sabe-se que  $\zeta(s)$  tem uma continuação meromórfica para todo o plano complexo com um único polo no ponto  $s = 1$ . Sabe-se que  $\zeta(s)$  não tem nenhum zero para  $\Re(s) > 1$  e os zeros com  $\Re(s) < 0$  são os inteiros pares negativos. A famosa **hipótese de Riemman** conjectura que todos os restantes zeros de  $\zeta(s)$  se situam sobre a recta  $\Re(s) = 1/2$ .



conjunto de Kraft-Chaitin. Isto não assegura, porém, que  $Y'$  seja suficientemente aleatório. Se adicionalmente se tiver  $|y_n| \leq d_n + O(1)$ , então  $\sigma_n \geq K(y_n) + O(1)$  e, por isso,  $Y'$  já será suficientemente aleatório.

4. O conjunto  $U_c = \{y \mid K(y) \leq |y| - c\}$  é recursivamente enumerado; seja  $\alpha$  uma qualquer enumeração recursiva de  $U_c$ . Vimos que  $U'_c = \{|\alpha(n)| - c \parallel \alpha(n)\}_n$  é um conjunto de Kraft-Chaitin.

Adicionalmente temos  $\sigma_n = K(\alpha(n)) + (|\alpha(n)| - c) - |\alpha(n)| = K(\alpha(n)) - c$ ; portanto  $\sum_n 2^{-\sigma(n)} = 2^c \sum_{y \in U_c} 2^{-K(y)} < \infty$ . Consequentemente  $U'_c$  é suficientemente aleatório.

O seguinte resultado ilustra a intuição por detrás da noção de KC-conjunto suficientemente aleatório.

## 92 FACTO

*Seja  $A$  um conjunto. Se existir  $d$  tal que  $R = \{d(n) \parallel A \upharpoonright n\}_{n \in \omega}$  é um KC-conjunto suficientemente aleatório, então existe um conjunto KLC-aleatório  $A'$  tal que  $K(A' \upharpoonright n) \leq K(A \upharpoonright n) + d(n) + O(1)$ , para todo  $n$ .*

**Prova** Seja  $\sigma(n) = K(A \upharpoonright n) + d(n) - n$ ; por hipótese  $R$  é suficientemente aleatório e, por isso,  $\sigma$  é  $\omega$ -similar. Então, pelo teorema de Miller (teorema 88, pag. 83), existe um conjunto KLC-aleatório  $A'$  tal que  $K(A' \upharpoonright n) \leq n + \sigma(n) + O(1)$ ; ou seja  $K(A' \upharpoonright n) \leq K(A \upharpoonright n) + d(n) + O(1)$

Este resultado ilustra como um conjunto  $A$  não aleatório (nomeadamente, é recursivo) pode “aleatorizar-se” à custa de uma sequência de incertezas  $d(n)$  que façam suficientemente aleatório o conjunto de Kraft-Chaitin definido pelas truncaturas de  $A$ ; é possível “randomizar” um conjunto  $A$ , mesmo que as complexidades  $K(A \upharpoonright n)$  sejam baixas,



juntando-lhes valores  $d(n)$ . A intuição é os  $d(n)$  representam a incerteza adicional que falta a  $A$  para ser aleatório; de certa forma  $d$  mede a “falta de aleatoriedade” de  $A$ .

O que este resultado não diz é qual é a relação entre os elementos do conjunto “pouco aleatório”  $A$  e os do conjunto aleatório  $A'$ .

### Aleatoriedade de Martin-Löf

A aleatoriedade de Martin-Löf baseia-se no conceito de “teste raro”. Intuitivamente um teste raro sobre conjuntos é uma propriedade que pode ser verificada de forma computacionalmente eficiente sobre qualquer conjunto  $A$ , que falha quase sempre mas não é impossível; isto é, são “raros” os conjuntos que verificam a propriedade mas existem sempre alguns.

Assim, dado um conjunto  $A$ , se for possível determinar um teste raro que  $A$  verifique, então é porque se conhece informação sobre  $A$  a partir da qual é possível fazer essa selecção de teste; nestas circunstâncias, o conjunto não será aleatório. Se não for possível escolher um tal teste, é porque não é possível encontrar padrões de regularidade em  $A$ ; isto é,  $A$  é aleatório.

Estas intuições formalizam-se na chamada **complexidade de Martin-Löf** e, para a formalizar, começamos por clarificar a nossa noção de “teste computável raro”.

□

Começamos por analisar a noção genérica de **teste**.

Classes efectivamente enumeráveis de conjuntos representam “continuações computáveis” de objectos previamente computados. Estas continuações são aproximações computáveis de objectos que, normalmente, não podem ser computados. Por isso faz sentido definir sequências de classes efectivamente enumeráveis representando sequências de aproximações sucessivas a objectos que podem ser descritos mas não computados.

### 93 NOÇÃO

Um **teste**  $\mathcal{T} = \{\mathcal{A}_n\}$  é uma sequência decrescente ( $\mathcal{A}_{n+1} \subseteq \mathcal{A}_n$ , para todo  $n$ ) de classes efectivamente enumeráveis de conjuntos.

Um teste  $\{\mathcal{A}_n\}$  é **uniformemente recursivo** se existe uma sequência uniforme de funcionais efectivas  $\{F_n\}$  de tal forma que cada uma das classes  $\mathcal{A}_n$  verifica  $\mathcal{W}_x \in \mathcal{A}_n \Leftrightarrow \varphi_x \in F_n \sim$ .

Uma classe  $U \subseteq 2^\omega$  **satisfaz** o teste  $\mathcal{T}$  quando  $U \subseteq \bigcap_n \mathcal{A}_n$ . Um conjunto  $A \in 2^\omega$ , satisfaz o teste  $\mathcal{T}$  quando a classe singular  $\{A\}$  satisfaz esse teste.

### 94 FACTO

A colecção de todos os testes uniformemente recursivos é enumerável.

#### Esboço de prova

Dado que cada sequência uniforme de funcionais efectivas é completamente determinada por uma função recursiva, a enumeração das funções recursivas induz a enumeração dos testes uniformemente recursivos.



No entanto testes genéricos, definidos desta forma não são mensuráveis porque as classes  $\mathcal{A}_n$  não são eventos. Por isso faz sentido substituir as diferentes classes  $\mathcal{A}_n$  por coberturas que possam ser mensuráveis.

### 95 NOÇÃO (TESTE DE MARTIN-LÖF)

Fixemos uma medida de Lebesgue  $\mu$ . Uma sequência de eventos recursivamente enumeráveis  $\mathcal{U} = \{U_k\}$  é um  **$\mu$ -teste de Martin-Löf (ML-teste)** se, para todo  $k$ ,  $\lambda(U_k) > k$  e existe um teste uniformemente recursivo  $\mathcal{T} = \{\mathcal{A}_k\}$  que é coberto por  $\mathcal{U}$ ; isto é, para todo  $k$ ,  $\mathcal{A}_k \subseteq U_k$ .

Convém focar nas três componentes essenciais desta definição:

- Cada classe  $\mathcal{A}_k$  é efectivamente enumerável. Isto significa que existe uma “array”  $A_k$  que a representa e que os conjuntos na colecção são exactamente os recursivamente enumeráveis que contém um conjunto finito indexado por esse “array”.
- Os teste de Martin-Löf são uniformemente recursivos. Isto implica, imediatamente, que existe uma enumeração de todos estes testes.
- Porque  $\mathcal{A}_k$  é coberto por  $U_k$  tem-se  $\lambda(\mathcal{A}_k) > k$ . Assim a medida de Hausdorff de  $H(\mathcal{A}_k)$  está limitada a  $2^{-\lambda(\mathcal{A}_k)} < 2^{-k}$ .

### 96 FACTO

Se  $\mathcal{T} = \{\mathcal{A}_k\}$  é um teste uniformemente recursivo tal que  $\lambda(\mathcal{A}_k) > k$ , para todo  $k$ , então existe um teste de Martin-Löf  $\mathcal{U} = \{U_k\}$  que cobre  $\mathcal{T}$ .



**Notas**

O menor que evento que cobre cada  $\mathcal{A}_k$  é a sua cobertura mínima  $\hat{\mathcal{A}}_k$ ; assim, se for  $\lambda(\mathcal{A}_k) > k$ , tem-se também  $\lambda(\hat{\mathcal{A}}_k) > k$ .

No entanto não é possível garantir que estas coberturas mínimas sejam recursivamente enumeráveis.

Recordemos, porém, que a cobertura mínima é sempre construída como o limite de  $n$ -coberturas  $\hat{\mathcal{A}}_k = \bigcap_n \mathcal{A}_k^{(n)}$ , e que todas estas  $n$ -coberturas são recursivamente enumeráveis. Se for  $\lambda(\hat{\mathcal{A}}_k) > k$  então tem de existir algum  $n$  tal que  $\lambda(\mathcal{A}_k^{(n)}) > k$ .

Definindo  $U_k = \mathcal{A}_k^{(n)}$ , para um  $n$  tal que  $\lambda(\mathcal{A}_k^{(n)}) > k$  obtemos o desejado teste de Martin-Löf.

**97 NOÇÃO (ALEATORIEDADE DE MARTIN-LÖF)**

Fixe-se uma medida de Lebesgue  $\mu$ . Uma classe  $\mathcal{A} \subseteq 2^\omega$  diz-se  $\mu$ -**Martin-Löf nula (ML  $\mu$ -nula)** se existe algum ML  $\mu$ -teste  $\{\mathcal{U}_n\}$  tal que  $\mathcal{A} \subseteq \bigcap_n \mathcal{U}_n$ . Um conjunto  $A$  diz-se **ML  $\mu$ -aleatória** quando a classe  $\{A\}$  não for ML  $\mu$ -nula; isto é, quando  $A$  não verifica qualquer ML  $\mu$ -teste.

Uma análise detalhada das noções de teste e aleatoriedade de Martin-Löf, assim como a relação destes conceitos com outras noções de aleatoriedade, pode ser vista no excelente artigo de Downey et al.<sup>19</sup>.

Os conceitos de KLC-aleatoriedade e ML-aleatoriedade estão intimamente ligados. De facto

<sup>19</sup>Rod Fowney, Denis R. Hirschfeldt, André Nies, Sebastain Terwijn, CALIBRATING RANDONNESS, The Bulletin of Symbolic Logic, Vol. 12, Nº 3, Setembro 2006.

## 98 TEOREMA (SCHORR)

Seja  $\mu$  a medida standard de Lebesgue; um conjunto é KLC-aleatório se e só se for ML  $\mu$ -aleatório.

**Esboço de Prova**

Para vermos a intuição por detrás desta relação entre estas duas abordagens à aleatoriedade, considere-se um qualquer teste de Martin-Löf na medida standard de Lebesgue. O teste é completamente determinado pela classe uniformemente enumerável  $\mathcal{U} = \{U_k\}$  de apresentações  $U_k$ , que são livres de prefixos, de tal forma que  $\sum_{y \in U_k} 2^{-|y|} < 2^{-k}$ .

Nesta representação, um conjunto  $A$  verifica o teste sse, para todo  $k$  existe um  $n$  tal que  $A \upharpoonright n \in U_k$ . Define-se, agora, um conjunto  $R$  tal que

$$d \parallel y \in R \Leftrightarrow k > 0 \wedge d = |y| - k \wedge y \in U_{2^k}$$

É fácil verificar que  $R$  é recursivo (já que os  $U_k$  formam uma sequência uniformemente recursiva) e que o seu peso é

$$\sum_{k>0} \sum_{y \in U_{2^k}} 2^{-(|y|-k)} = \sum_{k>0} 2^k \left( \sum_{y \in U_{2^k}} 2^{-|y|} \right) < \sum_{k>0} 2^k 2^{-2^k} < 1$$

Portanto  $R$  é um conjunto de Kraft-Chaitin e, pelo teorema 90, existe uma constante  $c$  tal que, para todo  $k > 0$  e todo  $y \in U_{2^k}$ ,  $K(y) \leq |y| - k + c$ ; equivalentemente  $K(y) - |y| \leq c - k$ .

Se  $A$  for KC-aleatório, e se for  $y = A \upharpoonright n$ , existe um  $k$  tal que  $K(y) - |y| > c - k$ . Por isso não podemos ter  $A \upharpoonright n = y$  para nenhum  $y \in U_{2^k}$  e, conseqüentemente, o teste falha. Ou seja, **KLC-aleatoriedade implica ML-aleatoriedade**.



A prova inversa é uma simples consequência do facto 84 cuja prova se repete aqui. Para cada  $k$  considere-se a linguagem  $U_k$  definida por

$$U_k = \{ y \mid K(y) \leq |y| - k \wedge (\forall z < y)[K(z) > |z| - k] \} \quad (41)$$

Pelo teorema 76  $U_k$  é recursivamente enumerado; claramente  $U_k$  é livre de prefixos. Portanto a sequência de linguagens  $\{U_k\}_{k \in \omega}$  é uniformemente recursiva. Como a indexação é livre de prefixos, tem-se  $\sum_{y \in U_k} 2^{-K(y)} \leq 1$ ; logo

$$\mu(\uparrow U_k) = \sum_{y \in U_k} 2^{-|y|} \leq \sum_{y \in U_k} 2^{-(K(y)+k)} \leq \sum_{y \in U_k} 2^{-K(y)} \cdot 2^{-k} \leq 2^{-k}$$

Portanto a sequência  $\{\uparrow U_k\}$  é um teste de Martin-Löf. Se  $A$  for ML-aleatória tem-se  $A \notin \bigcap_k \uparrow U_k$ . Isto significa que existe um  $k$  tal que  $A \notin \uparrow U_k$ . Ou seja, nenhum prefixo de  $A$  pode pertencer a  $U_k$ ; isto é  $(\forall n)[A \upharpoonright n \notin U_k] \equiv (\forall n)[K(A \upharpoonright n) > n - k]$ . Mas isto significa que  $A$  é KLC-aleatório. Portanto: *ML-aleatoriedade implica KLC-aleatoriedade*.

## Aleatoriedade de Shannon-Schorr

O terceiro paradigma de aleatoriedade designa-se por **paradigma da imprevisibilidade** e invoca uma intuição muito óbvia: *se uma string infinita  $u$  é aleatória, o conhecimento dos seus  $n$  primeiros bits, não fornece qualquer informação que permita prever o bit seguinte*.

Esta intuição pode ser concretizada através de uma noção de “estratégia ganhadora de aposta”: um jogador define uma regra que, conhecendo o passado  $x$  de resultados do jogo, lhe permite dividir o lucro que acumulou em  $x$  pelas hipóteses de jogo que podem continuar  $x$ .





No caso do jogo ser binário (lançar uma moeda ao ar, por exemplo), teremos uma relação da forma

$$\text{lucro}(x) = \text{aposta}(x 0) + \text{aposta}(x 1)$$

Como o resultado do jogo é binário, só faz sentido ter  $\text{lucro}(x) \leq 2 \text{aposta}(x)$  (o lucro está limitado ao dobro da aposta). A generalização é óbvia

### 99 NOÇÃO (MARTINGALES)

Uma função  $d: \varpi \rightarrow \mathbb{R}_+$  é um **super-martingale** se, para todo  $x \in \varpi$ ,

$$2 d(x) \geq d(1||x) + d(0||x)$$

Quando a igualdade  $2 d(x) = d(1||x) + d(0||x)$  se verifica para todo  $x$ ,  $d$  designa-se por **martingale**.

$d$  **sucede** em  $A \in 2^\varpi$ , e escreve-se  $A \sqsubseteq d$ , se  $\lim_{n \rightarrow \infty} \sup d(A \upharpoonright n) = \infty$ ;  $d$  sucede numa classe  $\mathcal{A}$  de conjuntos se sucede em todos conjuntos dessa classe.

É muito importante notar-se que cada valor  $d(A \upharpoonright n)$  é um real positivo e, por isso, só são conhecidos os seus limites superiores racionais. Daí o limite  $\lim_{n \rightarrow \infty} \sup d(A \upharpoonright n)$  se aplicar aos seus maximizantes racionais.

Os martingales são super-martingales que representam “estratégias ganhadores eficientes”: todo o lucro é investido em apostas e o lucro é sempre o dobro da aposta ganhadora. A sua importância deriva do seguinte teorema:



## 100 TEOREMA (VILLE)

Uma classe  $\mathcal{A} \subseteq 2^{\omega}$  é Lebesgue nula (é coberta por um evento nulo) se e só se existe um martingale  $d$  que sucede em  $\mathcal{A}$ .

A definição genérica de super-martingale (ou de martingale) não dá qualquer informação sobre as características computacionais destes objectos. Obviamente, só faz sentido uma estratégia ganhadora que possa ser computada!

Em primeiro lugar os diversos reais  $\{d(x)\}$  devem se reais recursivos. Isto significa que cada um destes valores é determinado por um conjunto recursivo e limitado de racionais; como existe uma correspondência biunívoca entre racionais positivos e naturais, cada  $d(x)$  identifica-se com um conjunto recursivo de naturais: os códigos desses racionais. Exige-se algo mais: deve existir uma forma computacionalmente relevante de gerar todos os conjuntos  $d(x)$ . Estas considerações motivam a seguinte definição:

## 101 NOÇÃO (MARTINGALE EFECTIVO)

Um super-martingale (ou martingale)  $d$  diz-se **efectivo** (ou **computacionalmente enumerável**) de todo  $d(x)$  é um real recursivo e existe uma função recursiva  $f$  tal que  $d(x) = \mathcal{W}_{f(x)}$ .

**EXEMPLO 10:** Considere-se a seguinte função  $d: 2^{<\omega} \rightarrow \mathbb{R}_+$

$$d(x) = \sum_{y \leq x} 2^{|y|-K(y)} + \sum_{y > x} 2^{|x|-K(y)} \quad (42)$$



Note-se, em primeiro lugar, que  $d(x)$  é um real recursivo que depende recursivamente de  $x$ ; basta notar que a 1ª parcela é uma soma finita que depende recursivamente de  $x$ , que a 2ª parcela  $2^{|x|} \sum_{y>x} 2^{-K(y)} \leq 2^{|x|}$ . Adicionalmente, calculando  $d(x0)$  e  $d(x1)$ , é fácil verificar  $2d(x) = d(x0) + d(x1)$ . Portanto  $d(\cdot)$  é um martingale efectivo.

Considere-se de novo os conjuntos  $U_c = \{x \mid K(x) \leq |x| - c\}$ . Uma vez que  $d(x) \geq 2^{|x|-K(x)}$  (basta seleccionar em (42) só a parcela  $y = x$ ), então, para todo  $x \in U_c$ , tem-se  $d(x) \geq 2^c$ . Tome-se agora um qualquer conjunto  $A$  não-aleatório; pela definição sabemos que, para todo  $c$ , existe um  $n$  tal que  $A \upharpoonright n \in U_c$ . Portanto, para todo  $c$ , existe um  $n$  tal que  $d(A \upharpoonright n) \geq 2^c$ ; logo  $\lim_n d(A \upharpoonright n) = \infty$ ; i.e.  $d$  sucede em  $A$ .

Alternativamente, considere-se agora um qualquer conjunto  $A$  aleatório; tome-se  $x = A \upharpoonright n$ , par um qualquer  $n$ . Note-se que  $y \leq x$  sse  $y$  for da forma  $A \upharpoonright k$  com  $k \leq n$ . Então tem-se  $d(A \upharpoonright n) \geq \sum_{k \leq n} 2^{k-K(A \upharpoonright k)}$ . Portanto  $\lim_{n \rightarrow \infty} d(A \upharpoonright n) \geq \sum_{n \in \omega} 2^{n-K(A \upharpoonright n)}$ .

□

Temos agora o resultado definitivo que ajuda a fechar a comparação entre estes três paradigmas.

## 102 TEOREMA (SCHORR)

*Um classe de  $\mathcal{A}$  é Martin-Löf nula se e só se existe um super-martingale efectivo  $d$  que sucede em  $\mathcal{A}$*

Como corolário óbvio



- 103 COROLÁRIO *Um conjunto  $A$  será ML-aleatório (e também KLC-aleatório) se e só se não existe nenhum supermartingale efectivo que sucede em  $A$ .*

## 1.6 Comparar a Aleatoriedade

Na secção anterior consideramos vários pontos de vista sobre a aleatoriedade de um conjunto  $A$  e estudou-se as possíveis equivalências entre estas pontos de vista. A pergunta essencial era sempre: *dado um conjunto  $A$ , será que ele é aleatório?*

Porém as simples respostas a esta pergunta dão, frequentemente, informação insuficiente. Normalmente quer-se ter alguma forma de “grau de aleatoriedade” que permita medir quão aleatório é  $A$ . Ou, pelo menos, poder comparar a aleatoriedade de dois conjuntos  $A$  e  $B$ , testando se um é mais ou menos aleatório que o outro ou se são incomparáveis em termos de aleatoriedade. Dar algumas respostas a estas questões é o objectivo desta secção.

### Aleatoriedade e os graus de Turing

Parece intuitivo que conjuntos aleatórios não podem ser efectivamente computáveis. A intuição é mais difícil quando se lida com semi-decidibilidade: *“será que um conjunto aleatório pode ser recursivamente enumerável?”*. Esta questão pode-se colocar em termos de uma mais geral: *“será que existem conjuntos KLC-aleatórios  $A$  que sejam Turing redutíveis a algum conjunto recursivamente enumerado?”* e a sua resposta foi dada por Kučera

#### 104 TEOREMA (KUČERA)

*Se  $A$  tem grau recursivamente enumerável (i.e. é Turing redutível a algum conjunto recursivamente enumerado) e é KLC-aleatório, então, forçosamente,  $A \equiv_T \mathcal{K}$ .*



Recordemos que  $\mathcal{K}$  é o conjunto recursivamente enumerável (mas não recursivo) dado por  $x \in \mathcal{K} \Leftrightarrow \varphi(x||x)\downarrow$ . Vimos que não existência de uma enumeração recursiva para o complemento de  $\mathcal{K}$  prende-se directamente com o problema da indecidibilidade da paragem em máquinas de Turing e, por isso,  $\mathcal{K}$  pode ser visto como o “menos previsível” de todos os conjuntos recursivamente enumerados.

Este resultado diz-nos que qualquer conjunto aleatório  $A$  que tenha grau r.e. se comporta (em termos de eventuais reduções recursivas) exactamente como  $\mathcal{K}$ . Nomeadamente, dado que qualquer conjunto recursivamente enumerável  $R$  verifica  $R \leq_T \mathcal{K}$ , também verificará  $R \leq_T A$ .

### Comparação de Kolmogorov

Pode-se comparar a aleatoriedade de dois conjuntos  $A$  e  $B$  comparando os seus graus de compressibilidade através da comparação das complexidades de Kolmogorov dos seus diversos prefixos.

#### 105 NOÇÃO (REDUÇÃO DE TURING)

O conjunto  $A$  é ***K-reduzível*** ao conjunto  $B$ , e escreve-se  $A \leq_K B$ , quando  $K(A \upharpoonright n) = O^{\log}(K(B \upharpoonright n))$ . Isto é, quando  $(\forall_{\infty} n) [K(A \upharpoonright n) \leq K(B \upharpoonright n) + O(1)]$ . Se  $A \not\leq_K B$  e  $B \not\leq_K A$ , os conjuntos dizem-se ***K-incomparáveis*** e escreve-se  $A \not\leq_K B$ .

Esta relação de ordem permite, desde já, uma outra questão: *que construções de conjuntos preservam aleatoriedade?*



Usaremos a notação  $A \uplus B$  para designar a união disjunta<sup>20</sup> de conjuntos definida como

$$A \uplus B = \{2x \mid x \in A\} \cup \{2x + 1 \mid x \in B\} \quad (43)$$

O seguinte teorema relaciona a aleatoriedade de  $A$  e de  $B$  com a da sua união disjunta.

106 TEOREMA (MILLER, VAN LANBALGEN, KAUTZ, KUCERA)

*Se  $A \uplus B$  é KLC-aleatório, então ambos  $A$  e  $B$  são KLC-aleatórios. Adicionalmente  $A$  é KLC-aleatório relativo a  $B$  (e  $B$  é KLC-aleatório relativo a  $A$ ) e ainda tanto  $A$  como  $B$  são  $K$ -incomparáveis com  $A \uplus B$ .*

□

Outra abordagem à comparação entre conjuntos recorre à sua interpretação como reais e é devida, principalmente, ao trabalho de Solovay.

Recordemos que cada  $u \in 2^{<\omega}$  pode ser vista como um racional  $\sum_{u_i=1} 2^{-i-1}$ . Desta forma se, como strings, for  $u \leq v$  ( $u$  é um prefixo de  $v$ ), então na ordem usual dos racionais também se tem  $u \leq v$ . Equivalentemente, cada  $x \in \omega$  está associado ao conjunto finito  $D_x$  e ao racional  $\sum_{i \in D_x} 2^{-i-1}$ .

<sup>20</sup>Os textos de computabilidade usam, normalmente, a notação  $A \oplus B$  para designar união disjunta; no entanto, neste curso, a notação  $x \oplus y$  será usada para designar o **xor** bit a bit de duas strings. A string que representa  $A \uplus B$  é, ao invés, formada pelo “merge” alternado dos bits de  $A$  e de  $B$ .



Um conjunto  $A$  pode ser visto como o real que é limite da sequência crescente de racionais  $\{A \upharpoonright n\}_n$ . Recordemos também (ver noção 60, página 59) que um **real recursivo** é o limite de uma sequência recursiva crescente de racionais.

O real  $A$  é **fortemente recursivo** se a sequência  $\{A \upharpoonright n\}_n$  é recursiva. Note-se que qualquer real fortemente recursivo é, obviamente, recursivo mas o inverso não é verdade: existem reais recursivos que não são fortemente recursivos. Para ilustrar esta diferença, e como consequência do teorema 59 (página 58), temos o seguinte resultado.

No que se segue,  $\varphi$  é uma função universal livre de prefixos, tida como referência, e  $\{\varphi_e\}_{e \in \omega}$  é a enumeração, induzida por  $\varphi$ , das funções parciais com domínios livres de prefixos.

### 107 PROPOSIÇÃO

*$A$  é um real recursivo se e só se for da forma  $\sum_{\varphi_e(x) \downarrow} 2^{-|x|}$  para algum programa  $e$ .*<sup>21</sup>

#### Esboço de prova

Se  $A$  for o limite de uma sequência recursiva e crescente de racionais  $\{r_n\}$ , seja  $\{l_n\}$  a sequência crescente de racionais de Lebesgue que verifica  $l_n \leq r_n < l_n + 2^{-n}$  e  $|l_n| = n$ . Claramente a sequência  $\{l_n\}$  é recursiva e tem  $A$  como limite; a partir desta sequência é possível construir uma sequência recursiva  $\{d_n\}$  tal que a série  $\sum_n 2^{-d_n}$  tem exactamente o mesmo limite  $A$ . Usando o facto 73, página 75 conclui-se que  $A$  é a medida de um domínio livre de prefixos de uma função parcial recursiva  $\varphi_e$ . Inversamente, dada uma linguagem recursiva livre de prefixos  $L$ , a sua medida é claramente um real recursivo.

<sup>21</sup>Equivalentemente, se for a medida do domínio de uma máquina de Turing livre de prefixos.





Como consequência deste resultado faz sentido introduzir a seguinte notação:

### 108 DEFINIÇÃO

Para uma função universal livre de prefixos  $\varphi$ ,  $\Omega \doteq \sum_{\varphi(x)\downarrow} 2^{-|x|}$  é o real recursivo definido por  $\varphi$  e que se designa por  **$\Omega$ -conjunto de Chaitin**. Genericamente, para um índice  $e$ ,  $\Omega_e \doteq \sum_{\varphi_e(x)\downarrow} 2^{-|x|}$  é o real recursivo determinado pela função  $\varphi_e$ .

A proposição 107 diz que todo o real recursivo é da forma  $\Omega_e$  para algum índice  $e$ . Nomeadamente os reais recursivos são recursivamente enumeráveis.

Note-se que  $\sum_{\varphi(x)\downarrow} 2^{-|x|}$  e  $\sum_{\varphi_e(x)\downarrow} 2^{-|x|}$  representam as medidas das linguagem livres de prefixos que são os domínios de  $\varphi$  ou  $\varphi_e$ : tem-se  $\Omega = \mu(\text{dom}(\varphi))$  e, genericamente,  $\Omega_e = \mu(\text{dom}(\varphi_e))$ . Se  $M_e$  for uma qualquer máquina de Turing que compute  $\varphi_e$ , o real  $\Omega_e$  mede a probabilidade de uma bit-strings infinita arbitrária ser aceite por tal máquina. Por isso  $\Omega_e$  é designada por **probabilidade de paragem** do programa  $e$ . Em particular  $\Omega$  é a **probabilidade de paragem** da máquina de Turing universal livre de prefixos ou, simplesmente, a probabilidade de paragem.

Limitando a recursividade, obtém-se variantes dos reais de Chaitin. Sendo  $\varphi^l$  a função universal livre de prefixos obtida de  $\varphi$  com recursividade limitada a  $l$  passos, então,  $\Omega^l \doteq \mu(\text{dom}(\varphi^l))$  e  $\Omega_e^l \doteq \mu(\text{dom}(\varphi_e^l))$ .

Na teoria clássica da computabilidade o grau r.e. de  $\mathcal{K} = \{x \mid \psi_x(x)\downarrow\}$  não depende da enumeração aceitável



$\{\psi_e\}$  das funções parciais recursivas; isto porque existe um teorema que diz que qualquer outro sistema aceitável  $\{\psi'_e\}$  determina um  $\mathcal{K}'$  que é Turing-equivalente a  $\mathcal{K}$  e, por isso, tem o mesmo grau.

Na aleatoriedade algorítmica,  $\Omega$  depende da função universal  $\varphi$  tomada por referência. Por isso, para sermos precisos, deve-se dizer que determina uma probabilidade de paragem. Para que  $\Omega$  determine a probabilidade de paragem, é necessário um resultado que estabeleça uma redução forte entre os diversos  $\Omega$  definidas pelas diversos candidatos a funções universais livres de prefixos. Isto exige uma nova noção de redução algorítmica.

Antes, porém, convém apresentar um resultado que está na origem da importância atribuída ao real  $\Omega$ .

109 LEMA  $\Omega$  é KLC-aleatório.

### Esboço de prova

Pela definição de KLC-aleatoriedade necessitamos de provar que existe um  $c > 0$  tal que, para todo  $n$ ,  $K(\Omega \upharpoonright n) \geq n - c$ . Equivalentemente, que existe um  $c > 0$  tal que, para todo  $n$  e todo  $x \in \text{dom}(\varphi)$  com  $|x| < n - c$ , se tem  $\varphi(x) \neq \Omega \upharpoonright n$ .

Pelo teorema 104, página 100,  $\Omega$  é Turing equivalente a  $\mathcal{K}$ ; nomeadamente, isto significa que, para todo  $n$  existe um limite à recursividade  $l$  tal que o prefixo  $\Omega \upharpoonright n$  está no contradomínio de  $\varphi^l$ ; i.e., para algum  $x$  tem-se  $\varphi^l(x) \simeq \Omega \upharpoonright n = \Omega^l \upharpoonright n$ . Vamos definir um programa  $e$  tal que  $\varphi_e(x) \notin \text{rng}(\varphi^l)$  para todo  $x$  tal que  $|x| < n - c$  sendo  $c = |e| + K(e)$ ; equivalentemente, quando  $|e||x| < n$ , temos  $|e||x| \notin \text{dom}(\varphi^l)$ . Isto significa que  $\Omega - \Omega^l \geq 2^{-|e||x|} > 2^{-n}$  e, portanto,  $\Omega \upharpoonright n \neq \Omega^l \upharpoonright n$ . Assim  $|x| < n - c$  implica que  $\varphi(x) \neq \Omega \upharpoonright n$ .



Este lema mostra que o real recursivo  $\Omega$  é aleatório<sup>22</sup>; inversamente gostaríamos de mostrar que todo o real recursivo que seja aleatório é, de alguma forma, redutível a  $\Omega$ . Para isso precisamos de uma noção de redução em que, nos reais,  $\Omega$  tenha propriedades semelhantes às que  $\mathcal{K}$  tem nos graus de Turing.

#### 110 NOÇÃO (REDUÇÃO DE SOLOVAY)

Diz-se que o real  $A$  é **Solovay-redutível** ao real  $B$  (equivalentemente,  $B$  **domina**  $A$ ), e escreve-se  $A \leq_S B$ , quando existe uma constante  $C > 0$  e uma função computável<sup>23</sup>  $f$  tais que, para todo racional  $q$ , se verifica  $A - f(q) < C(B - q)$ .

Essencialmente a definição diz-nos que, se existir uma sucessão crescente de racionais  $\{q_n\}$  que tenha  $B$  como limite, é possível construir uma sucessão  $\{f(q_n)\}$  que converge para  $A$  de forma tão ou mais rápida. Isto implica, nomeadamente, que  $A$  deve ser “menos aleatório” do que  $B$ . De facto Solovay provou

#### 111 LEMA (SOLOVAY) Se $A \leq_S B$ então, para todo $n$ , verifica-se $K(A \upharpoonright n) \leq K(B \upharpoonright n) + O(1)$ .

O facto de reais recursivos se identificarem com probabilidades de paragem de máquinas de Turing livres de prefixos permite obter uma forma explícita para esta relação quando aplicada a reais recursivos.

<sup>22</sup>No que se segue, a menos que seja explícito o modelo de aleatoriedade usado, “aleatório” significa indistintamente KLC-aleatório, Martin-Löf aleatório ou Shannon aleatório.

<sup>23</sup>Calculada dentro de uma determinada classe de complexidade de máquinas de Turing; por exemplo, como é convencional em Criptografia, a classe **PPT**.

## 112 TEOREMA

Sejam  $\Omega_a, \Omega_b$  reais recursivos determinados por programas  $a$  e  $b$  respectivamente. Então verifica-se  $\Omega_a \leq_S \Omega_b$  sse existe uma função total computável  $f$  e uma constante  $C > 0$  tais que,

$$(\forall l) \quad \left[ \Omega_a - \Omega_a^{f(l)} < C (\Omega_b - \Omega_b^l) \right] \quad (44)$$

113 COROLÁRIO Se  $A$  e  $B$  são limites de seqüências crescentes e computáveis de racionais  $\{p_n\}$  e  $\{q_n\}$ , respectivamente, então  $A \leq_S B$  sse existe uma função total computável  $f$  e uma constante  $C > 0$  tais que

$$(\forall n) \quad \left[ A - p_{f(n)} < C (B - q_n) \right] \quad (45)$$

Todos estes resultados traduzem a noção fundamental da redução de Solovay  $A \leq_S B$ : a partir de uma qualquer aproximação computável a  $B$  é possível construir uma aproximação a  $A$  que converja tão ou mais rapidamente que a aproximação de  $B$ .

Adicionalmente, aplicada a reais recursivos, a redução de Solovay assume propriedades idênticas à da redutibilidade de Turing. Para tal necessitamos de uma noção análoga ao grau  $0'$  e um resultado análogo ao teorema de Post.

## 114 NOÇÃO

Um real recursivo  $A$  diz-se  $\Omega$ -**equivalente** (ou **Solovay-completo**) quando domina  $\Omega$  (i.e.  $\Omega \leq_S A$ ).



O seguinte resultado resume a relação entre a aleatoriedade dos reais recursivos e a redução de Solovay.

## 115 TEOREMA

$\Omega$  domina todo real recursivo  $A$ . Adicionalmente,

- (i) Todo o real recursivo  $\Omega$ -equivalente é a probabilidade de paragem de uma função universal livre de prefixos; portanto, é KLC-aleatório.
- (ii) Todo o real recursivo KLC-aleatório é  $\Omega$ -completo.

Este resultado é fundamental ao nosso estudo já que estabelece uma correspondência biunívoca entre quaisquer duas de três classes de entidades: os reais recursivos  $\Omega$ -completos, as probabilidades de paragem de máquinas universais de Turing livres de prefixos e reais recursivos KLC-aleatórios.

Um aspecto interessante é a relação entre a redução de Solovay e a adição de reais recursivos. O seguinte resultado resume as relações essenciais.

## 116 FACTO

Sejam  $A$  e  $B$  reais recursivos arbitrários. Então

1. Se for  $A + B = \Omega$  então pelo menos uma das parcelas,  $A$  ou  $B$ , é  $\Omega$ -equivalente.
2. Verifica-se  $A \leq_S B$  sse existe um real recursivo  $X$  e uma constante  $c$  tais que  $A + X = cB$ .



Uma das construções mais importantes em Criptografia é o de **geradores pseudo-aleatórios**; isto é, sequências de “bits” que, até um certo ponto, são indistinguíveis de uma sequência aleatória. Sequências diferentes são geradas por “seeds” diferentes.

A definição aqui apresentada é expressa em termos de reais-recursive; o gerador é uma sequência, indexada pelos “seeds”, de reais recursive. A noção de “indistingibilidade” é expressa limitando os níveis de recursividade com que as aproximações são feitas.

#### 117 NOÇÃO

Uma classe efectivamente computável de reais recursive  $\{\Omega_{e(s)}\}$  é um **gerador pseudo-aleatório** para a **função de expansão**  $L$ , se existe uma função computável  $f$  e uma constante  $C > 0$  tais que, para cada  $s$ ,

$$\forall l \leq L(|s|) \quad \left[ \Omega - \Omega^{f(l)} \leq C \left( \Omega_{e(s)} - \Omega_{e(s)}^l \right) \right] \quad (46)$$

Note-se que, para qualquer “seed”  $s$ , a quantificação nos níveis de recursividade  $l$  não é ilimitada; se o fosse para algum  $s$ , então esse  $\Omega_{e(s)}$  seria sempre  $\Omega$ -equivalente (teorema 115) e, portanto, realmente aleatório. A função  $L$  estabelece um limite nos níveis de recursividade que podem ser usados para comparar as razões de convergência para  $\Omega$  e  $\Omega_{e(s)}$  das respectivas aproximações.





Considere-se uma qualquer conjunto de Kraft-Chaitin  $X = \{d_n || x_n\}_{n \in \omega}$ ; a essência do teorema de Chaitin é que existe um programa  $p$  que determina uma máquina de Turing livre de prefixos  $M_p$  cujo domínio é enumerado por uma sequência recursiva  $\{u_n\}$  verificando  $|u_n| = d_n$  e  $M_p(u_n) \simeq x_n$ , para todo  $n$ .

Nomeadamente peso  $\sum_n 2^{-d_n}$  do KC-conjunto  $X$  coincide com o real recursivo  $\Omega_p$ .

Obviamente que vários programas  $p$  conduzem a máquinas de Turing  $M_p$  que satisfazem estas condições; nada se diz sobre a sua natureza ou a sua complexidade; pode acontecer, por exemplo, que uma máquina  $M_p$ , para cumprir estas condições, tenha uma complexidade temporal que é exponencial com o comprimento do “input”<sup>24</sup>. No entanto, independentemente do programa  $p$ , o real recursivo  $\Omega_p$  é sempre o mesmo e é determinado por  $X$ .

Por isso é importante identificarmos classes de complexidade nos conjuntos KC análogas às que são definidas para as máquinas de Turing. Assim define-se

## 118 NOÇÃO

Diremos que o conjunto KC  $X = \{|u_i| || x_i\}$  é **computável** se existe uma máquina de Turing efectivamente computável  $M_p$  que verifique  $(\forall i) [M_p(u_i) \simeq x_i]$  e  $M_p(u) \downarrow \Rightarrow (\exists i) [u = u_i]$ .

<sup>24</sup>Neste caso o comportamento aleatório dos “inputs” não permite, realisticamente, inferir qualquer comportamento das respostas  $\{x_i\}$ .



Se se convencionar, como é usual em análise de segurança, que “efectivamente computável” significa que a máquina de Turing está na classe **PPT**, então um conjunto KC será computável quando existir um algoritmo probabilístico, com tempo polinomial que, com margem de erro desprezável, rejeite qualquer “input” que não pertença à sequência  $\{u_i\}$  e compute, para todo  $i$ ,  $x_i$  a partir de  $u_i$ .

Intuitivamente este algoritmo estabelece uma ligação entre a aleatoriedade da sequência de respostas  $\{x_i\}$  e dos “inputs”  $\{u_i\}$ : se a classe de complexidade for suficientemente restrita, a primeira nunca é mais aleatória do que a segunda.

Vimos que o peso  $w(X)$  de um KC-conjunto  $X = \{|u_i| \parallel x_i\}$  identifica-se com um real recursivo  $\Omega_p$ ; inversamente qualquer real recursivo  $\Omega_e$  pode ser identificado com o conjunto de Kraft-Chaitin da forma  $\{|u| \parallel u\}_{u \in \text{dom}(\varphi_e)}$ .

Faz sentido, então, analisar a forma como a ordem  $\leq_S$  de Solovay nestes pesos induz uma forma de comparação da aleatoriedade associada a KC-conjuntos.

## 119 NOÇÃO

*Sejam  $X, Y$  conjuntos Kraft-Chaitin e  $\Omega_p, \Omega_q$  os reais recursivos que representam os respectivos pesos.  $X$  é **reduzível** a  $Y$ , e escreve-se  $X \leq Y$ , quando  $\Omega_p \leq_S \Omega_q$ .*

A redutibilidade de Solovay estabelece-se na forma como as aproximações a um real recursivo convergem. Quando os reais recursivos são pesos  $w(X)$ , as aproximações podem ser expressas de uma forma específica.





Defina-se  $w(X; n) = \sum_{i \geq n} 2^{-|u_i|}$ .

Note-se que  $w(X) - w(X; n) = \sum_{i < n} 2^{-|u_i|}$  é um racional que aproxima o real  $w(X) = \Omega_p$ ; assim  $w(X; n)$  pode ser visto como o erro dessa aproximação: avalia o erro que se comete quando, em  $X$ , se ignoram “inputs” (e respostas) de ordem  $\geq n$ . Obviamente que  $\lim_n w(X; n) = 0$ .

Defina-se também  $\lambda(X; n) = \max\{ \lambda \in \varpi \mid \sum_{i \geq n} 2^{\lambda - |u_i|} \leq 1 \}$ .

Tem-se sempre  $\lim_n \lambda(X; n) = +\infty$ . Pode-se interpretar  $\lambda(X; n)$  como um limite inferior para os comprimentos de dos “inputs”  $u_i$  que determinam as respostas  $x_i$  de ordem  $i \geq n$ ; dito de outro modo,  $\lambda(X; n)$  mede a incerteza nas respostas  $x_i$  de ordem  $i \geq n$ .

$$\varphi_e(u) \simeq x_i \wedge i \geq n \Rightarrow |u| \geq \lambda(X; n)$$

Equivalentemente, a informação necessária para determinar as respostas de ordem inferior a  $n$ , está concentrada nos “inputs”  $u$  de comprimento limitado a  $\lambda(X; n)$ . Nesta perspectiva, quanto mais a informação global sobre  $X$  estiver “concentrada” nas respostas até  $n$ , maior será o “comprimento”  $\lambda(X; n)$ .

Considere-se agora um segundo conjunto Kraft-Chaitin  $Y = \{|v_i| | y_i\}$ . Como em qualquer KC-conjunto tem-se também  $\lim_n w(Y; n) = 0$  e  $\lim_n \lambda(Y; n) = +\infty$ . Comparando as razões de convergência, para zero dos pesos  $w(X; n)$  e  $w(Y; n)$  ou para  $+\infty$  dos comprimentos  $\lambda(X; n)$  e  $\lambda(Y; n)$ , é possível relacionar a aleatoriedade dos dois KC-conjuntos. Do corolário 253 conclui-se

## 120 FACTO

Verifica-se  $X \leq Y$  se e só se existe uma função computável  $f$  tal que, para todo  $n$ ,

$$\lambda(X; f(n)) \geq \lambda(Y; n) + O(1)$$

A relação de redução  $\leq$  é uma ordem parcial e, naturalmente, define classes de equivalências que serão designados por KC-**graus**;  $[X]$  denota o KC-grau que contém  $X$ . Diz-se que  $X$  é  $\Omega$ -**equivalente** quando  $\Omega \leq X$ ; assim, o KC-grau formado pelos KC-conjuntos  $\Omega$ -equivalentes é representada por  $[\Omega]$ .

□

A relação de redução de conjuntos Kraft-Chaitin pode ser transferida para sequência de respostas.

Note-se que vários KC-conjuntos  $X = \{d_i || x_i\}$  podem ter a mesma sequência  $\alpha = \{x_i\}$  de respostas. Também, como as sequências de respostas de um KC-conjunto é obrigatoriamente recursiva, é natural que seja necessário lidar com sequências que não coincidem exactamente com as respostas de nenhum KC-conjunto; por exemplo, a sequência  $\{A \upharpoonright n\}$  das truncaturas de um conjunto aleatório  $A$  não é recursiva.

Por isso convém estabelecer explicitamente uma relação entre sequências e KC-conjuntos.

## 121 NOÇÃO

Uma sequência  $\alpha = \{x_n\}_{n \in \omega}$  é **prevista** pelo conjunto de Kraft-Chaitin  $X$  quando for uma subsequência da



sequência de respostas de  $X$ .

O **fecho computável**, representado por  $[[\alpha]]$ , é a classe de todos os KC-conjuntos computáveis que prevêm  $\alpha$ .

Pode-se agora estabelecer uma relação de redução entre sequências (recursivas ou não).

## 122 NOÇÃO

A sequência  $\alpha$  é **reduzível** à (ou menos **aleatória** do que a) sequência  $\beta = \{y_n\}_{n \in \omega}$ , e escreve-se  $\alpha \preceq \beta$ , quando, para todo o KC-conjunto computável  $Y$  que prevê  $\beta$ , existe um KC-conjunto computável  $X$  que prevê  $\alpha$  e é reduzível a  $Y$ .

A sequência  $\alpha$  é  **$\Omega$ -completa** quando todo o KC-conjunto computável que preveja  $\alpha$  é  $\Omega$ -completo.

A definição anterior pode-se reescrever usando fechos computáveis.

$$\alpha \preceq \beta \Leftrightarrow (\forall Y \in [[\beta]]) (\exists X \in [[\alpha]]) [X \preceq Y] \quad (47)$$

$$\alpha \text{ é } \Omega\text{-completa} \Leftrightarrow [[\alpha]] \subseteq [\Omega] \quad (48)$$

De forma natural se introduz a noção de “pseudo-completude” de KC-conjuntos e sequências substituindo  $\Omega$  por um gerador pseudo-aleatório  $\{\Omega_{e(s)}\}$  e quantificando em relação a todas as “seeds” de tamanho limitado.



## 1.7 Incerteza

Na noção 81, página 79 vimos duas formas simples para avaliar a incerteza de um conjunto  $A$ . Vimos a **incerteza de Hartley** (ou simplesmente **incerteza**) de um conjunto  $A \neq \emptyset$ , representada por  $\vartheta(A)$ , definida por  $\vartheta(A) = \log_2 \llbracket A \rrbracket$ , sendo  $\llbracket \cdot \rrbracket$  a função recursiva que mede a cardinalidade de um conjunto. Esta incerteza só faz sentido em conjuntos finitos. Vimos também como a **incerteza uniforme** contorna este problema, apresentando-se como uma funcional  $\vartheta[\cdot]$  que, a cada conjunto  $A$ , associa uma função  $\vartheta[A]$  tal que  $\vartheta[A](n) \simeq \vartheta(A \upharpoonright n)$ .

Outra forma de incerteza é a **incerteza limite**, representada por  $\vartheta_l[A]$  e já aplicável a conjuntos infinitos, definida como o tamanho do menor elemento de  $A$ ; isto é,  $\vartheta_l[A] = \min_{x \in A} |x|$ .

Em qualquer destas formas de incerteza vamos convencionar que a incerteza de um conjunto vazio é  $-\infty$ .

A incerteza de Hartley  $\vartheta(A)$  depende apenas do número de elementos do conjunto  $A$ ; não depende dos elementos do conjunto. Desta forma a incerteza de um conjunto singular é sempre nula independentemente do inteiro que determina o conjunto. Já a incerteza limite desse conjunto singular depende do inteiro que o determina; de facto coincide com o tamanho desse inteiro.

O **tamanho** é uma outra medida aplicável apenas a conjuntos finitos; recorde-se que  $|A|$  é definido como o tamanho do índice canónico do conjunto; isto é,  $|A| = \log_2(1 + \sum_{a \in A} 2^a)$ . Ou  $|A| = |x|$  tal que  $A = D_x$ .

Em geral as várias medias de um conjunto finito (cardinalidade, incerteza e tamanho) produzem resultados distintos como se ilustra no seguinte exemplo.

### EXEMPLO 11:

Suponhamos que  $A$  é o conjunto formado por todos os inteiros no intervalo  $[0..n - 1]$ . A cardinalidade é  $\llbracket A \rrbracket = n$ . O código que o representa é  $\sum_{k=0}^{n-1} 2^k = 2^n - 1$  cujo tamanho é  $|A| = \log_2(1 + (2^n - 1)) = n$ . Portanto, neste caso,  $\llbracket A \rrbracket = |A| = n$  e  $\vartheta(A) = \lceil \log_2 n \rceil$ .

Suponhamos, agora, que  $n$  é par e o conjunto  $B$  é formado pelos inteiros pares no mesmo intervalo  $[0..n - 1]$ . Obviamente que  $\llbracket B \rrbracket = \llbracket A \rrbracket / 2$  e que  $\vartheta(B) \sim \vartheta(A) - 1$ . O código de  $B$  é  $\sum_{k=0}^{n/2-1} 2^{2k} = (2^n - 1) / 3$ ; o tamanho deste código é  $n - 2$ ; logo, neste caso,  $|B| = |A| - 2$  é distinto da cardinalidade  $\llbracket B \rrbracket$ .

Generalizando, pode-se verificar que se formarmos um conjunto dos múltiplos de  $p$  contidos em  $A$  (isto é  $\{ip \mid i = 0..n/p - 1\}$ ) temos uma incerteza que é aproximadamente  $\vartheta(A) - \log_2 p$  e um tamanho que é aproximadamente  $|A| - p$ .

A intuição por detrás da noção de incerteza pode ser vista por diferentes perspectivas das chamadas **propriedades discriminantes**

## 123 NOÇÃO

*Seja  $X$  um domínio e  $\sim$  uma relação de equivalência nesse domínio. Seja  $\mathcal{T} = \{\rho_e\}$  uma família de relações*



1. Sob o ponto de vista da **complexidade computacional**

## 1.8 Unidireccionalidade, Fuga de Informação e Geradores Pseudo-Aleatórios

Em Criptografia são essenciais as funções computáveis  $f: \varpi \rightarrow \varpi$  cuja inversa  $f^{-1}$  não seja computável; informalmente pode-se exprimir esta exigência da seguinte forma:

*dado um qualquer  $y$ , não deve ser computável decidir se  $f^{-1}(y) \neq \emptyset$  ou, caso esta condição se verifique, encontrar um  $x$  em  $f^{-1}(y)$ .*

Caso existam, essas funções são designada por **unidirecionais** (“one-way”).

A condição de unidireccionalidade pode ser refinada de vários modos; nomeadamente pode-se colocar uma condição de ausência de **fuga de informação** (“information leakage”) do “input” para o “output”. Informalmente existe

Vamos reformular a noção de teste de Martin-Löf em termos de funções parciais livres de prefixos. Vamos considerar, como referência, uma função universal livre de prefixos  $\varphi$ . Designaremos por  $\varphi_e$  e  $\mathcal{W}_e$ , respectivamente, a função universal livre de prefixos determinada pelo programa  $e$ , e o respectivo domínio.

### 124 NOÇÃO

*Um real recursivo  $A \in 2^\varpi$  é **aceite** por  $\varphi_e$  (ou  $\varphi_e$  **sucede** em  $A$ ), e escreve-se  $\varphi_e|A$ , quando algum prefixo de  $A$  pertence ao domínio de  $\varphi_e$ ; isto é  $\downarrow A \cap \mathcal{W}_e \neq \emptyset$  ou, equivalentemente,  $A \in \uparrow \mathcal{W}_e$ .*



Quando  $\varphi_e|A$  é decidível para todo o r.r.  $A$ , então  $\varphi_e$  diz-se uma **observação**.

Uma seqüência de observações uniformemente recursiva  $\tau = \{\varphi_{e(k)}\}_{k \in \omega}$  determina um **teste de Martin-Löf** quando, para todo  $k$ ,  $\mathcal{W}_{e(k+1)} \subseteq \mathcal{W}_{e(k)}$  e  $\lambda(\mathcal{W}_{e(k)}) > k$ .

$A \in 2^\omega$  **satisfaz** o teste  $\tau$ , e escreve-se  $\tau \| A$ , se e só  $\varphi_{e(k)}|A$ , para todo  $k$ .

A seqüência  $\alpha = \{x_n\}_{n \in \omega}$  **satisfaz** o teste  $\tau$ , e escreve-se  $\tau \| \alpha$ , quando para todo o KC-conjunto computável  $X$  que preveja  $\alpha$ , verifica-se  $\tau \| w(X)$ .

## 125 NOÇÃO

Dada uma função parcial recursiva  $f$ , uma **fuga de informação de “input”** (“inputs information leak”) para  $f$ , é uma par de testes de Martin-Löf,  $\langle \tau, \tau^f \rangle$  tais que, para todo o real recursivo  $A$ , verifica-se  $\tau \| A$  se e só se  $\tau^f \| f^{-1}(A)$ .

Dada uma função parcial recursiva  $f$ , uma **fuga de informação de “input”** (“inputs information leak”) para  $f$ , é uma par de testes de Martin-Löf,  $\langle \tau, \tau^f \rangle$  tais que, para toda a seqüência  $\alpha = \{x_n\}_{n \in \omega}$ , verifica-se  $\tau \| \alpha$  se e só se  $\tau^f \| f^{-1}(\alpha)$ .

$f$  é **fortemente unidirecional** se não tem quaisquer fugas de informação.





$f$  é  $\tau$ -**fortemente unidirecional** quando dado  $\tau$  é intratável encontrar uma fuga de informação com teste  $\tau$  no output.

## 2. Teoria dos Números

As técnicas criptográficas lidam essencialmente com domínios de informação finitos. Isto significa que a representatividade matemática dos items de informação vai ser realizada por domínios discretos de informação.

Domínios discretos e finitos são representáveis, em último caso, por conjuntos de *strings* de bits. Um outra representação possível assenta nos inteiros.

Ao longo deste capítulo procuraremos caracterizar alguns dos domínios matemáticos que têm estas características e que são amplamente usados nas técnicas criptográficas.

## 2.1 Divisibilidade

$\mathbb{Z}$  – conjunto dos inteiros com a estrutura algébrica de um **anel** com adição  $+$  e multiplicação  $\times$ .

$\mathbb{Z}_n$  – conjunto dos inteiros  $0, 1, \dots, (n - 1)$  com a estrutura de um **anel** com a adição  $+$  e multiplicação efectuadas módulo  $n$

EXEMPLO 12:  $\mathbb{Z}_5$

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\times$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

$\mathbb{Z}_2 = \{0, 1\}$  é um corpo particularmente importante pois representa a estrutura de uma álgebra booleana; as tabelas de adição e multiplicação são:

+	0	1
0	0	1
1	1	0

$\times$	0	1
0	0	0
1	0	1

Note-se que a multiplicação é equivalente à conjunção lógica **and** e a adição é equivalente à disjunção exclusiva **xor**.



$\mathbb{Z}_n^*$  – **grupo multiplicativo** formado por todos os elementos invertíveis de  $\mathbb{Z}_n$ ; i.e., elementos  $a \in \mathbb{Z}_n$  para os quais existe  $b \in \mathbb{Z}_n$  tal que  $a \times b = 1 \pmod{n}$ .

126 FACTO

$a \in \mathbb{Z}_n$  é invertível em  $\mathbb{Z}_n$  se e só se  $\gcd(a, n) = 1$

### Notas

- Sendo  $p$  primo todos os elementos de  $\mathbb{Z}_p$ , excepto 0, são invertíveis.
- $\mathbb{Z}_2^*$  é o conjunto singular  $\{1\}$  e representa o grupo multiplicativo mais simples (reduz-se à unidade).
- $\mathbb{Z}_9^* \equiv \{1, 2, 4, 5, 7, 8\}$  com  $2^{-1} = 5$ ,  $4^{-1} = 7$  e  $8^{-1} = 8$ .

A **ordem** de um grupo  $\langle \mathcal{G}, \cdot \rangle$  é o número de elementos do grupo e representa-se por  $|\mathcal{G}|$ .

O grupo é **cíclico**, quando existe um elemento  $g$  (dito **gerador** do grupo) tal que,  $\forall x \in \mathcal{G}$  existe um inteiro positivo  $n \in \mathbb{N}$  tal que  $x = g \cdot g \cdot g \dots \cdot g$  ( $n$  vezes).<sup>25</sup>

<sup>25</sup>Quando o grupo é aditivo é costume representar  $g \cdot g \cdot \dots \cdot g$  ( $n$  vezes) por  $ng$  e chama-se a esta operação, o **produto escalar discreto**. Quando o grupo é multiplicativo a mesma expressão representa-se por  $g^n$  e chama-se **exponenciação discreta**.



## 127 FACTO

Se  $\mathcal{G}$  é um grupo cíclico finito de gerador  $g$ , então

$$n = m \pmod{|\mathcal{G}|} \Leftrightarrow g^n = g^m$$

## EXEMPLO 13:

$\mathbb{Z}_9^*$  é um **grupo cíclico** de **ordem 6**.  
 2 e 5 são **geradores do grupo**  
 4 e 7 geram **subgrupos de ordem 3**

Exemplos de  $g^i \pmod{9}$

$i$	0	1	2	3	4	5
$g = 2$	1	2	4	8	7	5
$g = 4$	1	4	7	1	4	7
$g = 5$	1	5	7	8	4	2
$g = 7$	1	7	4	1	7	4

## 2.2 Resultados Fundamentais da Divisibilidade

128 TEOREMA (FUNDAMENTAL DA ARITMÉTICA)

Cada  $m > 1$  admite uma única **factorização**

$$m = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$$

em que  $1 < p_1 < p_2 < \cdots < p_k$  são primos.

129 TEOREMA (PEQUENO DE FERMAT)

Se  $p$  é primo então, para todo  $a \geq 0$ , verifica-se  $a^p = a \pmod{p}$ .

130 COROLÁRIO Se  $p$  é primo,  $a \in \mathbb{Z}_p^*$  e  $k \geq 0$ , então  $a^{p-k-1} = a^{-k}$  em  $\mathbb{Z}_p^*$ .

Se  $a^{-1}$  existe em  $\mathbb{Z}_p^*$  então, tomando congruências  $\pmod{p}$ , tem-se  $a^{-k} = a \cdot a^{-k-1} = a^p \cdot a^{-k-1} = a^{p-k-1}$ .

131 COROLÁRIO Se  $p$  é primo e  $k$  é um qualquer divisor de  $(p-2)$  então a função  $\alpha_k : x \mapsto x^k \pmod{p}$  é um automorfismo no grupo multiplicativo  $\mathbb{Z}_p^*$ .

Claramente  $\alpha_k$  preserva a a estrutura do grupo multiplicativo  $\mathbb{Z}^*$ . Basta provar, portanto, que é um isomorfismo.



Tomando sempre congruências  $(\text{mod } p)$ , suponhamos que existiam  $x, y \in \mathbb{Z}_p^*$  tais que  $x^k = y^k$ ; sendo  $(p-2)$  um múltiplo de  $k$  será também  $x^{(p-2)} = y^{(p-2)}$ ; pelo facto anterior,  $x^{-1} = x^{p-2}$  e  $y^{-1} = y^{p-2}$ ; conclui-se que  $x^{-1} = y^{-1}$  e, portanto,  $x = y$ .

## 132 DEFINIÇÃO

A **função de Euler-phi**, representada por  $\phi(\cdot)$ , associa a cada  $m > 1$  o número de inteiros  $0 \leq a < m$  tais que  $\text{gcd}(a, m) = 1$ .

## PROPRIEDADES DA FUNÇÃO DE EULER-PHI

- $\phi(m) = \prod_{i=1}^k p_i^{(e_i-1)} \times (p_i - 1)$
- $a^{\phi(m)} = 1 \pmod{m}$  (Teorema de Euler)
- Seja  $m = p * q$  o produto de dois primos distintos e  $\varphi = \phi(m)/2$

$$i = j \pmod{\varphi} \implies x^i = x^j \pmod{m}$$

Em particular (*Teorema do RSA*)

$$a = b^{-1} \pmod{\varphi} \implies (x^a)^b = x \pmod{m}$$

- $m = \sum_{a|m} \phi(a)$  ( $a|m$  representa a asserção " $a$  divide  $m$ ").

Note-se que o teorema RSA continua a verificar-se sempre que  $\varphi$  for um múltiplo comum de  $(p - 1)$  e  $(q - 1)$ .

Nomeadamente quando  $\varphi = \phi(m) = (p - 1) * (q - 1)$  e, ainda, quando  $\varphi = \text{lcm}(p - 1, q - 1) = (p - 1) * (q - 1) / \text{gcd}(p - 1, q - 1)$ .

#### EXEMPLO 14:

- $\phi(12) = \phi(2^2 \times 3) = 2^1 \times (2 - 1) \times (3 - 1) = 4$
- $5^4 = 625 = 52 \times 12 + 1 = 1 \pmod{12}$
- $15 = 3 \times 5$      $\phi(15) = 8$      $1025 = 1 \pmod{8}$  . Logo  $a^{1025} = a \pmod{15}$ .
- $\phi(1) + \phi(2) + \phi(3) + \phi(4) + \phi(6) + \phi(12) = 1 + 1 + 2 + 2 + 2 + 4 = 12$

#### 133 DEFINIÇÃO

O menor  $t > 0$  tal que  $a^t = 1 \pmod{m}$  é a **ordem** do elemento  $a \in \mathbb{Z}_m^*$ . Se  $t \equiv \phi(m)$ ,  $a$  diz-se **elemento primitivo** de  $\mathbb{Z}_m^*$





## 134 FACTO

- (a) Se  $a^s = 1 \pmod{m}$  então  $s$  é um múltiplo da ordem de  $a$ .
- (b)  $\mathbb{Z}_m^*$  tem um elemento primitivo se e só se  $m = 2, 4, p^n$  ou  $2p^n$ , sendo  $p$  um primo ímpar.
- (c) Se  $\mathbb{Z}_m^*$  tem um elemento primitivo então é um grupo cíclico em que cada um dos seus elementos primitivos é um gerador do grupo.
- (d) Se  $a \in \mathbb{Z}_m^*$  é um elemento primitivo então qualquer outro  $b \in \mathbb{Z}_m^*$  é um elemento primitivo se e só se tiver a forma  $a^k \pmod{m}$  com  $k \in \mathbb{Z}_{\phi(m)}$ . Donde, se  $\mathbb{Z}_m^*$  tiver algum elemento primitivo, terá  $\phi(\phi(m))$  elementos primitivos distintos.

$\mathbb{Z}_{21}^*$  não é cíclico. Porém  $\mathbb{Z}_{22}^*$  e  $\mathbb{Z}_{23}^*$  são ambos cíclicos. De facto  $\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$  não tem nenhum elemento de ordem superior a 6 (note-se que  $\phi(21) = 12$ ).



## 2.3 Raízes Quadradas

Um **resíduo quadrático** em  $\mathbb{Z}_n$  é um número  $a \neq 0$  para o qual existe  $x \in \mathbb{Z}_n$  tal que

$$a = x^2 \pmod{m}$$

Nestas circunstâncias, o valor  $x$  é uma **raíz quadrada modular** de  $a$ .

### 135 DEFINIÇÃO

Seja  $p > 2$  um primo. O **símbolo de Legendre**  $\left(\frac{a}{p}\right)$  é definido como sendo 0 se  $p|a$ , é igual a 1 se  $a$  é um resíduo quadrático módulo  $p$  e é igual a  $(-1)$  em caso contrário.

Se  $m > 1$  tem uma factorização  $p_1 p_2 \dots p_n$  por primos não necessariamente distintos então o **símbolo de Jacobi**  $\left(\frac{a}{m}\right)$  define-se como  $\prod_{i=1}^n \left(\frac{a}{p_i}\right)$ .

O valor do símbolo de Jacobi  $\left(\frac{a}{m}\right)$  é 0 se e só  $\gcd(a, m) \neq 1$ . Se  $a \in \mathbb{Z}_n^*$  o símbolo  $\left(\frac{a}{m}\right)$  é sempre  $\pm 1$ , com igual probabilidade (excepto quando  $m$  é uma quadrado onde o valor é sempre 1).

Existe um algoritmo bastante eficiente para determinar símbolos de Legendre e de Jacobi (sem recorrer à factorização de  $m$ ).



## 136 FACTO

Se  $p$  é primo então, para todo  $a$

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \pmod{p}$$

Se  $a$  é um resíduo quadrático (i.e.  $a^{(p-1)/2} = 1 \pmod{p}$ ) tem duas raízes quadradas modulares  $x$  calculadas da seguinte forma<sup>26</sup>:

- Se  $(p + 1)$  é divisível por 4 então  $x = \pm a^{(p+1)/4} \pmod{p}$ .
- Se  $(p - 5)$  é divisível por 8 e se for  $k \doteq (p - 5)/8$  então

$$x = \pm a y (2 a y^2 - 1) \pmod{p}$$

sendo  $y \doteq (2 a)^k \pmod{p}$ .

- Se  $p = 2^t q + 1$ , com  $q$  ímpar e  $t > 2$  então

$$x = a^{(q+1)/2} \cdot (u^q)^s \pmod{p}$$

com  $u$  um qualquer não-resíduo quadrático módulo  $p$  e  $s < 2^{t-1}$ , um expoente encontrado por tentativas.

<sup>26</sup>Neste caso,  $a^{(p+1)/2} = a^{(p+1)/2} * a^{(p-1)/2} = a^p = a \pmod{p}$ .



Existe um algoritmo para encontrar  $s$  com complexidade  $\mathcal{O}(t)$  baseado na sua expansão em bits e no seguinte facto:

137 FACTO

Se for

$$s = s_0 + 2 \cdot s_1 + 2^2 \cdot s_2 + \dots + 2^{t-2} \cdot s_{t-2} \quad s_i \in \mathbb{Z}_2$$

então verifica-se

$$a^{(p-1)/4} \cdot (u^{(p-1)/2})^{s_0} \equiv 1 \pmod{p}$$

$$a^{(p-1)/8} \cdot (u^{(p-1)/4})^{s_0} \cdot (u^{(p-1)/2})^{s_1} \equiv 1 \pmod{p}$$

$$\dots \equiv \dots$$

$$a^{(p-1)/2^t} \cdot (u^{(p-1)/2^{t-1}})^{s_0} \dots \dots (u^{(p-1)/2})^{s_{t-2}} \equiv 1 \pmod{p}$$

A 1ª equação determina  $s_0$ , a segunda determina  $s_1$ , etc. . .



## 2.4 Algoritmos

### Algoritmo de Euclides

Determina  $\gcd(a, b)$  quando  $a \geq b > 0$ .

enquanto  $b > 0$  {  $r = a \bmod b$  ;  $a = b$  ;  $b = r$  }

Existe uma versão estendida que, quando  $\gcd(a, b) = 1$ , determina o inverso de  $a$  módulo  $b$ .

Estes algoritmos estendem-se a qualquer anel comutativo, nomeadamente aos anéis de polinómios.

### Teorema Chinês dos Restos

Se  $N = n_1 \times n_2 \times \cdots \times n_k$  é um produto de números primos entre si, existe um único  $x < N$  que é solução do sistema de equações

$$x = a_1 \pmod{n_1}, \quad x = a_2 \pmod{n_2}, \quad \cdots \quad x = a_k \pmod{n_k}$$

dados os “restos”  $0 < a_i < n_i$ , com  $i = 1 \dots k$ .

$$x = \sum_{i=1}^k a_i x_i N_i \pmod{N} \quad \text{com} \quad N_i = N/n_i, \quad x_i = N_i^{-1} \pmod{n_i}$$



### Cálculo eficiente de exponenciais

Para um qualquer grupo multiplicativo  $\langle G, \cdot \rangle$  calcula  $x = a^b$ , com  $a \in G$  e um inteiro  $b \geq 0$ , a partir da representação binária de  $b$

Seja  $b = b_1 + 2b_2 + 2^2b_3 + \dots + 2^{n-1}b_n$  com  $b_i \in \mathbb{Z}_2$ .

$x = 1$  ;

para  $i = n$  até 1 {  $x = x * x$ ; se  $b[i]$  {  $x = x * a$  } }

O número de multiplicações está limitado ao dobro do número de bits necessários para representar  $b$

### Teste e Geração de Primos

Primos são essenciais em várias técnicas criptográficas. Distribuem-se de modo basicamente uniforme; para cada  $n > 2$ , o número de primos menores ou iguais a  $n$  tende para  $\frac{n}{\ln n}$

Os algoritmos mais eficientes para **geração** de grandes primos são baseados no algoritmo

```
repetir { gerar um número aleatório P }
até \ 'e { é-primo?(P) = verdadeiro }
```



Donde assentam em duas operações básicas: **geração** de números aleatórios e **teste** da propriedade “*ser primo*”.

Normalmente um teste determinístico é computacionalmente intratável. Por isso usam-se testes não determinísticos que se baseiam na existência, para cada  $p \gg 3$  de conjuntos  $W(p) \subset \mathbb{Z}_p$  com a propriedade

- (i)  $|W(p)| = 0$  se  $p$  é primo e  $|W(p)| \geq p/2$  se  $p$  não é primo.
- (ii) Para cada  $a \in \mathbb{Z}_p$ , o teste  $a \in W(p)$  é computacionalmente tratável.

### Algoritmo de teste

1. Escolhe-se um número  $a \in \mathbb{Z}_p$  aleatoriamente; se ocorrer  $a \in W(p)$  então, com probabilidade 1,  $p$  não é primo. O algoritmo termina com a resposta **não** e sem erro.  
Se  $a \in \overline{W(p)}$  então  $p$  pode ou não ser primo; porque  $|W(p)| \geq |\overline{W(p)}|$  a probabilidade de ser primo é pelo menos igual à probabilidade de não ser.
2. Repete-se (1) até se atingir um **limite de tentativas**  $t$  ou aí terminar.
3. Se o limite de tentativas  $t$  for alcançado a resposta é **sim** e tem uma probabilidade de erro inferior a  $2^{-t}$ .

**Critério de Fermat** Se  $p > 2$  é primo,  $a^{p-1} = 1 \pmod{p}$  para todo  $0 < a < p$ .

$$W(p) \equiv \{0 < a < p \mid a^{p-1} \neq 1 \pmod{p}\}$$

Infelizmente  $W(p)$  pode ser vazio mesmo quando  $p$  não é primo; os chamados **números de Carmichael** satisfazem essa propriedade.

**Critério de Euler** Se  $p > 2$  é primo,  $a^{(p-1)/2} = \left(\frac{a}{p}\right) \pmod{p}$  para todo  $0 < a < p$ .

$$W(p) \equiv \{0 < a < p \mid a^{(p-1)/2} \neq \left(\frac{a}{p}\right) \pmod{p}\}$$

**Nota** Se  $p$  não é primo, formalmente o símbolo de Legendre não existe; existe o **símbolo de Jacobi** que é o produto dos símbolos de Legendre de  $a$  em relação a cada um dos factores primos de  $p$ . O algoritmo para calcular  $\left(\frac{a}{p}\right)$  é, no entanto semelhante, e muito eficiente (ver **Koblitz**).

O *algoritmo de Solovay-Strassen* implementa este critério. Tem vindo a ser substituído pelo *algoritmo de Miller-Rabin* baseado em

**Critério de Miller-Rabin** Se  $p$  é primo e  $q = 2^{-s}(p-1)$  é o maior divisor ímpar de  $(p-1)$  então, para todo  $0 < a < p$ , uma de duas situações ocorre

$$a^q = 1 \pmod{p} \quad \text{ou}$$

$$a^{2^i q} = -1 \pmod{p} \quad \text{para algum } 0 \leq i < s$$



Os conjuntos  $W(p)$  definem-se apropriadamente a partir deste critério.

Ver, no [Handbook of Applied Cryptography](#), detalhes, comparações e implementações destes algoritmos.

## 2.5 Corpos Finitos

Um **corpo finito** é um anel finito e comutativo em que todos os elementos, excepto o 0, têm inversa multiplicativa.

A **característica** de um corpo finito é o menor inteiro  $v > 0$  tal que  $\overbrace{1 + 1 + \cdots + 1}^{v \text{ vezes}} = 0$ .

$\mathbb{Z}_p$ , para  $p$  primo, é um **corpo finito** de característica  $p$ ; note-se que, neste caso,  $\mathbb{Z}_p \setminus \{0\} \equiv \mathbb{Z}_p^*$

138 FACTO

Seja  $\mathbb{F}_m$  um corpo finito com  $m$  elementos.

- (a) O conjunto  $\mathbb{F}_m$  é identificável com o conjunto das raízes do polinómio  $X^m - X \in \mathbb{F}_m[X]$ , numa extensão do corpo  $\mathbb{F}_m$
- (b) Se  $\mathbb{F}_m$  tem característica  $p \neq 0$  então  $p$  é primo e existe uma dimensão  $n$  tal que  $m = p^n$  e

$$\mathbb{F}_m \sim (\mathbb{Z}_p)^n$$

- (c) Existe um **polinómio primitivo**<sup>27</sup>  $c \in \mathbb{Z}_p[X]$  de grau  $n$  tal que

$$\mathbb{F}_m \cong \mathbb{Z}_p[X]/c\mathbb{Z}_p[X]$$

<sup>27</sup>Um polinómio em  $\mathbb{Z}_p[X]$ , mónico e irreduzível, é **primitivo** em  $\mathbb{F}_m$  se divide  $X^m - X$  mas não divide nenhum  $X^k - X$  para  $k < m$ .



**Nota:**  $\mathbb{Z}_p[X]/\mathfrak{c}\mathbb{Z}_p[X]$  é o anel dos polinómios de coeficientes em  $\mathbb{Z}_p$  em que a adição e multiplicação de polinómios é efectuada módulo  $\mathfrak{c}(X)$ .

- (d) O grupo multiplicativo  $\mathbb{F}_m^*$ , formado pelos elementos invertíveis de  $\mathbb{F}_m$ , é cíclico e existem  $\phi(m-1)$  geradores diferentes deste grupo.
- (e) Se  $\mathfrak{c} \in \mathbb{Z}_p[X]$  é um polinómio primitivo seja  $K$  uma qualquer extensão de  $\mathbb{Z}_p$  (i.e.,  $K$  contém  $\mathbb{Z}_p$  como sub-corpo) que contém todas as raízes desse polinómio<sup>28</sup>; seja  $\beta$  uma raiz de  $\mathfrak{c}$  em  $K$  que não seja raiz de nenhum polinómio  $(X^k - 1)$  com  $k < m - 1$ <sup>29</sup>.

Então:

- (i) O menor anel que contém  $\mathbb{Z}_p$  e o elemento  $\beta$  forma um sub-corpo de  $K$  que representamos por  $\mathbb{Z}_p(\beta)$ ; sendo  $n = \text{grau}(\mathfrak{c})$ , o elemento genérico desse corpo tem a forma

$$a_0 + a_1 \beta + a_2 \beta^2 + \cdots + a_{n-1} \beta^{n-1} \quad \text{com } a_i \in \mathbb{Z}_p \quad (49)$$

- (ii) A menos de um isomorfismo  $\mathbb{F}_m$  identifica-se com  $\mathbb{Z}_p(\beta)$ .

- (f) Os elementos  $\beta, \beta^p, \beta^{p^2}, \dots, \beta^{p^{n-1}}$  estão contidos  $\mathbb{Z}_p(\beta)$  e formam o conjunto das  $n$  raízes do polinómio  $\mathfrak{c}$ .

### Comentários

<sup>28</sup>Diz-se que  $K$  é um corpo que **fractura** o polinómio  $\mathfrak{c}[X]$  sobre  $\mathbb{F}_p$ .

<sup>29</sup>Estas raízes designam-se por **raízes primitivas**.



- (a) Este facto define a representação essencial para os elementos de um corpo finito. O polinómio  $X(X^{m-1} - 1)$  tem  $m$  raízes: o 0 e as  $m - 1$  soluções distintas da equação  $X^{m-1} = 1$  (designadas **raízes da unidade**).

O resultado é uma simples consequência do facto: se  $x$  e  $y$  verificam  $x^m = x$  e  $y^m = y$  então, em  $\mathbb{F}_m$ , necessariamente se verifica  $(x + y)^m = (x + y)$  e  $(xy)^m = x^m y^m$ .

Esta representação é muito importante sob o ponto de vista da análise das propriedades algébricas de  $\mathbb{F}_m$ , mas não é muito útil em termos de manipulação. Por isso são necessárias outras representações.

- (b) Este facto significa que cada elemento  $x \in \mathbb{F}_m$  é representável por um vector de  $n$  componentes em que todas essas componentes são elementos de  $\mathbb{Z}_p$ .

Em caso particular muito importante ocorre quando  $p = 2$ . Quando a característica é 2 o corpo finito diz-se binário e cada um dos seus  $2^n$  elementos é representável por um vector de  $n$  componentes em  $\mathbb{Z}_2$ ; i.e., uma palavra de  $n$  bits.

Esta 2ª representação já permite uma forma simples de somar elementos de  $\mathbb{F}_m$ : para somar dois elementos  $x, y \in \mathbb{F}_m$  basta somar os respectivos vectores componente a componentes.

A multiplicação destes elementos já não pode ser feita nesta representação vectorial porque não está, genericamente, definida a multiplicação de vectores.

- (c) A procura dos polinómios primitivos  $c[X]$  é muito importante porque permite expressar um corpo finito como um corpo de polinómios  $\mathbb{F}_p[X]/c$ .

As raízes de um polinómio primitivo devem estar associadas ao elemento 0 de  $\mathbb{F}_m$ ; por isso o polinómio primitivo tem de dividir  $X^m - X$ . Por outro lado não pode dividir nenhum outro polinómio da forma  $X^k - X$ , com  $k < m$ , porque neste caso o espaço quociente teria menos elementos distintos dos que os  $m$  exigidos pelo corpo  $\mathbb{F}_m$ .

Esta 3ª representação já permite fazer multiplicações. As  $n$  componentes do vector são agora interpretadas como coeficientes de um polinómio.

Este facto diz essencialmente que existe um polinómio de grau  $n$ ,  $c \in \mathbb{Z}_p[X]$  tal que para multiplicar  $x, y \in \mathbb{F}_m$  basta multiplicar os dois polinómios que os representam e reduzir o resultado módulo  $c$ .

- (d) Sendo  $\mathbb{F}_m$  um corpo todos os seus elementos, excepto 0, são invertíveis. Portanto o grupo multiplicativo  $\mathbb{F}_m^*$  tem  $m - 1$  elementos. Adicionalmente o grupo é cíclico: todo o elemento de  $\mathbb{F}_m$ , excepto o 0, tem a forma  $g^i$  com  $i \in \mathbb{Z}_{m-1}$ .

Os elementos do grupo cíclico são as raízes de  $X^{m-1} - 1$  na extensão de  $\mathbb{F}_m$  que as contém.

Como elemento de corpo quociente de polinómios  $\mathbb{Z}_p[X]/\mathfrak{c}\mathbb{Z}_p[X]$  o monómio  $X$  é um gerador uma vez que, reduzido módulo  $\mathfrak{c}$  todas as potências  $X^k$ , com  $k < m$ , têm de ser distintas; de facto, se fosse  $X^k = X^i \pmod{\mathfrak{c}}$ , então  $X^k - X^i$  seria divisível por  $\mathfrak{c}[X]$  o que contraria a definição de polinómio primitivo.

- (e) A representação  $\mathbb{Z}_p(\beta)$  é, geralmente, a representação preferida de  $\mathbb{F}_m$ . Porque  $\beta$  é raiz de um polinómio de grau  $n$  com coeficientes em  $\mathbb{Z}_p$ , é sempre possível escrever  $\beta^n = c_0 + c_1\beta + \dots + c_{n-1}\beta^{n-1}$ , com  $c_i \in \mathbb{Z}_p$ ; em qualquer multiplicação de dois elementos da forma (49), sempre que o ocorra um termo da forma  $\beta^k$  com  $k \geq n$ , pode-se fazer substituições sucessivas de  $\beta^n$  de acordo com esta igualdade de forma a reduzir o resultado, sempre, a um expressão da forma (49).

Os geradores de  $\mathbb{F}_m \sim \mathbb{Z}_p(\beta)$  podem ser determinados a partir de  $\beta$ . De facto o elemento  $\beta$  é um gerador de  $\mathbb{Z}_p(\beta)^*$ : qualquer  $x \neq 0$  pode ser escrito como  $x = \beta^i$  com  $i \in 0..p^n - 1$ .

Isto facilita o cálculo de multiplicações ( $\beta^i \cdot \beta^j = \beta^{i+j}$ ) mas dificulta o cálculo das somas.

Seja  $\tau : \mathbb{Z}_{m-1} \rightarrow \mathbb{Z}_{m-1}$  a função parcial que associa cada  $i \in \mathbb{Z}_{m-1}$ , tal que  $\beta^i + 1 \neq 0$ , ao índice  $\tau(i)$  que verifica  $\beta^{\tau(i)} = \beta^i + 1$ . Esta função chama-se o *logaritmo de Zech* de base  $\beta$ ; então

$$\beta^i + \beta^j = (\beta^{i-j} + 1)\beta^j = \beta^{\tau(i-j)+j}$$

Isto dá uma forma “imediate” de calcular somas de potências ( $\beta^i + \beta^j$ ) quando  $\tau(i-j)$  é definido; se não for definido é porque o resultado da soma é zero.

Obviamente que isto presupõe o cálculo *à priori* do logaritmo de Zech. O que não é, geralmente, uma tarefa simples.

- (f) É muito simples provar que, para quaisquer dois elementos  $x, y \in \mathbb{Z}_p(\beta)$ , se verifica sempre  $(x + y)^p = x^p + y^p$  e



$(x \cdot y)^p = x^p \cdot y^p$ ; basta fazer a expansão apropriada e notar que, para todo  $a \in \mathbb{Z}_p$ ,  $a^p = a$ . Deste modo, para todo o polinómio  $\mathbf{p}[X] \in \mathbb{Z}_p[X]$ , tem-se  $(\mathbf{p}(x))^p = \mathbf{p}(x^p)$ .

Por isso, se  $\alpha \in \mathbb{Z}_p(\beta)$  é raiz de um qualquer polinómio  $\mathbf{p}[X] \in \mathbb{Z}_p[X]$ , teremos  $\mathbf{p}(\alpha^p) = (\mathbf{p}(\alpha))^p = 0$ . Logo  $\alpha^p$  também será raiz do mesmo polinómio.

Por este processo, a partir da raiz  $\beta$ , gera-se  $\beta^p, \beta^{p^2}, \dots, \beta^{p^{n-1}}$  todas raízes de  $\mathbf{c}$ ; uma vez que os diferentes expoentes  $p^i$ , com  $i \in 0..n-1$  são todos distintos em  $\mathbb{Z}_{p^{n-1}}$ , e  $\beta$  é um gerador do grupo cíclico  $\mathbb{Z}_p(\beta)^*$ , todas as raízes  $\beta^{p^i}$  serão distintas.

Por isso  $\mathbb{Z}_p(\beta)$  contém, não só a raiz  $\beta$ , como também todas as restantes raízes do polinómio primitivo  $\mathbf{c}[X]$ . Deste modo  $\mathbb{Z}_p(\beta)$  também fratura o polinómio sobre  $\mathbb{Z}_p$ . De facto, é o menor corpo (a menos de um isomorfismo) que verifica esta propriedade e, por isso, se chama o *corpo mínimo de fratura* de  $\mathbf{c}[X]$  sobre  $\mathbb{Z}_p$ .

**EXEMPLO 15:** Considere-se o corpo  $\mathbb{F}_9$ . Como representá-lo?

Dado que  $9 = 3^2$  a característica do corpo é 3 e a dimensão é 2. Isto significa que o polinómio característico é um monómio de grau 2 com coeficientes em  $\mathbb{Z}_3$ .

Os polinómios primitivos de  $\mathbb{F}_9$  serão monómios de grau 2 em  $\mathbb{Z}_3[X]$ , irredutíveis, que dividem  $X^9 - X$  mas não dividem nenhum outro  $X^k - X$  com  $k < 9$ .

Usando o sistema **Pari** (ou qualquer sistema de computação análogo) encontram-se os seguintes factores de



$X^9 - X$  que são irredutíveis e têm grau 2.

$$(X^2 + 1) \quad (X^2 + X - 1) \quad (X^2 - X - 1)$$

Note-se que, em  $\mathbb{Z}_3$ ,  $-1$  é equivalente a 2. Neste corpo Em  $\mathbb{Z}_3$  o polinómio  $(X^2 + 1)$  divide  $(X^5 - X)$ ; de facto  $(X^5 - X) = X \cdot (X^4 - 1) = X \cdot (X^2 - 1) \cdot (X^2 + 1)$ ; isso exclui-o de ser um polinómio primitivo.

Restam os polinómios  $(X^2 + X - 1)$  e  $(X^2 - X - 1)$ . Usando **Pari** de novo, verifica-se que nenhum deles divide qualquer polinómio  $(X^k - X)$ , com  $k \in 2..8$ . Portanto ambos os polinómios são primitivos e qualquer um pode ser usado como polinómio característico.

Tomemos, por exemplo,  $X^2 - X - 1$  como característico<sup>30</sup>; a partir daqui existem duas representações possíveis para os elementos de  $\mathbb{F}_9$

1. Tomemos uma extensão  $K$  qualquer de  $\mathbb{Z}_3$  que contenha todas as raízes do polinómio característico  $X^2 - X - 1$ . Tomemos uma qualquer raíz  $\beta$  desse polinómio. Como consequências imediatas,  $\beta$  verifica  $\beta^2 = \beta + 1$  e a outra raíz do polinómio é  $1 - \beta$ .

---

<sup>30</sup>Por curiosidade, o número irracional  $(1 + \sqrt{5})/2$ , que é a raiz real e positiva deste polinómio, é um dos números mais famosos da história da Matemática; é chamado *razão áurea* ou *número de ouro* ou *golden rule*, ou ainda várias outras designações análogas. É considerado a proporção ideal para a harmonia de dimensões de edifícios, dimensões de instrumentos musicais, progressão de escalas musicais, etc.

O sub-anel  $\mathbb{Z}_3(\beta)$  de  $K$ , formado por todos os elementos da forma  $a + b\beta$ , é um corpo que verifica:  $(a + b\beta) + (c + d\beta) = (a + c) + (b + d)\beta$  e  $(a + b\beta)(c + d\beta) = (ac + bd) + (ad + bc + bd)\beta$ .

2. A segunda representação é polinomial. Cada elemento de  $\mathbb{F}_9$  é descrito por um polinómio  $(x + yX) \in \mathbb{Z}_3[X]/\mathfrak{c}, \mathbb{Z}_3[X]$  em que  $\mathfrak{c}$  denota o polinómio característico  $X^2 - X - 1$ .

As operações de soma e multiplicação são efectuadas como em quaisquer polinómios tendo em atenção, apenas, que operações nos coeficientes são sempre efectuadas em  $\mathbb{Z}_3$  e que os polinómios de grau superior a 2 são reduzidos módulo o polinómio característico.

Obviamente que as duas representações são isomórficas: cada uma delas se converte na outra de uma forma única. No entanto é preciso constatar que lida com entidades diferentes: polinómios no segundo caso e extensões algébricas no primeiro caso.

Considere-se a representação de um elemento genérico  $x \in \mathbb{F}_9$  por um elemento de  $\mathbb{Z}_3(\beta)$ .

A menos de um isomorfismo o corpo  $\mathbb{Z}_3(\beta)$  é único e os seus elementos são

$$\{0, -1, 1, \beta, -\beta, 1 + \beta, 1 - \beta, -1 + \beta, -1 - \beta\}$$

Todo  $x \in \mathbb{F}_9$ , excepto 0, é invertível; também  $\mathbb{F}_9^*$  é cíclico de gerador  $\beta$ ; isto significa todo  $x \in \mathbb{F}_9^*$  é escrito na forma  $x = \beta^k$  para algum  $k \in \mathbb{Z}_8$ .



Usando **Pari** pode-se calcular uma tabela das potências  $\beta^k$  e verificar

$\beta^0$	$\beta^1$	$\beta^2$	$\beta^3$	$\beta^4$	$\beta^5$	$\beta^6$	$\beta^7$
1	$\beta$	$1 + \beta$	$1 - \beta$	-1	$-\beta$	$-1 - \beta$	$-1 + \beta$

Para calcular o inverso de um elemento pode-se usar esta tabela; por exemplo (note-se que  $6 = -2 \pmod{8}$ )

$$(1 + \beta)^{-1} = (\beta^2)^{-1} = \beta^6 = -1 - \beta$$

O logaritmo de Zech pode ser calculado usando a tabela anterior e tem-se

$k$	0	1	2	3	4	5	6	7
$\tau(k)$	4	2	7	6	$\perp$	3	5	1

Por exemplo, para calcular

$$- \beta^5 + \beta^3 = \beta^{\tau(5-3)+3} = \beta^{7+3} = \beta^2$$

porque os expoentes são vistos  $\pmod{8}$ .

$$- \beta^6 + \beta^2 = 0$$

porque  $\tau(6 - 2) = \perp$ .

**EXEMPLO 16:** Considere-se agora a representação de  $\mathbb{F}_{256}$ .

Como  $256 = 2^8$  temos um corpo de característica 2 isomórfico com  $(\mathbb{Z}_2)^8$ : os elementos de  $\mathbb{F}_{256}$  devem, assim, ser representados por uma palavra de 8 *bits* (um “byte”).

Usando **Pari** é possível determinar todos os polinómios irredutíveis de grau 8 que são factores de  $X^{255} + 1$  (note-se que, com característica 2, tem-se  $X = -X$ ). Desses polinómios seleccionam-se aqueles que não dividem nenhum  $X^k + 1$ , com  $k < 255$ .

Por este processo verifica-se que não existe nenhum polinómio primitivo com menos de 5 coeficientes não-nulos. Um desses polinómios é  $(X^8 + X^4 + X^3 + X + 1)$  que pode ser usado como polinómio característico.

A estrutura de um corpo finito  $\mathbb{F}_m$  é também determinada pelos seus automorfismos; isto é, as aplicações  $\mathbb{F}_m \rightarrow \mathbb{F}_m$  que preservam a estrutura do corpo (endomorfismos) e são injectivas.

Os automorfismos de  $\mathbb{F}_m$  que fixam<sup>31</sup> os elementos de  $\mathbb{F}_p$  formam o **grupo de Galois**  $\text{Gal}(\mathbb{F}_m/\mathbb{F}_p)$ . A operação de grupo é a composição de morfismos com a identidade 1. Considera-se o grupo multiplicativo:  $\sigma \cdot \delta$  é o morfismo  $x \mapsto \sigma(\delta(x))$ ,  $\sigma^0 = 1$  e  $\sigma^k$  é  $\underbrace{\sigma \cdot \sigma \cdots \sigma}_{k \text{ vezes}}$ .

### 139 FACTO

Seja  $p \neq 0$  a característica de  $\mathbb{F}_m$ . Para todos  $x, y \in \mathbb{F}_m$ ,  $(x + y)^p = x^p + y^p$ ,  $(x \cdot y)^p = x^p \cdot y^p$ ,  $x^p = 0$  se e só se  $x = 0$  e  $x^p = x$  se e só se  $x \in \mathbb{F}_p$ .

A aplicação  $\sigma : x \mapsto x^p$  designa-se por **morfismo de Frobenius** e este resultado afirma que se trata de um automorfismo em  $\mathbb{F}_m$  que discrimina os elementos de  $\mathbb{F}_p$  por serem os que são fixos por  $\sigma$ .

### Comentários

1. O morfismo de Frobenius fixa os elementos de  $\mathbb{F}_p$  contidos em  $\mathbb{F}_m$ ; de facto, pelo pequeno teorema de Fermat,  $x^p = x$  para todo  $x \in \mathbb{F}_p$ . Por outro lado, sabemos que  $\mathbb{F}_p$  se identifica com o conjunto das raízes numa sua extensão (como é o caso de  $\mathbb{F}_m$ ) do polinómio  $X^p - X$ . Portanto qualquer elemento de  $\mathbb{F}_m$  que seja fixo por  $\sigma$  se identifica com um elemento de  $\mathbb{F}_p$ .

---

<sup>31</sup>O morfismo  $\sigma$  fixa  $x$  quando  $\sigma(x) = x$ .

2. Usando a noção de característica é muito simples mostrar que  $\sigma$  é realmente um endomorfismo; isto é,  $\sigma(x + y) = (x + y)^p = x^p + y^p = \sigma(x) + \sigma(y)$  e  $\sigma(x \cdot y) = (x \cdot y)^p = x^p \cdot y^p = \sigma(x) \cdot \sigma(y)$ .
3. Como primeira consequência, tem-se que, para todo o polinómio  $\mathbf{p}[X] \in \mathbb{F}_p[X]$  se verifica  $(\mathbf{p}[X])^p = \mathbf{p}[X^p]$ .

De facto, se for  $\mathbf{p}[X] = p_0 + p_1X + p_2X^2 + \dots + p_dX^d$  então

$$\begin{aligned} (\mathbf{p}[X])^p &= a_0^p + a_1^p X^p + a_2^p (X^p)^2 \dots + a_d^p (X^p)^d = \\ &= a_0 + a_1 X^p + a_2 (X^p)^2 \dots + a_d (X^p)^d = \mathbf{p}[X^p] \end{aligned}$$

já que, por serem elementos de  $\mathbb{F}_p$ , todos os  $a_i$  verificam  $a_i^p = a_i$ .

4. Como corolário da observação anterior, se  $\alpha$  é uma raiz do polnómio  $\mathbf{p}[X]$  numa qualquer extensão de  $\mathbb{F}_p$ , então  $\alpha^p$  será também uma raiz.

$$\mathbf{p}[\alpha] = 0 \implies \mathbf{p}[\alpha^p] = (\mathbf{p}[\alpha])^p = 0$$

5. Para verificar-mos que o morfismo de Frobenius  $\sigma$  é um isomorfismo, dado que é linear, basta verificar que não existe nenhum  $x \neq 0$  que verifique  $\sigma(x) = 0$ .

Tomando a representação polinomial genérica, definida em (49), para um elemento  $x \in \mathbb{F}_m$ ; pelas observações anteriores

$$x^p = a_0 + a_1 \beta^p + a_2 (\beta^p)^2 + \dots + a_{n-1} (\beta^p)^{n-1}$$

Sendo  $\beta$  uma raiz do polinómio característico,  $\beta^p$  é também uma raiz do mesmo polinómio. Portanto o morfismo de Frobenius toma um elemento  $x \in \mathbb{F}_m$  e limita-se a produzir uma mudança de base: produz a soma formal com os mesmos coeficientes mas efectuada com uma raiz diferente do polinómio característico. Se fosse  $x^p = 0$  todas as componentes  $a_i$  teriam de ser nulas e, assim, seria também  $x = 0$ .



Um dos resultados mais importantes do estudo dos corpos finitos pode ser enunciado da forma seguinte

140 FACTO

*Se  $L \simeq \mathbb{F}_m$  e  $K \simeq \mathbb{F}_q$  são dois corpos finitos com a mesma característica  $p$  (com  $m = p^n$  e  $q = p^r$ ) então  $L$  é uma extensão de  $K$  se e só se existe  $s \geq 1$  tal que  $n = s r$ .*

*Nestas circunstâncias o grupo de Galois  $\text{Gal}(L/K)$  é cíclico, tem ordem  $s$  e é gerado pelo morfismo de Frobenius  $\sigma : x \mapsto x^q$  de  $L$  em  $K$ .*

### Comentários

Recordemos que o grupo de Galois  $\text{Gal}(L/K)$  é formado por todos os automorfismos em  $L$  que fixam os elementos de  $K$ . A operação de grupo é a composição de morfismos e o elemento neutro é o morfismo identidade.

Como caso particular temos  $K \equiv \mathbb{F}_p$  (corresponde a ser  $r = 1$ ,  $p = q$  e  $s = n$ ). Aqui o resultado diz-nos que  $\mathbb{F}_{p^n}$  é sempre uma extensão de  $\mathbb{F}_p$ ; diz-nos também que o corpo dos automorfismos é formado pelas  $n$  potências  $\{\sigma^k \mid k \in 0..n-1\}$  do morfismo de Frobenius (que, aqui, é simplesmente  $x \mapsto x^p$ ).



## 141 DEFINIÇÃO

Sejam  $L$  e  $K$  corpos finitos como no facto 140. O **traço** de  $L$  em  $K$  é a aplicação definida por

$$\text{tr}_{L/K}(x) \doteq \sum_{k=0}^{s-1} \sigma^k(x)$$

A **norma** de  $L$  em  $K$  é a aplicação

$$\text{nr}_{L/K}(x) \doteq \prod_{k=0}^{s-1} \sigma^k(x)$$

em que  $\sigma : x \mapsto x^q$  denota o morfismo de Frobenius de  $L$  em  $K$ .

## 142 FACTO

Subentendendo-se os corpos  $L$  e  $K$ . Para todo  $x, y \in L$  e todo  $a \in K$ , verifica-se:

- (i)  $\text{tr}(x) \in K$  e  $\text{nr}(x) \in K$
- (ii)  $\text{tr}(x + y) = \text{tr}(x) + \text{tr}(y)$  e  $\text{tr}(ax) = a \text{tr}(x)$ ,
- (iii)  $\text{nr}(x \cdot y) = \text{nr}(x) \cdot \text{nr}(y)$  e  $\text{nr}(ax) = a^s \text{nr}(x)$
- (iv)  $\text{tr}(\cdot)$  é uma aplicação sobrejectiva de  $L$  em  $K$  e  $\text{nr}(\cdot)$  é uma aplicação sobrejectiva de  $L^*$  em  $K^*$ .

Esboço de prova

Este resultado pretende afirmar que o traço  $\text{tr}(x)$  é sempre um elemento de  $\mathbb{F}_p$ , que todo  $c \in \mathbb{F}_p$  é traço de algum  $x \in \mathbb{F}_m$  e que a aplicação preserva somas e multiplicações escalares.

Seja  $t = \text{tr}(x)$ ; então  $\sigma(t) = \sum_{k=1}^n \sigma^k(x)$ ; como  $\sigma^n(x) = x$  então  $\sigma(t) = x + \sum_{k=1}^{n-1} \sigma^k(x) = t$ ; concluímos que  $t$  é fixo pelo morfismo de Frobenius e, por isso, tem de ser um elemento de  $\mathbb{F}_p$ .

Transformações semelhantes (usando as propriedades de  $\sigma$ ) permitem concluir facilmente que  $\text{tr}(\cdot)$  preserva somas e multiplicação escalar.

Para provar que é sobrejectiva, considere-se um qualquer  $c \in \mathbb{F}_p$ ; seja  $y \in \mathbb{F}_m$  tal que  $\text{tr}(y) \neq 0$ ; um tal  $y$  tem de existir por que existem, quanto muito,  $p^{n-1}$  raízes do polinómio  $x + x^p + x^{p^2} + \dots + x^{p^{n-1}}$  e existem  $p^n$  elementos em  $\mathbb{F}_m$ . Definindo  $x \doteq (c/\text{tr}(y)) y$ , temos um elemento com traço  $c$ .

## 2.6 Corpos de Galois

São os corpos finitos de característica 2.

143 FACTO

Se  $\mathbb{F}_m$  tem característica 2 e dimensão  $n$  e  $\mathcal{B} \subseteq \mathbb{F}_m$  é um qualquer subconjunto com  $n$  elementos de tal forma que nenhum seu subconjunto não vazio soma zero, então o seu "power set"  $\wp(\mathcal{B})$  gera o corpo  $\mathbb{F}_m$  da seguinte forma: cada  $x \in \mathbb{F}_m$  é representado pelo único  $\alpha \subseteq \mathcal{B}$  cuja soma de elementos coincide com  $x$ .

$$x = \sum_{a \in \alpha} a$$

Neste caso  $\mathcal{B}$  diz-se uma **base** de  $\mathbb{F}_{2^n}$ . As bases mais frequentes são

<b>Base Polinomial Tipo 0</b>	$\mathcal{B} = \{1, \beta, \beta^2, \dots, \beta^{n-1}\}$
<b>Base Polinomial Tipo 1</b>	$\mathcal{B} = \{\beta, \beta^2, \dots, \beta^{n-1}, \beta^n\}$
<b>Base Normal</b>	$\mathcal{B} = \{\rho, \rho^2, \rho^4, \dots, \rho^{2^{n-1}}\}$

com  $\beta$  uma raiz do polinómio característico em  $\mathbb{F}_m$  e  $\rho$  um elemento com traço 1.



**EXEMPLO 17:** O corpo finito  $\mathbf{GF}(2^4)$  determinado pelo polinómio característico  $c[X] = (X^4 + X + 1)$  está representado na tabela 3 numa base polinomial do tipo 0 e usando polinómios na variável  $X$ . Note-se que, genericamente, o gerador do grupo cíclico  $\mathbf{GF}(2^m)^*$  é muito simples de encontrar: é o polinómio  $X$ .

$X^i$	$X^i \bmod c[X]$	$(\mathbb{Z}_2)^4$	$X^i$	$X^i \bmod c[X]$	$(\mathbb{Z}_2)^4$
—	0	0000			
$X^0$	1	0001	$X^1$	$X$	0010
$X^2$	$X^2$	0100	$X^3$	$X^3$	1000
$X^4$	$X + 1$	0011	$X^5$	$X^2 + X$	0110
$X^6$	$X^3 + X^2$	1100	$X^7$	$X^3 + X + 1$	1011
$X^8$	$X^2 + 1$	0101	$X^9$	$X^3 + X$	1010
$X^{10}$	$X^2 + X + 1$	0111	$X^{11}$	$X^3 + X^2 + X$	1110
$X^{12}$	$X^3 + X^2 + X + 1$	1111	$X^{13}$	$X^3 + X^2 + 1$	1101
$X^{14}$	$X^3 + 1$	1001	$X^{15}$	1	0001

Figura 3:  $\mathbf{GF}(2^4)$

Usando essa tabela pode-se ver alguns exemplos de codificação de  $\mathbb{Z}_{16}$  em  $\text{GF}(2^4)$

$$7 + 11 = 0111 + 1011 = 1100 = 12$$

$$7 * 11 = 0111 * 1011 = X^{10} * X^7 = X^{17} = X^2 = 0100 = 4$$

$$(9)^2 = (1001)^2 = (X^{14})^2 = X^{28} = X^{13} = X^3 + X^2 + 1 = 1101 = 13$$

$$9^{-1} = (X^{14})^{-1} = X^{-14} = X^1 = 0010 = 2$$

Os corpos finitos binários  $\text{GF}(2)$  são particularmente importantes em Criptografia e, por isso, é necessário pensar em mecanismos computacionais eficientes para manipular estas estruturas.

Note-se que, sendo  $\text{GF}(2)$  é isomórfico com  $\mathbb{Z}_2^n$  (vectores de  $n$  bits), o que representa cada uma destas componentes vai determinar o algoritmo usado para realizar cada uma das operações básicas.

Um primeiro ponto importante é a especialização das noções de *traço* e *norma* em  $\text{GF}(2^n)$ . Se atendermos à definição 141 temos, neste caso, característica  $p = 2$  e extensão em cause é  $[\text{GF}(2^n)/\text{GF}(2)]$ , temos  $s = n$  e o morfismo de Frobenius é  $\sigma : x \rightarrow x^2$ . Portanto

$$\text{tr}(x) = \sum_{k=0}^{n-1} x^{2^k} \quad , \quad \text{nr}(x) = \prod_{k=0}^{n-1} x^{2^k}$$

Falando brevemente da noção de norma em  $\text{GF}(2^n)$  tem-se

$$\text{nr}(x) = \prod_{k=0}^{n-1} x^{2^k} = x^{(\sum_{k=0}^{n-1} 2^k)} = x^{2^n - 1}$$

Portanto  $\text{nr}(x) = 1$  se  $x \neq 0$  e  $\text{nr}(x) = 0$  se  $x = 0$ . A norma é, em  $\text{GF}(2^n)$  o simétrico do **símbolo de Kronecker**:  $\delta(x) = 1$  sse  $x = 0$ .

O processo para cálculo do traço é mais complexo e vai depender do tipo de base utilizada.

### Bases Polinomiais

Usando a representação  $\mathbb{Z}_2(\beta)$  (com  $\beta$  uma raiz do polinómio característico), a forma (49) no facto 138

$$x_0 + x_1 \cdot \beta + \cdots + x_{n-1} \cdot \beta^{n-1} \quad x_i \in \text{GF}(2) \quad (50)$$

indica que o conjunto

$$\mathcal{B}_{\beta,0} = \{1, \beta, \beta^2, \dots, \beta^{n-1}\} \quad (51)$$

forma uma base polinomial do tipo 0.

Um vector de bits  $x \sim \in \text{GF}(2)^n$  identifica (de forma única) um elemento  $x \in \text{GF}(2^n)$  através da representação (50).

*Genericamente o operador  $(\cdot) \sim$  associa um elemento  $x \in \text{GF}(2^n)$  à sua representação  $x \sim \in \text{GF}(2)^n$  numa base  $\mathcal{B}$  implícita no contexto.*

Um elemento importante é o que representa  $\beta^n$ . Note-se que, se for  $c[X]$  o polinómio característico, tem-se

$$c_0 + c_1 \cdot \beta + \cdots + c_{n-1} \cdot \beta^{n-1} = \beta^n$$

porque  $\beta$  é raiz do polinómio; isto significa que  $\beta^n$  é representado pelo vector  $c = (c_0, c_1, \dots, c_{n-1}) \in \text{GF}(2)^n$  formado pelos coeficientes do polinómio característico para termos de ordem  $< n$ .

No exemplo 17 note-se como o polinómio  $X^4$  é equivalente a  $1 + X$ ; portanto a representação de  $\beta^4$ , neste caso, seria o vector  $(1, 1, 0, 0)$ .

Com representação polinomial de tipo 0 o cálculo da soma é muito simples: para somar dois elementos  $x, y \in \text{GF}(2^n)$  representados pelos vectores  $x \sim, y \sim \in \text{GF}(2)^n$  basta somar as componentes bit a bit: a representação de  $x + y$  será  $x \sim \oplus y \sim$ .

O cálculo da multiplicação é mais complexo e exige a seguinte definição



## 144 DEFINIÇÃO

A **matriz companheira** do polinómio  $c[X]$  é uma matriz  $\mathbf{A} \in \text{GF}(2)^{n \times n}$  dada por

$$\begin{cases} \mathbf{A}_{ij} = 1 & \text{sse } i = j + 1 & \text{para } j < n - 1 \\ \mathbf{A}_{ij} = c_i & & \text{para } j = n - 1 \end{cases}$$

A matriz companheira de  $c[X]$  é a matriz que tem esse polinómio como polinómio característico e é “o mais simples possível”: a última coluna coincide com o vector das componentes de  $\mathbf{c}$  e as  $n - 1$  primeiras colunas têm o primeiro elemento sempre 0 e os restantes são determinados pela matriz identidade  $\mathbf{I}_{n-1}$  de dimensão  $n - 1$ .

No exemplo 17 a matriz companheira seria

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

A relação entre multiplicação e matriz companheira deriva do seguinte facto

145 FACTO

Para quaisquer  $x, y \in \text{GF}(2^n)$ , verifica-se  $(\beta \cdot x)^\sim = \mathbf{A} x^\sim$  e, genericamente,

$$(y \cdot x)^\sim = \bigoplus_{y_i=1} \mathbf{A}^i x^\sim \quad (52)$$

Para um qualquer vector  $u \in \text{GF}(2)^n$ , o cálculo de  $\mathbf{A} u$  é particularmente simples; se representarmos por  $\vec{u}$  o vector formado pelo desvio ("shift") de um bit de  $u$  para a direita, então facilmente se confirma que  $\mathbf{A} u = \vec{u} \oplus u_{n-1} \cdot c$ .

Um algoritmo eficiente para a multiplicação em (52), será

```

M := 0
para i = 0 ate n-1 fazer
  se bit0(y) entao M := M + x ; shiftright(y)
  shiftright(x) ; se carry entao x := x + c
fim

```

Numa base polinomial o cálculo do traço segue directamente a definição e usa este algoritmo de multiplicação para construir os diferentes quadrados. Para calcular o traço de  $\tilde{x}$  faz-se

```
T := x
```



```

para i = 1 ate n-1 fazer
  T := x + T*T
fim

```

Uma base polinomial do tipo 1 tem a forma

$$\mathcal{B}_{\beta,1} = \{ \beta, \beta^2, \dots, \beta^{n-1}, \beta^n \} \quad (53)$$

Aqui é o elemento 1 que é escrito como uma soma de elementos da base; sabendo que o polinómio característico tem sempre  $c_0 = 1$ , temos  $c(\beta) = 0$  e portanto

$$1 = c_1 \cdot \beta + c_2 \cdot \beta^2 + \dots + c_{n-1} \cdot \beta^{n-1} + c_n \cdot \beta^n$$

tendo em atenção que se tem sempre  $c_n = 1$ .

### Bases Normais

Escolhendo um elemento  $\rho \in \text{GF}(2^n)$  tal que  $\text{tr}(\rho) = 1$  então

$$\mathcal{N}_\rho = \{ \rho, \rho^2, \dots, \rho^{2^k}, \dots, \rho^{2^{n-1}} \}$$

forma uma base normal. De facto a soma dos elementos de  $\mathcal{N}_\rho$  é igual a 1 porque coincide com o traço de  $\rho$  (que, por hipótese, é igual a 1); nenhum outro subconjunto não-vazio de  $\mathcal{N}_\rho$  pode somar 0 porque, por aplicação sucessiva da função quadrado aos seus elementos, seriam gerados todos os elementos de  $\mathcal{N}_\rho$  que, somados, dariam 0 (contradizendo a conclusão anterior).

Um vector  $x^\sim = (x_0, \dots, x_{n-1}) \in \text{GF}(2)^n$  representa um elemento  $x \in \text{GF}(2^n)$  através da soma

$$x = x_0 \cdot \rho + x_1 \cdot \rho^2 + x_2 \cdot \rho^4 + \dots + x_{n-1} \cdot \rho^{2^{n-1}} \quad (54)$$

As igualdades essenciais para os diversos algoritmos são

$$1 = \rho + \rho^2 + \rho^4 + \dots + \rho^{2^{n-1}} \quad , \quad \rho = \rho^{2^n}$$

A primeira resulta da hipótese  $\text{tr}(\rho) = 1$ ; a segunda deriva da construção do corpo finito.

Tal como nas bases polinomiais, a representação da soma é a soma bit-a-bit das representações:  $(x + y)^\sim = x^\sim \oplus y^\sim$ , com  $x, y \in \text{GF}(2^n)$ .



Outras operações simples são a construção de quadrados e de traços; de facto, usando (54), tem-se

$$\begin{aligned} x^2 &= \left( \sum_{k=0}^{n-1} x_k \cdot \rho^{2^k} \right)^2 = \sum_{k=0}^{n-1} x_k \cdot \rho^{2^{k+1}} = \\ &= x_{n-1} \cdot \rho + \sum_{k=1}^{n-1} x_{k-1} \cdot \rho^{2^k} \end{aligned}$$

isto porque  $x_{n-1} \cdot \rho^{2^n} = x_{n-1} \cdot \rho$ . Consequentemente a representação de  $x^2$  obtém-se fazendo um desvio com rotação para a direita dos bits de  $x$ ; esta operação será representada por  $x^{\circlearrowright}$ .

$$(x^2)^{\sim} = (x^{\circlearrowright})$$

Para o cálculo do traço note-se que, para todo  $k$ ,  $\text{tr}(\rho^{2^k}) = \text{tr}(\rho) = 1$ . Portanto

$$\text{tr}(x) = \sum_{k=0}^{n-1} x_k = \text{Tr}(x^{\sim})$$

**Nota:** Para qualquer  $u \in \text{GF}(2)^n$  define-se  $\text{Tr}(u) = \sum_{i=0}^{n-1} u_i$

## A multiplicação

$$x \cdot y = \sum_{x_i=1} \sum_{y_j=1} \rho^{2^i} \cdot \rho^{2^j} \quad (55)$$

é bastante mais complicada e, genericamente, mais complexa nas bases normais do que nas bases polinomiais. A dificuldade reside no facto de (55) ser formado por termos da forma  $\rho^{2^i} \cdot \rho^{2^j} = \rho^{2^i+2^j}$  que é necessário converter para termos do tipo  $\rho^{2^k}$ .

Porém, para alguns valores particulares de  $n$ , a conversão é simples e a multiplicação nas bases normais é tão ou mais eficiente que em bases polinomiais. São as chamadas **bases normais ótimas**.

## 146 FACTO

Se  $p = n + 1$  é primo e  $2$  é gerador do grupo cíclico  $\mathbb{Z}_p^*$  então existe  $\rho$  que verifica  $\text{tr}(\rho) = 1$  e em que o conjunto

$$\left\{ 1, \rho, \rho^2, \rho^4, \dots, \rho^{2^{n-1}} \right\}$$

determina um sub-grupo cíclico de  $\text{GF}^*(2^n)$  de ordem  $p$ .

**Prova:** Se  $p$  for primo então  $2^{p-1} = 1 \pmod{p}$  e, portanto,  $2^n - 1 = 2^{p-1} - 1$  é divisível por  $p$ . Como  $2^n - 1$  é a ordem e  $\text{GF}^*(2^n)$ , concluímos que  $\text{GF}^*(2^n)$  contém um sub-grupo cíclico  $G \subseteq \text{GF}^*(2^n)$  de ordem  $p$ .



Seja  $\rho$  um gerador de  $G$ ; deste modo  $G = \{\rho^s \mid s \in \mathbb{Z}_p\}$ . Como, por hipótese, 2 é um gerador de  $\mathbb{Z}_p^*$  (que tem ordem  $p - 1 = n$ ) as potências  $2^k \pmod{p}$  (com  $k \in \mathbb{Z}_n$ ) geram os expoentes  $s = 1, 2, \dots, p - 1$  (mas não o expoente  $s = 0$  que determina o elemento  $\rho^0 = 1$ ); portanto temos

$$G = \{1\} \cup \{\rho^{2^k} \mid k \in \mathbb{Z}_n\} = \{1, \rho, \rho^2, \dots, \rho^{2^{n-1}}\}$$

Resta provar que  $\text{tr}(\rho) = 1$ ; tem-se

$$\text{tr}(\rho) = \sum_{k=0}^{n-1} \rho^{2^k} = \sum_{s=1}^{p-1} \rho^s = (\rho^p + \rho) \cdot (1 + \rho)^{-1} = 1$$

porque, dada a ordem de  $G$ ,  $\rho^p = 1$ .

147 FACTO

*Nas condições do facto 146 tem-se:*

(i) *Se  $2^i + 2^j = 0 \pmod{p}$  então*

$$\rho^{2^i} \cdot \rho^{2^j} = \rho + \rho^2 + \rho^4 + \dots + \rho^{2^{n-1}}$$

(ii) Se  $2^i + 2^j \neq 0 \pmod{p}$  então

$$\rho^{2^i} \cdot p^{2^j} = \rho^{2^{\lambda_{ij}}} \quad \text{com} \quad \lambda_{ij} \doteq i + \tau(j - i) \pmod{n}$$

em que  $\tau(\cdot)$  denota o logaritmo de Zech de base 2 em  $\mathbb{Z}_p^*$ .

### Prova

O resultado anterior diz-nos que os elementos  $\rho^{2^i}, p^{2^j}$  pertencem a um subgrupo multiplicativo de ordem  $p$ . Portanto  $\rho^{2^i} \cdot p^{2^j} = \rho^{2^{i+2^j}}$  é um elemento do mesmo subgrupo; o que significa que é 1 ou então é da forma  $\rho^{2^k}$  para algum  $k \in \mathbb{Z}_n$ .

Se  $2^i + 2^j = 0 \pmod{p}$  então  $\rho^{2^i} \cdot p^{2^j} = 1 = \text{tr}(\rho) = \rho + \rho^2 + \dots + \rho^{2^{n-1}}$ .

Se  $2^i + 2^j \neq 0 \pmod{p}$  então pode-se escrever na forma  $2^i (1 + 2^{j-i}) \pmod{p}$ . Se  $2^k \neq -1 \pmod{p}$ , o logaritmo de Zech  $\tau(k) \in \mathbb{Z}_n$  está definido e é um elemento que verifica  $2^{\tau(k)} = 2^k + 1 \pmod{p}$ ; então concluímos (com os expoentes vistos sempre como elementos de  $\mathbb{Z}_n$ )

$$2^i + 2^j = 2^i (1 + 2^{j-i}) = 2^i 2^{\tau(j-i)} = 2^{i+\tau(j-i)} \pmod{p}$$

Tabelando o logaritmo de Zech, o que não é muito difícil para os valores usuais de  $n$ , este resultado permite construir uma forma computacionalmente eficiente de implementar (55): permite construir o vector de bits, que representa  $\tilde{x} \cdot \tilde{y}$ , usando apenas operações básicas *xor* e *shift* sobre os vectores de bits  $x$  e  $y$ .



**EXEMPLO 18:** Tomemos de novo  $\text{GF}(2^4)$ ; note-se que  $p = n + 1$  é primo ( $p = 5$ ) e que 2 é gerador de  $\mathbb{Z}_5^*$ ; de facto, em  $\mathbb{Z}_5^*$ ,  $2^0 = 1$ ,  $2^1 = 2$ ,  $2^2 = 4$ ,  $2^3 = 3$ .

Portanto  $\text{GF}(2^4)$  tem uma base normal óptima e as multiplicações podem ser facilmente efectuadas. Tabelando o logaritmo de Zech pela definição ( $2^{\tau(k)} = 2^k + 1$ ) tem-se

$$\tau(0) = 1, \tau(1) = 3, \tau(2) = \perp, \tau(3) = 2$$

Suponhamos que se se pretende efectuar a multiplicação dos elementos  $x, y \in \text{GF}(2^n)$  que têm as representações  $x^\sim = (1, 0, 1, 0)$  e  $y^\sim = (0, 1, 1, 0)$ . Isto significa que  $x = \rho + \rho^4$  e  $y = \rho^2 + \rho^4$ .

Denotamos por  $\Lambda_{ij}$  a representação vectorial do elemento  $\rho^{2^i} \cdot \rho^{2^j}$ . Usando (55) vemos que só interessa calcular  $\Lambda_{ij}$  quando se verifica  $x_i = 1$  e  $y_j = 1$ .

$$(x \cdot y)^\sim = \Lambda_{01} \oplus \Lambda_{02} \oplus \Lambda_{21} \oplus \Lambda_{22}$$

Sempre que for  $j - i = 2 \pmod{4}$  obtemos um valor para o qual o logaritmo de Zech é indefinido e, neste caso, o resultado anterior diz-nos que  $\Lambda_{ij} = (1, 1, 1, 1)$ . Nas restantes situações  $\Lambda_{ij}$  é um vector que tem uma única componente igual a 1 determinada pelo índice  $\lambda_{ij} \doteq i + \tau(j - i) \pmod{4}$ .

Para facilitar o cálculo construímos a seguinte tabela



$i$	$j$	$j - i$	$\lambda_{ij}$
0	1	1	3
0	2	2	$\perp$
2	1	3	0
2	2	0	3

$$\Lambda_{01} = \Lambda_{22} = (0, 0, 0, 1)$$

$$\Lambda_{02} = (1, 1, 1, 1), \quad \Lambda_{21} = (1, 0, 0, 0)$$

$$\text{portanto } (x \cdot y)^\sim = (0, 1, 1, 1)$$

$$\text{ou seja } x \cdot y = \rho^2 + \rho^4 + \rho^8$$

## 2.7 Geração de sequências aleatórias e pseudo-aleatórias

A segurança de muitas técnicas criptográficas depende da capacidade de se gerar quantidades imprevisíveis. Sem perda de generalidade pode-se considerar que essas quantidades são bits e são geradas por duas classes de processos:

**Sequências aleatórias** são produzidas por dispositivos que dependem de fontes aleatórias naturais.

Em *hardware*: o ruído térmico em resistências ou semicondutores, as flutuações na frequência de um oscilador, etc...

Em *software*: o relógio do sistema, o tempo entre eventos de entrada (toques de tecla, movimentos do rato,...), parâmetros de carga do sistema operativo (tamanho do *heap* ou do *stack* de processos do sistema, tamanho de *buffers* ou filas de espera, ...).

Nenhuma destas fontes, isoladamente, pode ser considerada à prova de ataque. Recomenda-se sempre uma **mistura de fontes** que é feita concatenando valores de várias fontes, passando este valor por uma função de *hash* (como o SHA-1) e seleccionando apenas alguns dos bits do resultado.

Neste caso é considerado intratável (apesar de não ser formalmente demonstrado) atacar uma das fontes de modo a criar uma **tendência** (“*bias*”) em relação à emissão de um determinado valor (0 ou 1) ou **correlacionar** a emissão de um bit com emissões anteriores.

**Sequências Pseudo-aleatórias** de elementos de um domínio finito  $X$

$$x_1, x_2, \dots, x_i, \dots$$

É produzida a partir de outra sequência  $s_0, s_1, \dots, s_k, \dots$  por:

$$s_k = G(s_{k-1}), \quad x_k = h(s_k), \quad \text{com } k > 0$$

(sendo  $h$  uma função *one-way*) de tal forma que é satisfeita a condição: *Conhecidos os primeiros  $k$  valores da sequência  $x_1, \dots, x_k$  é computacionalmente intratável prever o próximo elemento  $x_{k+1}$  com probabilidade superior a  $\varepsilon + |X|^{-1}$ ;  $\varepsilon > 0$  é arbitrariamente pequeno.*

Seleccionando alguns dos bits dos vários  $x_k$  constrói-se a sequência de bits pretendida.

O valor  $s_0$ , que determina toda a sequência dos  $s_k$  (e portanto dos  $x_k$ ), chama-se *semente* (ou “*seed*”).

Sequências que não verificam a condição anterior não são consideradas **criptograficamente seguras** mas podem ser usadas noutro tipo de aplicações (por exemplo, na geração de valores de teste nos algoritmos de teste de números primos).



**EXEMPLO 19: (gerador linear)**

$$s_k = a \times s_{k-1} + b \qquad a, b, s_k \in \mathbb{Z}_p, \quad a \neq 0$$

$$x_k = s_k \pmod{2} \qquad x_k \in \mathbb{Z}_2$$

Prova-se que este gerador não é criptograficamente seguro pois consegue-se determinar  $s_0$  a partir do conhecimento de  $p$  e de poucos bits  $x_k$ .

**Gerador RSA**

Parâmetros: dois primos  $p$  e  $q$ , para os quais a factorização de  $m = p \cdot q$  é intratável, e  $a \in \mathbb{Z}_{\phi(m)}^*$ ; os valores  $a, m$  podem ser tornados públicos, o valor  $\phi(m) = (p - 1) \cdot (q - 1)$  é secreto e  $p$  e  $q$  são destruídos.

Semente: um valor aleatório  $s_0 \in \mathbb{Z}_m^*$  (gerado um por dos mecanismos de *hardware* ou *software* atrás referidos).

Gerador:

$$s_k = s_{k-1}^a \pmod{m}, \quad x_k = s_k \pmod{2}$$

Este gerador é criptograficamente seguro mas pouco eficiente porque necessita de uma exponenciação para cada bit gerado. Existe uma variante, chamado *gerador de Micalli-Schnorr*, que dá, por iteração, tantos bits quantos os necessários para representar  $m$  (tipicamente 1024).



## Gerador BBS (Blum-Blum-Shub)

Parâmetros dois primos  $p$  e  $q$  para os quais a factorização de  $m = p \cdot q$  é intratável e que verificam  $p = q = 3 \pmod{4}$ .

Semente: um valor  $s_0 \doteq \omega^2 \pmod{m}$  em que  $\omega \in \mathbb{Z}_m^*$  é aleatório.

Gerador

$$s_k = s_{k-1}^2 \pmod{m}, \quad x_k = s_k \pmod{2}$$

O gerador BBS é criptograficamente seguro mesmo quando se aproveita mais do que um bit por iteração. Porém não existe actualmente um processo de determinar quantos bits a mais é possível retirar de cada iteração mantendo a condição de segurança criptográfica.

## Norma ANSI X9.17

Parâmetros: usa a função de cifragem do triplo DES, com uma chave composta  $\kappa$ , e um parâmetro  $I \doteq \{d\}_\kappa$ , sendo  $d$  uma representação da data+hora com 64 bits.

Semente:  $s_0$  é um valor aleatório com 64 bits.

Gerador:

$$s_1 = I \oplus s_0 \quad x_i = \{s_i\}_\kappa, \quad s_{i+1} = I \oplus \{I \oplus x_i\}_\kappa \quad i > 0$$



## Gerador FIPS 186

O *Federal Information Processing Standard* (FIPS) 186 acompanha a norma do DSA (*Digital Signature Algorithm*) e define dois geradores de números pseudo-aleatórios: uma versão para a geração de pares de chaves públicos-privada e uma versão para geração de chaves de sessão.

Na primeira versão o utilizador pode modificar a semente produzida pelo implementador com uma semente por ele escolhida.

Essencialmente o gerador tem uma construção do tipo da anterior envolvendo, como “misturador” de bits em cada iteração a função de *hash* SHA-1.

Para detalhes consultar o *Handbook of Applied Cryptography* de Menezes *et al.*.

## 2.8 Factorização de Inteiros

Dado um inteiro  $m > 1$ , determinar primos  $1 < p_1 < \dots < p_k$  e expoentes  $e_1, \dots, e_k$  tais que  $m = p_1^{e_1} \times \dots \times p_k^{e_k}$ .

A factorização determina uma classe de funções *one-way*: computacionalmente a multiplicação é trivial mas a sua “inversa” factorização é, para valores apropriados de  $m$ , intratável. Existem porém algumas **factorizações triviais**.

**Factorização por divisão** Faz-se variar  $p$  ao longo dos “pequenos primos”  $(2, 3, 5, \dots)$  e testa-se  $m \equiv 0 \pmod{p}$ . Se tal for possível faz-se  $m \leftarrow m/p$  reduzindo a complexidade do problema da factorização.

Formalmente esta técnica encontra qualquer factorização de  $m$  só que implica  $\lceil \sqrt{m} \rceil$  cálculos do módulo o que é intratável para grandes  $m$ .

**Factorização de Fermat** Se  $m = p \times q$ , com  $q < p$  primos ímpares “próximos”, encontrar  $p$  e  $q$  é simples.

Define-se  $\mu = (p + q)/2$  e  $v = (p - q)/2$ ; donde  $m = p \times q = (\mu^2 - v^2)$  e

$$\mu^2 = m + v^2 \quad \text{com} \quad v^2 \ll m \quad (\dagger)$$

A partir do valor inicial  $\mu = \lceil (\sqrt{m}) \rceil$  e incrementando sucessivamente  $\mu \leftarrow \mu + 1$ , testam-se, para cada  $\mu$ , valores  $v = 0, 1, \dots, \lceil (\sqrt{\mu^2 - m}) \rceil$  até que  $(\dagger)$  se verifique.



Conhecidos  $\mu$  e  $v$  determina-se  $p = \mu + v$  e  $q = \mu - v$ .

**EXEMPLO 20:** Para  $m = 1127843$  temos o valor inicial  $\mu = \lceil(\sqrt{m})\rceil = 1062$ ; como  $m - \mu^2 = 1$  é logo um quadrado perfeito, conclui-se  $v = 1$  e  $\mu = 1062$ . Assim  $p = 1063$  e  $q = 1061$ .

### Factorização de Pollard

Gerando uma sequência pseudo-aleatória finita  $\rho_0, \rho_1, \dots, \rho_n$  em  $\mathbb{Z}_m$ , calculam-se os valores  $d_i = \gcd(\rho_i, m)$  até se obter algum  $d_s > 1$  (como resultado do processo) ou se atingir o limite da sequência dos  $\rho_i$  (com indicação de falha).

**Método rho** Os  $\rho_i$  são determinados por um gerador quadrático

$$\begin{aligned} x_i &= x_{i-1}^2 + 1 \pmod{m} & y_i &= (y_{i-1} + 1)^2 + 1 \pmod{m} & (\dagger) \\ \rho_i &= x_i - y_i \pmod{m} \end{aligned}$$

**Justificação** Se  $x_0 = y_0$  então  $y_i = x_{2i}$ , para todo  $i \geq 0$ : a solução é atingida quando for  $x_{2s} = x_s \pmod{p}$  para algum  $p$  (desconhecido) divisor de  $m$ .

Todo o eventual  $p$ , divisor de  $m$ , determina uma sequência  $x'_i = x_i \pmod{p}$  que verifica a mesma igualdade  $(\dagger)$  e que, eventualmente, será periódica; quando se atingir um  $s$  que seja múltiplo do período teremos  $x'_{2s} = x'_s$  e portanto atinge-se a solução. O número de operações necessárias para encontrar  $s$  é da ordem de  $\sqrt{\sqrt{m}}$ .



**Método (p-1)** Dado um limite  $B$ , os  $\rho_i$  são gerados por

$$X_1 = 2, \quad X_i = (X_{i-1})^i \pmod{m} \quad i = 2, \dots, B$$

$$\rho_i = X_i - 1$$

**Justificação** Se  $p$  for um divisor primo de  $m$  e for  $q \leq B$  para todo o divisor primo de  $(p-1)$ , então  $(p-1)$  tem de dividir  $B!$ . Pelo teorema de Fermat, será  $2^{B!} = 1 \pmod{p}$ .

No final do ciclo temos  $X_B = 2^{B!} \pmod{m}$ ; portanto  $X_B = 2^{B!} = 1 \pmod{p}$ . Logo  $(X_B - 1)$  é múltiplo de  $p$  e será  $\gcd(X_B - 1, m) > 1$ .

A condição pode-se verificar antes de  $i$  atingir  $B$ ; basta que  $X_i$  acumule um expoente múltiplo de  $(p-1)$ .

O problema está na determinação do valor  $B$  que seja limite superior de todos os factores de  $(p-1)$

As técnicas de factorização poderiam, por si só, dar origem a um curso tanto ou mais extenso quanto o que aqui apresentamos. Recentemente tem aparecido algoritmos que diminuiriam substancialmente a complexidade computacional deste problema. O criptógrafo preocupado com a segurança dos seus sistemas deve estar atento a estes métodos.

**ECM (Elliptic Curve Method)** É uma variante do método  $(p-1)$  de Pollard; note-se que  $(p-1)$  é a ordem do grupo cíclico  $\mathbb{Z}_p^*$  e os elementos  $X_i$  percorrem esse grupo. O ECM substitui o grupo  $\mathbb{Z}_p^*$  por uma curva elíptica aleatória sobre  $\mathbb{Z}_p$  cuja ordem  $B$  é computacionalmente limitada.



Detalhes deste e dos outros algoritmos desta secção podem-se obter em [A Course in Computational Algebraic Number Theory](#) de H.Cohen.

## Factorização Quadrática

148 FACTO

Se se verificar

$$x^2 = y^2 \pmod{m} \quad \& \quad x \neq \pm y \pmod{m} \quad (\dagger)$$

então  $\gcd(x - y, m)$  é um divisor não trivial de  $m$ . Se  $m$  for ímpar e tiver  $k$  factores primos distintos então, para cada  $y \in \mathbb{Z}_m^*$ , existem  $2^{k-1}$  soluções  $x \in \mathbb{Z}_m$  de  $(\dagger)$ .

Seja  $\mathcal{B} = \{p_1, \dots, p_k\}$  uma **base de primos**; um **número- $\mathcal{B}$**  é um inteiro cujos factores primos são todos elementos de  $\mathcal{B}$ . Se  $a = p_1^{e_1} \times \dots \times p_k^{e_k}$ , com  $e_i \geq 0$ , é um número- $\mathcal{B}$  representamos por  $\|a\|$  o vector em  $(\mathbb{Z}_2)^k$  formado pelas paridades dos vários expoentes:  $\langle e_1 \pmod{2}, \dots, e_k \pmod{2} \rangle$

Por tentativas geram-se pares  $\langle x_i, a_i \rangle$  tais que  $x_i^2 = a_i \pmod{m}$ , e os  $a_i$  são números- $\mathcal{B}$  de factorização conhecida.

Se forem encontrados  $a_1, \dots, a_\mu$  tais que  $\|a_1\| + \dots + \|a_\mu\| = 0$  então o produto  $a_1 \times \dots \times a_\mu$  é um quadrado perfeito cuja raiz quadrada  $y$  é facilmente calculada já que as factorizações dos  $a_i$  são conhecidas; se for



$x_1 \times \dots \times x_\mu \not\equiv \pm y \pmod{m}$ , como temos

$$(x_1 \times \dots \times x_\mu)^2 = a_1 \times \dots \times a_\mu = y^2 \pmod{m}$$

pode-se calcular  $\gcd(x_1 \times \dots \times x_\mu - y, m)$  e obter o factor pretendido.

Os melhores métodos generalistas conhecidos, nomeadamente o **Quadratic Sieve Method**, são baseados neste processo e definem critérios eficientes para gerar as diversas escolhas aqui indicadas.



## 2.9 Logaritmo Discreto

A segurança de muitas técnicas criptográficas depende crucialmente da característica *one-way* da “exponenciação” em **grupos cíclicos**: a função directa é computacionalmente tratável mas a função inversa deve ser computacionalmente intratável.

A propriedade criptograficamente significativa de um grupo cíclico finito  $\langle G, \cdot \rangle$  é a existência do **isomorfismo de grupos**  $\mathbb{Z}_{|G|} \simeq G$  estabelecida pela exponenciação  $\exp_g : i \mapsto g^i$  e pela sua função inversa  $\log_g : G \rightarrow \mathbb{Z}_{|G|}$  (o **logaritmo discreto** em  $G$ ).

O *Problema do Logaritmo Discreto* (PLD) em  $G$  é a determinação, dado o gerador  $g \in G$  e um elemento  $x \in G$ , do único  $i \in \mathbb{Z}_{|G|}$  tal que  $x = g^i$ . Em particular para  $G \equiv \mathbb{Z}_p^*$  temos

### 149 DEFINIÇÃO

*Dado um primo  $p$  ímpar, um algoritmo resolve LOGDISC( $p$ ) quando, dados um gerador  $\alpha \in \mathbb{Z}_p^*$  e um elemento  $\beta \in \mathbb{Z}_p^*$ , determina o único  $a \in \mathbb{Z}_{p-1}$  tal que  $\beta = \alpha^a \pmod{p}$ .*

### Procura exaustiva em memória

Constrói-se uma tabela de pares  $\langle i, \alpha^i \pmod{p} \rangle$ , com  $i = 0, \dots, (p-2)$  e ordena-se pela segunda componente. Dado  $\beta$  procura-se este valor na segunda coluna da tabela; o resultado  $a$  é dado pela primeira componente do par encontrado.



*Complexidade* A ordenação é feita para facilitar a procura; o algoritmo exige memória proporcional a  $p$  e um número de comparações proporcional a  $(p - 1)$ .

### Procura exaustiva no tempo

É gerada a sequência de elementos de  $\mathbb{Z}_p^*$

$$x_0 = 1, \quad x_i = \alpha \times x_{i-1} \pmod{p} \quad i = 1, 2, \dots, (p - 2)$$

que termina quando se verificar  $x_i = \beta$ . O índice  $i$  correspondente é o resultado pretendido.

*Complexidade* Exige memória constante mas tempo de procura é proporcional a  $(p - 1)$  multiplicações.

### Algoritmo de Shank (“baby-step, giant-step”)

É um compromisso entre os dois algoritmos anteriores. Escolhe-se  $n = \lceil \sqrt{p} \rceil$ , calcula-se  $\alpha_n \doteq \alpha^{-n} \pmod{p}$  e constrói-se a tabela de pares  $\langle j, \alpha^j \pmod{p} \rangle$ , com  $j = 0, \dots, n - 1$ .

Como cada  $a \in \mathbb{Z}_{p-1}$  pode ser escrito na forma  $a = i \times n + j$  com  $i, j \in \mathbb{Z}_n$ , temos

$$\beta = \alpha^a \pmod{p} \quad \text{sse} \quad \beta \times (\alpha_n)^i = \alpha^j \pmod{p}$$

Para  $i = 0, 1, \dots, n - 1$  calcula-se  $\beta_i \equiv \beta \times (\alpha_n)^i \pmod{p}$  e procura-se, na segunda coluna da tabela, algum elemento igual a  $\beta_i$ ; se existir, com o respectivo índice  $j$  e com o índice  $i$  de  $\beta_i$  pode-se calcular  $a$ .

*Complexidade* A tabela exige memória proporcional a  $\sqrt{p}$ ; o número total de procuras é proporcional a  $p$  e cada uma exige um número de multiplicações proporcional a  $\sqrt{p}$ .

Em relação aos dois algoritmos anteriores, este é um compromisso memória-tempo.

### Algoritmo de Pohlig-Hellman

Pretende-se determinar  $a \in \mathbb{Z}_{p-1}$  tal que  $\beta = \alpha^a \pmod{p}$ . Suponhamos<sup>32</sup> que é conhecida a factorização de  $(p - 1)$ .

Fazendo  $q$  percorrer todos os factores primos de  $(p - 1)$ , se for possível determinar  $a_q \in \mathbb{Z}_q$  tal que

$$a = a_q \pmod{q} \quad (\dagger)$$

então é possível calcular  $a$  usando o teorema chinês dos restos.

1º caso:  $q$  é primo

---

<sup>32</sup>Muitas técnicas criptográficas usam primos da forma  $p = q2^t + 1$ , com  $q$  primo.

Seja  $r = (p - 1)/q$  e  $\gamma = \alpha^r \pmod{p}$ ; de (†) concluímos  $q|(a - a_q)$  e portanto  $(p - 1)|(a - a_q)r$  já que  $(p - 1) = qr$ .

Donde  $a_r = r a_q \pmod{p - 1}$  e, pelo isomorfismo  $\mathbb{Z}_{p-1} \simeq \mathbb{Z}_p^*$ , temos

$$(\alpha^a)^r = (\alpha^r)^{a_q} \pmod{p} \quad \text{ou seja} \quad \beta^r = (\gamma)^{a_q} \pmod{p} \quad (\ddagger)$$

Pode-se ver (‡) como um problema de cálculo do logaritmo discreto  $a_q$  de um novo elemento  $\beta^r \pmod{p}$  num grupo multiplicativo de gerador  $\gamma$  e ordem  $q$ . Este problema é resolvido por um dos algoritmos anteriores.

2º caso:  $q \equiv \mu^e$  com  $\mu$  primo.

Representa-se  $a_q$  na base  $\mu$ : temos  $a_q \equiv x_1 + \mu x_2 + \dots + \mu^{e-1} x_e$ .

De (†) temos  $\mu|(a - x_1)$ ; tal como anteriormente, com  $r = (p - 1)/\mu$ , tem-se  $a_r = r x_1 \pmod{p - 1}$  o que permite, determinar  $x_1$  resolvendo um PLD num sub-grupo de ordem  $\mu$  e gerador  $\gamma = \alpha^r \pmod{p}$ .

Tem-se  $\mu^2|(a - x_1 - x_2 \mu)$  donde  $(a - x_1)(r/\mu) = r x_2 \pmod{p - 1}$ . Daí determina-se  $x_2$  resolvendo um PLD no mesmo sub-grupo.

E assim sucessivamente determinam-se todos os dígitos  $x_1, x_2, \dots, x_e$ .

**Complexidade:** *proporcional ao maior divisor primo de  $(p - 1)$*



## Método $\rho$ -Pollard

Quando se pretende resolver o problema do logaritmo discreto num sub-grupo de ordem  $q$  de  $\mathbb{Z}_p^*$  pode-se usar um método análogo ao método da factorização  $\rho$  de Pollard.

**Problema** Dado  $q$  primo que divide  $(p - 1)$ , dados  $\alpha, \beta \in \mathbb{Z}_p^*$  de ordem  $q$ , determinar  $\mu \in \mathbb{Z}_q$  tal que  $\beta = \alpha^\mu \pmod{p}$ .

## Algoritmo

1. Divide-se o subgrupo gerado por  $\alpha$  em três conjuntos disjuntos  $S_0$ ,  $S_1$  e  $S_2$  de tamanho aproximadamente igual e facilmente computáveis.<sup>33</sup>
2. Define-se a sequência

$$x_0 = 1 \quad x_{i+1} = \begin{cases} \beta x_i \pmod{p} & \text{se } x_i \in S_0 \\ x_i^2 \pmod{p} & \text{se } x_i \in S_1 \\ \alpha x_i \pmod{p} & \text{se } x_i \in S_2 \end{cases}$$

<sup>33</sup>Por exemplo,  $S_i \equiv \{x \mid x = i \pmod{3}\}$   $i = 0, 1, 2$ .

3. Tem-se  $x_i = \alpha^{a_i} \beta^{b_i} = \alpha^{a_i + \mu b_i} \pmod{p}$  em que

$$(a_0, b_0) = (0, 0) \quad (a_{i+1}, b_{i+1}) = \begin{cases} (a_i, b_i + 1) \pmod{q} & \text{se } x_i \in S_0 \\ (2a_i, 2b_i) \pmod{q} & \text{se } x_i \in S_1 \\ (a_i + 1, b_i) \pmod{q} & \text{se } x_i \in S_2 \end{cases}$$

4. Usando a iteração de Pollard-Floyd, determina-se  $x_k$  tal que

$$x_k = x_{2k} \pmod{p}$$

Nesse caso  $\alpha^{a_k + \mu b_k} = \alpha^{a_{2k} + \mu b_{2k}} \pmod{p}$  e, portanto,

$$\mu = (b_k - b_{2k})^{-1} (a_{2k} - a_k) \pmod{q}$$

A sequência de pares definida em (3.) permite calcular  $(a_k, b_k)$  e  $(a_{2k}, b_{2k})$ ; donde é possível determinar  $\mu$ .

**Complexidade computacional:** *proporcional a  $\sqrt{q}$ .*