Técnicas Criptográficas

José Manuel E. Valença *

10 de Novembro de 2008

*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 algorimos 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 ε. $\mathbb{B}^*$ 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 ≥ 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 Assimptótica

Frequentemente lidamos com funções reais de argumento inteiro positivo $f : \mathbb{N} \longrightarrow \mathbb{R}_+$ 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 ∞ a $\mathbb{N}$. Como símbolo ∞ deve verificar

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

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* $\lim_{n\to\infty} f(n)/g(n) = 0$.
- $f(n) = \Theta(g(n))$ *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* $f = \Omega^{\log}(g)$ *equivalente a* $g = O^{\log}(f)$.

Notas

1. A notação pode ser usada para exprimir várias tipos de propriedades das funções $n \mapsto f(n)$.Por exemplo $f(n) = c + o(1)$ é apenas um modo de escrever $\lim_{n\to\infty} f(n) = c$; 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 $\Theta(.)$ 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

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

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

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

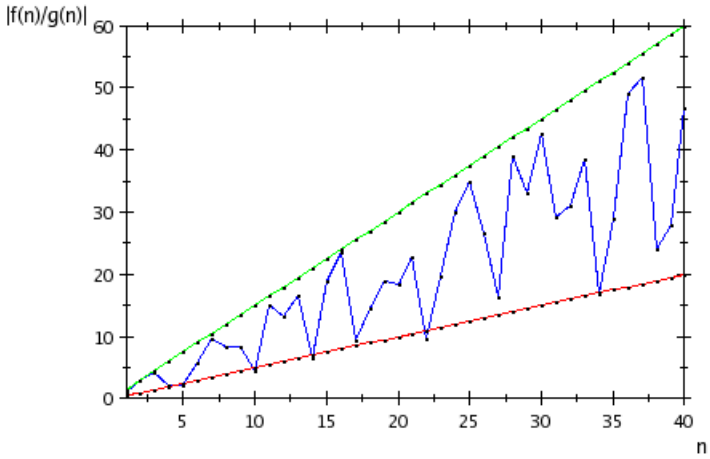
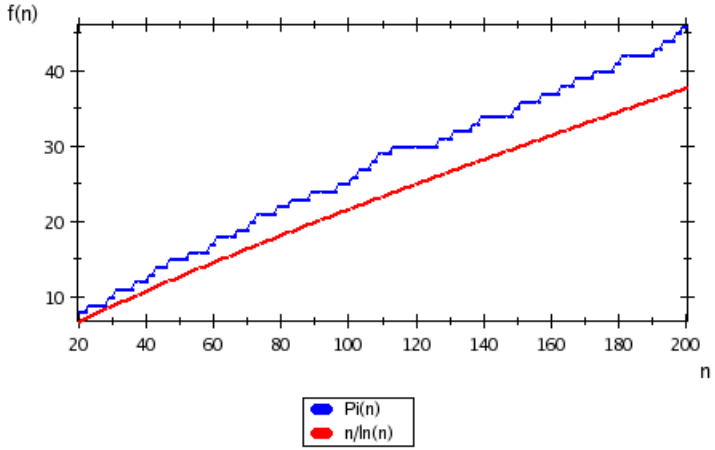


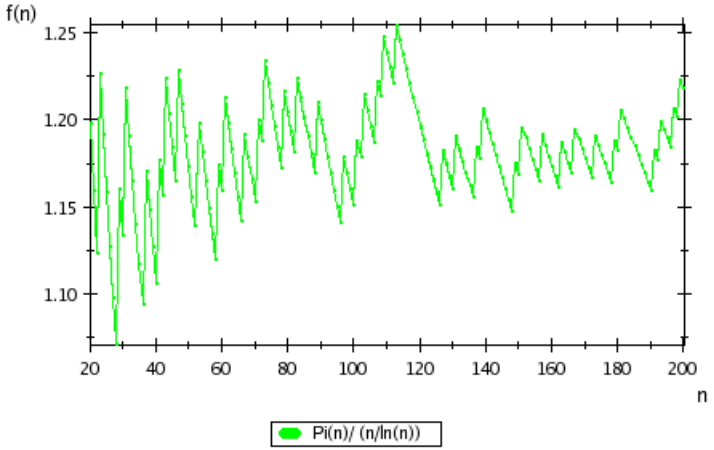
Figura 1: $f(n) = \Theta(n\,g(n))$.

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 π 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 ~ 1. Por isso faz sentido afirmar que

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

No estudo dos algoritmos a notação assimptótica 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 uma 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 $\tau : \mathbb{N} \longrightarrow \mathbb{R}_+$ 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*. [1]

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

2 NOÇÃO (ORDEM DE COMPLEXIDADE)
Diz-se que τ *tem* ***ordem***

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

[1] Em alternativa, a complexidade pode contar o número *máximo* ou o número *mínimo* de *slots* temporais ou unidades de memória.
[2] Como casos particulares temos as ordens **linear** ($\tau(n) = O(n)$), **quadrática** ($\tau(n) = O(n^2)$), **cúbica** ($\tau(n) = O(n^3)$), etc. . .
[3] 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 $L[p, c](n) = O(2^{c\, n^p\, (\log_2 n)^{1-p}})$ *, 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 $L[1, c](n) = O(2^{c\, n})$ 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).

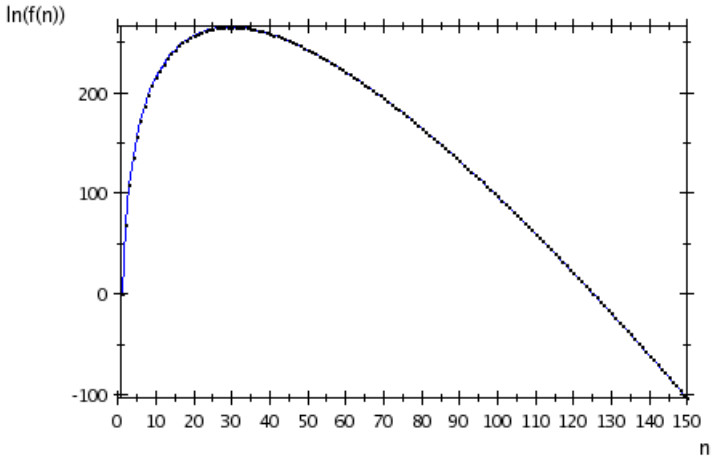
□

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[4]

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 $f, g : \mathbb{N} \longrightarrow \mathbb{N}$ 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. $\digamma f$ é equivalente a $(\forall_\infty x)[f(x)\uparrow]$; ou seja, é finito o número de argumentos x onde $f(x)$ é definido.
4. $f(x)\downarrow\downarrow = (\forall y < x)[f(y)\downarrow]$ é o predicado que indica que $f(y)$ é definido para todos os valores $y < x$.
5. Funções parciais admitem uma ordem parcial $\lesssim$ definida por

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

[4] Este capítulo e o seguinde (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).*

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: $(\forall x \in \mathbb{N})\,[\emptyset(x)\uparrow]$.

6. $f \simeq g$ 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

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

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

$$f(x)\downarrow \;\Rightarrow\; f(x) \simeq 0$$

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.

Um dos problemas essenciais na construção de funções é a representação dos pares de inteiros. Vamos assumir a existência de um operador binário $.\|. : \mathbb{N}\times\mathbb{N} \longrightarrow \mathbb{N}$ que codifica pares de inteiros em inteiros e é sobrejectivo. Vamos

assumir também que existem duas **funções projecção** sobrejectivas $\pi_1, \pi_2 : \mathbb{N} \longrightarrow \mathbb{N}$ que verificam $\forall x, y, z \in \mathbb{N}$

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

□

Em função da classe de máquinas de Turing em causa pode-mos propor várias opções para a noção de computabilidade:

Máquinas de Turing Determinístinas (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 selecionada pela posição, determina um novo estado e altera memória e posição.

O triplo *estado*×*posição*×*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* $f : \mathbb{N} \to \mathbb{N}$ *com compexidade* $T(n), S(n)$ *– e escreve-se* $f \lesssim_{T,S} M$ *– se, partindo de uma configuação 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 quado uma das possibilidades de transição atingir o estado final. A máquina é consistente se, nessas circusntâncias, 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 $f : \mathbb{N} \longrightarrow \mathbb{N}$ com compexidade $T(n), S(n), \varepsilon(n)$ 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 compuáveis 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 restricções 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 computabiliadade 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.

Teorema 1
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**.

7 Noção (Funções Primitivas e Primitivas Recursivas)
São **funções primitivas** *a constante* 0, *a função "sucessor"* $\mathsf{suc}\colon x \mapsto x+1$, *e as projecções* π_1, π_2.

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

$$f(x\|0) \simeq g(x) \quad , \quad f(x\|\mathsf{suc}(y)) \simeq h(x\|y\|f(x\|y)) \tag{2}$$

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**.

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* $z < y$.

$$P(x) \Rightarrow (\exists_1\, y \leq x)[P(y) \wedge (\forall z < y)\, \neg P(z)] \tag{3}$$

Denota-se por μP o valor y que verifica (3). Caso não exista nenhum x onde a propriedade P ocorra diz-se que μP é indefinido e representa-se esse facto por $\mu P{\uparrow}$.

Note-se que a capacidade para determinar esse valor μP depende crucialmente do facto de a propriedade ser sempre definida pelo menos até ao valor y.

Uma algoritmo possível para determinar μP 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 indicidível 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,

$$\mathcal{Z}[f](x) = f(x)\!\downarrow\!\downarrow \wedge f(x) \simeq 0 \tag{4}$$

$$\mathcal{Z}[l, f](x) = x \leq l \wedge \mathcal{Z}[f](x) \tag{5}$$

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.

8 NOÇÃO (RECURSIVIDADE)
A função parcial h define-se por **recursividade-μ** *não-limitada, a partir da função parcial f se*

$$h(y) \simeq \mu\mathcal{Z}[f_y] \qquad \textit{em que} \quad f_y(x) = f(y\|x) \tag{6}$$

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-μ 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-μ não-limitada a partir de uma função recursiva.

9 NOÇÃO
1. *Funções parciais recursivas f cujo domínio é finito (isto é, onde se verifica $\digamma f$) designam-e 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* $\mathcal{RS}$ [5].

Relativa a uma relação recursiva arbitrária ρ 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 $\mu[\rho(x) \simeq 1]$.
(iii) ρ^+ e ρ^- denotam as funções parciais cujo único valor definido é 0 e que verificam

$$\rho^+(x)\downarrow \;\Leftrightarrow\; \rho(x) \qquad , \qquad \rho^-(x)\downarrow \;\Leftrightarrow\; \neg\rho(x) \tag{7}$$

É simples provar que ambas são funções parciais recursivas.

Uma propriedade essencial destas sub-classe de funções:

10 FACTO
A sub-classe das funções características é fechada por composição, recursividade primitiva e recursividade-μ *não limitada. A sub-classe das relações recursivas é fechada por composição e recursividade primitiva.*

[5] De "recursive set".

□

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 fundametais da Matemática liga estes dois conceitos.

Teorema 2 (Computabilida 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 restricções de recursos. Para vários outros modelos da computação (autómatos, λ-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 restricções de recursos são modelos pouco realistas da computação; normalmente "computabilidade" subentende "computabilidade efectiva", no sentido em que só são interressantes 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 espistemologia (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 $\mathcal{I}\colon \mathcal{R} \longrightarrow \mathbb{N}$; 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

PROPOSIÇÃO 3 (AVALIAÇÃO DAS FUNÇÕES RECURSIVAS)
Existe uma avaliação recursiva $\varphi\colon \mathbb{N} \longrightarrow \mathbb{N}$ *tal que, para todo* $f \in \mathcal{R}$ *e todo* $p \in \mathbb{N}$

$$\mathcal{I}(f) = p \qquad \Longleftrightarrow \qquad (\forall x)\,[\,\varphi(p\|x) \simeq f(x)\,] \tag{8}$$

A função recursiva φ é um "interpretador universal"[6] 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 φ (que é recursiva) pode ser normalizada para uma construção que usa duas funções primitivas recursivas.

[6] Os programadores de linguagens funcionais reconhecem em φ a função `eval` do LISP ou o operador `$` do HASKELL.

Teorema 4 (Normalização das Funções Parciais Recursivas)
Existe uma função primitiva recursiva $\mathcal{U}$ *, e uma relação primitiva recursiva* $\mathcal{S}\colon \mathbb{N} \to \mathbb{N}$ *, que determinam uma função parcial recursiva* φ *por*

$$\varphi(x) \simeq \mathcal{U}(\mu \mathcal{S}_x) \qquad \text{sendo} \quad \mathcal{S}_x(y) = \mathcal{S}(x\|y) \tag{9}$$

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. φ_n.

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

Comparando com a proposição 3 nota-se uma restricção importante: a "avaliação" φ 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 φ diz-nos que a recursividade-μ 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-μ não-limitada.

Esta única utilização da recursividade-μ não-limitada sugere uma variante limitada da função φ substituindo a recursividade ilimitada por recursividade limitada.

12 NOÇÃO

Sejam $\mathcal{U}, \mathcal{S}$ as funções primitivas recursivas definidas no teorema 4. Para cada natural l, a **enumeração das aproximações** *é a sequência de funções $\{\varphi_n^l(x) = \varphi^l(n\|x\}$ com φ^l definida por*

$$\varphi^l(x) \simeq \mathcal{U}(\mu\mathcal{Z}[l, \mathcal{S}_x]) \qquad sendo \quad \mathcal{S}_x(y) = \mathcal{S}(x\|y) \tag{10}$$

As propriedades essenciais da aproximação φ^l são

$$\varphi^l \lesssim \varphi \quad , \quad \varphi^l(x) \simeq \varphi(x) \Leftrightarrow (\exists y < l)\, \mathcal{S}(x\|y) \tag{11}$$

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 extendida da seguinte forma.

13 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 η 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 progrma 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 ψ_e, de tal forma que, com esse valor x e o programa original e, se constrói um novo progrma $i = \eta(x\|e)$ 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:

TEOREMA 5

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 R, como **funções extensionais** nos índices. Formalmente

14 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 restricção 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,

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

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 φ, 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,

$$F[\varphi_e](x) \simeq \varphi(f(e)\|x) \tag{12}$$

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

$$F_n[\varphi_e](x) \simeq \varphi(f(n\|e)\|x) \tag{13}$$

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.

15 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

$$f \in F^{\sim} \quad \Leftrightarrow \quad (\exists e)\,[\, f \simeq \varphi_e \,] \wedge (\exists u \in F)\,[\, \digamma u \wedge u \lesssim f \,] \tag{14}$$

Uma classe de funções $\mathcal{A}$ diz-se **efectivamente enumerável (e.e.)**[7] *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.

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

$$\varphi_{\kappa(y)}(0) \simeq y \quad , \quad \varphi_e(0) \not\simeq y \quad \textit{par todo} \quad e < \kappa(y) \tag{15}$$

[7] Também designada por **completamente recursivamente enumerável** ou Σ^0_1-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-μ a partir da função recursiva parcial $g_y(e) \simeq \varphi(e\|0) - y$. No entanto a recursividade μ, apesar de possível, não produz o resultado pretendido já que $g_y(e)$ pode não ser definida para valores $e < \kappa(y)$. Por isso

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

18 NOÇÃO
Um inteiro y diz-se **compressível** *quando $y > \kappa(y)$. Os inteiros não compressíveis dizem-se κ-*__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 8, página 17) uma relação Φ (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** Φ

19 Noção (Recursividade com Oráculos)
$\mathcal{RP}^{\Phi}$ é a menor classe de funções que contém Φ, as funções primitivas e é fechada por composição, recursividade primitiva e recursividade-μ ilimitada.

A noção de oráculo deriva, originalmente, das máquinas de Turing. Numa máquina de Turing com oráculo Φ, cada passo pode, ou não, recorrer ao oráculo Φ 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 Φ 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.

20 Noção
Uma máquina de Turing com oráculo M^{Φ} **simula** *a função f com compexidade $T(n), S(n), \varepsilon(n), O(n)$ 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.

Teorema 6
Existem oráculos recursivos Φ *e* Ψ *tais que* $\mathrm{P}^{\Phi} = \mathrm{NP}^{\Phi}$ *e* $\mathrm{P}^{\Psi} \neq \mathrm{NP}^{\Psi}$.

Como sabemos, a asserção $\mathrm{P} \neq \mathbf{NP}$ 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 asociar 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.

Proposição 7
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 $\overline{f}$ são aqueles para os quais existe um relação recursiva ρ tal que $f \simeq \rho^{+}$ e $\overline{f} \simeq \rho^{-}$.

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 ϖ.

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

$$(f \cup g)(x)\downarrow \;\Leftrightarrow\; f(x)\downarrow \vee q(x)\downarrow \qquad , \qquad (f \cap g)(x)\downarrow \;\Leftrightarrow\; f(x)\downarrow \wedge q(x)\downarrow$$

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

$$\overline{f}(x)\downarrow \quad \Leftrightarrow \quad f(x)\uparrow$$

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

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

21 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*

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

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 ρ tal que $x \in A \Leftrightarrow \rho(x)$.*

O conjunto recursivo A, que verifica $(\forall x)[x \in A]$, designa-se por ϖ; extensionalmente ϖ 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.

Como consequência da proposição 7 e do facto 10 tem-se

Teorema 8 (Consistência Estrutural dos Conjuntos r.e.)
1. *Os conjuntos recursivamente enumeráveis formam um reticulado distributivo* **ReSet** *com o limite superior $\emptyset$ e*

limite superior ϖ. 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

22 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 comutáveis tem de ser suficientemente lata para que se verifique um resultado de consistência estrutural análogo ao teorema 8. 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 4, página 22). Seja φ 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.

TEOREMA 9

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" $\delta\colon x \mapsto \varphi_x(x) = \varphi(x\|x)$ 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

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

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.

23 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 desta definição basta mostrar que o contradomínio de uma qualquer função parcial recursiva $f \neq \emptyset$ é também contra-domínio de uma função recursiva. **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

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

Como $x \mapsto \varphi^l(e\|x)\downarrow$ 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. Vamos iniciar por algo intermédio.

Teorema 10 (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 f, então é o contradomínio da função parcial $r(x) \simeq x + f(x)$. 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-μ para procurar, por ordem crescente, os elementos do

conjunto. A seguinte função é recursiva, é monótona não decrescente e enumera os elementos de A.

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

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

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 φ_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 existem, 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 ρ 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

$$\mathcal{W}_x \in \mathcal{S}_a \Leftrightarrow D_a \subseteq \mathcal{W}_x \tag{16}$$

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 extender 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.

24 NOÇÃO (CLASSE EFECTIVAMENTE ENUMERÁVEL DE CONJUNTOS)
Uma classe de conjuntos recursivamente enumeráveis $\mathcal{A}$ é **efectivamente enumerável**[8]*, se existe um conjunto*

[8] Também designada por **completamente recursivamente enumerável** ou por Σ_1^0-**classe**.

recursivamente enumerável A, chamado **representação** *de $\mathcal{A}$, tal que*

$$\mathcal{W}_x \in \mathcal{A} \quad \Leftrightarrow \quad (\exists a)\,[a \in A \wedge D_a \subseteq \mathcal{W}_x\,] \tag{17}$$

Equivalentemente, se existe uma funcional efectiva F tal que

$$\mathcal{W}_x \in \mathcal{A} \quad \Leftrightarrow \quad \varphi_x \in F^{\tilde{}} \tag{18}$$

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

25 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

26 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, $\mathcal{A} = \{\mathcal{W}_{r(e)}\}_{e\in\varpi}$.*

Em (17), sem perda de generalidade, pode-se considerar que os elementos da representação A codificam conjuntos

finitos disjuntos dois a dois. Isto é, A verifica

$$(a \in A) \wedge (b \in A) \quad \Rightarrow \quad D_a \cap D_b = \emptyset \tag{19}$$

27 Noção (Array)
Um conjunto recursivamente enumerado A que verifica (19) *designa-se por* **array** [9].

28 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,

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

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

[9] No Odifreddi (pag. 228) este conceito é designado por "strong disjoint array".

Conjuntos "quase" recursivos

Vimos que a diferença essencial entre um conjunto recursivo (ou computável) e um conjunto r.e (ou c.e.) 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**.

30 NOÇÃO (CONJUNTOS SEMI-RECURSIVOS)
Uma função recursiva σ *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* σ *tal que, para todos* x, y

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

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 $A = \{a_x = r(x)\}_{x \in \varpi}$. O conjunto diz-se **retráctil** se existe uma função parcial recursiva σ tal que

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

31 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 [10].

32 NOÇÃO (CONJUNTOS SIMPLES)
*Um conjunto é **simples** se é recursivamente enumerável e o seu complemento é imune.*

O seguinte resultado exprime um conceito fundamental da aleatoriedade de Kolmogorov

TEOREMA 11 (KOLMOGOROV)
O conjunto dos números κ-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.

[10] Recordemos que imunes são os conjuntos infinitos que não contém qualquer subconjunto infinito recursivamente enumerável.

1.4 Domínios

...God made the integers, all else is the work of man...[11]

...God made the bit-strings, all else is the work of programmers...[12]

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" $\wp(\mathbb{N})$.

33 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 $\psi : \mathbb{N} \longrightarrow X$, o domínio X diz-se* ***enumerável****.*

Um domínio que está em correspondência biunívoca com <u>todo</u> $\mathbb{N}$ ***identifica-se*** *com ϖ.*

[11] LEOPOLD KRONECKER- 1823/1891
[12] 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 ϖ.*

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 $y = 0$.

34 Noção (Representação e Codificação)
Se X é um qualquer domínio enumerável, uma função sobrejectiva $r\colon \mathbb{N} \to X$ 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 $s\colon X \to \mathbb{N}$ que mapeie cada $x \in X$ num dos seus r-códigos[13] 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, $2^{<\varpi}$ e 2^{ϖ}.

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

35 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*

[13] 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.

representado por $|x|$. *Se* u *é uma string infinita, convenciona-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.

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

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

Representa-se por $x \leq v$ quando existe uma terceira string s tal que $x\,s = v$. 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.

$$\uparrow x \;=\; \{\, u \in \mathbb{B}^\infty \mid x \leq u \,\} \;=\; \{\, x\,v \mid v \in \mathbb{B}^\infty \,\} \tag{20}$$

Um sub-conjunto $L \subseteq 2^{<\varpi}$ 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 $L = (1\,1)^*$ e $H = 1\,(1\,1)^*$. 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

$$\upsilon \in \uparrow L \quad \Leftrightarrow \quad (\exists^1 x \in L)\,[\, x \leq \upsilon \,] \tag{21}$$

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.

36 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 $n \to \infty$, 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.

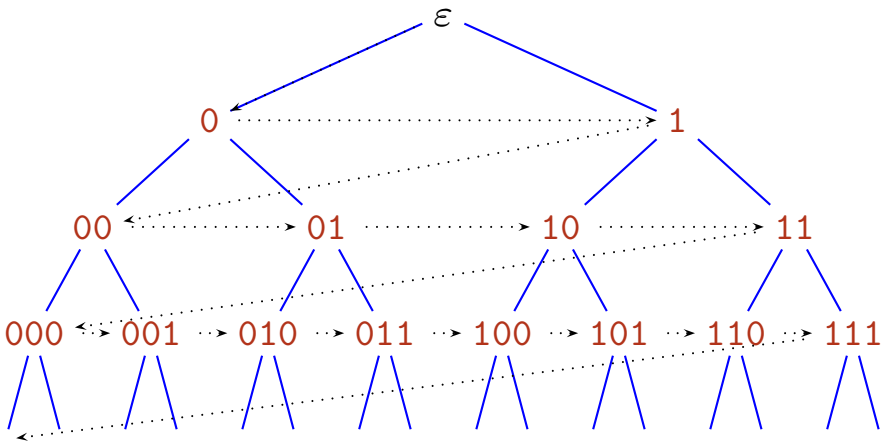


Figura 2: Enumeração das bit-strings finitas.

A função representação $r\colon \mathbb{N} \to \mathbb{B}^*$, e a respectiva codificação $s\colon \mathbb{B}^* \to \mathbb{N}$, constroem-se a partir das funções

biunívocas $r_n : [0, 2^n - 1] \longrightarrow \mathbb{B}^n$, 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

$$r(x) \;=\; r_{|x|}\Big(x - (2^{|x|} - 1)\Big) \qquad , \qquad s(u) \;=\; (2^{|u|} - 1) + s_{|u|}(u) \tag{22}$$

Considere-se a representação e codificação standard de bit-strings finitas defindas em (22). Dado que $2^{<\varpi}$ e ϖ se identificam, a cada linguagem $L \subseteq 2^{<\varpi}$ corresponde univocamente um conjunto de naturais formado pelos códigos das strings em L: a cada linguagem L está associado um único conjunto $A_L \in 2^{\varpi}$ dada por

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

Inversamente, a cada conjunto finito $A \in 2^{\varpi}$ corresponde uma única linguagem $L_A \subseteq 2^{<\varpi}$ 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 $2^{2^{<\varpi}}$ identifica-se com o conjunto de todos os conjuntos de naturais $\wp(\mathbb{N})$, e ambos são designados por 2^{ϖ}.

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

37 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** *(κ-aleatório) se é finita e A_L é compressível (κ-aleatório).*

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

38 NOÇÃO
- *Cada conjunto A identifica-se com a relação α tal que $\alpha(x) \simeq 1 \Leftrightarrow x \in A$.*
- *Cada relação α 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^{ϖ}; para cada uma destas facetas pode-se ver uma noção de "truncatura".

39 NOÇÃO
Vendo $A \in 2^{\varpi}$ como bit-strings infinita, a **truncatura** *$A \restriction n$ denota o prefixo de comprimento n de A .*

Vendo $A \in 2^{\varpi}$ *como um conjunto, linguagem ou relação,* $A \restriction n$ *denota o prefixo de comprimento* n *da string infinita identificada com o objecto* A.

Cada $A \restriction n$, por ser uma string finita, está identificada pelo seu código $s(A \restriction n)$. Caso não exista ambiguidade designaremos, indistintamente, por $A \restriction n$ tanto a string como o número natural que a codifica.

40 Noção
Dados $x \in \varpi$ *e* $A \in 2^{\varpi}$, *designaremos por* $x \prec A$ *o predicado* $(\exists n)\,[A \restriction n \simeq x]$.

- *Para cada* $A \in 2^{\varpi}$, *o conjunto* $\{\, x \in \varpi \mid x \prec A \,\}$ *é representado por* $\downarrow A$.
- *Para cada* $x \in \varpi$, *a classe* $\{\, A \in 2^{\varpi} \mid x \prec A \,\}$ *é representada por* $\uparrow x$.

Reais

Indistintamente 2^{ϖ} 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 $\mathbb{I} = [0, 1]$.

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

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

$$x^{\sim} = \sum_{k<|x|} x_k \, 2^{-k-1}$$

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 $\cdot^{\sim}\colon \mathbb{B}^* \to \mathbb{Q}_+$ é monótona crescente e limitada a 1.

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

$$u^{\sim} \quad = \quad \sum_{k=0}^{\infty} u_k \, 2^{-k-1} \tag{23}$$

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

$$u^{\sim} \; = \; \lim_{n \to \infty} (u \restriction n)^{\sim}$$

Para simplificar a notação designemos por $x_n \in 2^{<\varpi}$ a string finita $x_n = u \restriction n$ que se obtém truncando u a n bits. Designemos por $\downarrow u$ 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 $x_n = u \restriction n$ 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,

42 NOÇÃO (REAL RECURSIVO (R.R.))
Um **real recursivo** *é um sub-conjunto recursivo* $R \subseteq \mathbb{Q}_+$ *que tem um limite superior.*

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 uam linguagem recursiva cujos elementos formam uma ordem total, pode ser enumerada por uma função recursiva, monótona, não-decrescente (teorema 10, página 37). Os elementos de L, assim enumerado, originam uma sequência não-decrescente de racionais de Lebesgue.

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

Tomemos, como exemplo, uma string finita $x \in 2^{<\varpi}$ e a classe $\uparrow x = \{\, x\,v \mid v \in 2^{\varpi} \,\}$ formada por todas as strings infinitas $u = x\,v$ que têm x como prefixo. Pela definição (23) tem-se

$$u^{\tilde{}} \;=\; x^{\tilde{}} \,+\, 2^{-|x|}\, v^{\tilde{}}$$

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

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

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:

44 Noção (σ-Álgebra)
*Seja X um domínio. Uma família Σ de sub-conjuntos de X, fechada por uniões contáveis e complementos, designa-se por σ-**álgebra**. Os elementos de uma σ-álgebra designam-se por **eventos**.*

Notas Porque Σ 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 Σ é também fechado pela intersecção contável de eventos.

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

45 Noção (Medida em σ-álgebras)
*Se Σ é uma σ-álgebra, uma função $\mu : \Sigma \to \mathbb{R}_+$ é uma **medida** se verifica:*

1. $\mu(\bigcup_i \alpha_i) = \sum_i \mu(\alpha_i)$ *para qualquer enumeração* $\{\alpha_i\}$ *de eventos mútuamente disjuntos.*

2. Se $\{\alpha_i\}$ *é uma sequência decrescente* ($\alpha_{i+1} \subseteq \alpha_i$ *para todo* i), *então* $\mu(\bigcap \alpha_i) = \lim_{i\to\infty} \mu(\alpha_i)$.

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 destas cadeia tenha uma medida que é o limite das medidas dos elementos da sequência.

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

Na perspectiva do domínio genérico 2^{ϖ}, pode-se construir uma generalização desta σ-álgebra.

46 Noção (σ-álgebra de Lebesgue)
No domínio 2^{ϖ} *a* **σ-álgebra de Lebesgue**, *representada por* $\mathcal{L}$, *é a menor* σ*-álgebra que contém todos as classes da forma* $\uparrow x$ *com* $x \in \varpi$.

Nas σ-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**.

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

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

É importante nota-se que esta definição de σ-álgebra de Lebesgue é feita para o domínio abstracto 2^{ϖ} 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 μ (e não de x). Neste caso pode-se afirmar que $\lambda(\uparrow x) \geq \epsilon\,|x|$.

EXEMPLO 3 : Interpretando 2^{ϖ} 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 $x_n = u \restriction n$ 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

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

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

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

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

Uniões contáveis de eventos singulares $\mathcal{A} = \bigcup_i \Omega_i$, com $\Omega_i = \{A_i\}$ e $A_i \in 2^{\varpi}$, 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.

Aproximaçãoes 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.

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

49 Noção
Fixe-se uma medida de Lebesgue μ e uma classe $\mathcal{A} \subseteq 2^{\varpi}$. $\mathcal{A}$ 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 (16) na página 39) 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, consequentemente, 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.

$$z \in (\downarrow \mathcal{S}_x) \quad \Leftrightarrow \quad (\exists e)\,[\, z \prec \mathcal{W}_e \ \wedge\ \mathcal{W}_e \text{ não finito } \wedge\ \mathcal{W}_e \supseteq D_x\,] \tag{24}$$

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

$$z \in (\downarrow \mathcal{A}) \quad \Leftrightarrow \quad (\exists x)\,[\, x \in A \ \wedge\ z \in (\downarrow \mathcal{S}_x)\,] \tag{25}$$

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

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

$$\mathcal{A}^{(n)} = \bigcup_{|z| \geq n \,\wedge\, z \in (\downarrow \mathcal{A})} \uparrow z \tag{26}$$

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

52 Noção
A **cobertura mínima**[14] *de* $\mathcal{A}$ *é o evento* $\hat{\mathcal{A}} = \bigcap_n \mathcal{A}^{(n)}$. *Fixando uma medida de Lebegue* μ, *o real* $\mu(\hat{\mathcal{A}})$ *representa-se por* $H^\mu(\mathcal{A})$ *e designa-se por* μ-**medida exterior de Hausdorff** *de* $\mathcal{A}$.

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

Note-se que

$$H^\mu(\mathcal{A}) = \mu(\hat{\mathcal{A}}) = \lim_{n\to+\infty} \mu(\mathcal{A}^{(n)}) = \lim_{n\to+\infty} \sum_{|z|\geq n \wedge z \in (\downarrow\mathcal{A})} \mu(\uparrow z) \tag{27}$$

É importante ter em atenção que H^μ 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.

[14] Também designada por **cobertura de Hausdorff**.

EXEMPLO 5 :
Pode-se construir os elementos de $\mathbb{N}$ pela construção de Peano, partindo da constante 0 e da função "sucessor" 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 ε e duas funções inc1: $u \mapsto u\,1$ e inc0: $u \mapsto u\,0$ (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 (23) 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 ω_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 $h : \mathbb{I} \longrightarrow \mathbb{B} \times \mathbb{I}$ definida do seguinte modo

$$h(x) \quad = \quad \begin{cases} (0 \ , \ 2\,x) & \text{se} \quad x < 1/2 \\ (1 \ , \ 2\,x - 1) & \text{se} \quad x \geq 1/2 \end{cases} \tag{28}$$

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

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

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 [15].

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

[15] Aliás esta é a definição formal de real.

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 16, página 27) de um inteiro y como o inteiro $\kappa(y)$ tal que

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

Vimos que κ 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

$$\varphi^O_{\kappa(y)}(0) \simeq y \quad , \quad \varphi^O_e(0) \not\simeq y \quad \text{par todo} \quad e < \kappa(y)$$

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

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

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

O teorema de Kolmogorov (teorema 11, página 43) permite dar uma primeira caracterização dos objectos (inteiros, ou strings finitas, ou conjuntos finitos) κ-compressíveis e, por complemento, os objectos κ-aleatórios. O teorema diz-nos os inteiros κ-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.

54 Facto

Existe um sistema aceitável de índices $\{\varphi_e\}$ *tal que a função universal* φ *que verifica* $\varphi_e(x) \simeq \varphi(e\|x)$ *, para todo* x*, tem um domínio que é livre de prefixos.*

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

$$\varphi(x)\downarrow \wedge \varphi(y)\downarrow \quad \Rightarrow \quad \uparrow x \cap \uparrow y = \emptyset \tag{29}$$

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

O facto de o domínio de φ ser livre de prefixos tem consequências importantes. Nomeadamente tem-se $\sum_{\varphi(z)\downarrow} 2^{-|z|} \leq 1$ (ver facto 36, página 48).

55 Noção (Complexidade de Kolmogorov)
*Seja φ uma função universal livre de prefixos. Define-se φ-**complexidade de Kolmogorov**[16] de um $y \in \varpi$, representado por $K_\varphi(y)$, como* $\min\{\, |x| \mid \varphi(x) \simeq y \,\}$.

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

Pela definição da indexação de funções verifica-se facilmente um resultado que estabelece a relação entre a complexidade de Kolmogorov e o índice de Kolmogorova para indexações livres de prefixos.

56 Facto
A menos de uma constante $O(1)$ independente de y, tem-se $K_\varphi(y) = \min\{\, |e| \mid \varphi_e(0) \simeq y \,\} + O(1)$.

[16] Na generalidade dos textos esta comprexidade é normalmente designada "prefix free Kolmogorov complexity". No entanto, devido 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.

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

Teorema 12
Para toda a constante $c \geq 0$, o conjunto $\{ y \mid K_\varphi(y) \leq |y| - c \}$ é simples.

Esboço de prova
O conjunto $\{ y \mid K_\varphi(y) \leq |y| - c \}$ é um sub-conjunto do conjunto das 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 5, página 24) prova-se

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

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 φ em K_φ.

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,

58 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 π_1. Temos $\varphi(e\|(y\|0)) \simeq \pi_1(y\|0) \simeq y$. Pela propriedade de parametrização dos índices, existe uma função recursiva λ 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

59 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:

60 Noção

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

$$k_d(y) \quad = \quad K(y) - |y| - K(|y|) \tag{30}$$

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

$$K_d(y) \quad = \quad K(y) - |y| \;=\; k_d(y) + K(|y|) \tag{31}$$

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 "ouput" 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.

61 Noção (Incerteza de Hartley e uniforme)
Designamos por **incerteza de Hartley** *(ou, caso não exista ambiguidade, simplesmente* **incerteza**[17]*) de um conjunto finito A, e representamos por $\vartheta(A)$, e definida por $\vartheta(A) = \lceil \log_2 [\![A]\!] \rceil$, sendo $[\![\cdot]\!]$ 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 $\vartheta[A](n) \simeq \vartheta(A \restriction n)$.*

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

Teorema 13 (Contagem de Chaitin)
Para todo n, todo k, e para uma constante $O(1)$ independente de n e k,

$$\max_{|y|=n} k_d(y) = \pm O(1) \tag{32}$$

$$n - \vartheta[\{\, y \mid k_d(y) \leq -k \,\}](n) \geq k - O(1) \tag{33}$$

O teorema da contagem presupõe 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.

[17] A próxima secção, dedicada ao conceito de incerteza, vai introduzir outras formas para exprimir esta noção.

(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 $K(A\restriction n)$ das diferentes truncaturas de A.

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

62 NOÇÃO (ALEATORIEDADE DE KOLMOGOROV-LEVIN-CHAITIN)
Um conjunto A é ***KLC-aleatório*** *se existe um $c \in \varpi$ tal que, para todo n, $K(A\restriction n) > n - c$. O conjunto A é* ***KLC-aleatório relativo a O*** *se, para algum c, se verifica $K(A\restriction n|O) > n - c$ para todo n.*

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

$$U_c \quad = \quad \{\, y \mid K(y) \leq |y| - c \,\} \tag{34}$$

Note-se que, pelo teorema 12, todos estes conjuntos são simples. Então a noção 62 reescreve-se facilmente.

63 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 62 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.

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,

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

A complexidade de Kolmogorov está sempre definida a menos de uma constante; por isso usamos sempre quantidades da forma $K(A \upharpoonright n) \pm O(1)$ para medir a informação necessária para computar o prefixo de tamanho n do conjunto A.

A diferença $K(A \upharpoonright n) - n$ mede o "défice" inicial de informação que é preciso colocar na computação relativa à informação nos bits do prefixo. Esta diferença está limitada superiormente pela complexidade do comprimento n; se

a diferença for sempre positiva isto indica que é sempre preciso por "mais informação" no cálculo do prefixo do que aquela que se consegue retirar do próprio prefixo.

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.

A definição de KLC-aleatoriedade diz que são aleatórios os conjuntos A para os quais, para algum c, o déficite $K_d(A\upharpoonright n) = K(A\upharpoonright n) - n$ é superior a $-c$. Aliás existe um resultado mais forte que caracteriza os conjuntos KLC-aleatórios

Teorema 14 (Miller & Yu)
A é KLC-aleatório se e só se se verifica qualquer uma das seguintes condições:

1. *A função $n \mapsto K_d(A\upharpoonright n) = K(A\upharpoonright n) - n$ é ϖ-limitada; isto é, $\sum_n 2^{-K_d(A\upharpoonright n)} < \infty$.*
2. *Para toda a função total σ, que não é ϖ-limitada, verifica-se*

$$(\exists_\infty n)\,[K_d(A\upharpoonright n) > \sigma(n)]$$

Como complemento a este teorema temos algo que nos ajuda a responder a uma dúvida básica: *será que existem conjuntos aleatórios?*

Teorema 15 (Miller)
Para todo a função ϖ-limitada σ existe um conjunto KC-aleatório A tal que $K_d(A\restriction n) \leq \sigma(n) + O(1)$, para todo n.

□

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

64 Noção (Conjunto de Kraft-Chaitin)
Um ***conjunto de Kraft-Chaitin*** *é um conjunto recursivamente enumerável R em que $\sum_{r\in R} 2^{-\pi_1(r)} \leq 1$.*

Os elementos $r \in R$ são designados por **requests**; seja $\{r_i\}$ uma enumeração recursiva destes elementos. Vendo cada $r_i = d_i \| y_i$ como um par (com $d_i = \pi_1(r_i)$ e $y_i = \pi_2(r_i)$), o elemento d_i é designado por **incerteza** e o elemento y_i por **resposta**; o real $w_i = 2^{-d_i}$ designa-se por **peso** de r_i.

O real recursivo $w(R) = \sum_i w_i = \sum_i 2^{-d_i}$ designa-se por **peso** de R e o inteiro $\eta(R) = \lfloor -\log_2 w(R) \rfloor$ designa-se por **incerteza** de R. Num conjunto Kraft-Chaitin será $0 < w(R) \leq 1$ e $+\infty > \eta(R) \geq 0$.

Como exemplo, o domínio abstract ϖ determina, para cada $k \geq 1$, um conjunto Kraft-Chaitin $\{(n+k)\|n\}_n$ que representaremos por ϖ_k. De facto, temos $d_n = n + k$ e, por isso, $w(\varpi_k) = \sum_n 2^{-n-k} = 2^{1-k} \leq 1$.

Regressando a um conjunto KC genérico, a intuição agora diz-nos que podemos ver cada $w_i = 2^{-d_i}$ como uma **probabilidade** de um acto de selecção do índice i. Nesta perpectiva os d_i são visto como as **incertezas** sobre a ocorrência desse acto.

Por isso $R = \{d_i \| y_i\}$ pode ser visto como uma enumeração de respostas à selecção de um i condicionado por incertezas d_i. É como uma função recursiva $i \mapsto y_i$ mas associando incertezas d_i à selecção do argumento e com a restrição adicional de os pares incerteza+resposta serem recursivamente enumeráveis.

Com esta intuição o seguinte resultado parece (mas não é!) óbvio.

Teorema 16 (Levin, Schnorr, Chaitin)
Seja $R = \{r_i = d_i \| y_i\}$ um conjunto de Kraft-Chaitin; então, para todo i e para uma constante $O(1)$ independente de i, verifica-se $K(y_i) \leq d_i + O(1)$.

Essencialmente o resultado diz-nos que, para cada $r_i = d_i \| y_i$, existe um programa e_i com o comprimento da incerteza d_i, que é capaz de gerar a resposta y_i; ou seja,

$$\varphi(e_i \| 0) \simeq y_i \quad , \quad |e_i| = d_i$$

Vendo os valores 2^{-d_i} como probabilidades de selecção do "request" r_i, pode-se definir uma noção de **expectativa** de uma função f como uma forma de "média" dos valores $f(y_i)$.

65 Noção (Expectativa)

Seja R um conjuto de Kraft-Chaitin e f uma função total. A **expectativa** $\mathbb{E}[f|R]$ *define-se*

$$\mathbb{E}[f|R] \quad = \quad \sum_i 2^{-f(y_i)} \cdot 2^{-d_i} \tag{35}$$

*A função f é ϖ-**limitada** se $\sum_n 2^{-f(n)} < \infty$. Diz-se R-**limitada** se $\sum_i 2^{-f(y_i)} < \infty$.*

Se for a função f for R-**limitada** então a expectativa $\mathbb{E}[f|R]$ também é finita. Interessa-nos porém casos em que a função f pode não ser limitada em R mas, ainda assim, $\mathbb{E}[f|R] < \infty$.

Interessa-nos, em particular, a função $K_d(y) \simeq K(y) - |y|$. Neste caso $\mathbb{E}[K_d|R]$ vai dar um "integral" dos défices de informação nos bits dos diversos y_i. Será, pela definição,

$$\mathbb{E}[K_d|R] \quad = \quad \sum_i 2^{|y_i|-K(y_i)-d_i} \tag{36}$$

66 Noção

Um conjunto de Kraft-Chaitin, R, diz-se **suficientemente aleatório** *se*

$$\mathbb{E}[K_d|R] \; = \; \sum_i 2^{|y_i|-K(y_i)-d_i} \; < \; \infty \tag{37}$$

Note-se que, se os y_i fossem, por exemplo, trucaturas $Y \upharpoonright i$ de um conjunto Y e se esse Y fosse aleatório, então (37) seria válido mesmo com todos os d_i´s iguais a zero. A aleatoriedade estava toda "concentrada" nos y_i´s. O conjunto R que é suficientemente aleatório, distribui a aleatoriedade por duas componentes: as incertezas d_i e as respostas y_i.

Esta intuição conduz-nos também a uma definição de "aproximação" de um conjunto qualquer A.

67 NOÇÃO
Se $R = \{r_n = d_n \| y_n\}$ é um conjunto KC e A é um qualquer conjunto, escreve-se $A \sqsubseteq R$, e diz-se que A é ***previsto*** *por R, quando $(\forall n)(\exists i)\,[A \upharpoonright n \simeq y_i\,]$.*

Resta saber se, dado um conjunto A, existe algum conjunto KC que o preveja. Se A for recursivamente enumerado é simples construir esse conjunto: basta definir $r_n = n \| A \upharpoonright n$.

No caso geral (incluindo quando A é KC-aleatório) o seguinte teorema ajuda a responder a esta questão:

COROLÁRIO 17
Seja $R = \{d_n \| y_n\}$ um conjunto de Kraft-Chaitin suficientemente aleatório tal que $|y_n| = n$; então existe um conjunto KLC-aleatório A tal que $K(A \upharpoonright n) \leq K(y_n) + d_n + O(1)$.

Esboço de prova

Seja $\sigma(n) = K(y_n) + d_n - |y_n| = K_d(y_n) + d_n$. Temos $\sum_n 2^{-\sigma(n)} < \infty$ porque, por hipótese, R é suficientemente aleatório. Usando o teorema 15 temos a inequação pretendida.

Como consequência destes dois teoremas, temos

Teorema 18
Se A não é KLC-aleatório existe um conjunto de Kraft-Chaitin R suficientemente aleatório que prevê A.

O teorema 14 diz-nos que, se A for KLC-aleatório, verifica-se $\sum_n 2^{-K_d(A\restriction n)} < \infty$; esta quantidade pode ser lida como a expectativa de K_d para conjunto "pseudo" KC que tivesse todos os $d_n = 0$. Tal conjunto teria uma incerteza 0.

Quando A não é podem existir vários R que verificam $A \sqsubseteq R$; alguns deles vão ter expectativa $\mathbb{E}[K_d|R]$ finita mas, para outros, essa expectativa será infinita. Será possível usar a incerteza dos vários R´s que têm expectativa finita como medida do grau de aleatoriedade de A?

Tomemos um qualquer R tal que $A \sqsubseteq R$. Note-se que os d_i dão-nos os limites impostos aos programas de uma máquina que gere as vrárias truncaturas $A \restriction n$ de A; é óbvio que, se os d_i forem muito grandes é fácil satisfazer a condição $y_i = A \restriction n$. Neste caso a expectativa de K_d tenderá a ser finita porque os pesos $w_i = 2^{-d_i}$ são pequenos.

A expectativa $\mathbb{E}[K_d|R]$ tende a ser infinita quando os pesos w_i são grandes ou, equivalentemente, quando as incertezas d_i são pequenas.

Portanto uma incerteza $\eta(R)$ baixa faz com que a expectativa $\mathbb{E}[K_d|R]$ tenha tendência a ser infinita; uma incerteza alta faz com que a expectativa possa, eventualmente, ser finita. A incerteza tal que essa expectativa passa de infinita a finita, dá uma medida de quão pouco aleatório é o conjunto A.

Note-se que, quando A é KLC-aleatório, a "expectativa" $\sum_n 2^{-K_d(A\restriction n)}$ (que equivale a ter-se todos os $d_i = 0$) é sempre finita.

Desta forma, faz sentido definir,

68 NOÇÃO

O **grau de previsibilidade** *de um conjunto A é 0 se A for KC-aleatório e, caso não seja, é a menor incerteza dos conjuntos de Kraft-Chaitin sucientemente aleatórios que prevêm A.*

$$\chi(A) \quad = \quad \begin{cases} 0 & \text{se } A \text{ é aleatório} \\ \min\{\, \eta(R) \mid A \sqsubseteq R \wedge R \text{ é suficientemente aleatório} \,\} & \text{em caso contrário} \end{cases} \tag{38}$$

Pode-se comparar graus de compressibilidade de conjuntos através da comparação das complexidades de Kolmogorov dos seus diversos prefixos.

69 NOÇÃO

*Diz-se que o conjunto A é K-***redutível** *ao conjunto B, e escreve-se $A \leqslant_K B$, quando para todo n,*

$$K(A\restriction n) \leq K(B\restriction n) + O(1)$$

Se $A \not\leq_K B$ *e* $B \not\leq_K A$, *os conjunto dizem-se K-***incomparáveis** *e escreve-se* $A|_K B$.

Uma outra questão que vamos apenas aflorar é: *que construções de conjuntos preservam aleatoriedade?*

Usaremos a notação $A \oplus B$ para designar a união disjunta[18] de conjuntos. Pode-se definir $A \oplus B$ como

$$A \oplus B \quad = \quad \{\, 2\,x \mid x \in A \,\} \cup \{\, 2\,x+1 \mid x \in A \,\} \tag{39}$$

Teorema 19 (Miller, van Lanbalgen, Kautz, Kucera)
Se $A \oplus B$ *é KLC-aleatório, então:*

1. *Ambos* A *e* B *são KLC-aleatórios.*
2. A *(ou* B*) é K-incomparável com* $A \oplus B$.
3. A *é KLC-aleatório relativo a* B, *e* B *é KLC-aleatório relativo a* A.

□

Aleatoriedade de Martin-Löf

[18] A notação $x \oplus y$ será usada, ao longo do curso, para designar o `xor` bit a bit de duas strings. Neste capítulo iremos, no entanto, usar o símbolo $\oplus$ com o significado standard em textos de computabilidade.

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; ito 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 computávies" 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.

70 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* $\mathcal{W}_x \in \mathcal{A}_n \Leftrightarrow \varphi_x \in F_n\tilde{}$.

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

71 Facto

A colecção de todos os testes uniformemente recursivos é enumerável.

Esboço de prova

Dado que cada sequência uniforme de funcionais efectivas é completamente determinda por uma função recursiva, a enumeração das funções recusivas 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.

72 Noção (Teste de Martin-Löf)

Fixemos uma medida de Lebesgue μ*. Uma sequência de eventos recursivamente enumeráveis* $\mathcal{U} = \{U_k\}$ *é um* ***μ-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 essencias 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 $2^{-\lambda(\mathcal{A}_k)} < 2^{-k}$.

73 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 $\lambda\left(\hat{\mathcal{A}}_k\right) > k$.

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 $\lambda\left(\hat{\mathcal{A}}_k\right) > k$ então tem de existir algum n tal que $\lambda\left(\mathcal{A}_k^{(n)}\right) > k$.

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

74 NOÇÃO (ALEATORIEDADE DE MARTIN-LÖF)
Fixe-se uma medida de Lebesgue μ. Uma classe $\mathcal{A} \subseteq 2^{\varpi}$ diz-se **μ-Martin-Löf nula (ML μ-nula)** *se existe algum ML μ-teste $\{\mathcal{U}_n\}$ tal que $\mathcal{A} \subseteq \bigcap_n \mathcal{U}_n$. Um conjunto A diz-se* **ML μ-aleatória** *quando a classe $\{A\}$ não for ML μ-nula; isto é, quando A não verifica qualquer ML μ-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.[19].

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

TEOREMA 20 (SCHORR)
Seja μ a medida standard de Lebesgue; um conjunto é KLC-aleatório se e só se for ML μ-aleatório.

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

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 $A \restriction n \in U_k$. Define-se, agora, um conjunto R tal que

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

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

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

Portanto R é um conjunto de Kraft-Chaitin e, pelo teorema 16, existe uma constante c tal que, para todo $k > 0$ e todo $y \in U_{2k}$, $K(y) \leq |y| - k + c$; equivalentemente $K(y) - |y| \leq c - k$.

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

A prova inversa é uma simples consequência do facto 63 cuja prova se repete aqui. Para cada k considere-se a linguagem U_k definda por

$$U_k = \{ y \mid K(y) \leq |y| - k \wedge (\forall z < y)[K(z) > |z| - k] \} \tag{40}$$

Pelo teorema 12 U_k é recursivamente enumerado; claramente U_k é livre de prefixos. Portanto a sequência de linguagens $\{U_k\}_{k\in\varpi}$ 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

75 Noção (Martingales)
Uma função $d\colon \varpi \to \mathbb{R}_+$ *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* $\lim_{n\to\infty} \sup d(A \restriction n) = \infty$; d *sucede numa classe* $\mathcal{A}$ *de conjuntos se sucede em todos conjuntos dessa classe.*

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

Teorema 21 (Ville)
Uma classe $\mathcal{A} \subseteq 2^\varpi$ *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:

76 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 $d(x) = \mathcal{W}_{f(x)}$.*

Existe uma ligação próxima entre martingales e conjuntos de Kraft-Chaitin.

Teorema 22
Para todo conjunto Kraft-Chaitin $R = \{d_i \| y_i\}_i$, a função $d\colon 2^{<\varpi} \to \mathbb{R}_+$ definida a seguir, é um martingale efectivo.

$$d(x) \quad = \quad \sum_{y_i \leq x} 2^{|y_i| - d_i} \; + \; \sum_{x < y_i} 2^{|x| - d_i} \tag{41}$$

Adicionalmente, se $A \sqsubseteq R$, o martingale d sucede em A se e só se R é suficientemente aleatório.

A segunda parte do resultado diz-nos que, para qualquer conjunto A previsto por R, o martingale d sucede em A se e só se R é suficientemente aleatório.

Esboço de prova

Nesta construção os y_i´s são interpretados como bit-strings finitas. Dado que, num conjunto KC, se tem $\sum_i 2^{-d_i} < \infty$, ambas as parcelas são finitas; por isso $d(x)$ é um bem definido real recursivo. Dado que R é recursivo, se d for um martingale é claramente efectivo. Para verificar que é um martingale calcula-se $d(x\,0) + d(x\,1)$ em função de $d(X)$; vê-se todas as combinações de hipóteses de y_i' s que possam ser prefixos de $x\,0$ e $x\,1$ ou que possam conter estas strings como prefixos; conclui-se que d é, de facto, um martingale.

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

Teorema 23 (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

Corolário 24
Um conjunto A será ML-aleatório (e também KLC-aleatório) se e só se não existe nenhum super-martingale efectivo que sucede em A.

1.6 Incerteza

Na noção 61, página 73 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 $\vartheta(A) = \lceil \log_2 [\![A]\!] \rceil$, sendo $[\![\cdot]\!]$ a função recursiva que mede a cardinalidade de um conjunto.

Este tipo de incerteza só é aplicável a conjuntos finitos; a **incerteza uniforme** contorna este problema, apresentando-se como uma funcional $\vartheta[\cdot]$ que a cada conjunto A associa uma função $n \mapsto \vartheta(A \restriction n)$.

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 inceteza 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 código que representa o conjunto; isto é, $|A| = \lfloor \log_2(1 + \sum_{a \in A} 2^a) \rfloor$.

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

Exemplo 8 :
Suponhamos que A é o conjunto formado por todos os inteiros no intervalo $[0..n-1]$. A cardinalidade é $[\![A]\!] = n$. O código que o representa é $\sum_{k=0}^{n-1} 2^k = 2^n - 1$ cujo tamanho é $|A| = \lfloor \log_2(1 + (2^n - 1)) \rfloor = n$. Portanto, neste caso, $[\![A]\!] = |A| = n$ 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 $[\![B]\!] = [\![A]\!]/2$ e que $\vartheta(B) \sim \vartheta(A) - 1$. O código de B é $\sum_{k=0}^{n/2-1} 2^{2\,k} = (2^n - 1)/3$; o tamanho deste código é $n - 2$; logo, neste caso, $|B| = |A| - 2$ é distinto da cardinalidade $[\![B]\!]$.

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

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 fintos 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 $0, 1, \ldots, (n-1)$ com a estrutura de um **anel** com a adição $+$ e multiplicação efectuadas módulo n

Exemplo 9 : $\mathbb{Z}_5$

$+$	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 $a \times b = 1 \pmod{n}$.

77 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 $x = g \cdot g \cdot g \ldots \cdot g$ (n vezes). [20]

[20] Quando o grupo é aditivo é costume representar $g \cdot g \cdot \ldots \cdot g$ (n vezes) por $n\,g$ 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**.

78 Facto

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

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

Exemplo 10 :

$\mathbb{Z}_9^*$ 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

Teorema 25 (fundamental da aritmética)
Cada $m > 1$ admite uma única **factorização**

$$m = p_1^{e1} \times p_2^{e2} \times \cdots \times p_k^{ek}$$

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

Teorema 26 (pequeno de Fermat)
Se p é primo então, para todo $a \geq 0$, verifica-se $a^p = a \pmod p$.

Corolário 27
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 $a^{-k} = a \cdot a^{-k-1} = a^p \cdot a^{-k-1} = a^{p-k-1}$.

Corolário 28
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 α_k preserva a a estrutura do grupo multiplicativo $\mathbb{Z}^*$. Basta provar, portanto, que é um isomorfismo.

Tomando sempre congruências $(\mathrm{mod}\ p)$, 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, $x = y$.

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

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 $\varphi = \phi(m)/2$

$$i = j \pmod \varphi \Longrightarrow x^i = x^j \pmod m$$

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 φ for um múltiplo comum de $(p-1)$ e $(q-1)$.

Nomeadamente quando $\varphi = \phi(m) = (p-1) * (q-1)$ e, ainda, quando $\varphi = \text{lcm}(p-1, q-1) = (p-1) * (q-1)/\gcd(p-1, q-1)$.

Exemplo 11 :

- $\phi(12) = \phi(2^2 \times 3) = 2^1 \times (2-1) \times (3-1) = 4$
- $5^4 = 625 = 52 \times 12 + 1 = 1 \pmod{12}$
- $15 = 3 \times 5 \qquad \phi(15) = 8 \qquad 1025 = 1 \pmod{8}$. Logo $a^{1025} = a \pmod{15}$.
- $\phi(1) + \phi(2) + \phi(3) + \phi(4) + \phi(6) + \phi(12) = 1 + 1 + 2 + 2 + 2 + 4 = 12$

Definição 2
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* $\mathbb{Z}_m^*$

79 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* $b \in \mathbb{Z}_n^*$ *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.

Definição 3
*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 $p_1 p_2 \ldots p_n$ por primos não necessariamente distintos então o **símbolo de Jacobi** $\left(\frac{a}{m}\right)$ define-se como $\Pi_{i=1}^{n} \left(\frac{a}{p_i}\right)$.*

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 ± 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).

80 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*[21]*:*

- *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.*

[21] 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:

81 FACTO

Se for

$$s = s_0 + 2 \cdot s_1 + 2^2 \cdot s_2 + \ldots + 2^{t-2} \cdot s_{t-2} \qquad s_i \in \mathbb{Z}_2$$

então verifica-se

$$\begin{aligned}
a^{(p-1)/4} \cdot (u^{(p-1)/2})^{s_0} &\equiv 1 \pmod{p} \\
a^{(p-1)/8} \cdot (u^{(p-1)/4})^{s_0} \cdot (u^{(p-1)/2})^{s_1} &\equiv 1 \pmod{p} \\
\ldots &\equiv \ldots \\
a^{(p-1)/2^t} \cdot (u^{(p-1)/2^{t-1}})^{s_0} \cdot \ldots \cdot (u^{(p-1)/2})^{s_{t-2}} &\equiv 1 \pmod{p}
\end{aligned}$$

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 mod b ; a = b ; b = r }
```

Existe uma versão extendida que, quando $\gcd(a, b) = 1$, determina o inverso de a módulo b.

Estes algoritmos extendem-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 $i = 1 \ldots k$.

$$x = \sum_{i=1}^{k} a_i \, x_i \, N_i \pmod{N} \qquad \text{com} \quad N_i = N/n_i, \quad x_i = N_i^{-1} \pmod{n_i}$$

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 $b = b_1 + 2b_2 + 2^2 b_3 + \cdots + 2^{n-1} b_n$ 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 }
at\'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

$$a^q = 1 \pmod p \quad \textbf{ou}$$

$$a^{2^i q} = -1 \pmod p \quad \text{para algum} \quad 0 \leq i < s$$

Os conjuntos $W(p)$ definem-se apropriadamente a partir deste critério.

Ver, no Handbook of Applied Criptography, 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 $\upsilon > 0$ tal que $\overbrace{1+1+\cdots+1}^{\upsilon \text{ vezes}} = 0$.

$\mathbb{Z}_p$, para p primo, é um **corpo finito** de característica p; note-se que, neste caso, $\mathbb{Z}_p \setminus \{0\} \equiv \mathbb{Z}_p^*$

82 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**[22] $\mathbf{c} \in \mathbb{Z}_p[X]$ de grau n tal que*

$$\mathbb{F}_m \cong \mathbb{Z}_p[X]/\mathbf{c}\mathbb{Z}_p[X]$$

[22] 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: $\mathbb{Z}_p[X]/\mathbf{c}\mathbb{Z}_p[X]$ *é o anel dos polinómios de coeficientes em* $\mathbb{Z}_p$ *em que a adição e multiplicação de polinómios é efectuada módulo* $\mathbf{c}(X)$.

(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* $\mathbf{c} \in \mathbb{Z}_p[X]$ *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*[23]*; seja* β *uma raíz de* $\mathbf{c}$ *em* K *que não seja raíz de nenhum polinómio* $(X^k - 1)$ *com* $k < m-1$ [24]*.*

Então:

(i) *O menor anel que contém* $\mathbb{Z}_p$ *e o elemento* β *forma um sub-corpo de* K *que representamos por* $\mathbb{Z}_p(\beta)$ *; sendo* $n = \mathsf{grau}(\mathbf{c})$ *, o elemento genérico desse corpo tem a forma*

$$a_0 + a_1\,\beta + a_2\,\beta^2 + \cdots + a_{n-1}\,\beta^{n-1} \quad com \quad a_i \in \mathbb{Z}_p \tag{42}$$

(ii) *A menos de um isomorfismo* $\mathbb{F}_m$ *identifica-se com* $\mathbb{Z}_p(\beta)$ *.*

(f) *Os elementos* β , β^p , β^{p^2} , $\cdots$, $\beta^{p^{n-1}}$ *estão contidos* $\mathbb{Z}_p(\beta)$ *e formam o conjunto das* n *raízes do polinómio* $\mathbf{c}$ *.*

Comentários

[23] Diz-se que K é um corpo que **fractura** o polinómio $\mathbf{c}[X]$ sobre $\mathbb{F}_p$.

[24] 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 $(x\,y)^m = x^m\,y^m$.

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 $\mathbf{c}[X]$ é muito importante porque permite expressar um corpo finito como um corpo de polinómios $\mathbb{F}_p[X]/\mathbf{c}$.

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, $\mathbf{c} \in \mathbb{Z}_p[X]$ tal que para multiplicar $x, y \in \mathbb{F}_m$ basta multiplicar os dois polinómios que os representam e reduzir o resultado módulo $\mathbf{c}$.

(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 $\mathbb{Z}_p[X]/\mathbf{c}\mathbb{Z}_p[X]$ o monómio X é um gerador uma vez que, reduzido módulo $\mathbf{c}$ todas as potências X^k, com $k < m$, têm de ser distintas; de facto, se fosse $X^k = X^i \pmod{}c$, então $X^k - X^i$ seria divisível por $\mathbf{c}[\mathbf{X}]$ 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 β é raiz de um polinómio de grau n com coeficientes em $\mathbb{Z}_p$, é sempre possível escrever $\beta^n = c_0 + c_1\,\beta + \cdots + c_{n-1}\,\beta^{n-1}$, com $c_i \in \mathbb{Z}_p$; em qualquer multiplicação de dois elementos da forma (42), sempre que o ocorra um termo da forma β^k com $k \geq n$, pode-se fazer substituições sucessivas de β^n de acordo com esta igualdade de forma a reduzir o resultado, sempre, a um expressão da forma (42).

Os geradores de $\mathbb{F}_m \sim \mathbb{Z}_p(\beta)$ podem ser determinados a partir de β. De facto o elemento β 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 $\tau : \mathbb{Z}_{m-1} \rightharpoonup \mathbb{Z}_{m-1}$ 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 β; então

$$\beta^i + \beta^j \;=\; (\beta^{i-j}+1)\,\beta^j \;=\; \beta^{\tau(i-j)+j}$$

Isto dá uma forma "imediata" 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 α^p também será raiz do mesmo polinómio.

Por este processo, a partir da raiz β, gera-se $\beta^p, \beta^{p^2}, \cdots, \beta^{p^{n-1}}$ todas raizes de $\mathbf{c}$; uma vez que os diferentes expoentes p^i, com $i \in 0..n-1$ são todos distinctos em $\mathbb{Z}_{p^n-1}$, e β é um gerador do grupo cíclico $\mathbb{Z}_p(\beta)^*$, todas as raízes β^{p^i} serão distintas.

Por isso $\mathbb{Z}_p(\beta)$ contém, não só a raiz β, 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 12 : 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.

$$(X^2 + 1) \qquad (X^2 + X - 1) \qquad (X^2 - X - 1)$$

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[25]; 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 β desse polinómio. Como consequências imediatas, β verifica $\beta^2 = \beta + 1$ e a outra raíz do polinómio é $1 - \beta$.

[25] 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 $(x + y\,X) \in \mathbb{Z}_3[X]/\mathbf{c}, \mathbb{Z}_3[X]$ em que $\mathbf{c}$ 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 β; 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 β^k e verificar

β^0	β^1	β^2	β^3	β^4	β^5	β^6	β^7
1	β	$1+\beta$	$1-\beta$	-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 13 : 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 selecionam-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 $\mathbb{F}_m \longrightarrow \mathbb{F}_m$ que preservam a estrutura do corpo (endomorfismos) e são injectivas.

Os automorfismos de $\mathbb{F}_m$ que fixam[26] os elementos de $\mathbb{F}_p$ formam o **grupo de Galois** $\mathrm{Gal}(\mathbb{F}_m/\mathbb{F}_p)$. A operação de grupo é a composição de morfismos com a identidade 1. Considera-se o grupo multiplicativo: $\sigma \cdot \delta$ é o morfismo $x \mapsto \sigma(\delta(x))$, $\sigma^0 = \mathbf{1}$ e σ^k é $\underbrace{\sigma \cdot \sigma \cdots \sigma}_{k \text{ vezes}}$.

83 FACTO

Seja $p \neq 0$ a característica de $\mathbb{F}_m$. Para todos $x, y \in \mathbb{F}_m$, $(x+y)^p = x^p + y^p$, $(x \cdot y)^p = x^p \cdot x^p$, $x^p = 0$ 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 σ.

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 σ se identifica com um elemento de $\mathbb{F}_p$.

[26] O morfismo σ fixa x quando $\sigma(x) = x$.

2. Usando a noção de característica é muito simples mostrar que σ 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 $\mathbf{p}[X] = p_0 + p_1 X + p_2 X^2 + \cdots + p_d X^d$ então

$$(\mathbf{p}[X])^p = \quad a_0^p + a_1^p X^p + a_2^p (X^p)^2 \cdots + a_d^p (X^p)^d =$$

$$= \quad a_0 + a_1 X^p + a_2 (X^p)^2 \cdots + a_d (X^p)^d = \quad \mathbf{p}[X^p]$$

   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 α é uma raiz do polnómio $\mathbf{p}[X]$ numa qualquer extensão de $\mathbb{F}_p$, então α^p será também uma raiz.

$$\mathbf{p}[\alpha] = 0 \quad \Longrightarrow \quad \mathbf{p}[\alpha^p] = (\mathbf{p}[\alpha])^p = 0$$

5. Para verificar-mos que o morfismo de Frobenius σ 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 (42), para um elemento $x \in \mathbb{F}_m$; pelas observações anteriores

$$x^p \;=\; a_0 + a_1\, \beta^p + a_2\, (\beta^p)^2 + \cdots + a_{n-1}\, (\beta^p)^{n-1}$$

   Sendo β uma raiz do polinómio característico, β^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

84 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 $\mathsf{Gal}(L/K)$ *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 $\mathsf{Gal}(L/K)$ 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 $x \mapsto x^p$).

Definição 4
Sejam L e K corpos finitos como no facto 84. O **traço** *de L em K é a aplicação definida por*

$$\mathsf{tr}_{L/K}(x) \doteq \sum_{k=0}^{s-1} \sigma^k(x)$$

A **norma** *de L em K é a aplicação*

$$\mathsf{nr}_{L/K}(x) \doteq \prod_{k=0}^{s-1} \sigma^k(x)$$

em que $\sigma : x \mapsto x^q$ denota o morfismo de Forbenius de L em K.

85 Facto
Subentendendo-se os corpos L e K. Para todo $x, y \in L$ e todo $a \in K$, verifica-se:

(i) $\mathsf{tr}(x) \in K$ *e* $\mathsf{nr}(x) \in K$
(ii) $\mathsf{tr}(x+y) = \mathsf{tr}(x) + \mathsf{tr}(y)$ *e* $\mathsf{tr}(a\,x) = a\,\mathsf{tr}(x)$,
(iii) $\mathsf{nr}(x \cdot y) = \mathsf{nr}(x) \cdot \mathsf{nr}(y)$ *e* $\mathsf{nr}(a\,x) = a^s \mathsf{nr}(x)$
(iv) $\mathsf{tr}(\cdot)$ *é uma aplicação sobrejectiva de L em K e* $\mathsf{nr}(\cdot)$ *é uma aplicação sobrejectiva de L^* em K^*.*

Esboço de prova

Este resultado pretende afirmar que o traço $\mathrm{tr}(x)$ 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 $t = \mathrm{tr}(x)$; 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 σ) permitem concluir facilmente que $\mathrm{tr}(\cdot)$ 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 $\mathrm{tr}(y) \neq 0$; um tal y tem de existir por que existem, quanto muito, p^{n-1} raizes do polinómio $x + x^p + x^{p^2} + \cdots + x^{p^{n-1}}$ e existem p^n elementos em $\mathbb{F}_m$. Definindo $x \doteq (c/\mathrm{tr}(y))\, y$, temos um elemento com traço c.

2.6 Corpos de Galois

São os corpos finitos de característica 2.

86 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.

$$x = \Sigma_{a \in \alpha} a$$

Neste caso $\mathcal{B}$ diz-se uma **base** de $\mathbb{F}_{2^n}$. As bases mais frequentes são

Base Polinomial Tipo 0	$\mathcal{B} = \{1, \beta, \beta^2, \ldots, \beta^{n-1}\}$
Base Polinomial Tipo 1	$\mathcal{B} = \{\beta, \beta^2, \ldots, \beta^{n-1}, \beta^n\}$
Base Normal	$\mathcal{B} = \{\rho, \rho^2, \rho^4, \ldots, \rho^{2^{n-1}}\}$

com β uma raíz do polinómio característico em $\mathbb{F}_m$ e ρ um elemento com traço 1.

EXEMPLO 14 : O corpo finito $\mathbf{GF}(2^4)$ determinado pelo polinómio característico $\mathbf{c}[X] = (X^4 + X + 1)$ 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 $\mathrm{GF}(2^m)^*$ é muito simples de encontrar: é o polinómio X.

X^i	$X^i \mod \mathbf{c}[X]$	$(\mathbb{Z}_2)^4$	X^i	$X^i \mod \mathbf{c}[X]$	$(\mathbb{Z}_2)^4$
$-$	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: $\mathrm{GF}(2^4)$

Usando essa tabela pode-se ver alguns exemplos de codificação de $\mathbb{Z}_{16}$ em 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 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 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 GF(2^n). Se atender-mos à definição 4 temos, neste caso, característica $p = 2$ e extensão em cause é [GF(2^n)/GF(2)], temos $s = n$ e o

morfismo de Frobenius é $\sigma : x \longrightarrow x^2$. Portanto

$$\mathrm{tr}(x) = \sum_{k=0}^{n-1} x^{2^k} \quad , \quad \mathrm{nr}(x) = \prod_{k=0}^{n-1} x^{2^k}$$

Falando brevemente da noção de norma em $\mathrm{GF}(2^n)$ tem-se

$$\mathrm{nr}(x) = \prod_{k=0}^{n-1} x^{2^k} = x^{(\sum_{k=0}^{n-1} 2^k)} = x^{2^n-1}$$

Portanto $\mathrm{nr}(x) = 1$ se $x \neq 0$ e $\mathrm{nr}(x) = 0$ se $x = 0$. A norma é, em $\mathrm{GF}(2^n)$ 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 β uma raíz do polinómio característico), a forma (42) no facto 82

$$x_0 + x_1 \cdot \beta + \cdots + x_{n-1} \cdot \beta^{n-1} \qquad x_i \in \mathrm{GF}(2) \tag{43}$$

indica que o conjunto

$$\mathcal{B}_{\beta,0} = \{1 \ , \ \beta \ , \ \beta^2 \ , \ \cdots \ , \ \beta^{n-1}\} \tag{44}$$

forma uma base polinomial do tipo 0.

Um vector de bits $x^{\tilde{}} \in \mathsf{GF}(2)^n$ identifica (de forma única) um elemento $x \in \mathsf{GF}(2^n)$ através da representação (43).

> *Genericamente o operador* $(\cdot)^{\tilde{}}$ *associa um emento* $x \in \mathsf{GF}(2^n)$ *à sua representação* $x^{\tilde{}} \in \mathsf{GF}(2)^n$ *numa base* $\mathcal{B}$ *implícita no contexto.*

Um elemento importante é o que representa β^n. Note-se que, se for $\mathbf{c}[X]$ o polinómio característico, tem-se

$$c_0 + c_1 \cdot \beta + \cdots + c_{n-1} \cdot \beta^{n-1} = \beta^n$$

porque β é raíz do polinómio; isto significa que β^n é representado pelo vector $c = (c_0, c_1, \cdots, c_{n-1}) \in \mathsf{GF}(2)^n$ formado pelos coeficientes do polinómio característico para termos de ordem $< n$.

No exemplo 14 note-se como o polinómio X^4 é equivalente a $1 + X$; portanto a representação de β^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 $x, y \in \mathrm{GF}(2^n)$ representados pelos vectores $x\tilde{}, y\tilde{} \in \mathrm{GF}(2)^n$ basta somar as componentes bit a bit: a representação de $x + y$ será $x\tilde{} \oplus y\tilde{}$.

O cálculo da multiplicação é mais complexo e exige a seguinte definição

DEFINIÇÃO 5
A ***matriz companheira*** *do polinómio* $\mathbf{c}[X]$ *é uma matriz* $\mathbf{A} \in \mathrm{GF}(2)^{n\times n}$ *dada por*

$$\begin{cases} \mathbf{A}_{ij} = 1 \quad \textit{sse } i = j+1 & \textit{para } j < n-1 \\ \mathbf{A}_{ij} = c_i & \textit{para } j = n-1 \end{cases}$$

A matriz companheira de $\mathbf{c}[X]$ 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 14 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

87 FACTO

Para quaisquer $x, y \in \mathsf{GF}(2^n)$, *verifica-se* $(\beta \cdot x)^{\tilde{}} = \mathbf{A}\, x^{\tilde{}}$ *e, genericamente,*

$$(y \cdot x)^{\tilde{}} = \bigoplus_{y_i=1} \mathbf{A}^i\, x^{\tilde{}} \tag{45}$$

Para um qualquer vector $u \in \mathsf{GF}(2)^n$, o cálculo de $\mathbf{A}\, u$ é particularmente simples; se representar-mos por $\overrightarrow{u}$ o vector formado pelo desvio ("shift") de um bit de u para a direita, então facilmente se confirma que $\mathbf{A}\, u = \overrightarrow{u} \oplus u_{n-1} \cdot c$.

Um algoritmo eficiente para a multiplicação em (45), será

```
M := 0
para i = 0 ate n-1 fazer
  se bit0(y) entao M := M + x ;  shifleft(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

$$\mathcal{B}_{\beta,1} = \{\beta \ , \ \beta^2 \ , \ \cdots \ , \ \beta^{n-1} \ , \ \beta^n\} \tag{46}$$

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 $\mathbf{c}(\beta) = 0$ e portanto

$$1 = c_1 \cdot \beta + c_2 \cdot \beta^2 + \cdots + c_{n-1} \cdot \beta^{n-1} + c_n \cdot \beta^n$$

tendo em atenção que se tem sempre $c_n = 1$.

Bases Normais

Escolhendo um elemento $\rho \in \mathrm{GF}(2^n)$ tal que $\mathrm{tr}(\rho) = 1$ então

$$\mathcal{N}_\rho = \{\rho \ , \ \rho^2 \ , \ \cdots \ , \ \rho^{2^k} \ , \ \cdots \ , \ \rho^{2^{n-1}}\}$$

forma uma base normal. De facto a soma dos elementos de $\mathcal{N}_\rho$ é igual a 1 porque coincide com o traço de ρ (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 $x\tilde{} = (x_0, \cdots, x_{n-1}) \in \mathrm{GF}(2)^n$ representa um elemento $x \in \mathrm{GF}(2^n)$ através da soma

$$x = x_0 \cdot \rho + x_1 \cdot \rho^2 + x_2 \cdot \rho^4 + \cdots + x_{n-1} \cdot \rho^{2^{n-1}} \tag{47}$$

As igualdades essenciais para os diversos algoritmos são

$$1 = \rho + \rho^2 + \rho^4 + \cdots + \rho^{2^{n-1}} \quad , \quad \rho = \rho^{2^n}$$

A primeira resulta da hipótese $\mathrm{tr}(\rho) = 1$; 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: $(x+y)\tilde{} = x\tilde{} \oplus y\tilde{}$, com $x, y \in \mathrm{GF}(2^n)$.

Outras operações simples são a construção de quadrados e de traços; de facto, usando (47), tem-se

$$x^2 = (\sum_{k=0}^{n-1} x_k \cdot \rho^{2^k})^2 = \sum_{k=0}^{n-1} x_k \cdot \rho^{2^{k+1}} =$$

$$= x_{n-1} \cdot \rho + \sum_{k=1}^{n-1} x_{k-1} \cdot \rho^{2^k}$$

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 de $x^{\sim}$; esta operação será representada por $\overset{\circlearrowleft}{x^{\sim}}$.

$$(x^2)^{\sim} = (\overset{\circlearrowleft}{x^{\sim}})$$

Para o cálculo do traço note-se que, para todo k, $\mathrm{tr}(\rho^{2^k}) = \mathrm{tr}(\rho) = 1$. Portanto

$$\mathrm{tr}(x) = \sum_{k=0}^{n-1} x_i = \mathrm{Tr}(x^{\sim})$$

Nota: Para qualquer $u \in \mathrm{GF}(2)^n$ define-se $\mathrm{Tr}(u) = \sum_{i=0}^{n-1} u_i$

A multiplicação

$$x \cdot y \;=\; \sum_{x_i=1} \sum_{y_j=1} \rho^{2^i} \cdot \rho^{2^j} \tag{48}$$

é bastante mais complicada e, genericamente, mais complexa nas bases normais do que nas bases polinomiais. A dificuldade reside no facto de (48) 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 ρ^{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 óptimas**.

88 FACTO
Se $p = n + 1$ *é primo e* 2 *é gerador do grupo cíclico* $\mathbb{Z}_p^*$ *então existe* ρ *que verifica* $\mathsf{tr}(\rho) = 1$ *e em que o conjunto*

$$\{1 \;,\; \rho \;,\; \rho^2 \;,\; \rho^4 \;,\; \cdots \;,\; \rho^{2^{n-1}}\}$$

determina um sub-grupo cíclico de $\mathsf{GF}^*(2^n)$ *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 $\mathsf{GF}^*(2^n)$, concluímos que $\mathsf{GF}^*(2^n)$ contém um sub-grupo cíclico $G \subseteq \mathsf{GF}^*(2^n)$ de ordem p.

Seja ρ 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 $s = 1, 2, \cdots, p-1$ (mas não o expoente $s=0$ que determina o elemento $\rho^0 = 1$); portanto temos

$$G = \{1\} \cup \{\, \rho^{2^k} \mid k \in \mathbb{Z}_n \,\} = \{1 \ , \ \rho \ , \ \rho^2 \ , \ \cdots \ , \ \rho^{2^{n-1}}\}$$

Resta provar que $\mathrm{tr}(\rho) = 1$; tem-se

$$\mathrm{tr}(\rho) = \sum_{k=0}^{n-1} \rho^{2^k} = \sum_{s=1}^{p-1} \rho^s = (\rho^p + \rho) \cdot (1+\rho)^{-1} = 1$$

porque, dada a ordem de G, $\rho^p = 1$.

89 FACTO

Nas condições do facto 88 tem-se:

(i) Se $2^i + 2^j = 0 \pmod p$ *então*

$$\rho^{2^i} \cdot p^{2^j} = \rho + \rho^2 + \rho^4 + \cdots + \rho^{2^{n-1}}$$

(ii) Se $2^i + 2^j \neq 0 \pmod p$ *então*

$$\rho^{2^i} \cdot p^{2^j} = \rho^{2^{\lambda_{ij}}} \quad com \quad \lambda_{ij} \doteq i + \tau(j-i) \pmod n$$

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 ρ^{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 ρ^{2^k} para algum $k \in \mathbb{Z}_n$.

Se $2^i + 2^j = 0 \pmod p$ então $\rho^{2^i} \cdot p^{2^j} = 1 = \mathrm{tr}(\rho) = \rho + \rho^2 + \cdots + \rho^{2^{n-1}}$.

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 (48): 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 15 :Tomemos de novo 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 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 $x, y \in$ GF(2^n) que têm as representações $x^{\tilde{}} = (1, 0, 1, 0)$ e $y^{\tilde{}} = (0, 1, 1, 0)$. Isto significa que $x = \rho + \rho^4$ e $y = \rho^2 + \rho^4$.

Denotamos por Λ_{ij} a representação vectorial do elemento $\rho^{2^i} \cdot \rho^{2^j}$. Usando (48) vemos que só interessa calcular Λ_{ij} quando se verifica $x_i = 1$ e $y_j = 1$.

$$(x \cdot y)^{\tilde{}} \ = \ \Lambda_{01} \oplus \Lambda_{02} \oplus \Lambda_{21} \oplus \Lambda_{22}$$

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 Λ_{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	$j-i$	λ_{ij}
0	1	1	3
0	2	2	$\perp$
2	1	3	0
2	2	0	3

$$\Lambda_{01} = \Lambda_{22} = (0,0,0,1)$$

$$\Lambda_{02} = (1,1,1,1) \ , \ \Lambda_{21} = (1,0,0,0)$$

portanto $(x \cdot y)\tilde{} = (0,1,1,1)$

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

$$x_1, x_2, \ldots, x_i, \ldots$$

É produzida a partir de outra sequência $s_0, s_1, \ldots, s_k, \ldots$ por:

$$s_k = G(s_{k-1}), \qquad x_k = h(s_k), \quad \text{com} \quad k > 0$$

(sendo h uma função *one-way*) de tal forma que é satisfeita a condição: *Conhecidos os primeiros k valores da sequência $x_1, \ldots, x_k$ 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 **criptográficamente 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 16 :(**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:

$$s_k = s_{k-1}^a \pmod m, \qquad x_k = s_k \pmod 2$$

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 $m = p \cdot q$ e 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

$$s_k = s_{k-1}^2 \pmod m, \qquad x_k = s_k \pmod 2$$

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 κ, 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:

$$s_1 = I \oplus s_0 \qquad x_i = \{s_i\}_\kappa, \qquad s_{i+1} = I \oplus \{I \oplus x_i\}_\kappa \quad i > 0$$

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 $1 < p_1 < \cdots < p_k$ e expoentes $e_1, \ldots, e_k$ tais que $m = p_1^{e_1} \times \ldots \times p_k^{e_k}$.

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" $(2, 3, 5, \ldots)$ e testa-se $m = 0 \pmod{p}$. 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 $\upsilon = (p-q)/2$; donde $m = p \times q = (\mu^2 - \upsilon^2)$ e

$$\mu^2 = m + \upsilon^2 \qquad \text{com} \quad \upsilon^2 \ll m \tag{†}$$

A partir do valor inicial $\mu = \lceil (\sqrt{m}) \rceil$ e incrementando sucessivamente $\mu \leftarrow \mu + 1$, testam-se, para cada μ, valores $\upsilon = 0, 1, \ldots, \lceil (\sqrt{\mu^2 - m}) \rceil$ até que (†) se verifique.

Conhecidos μ e υ determina-se $p = \mu + \upsilon$ e $q = \mu - \upsilon$.

EXEMPLO 17 :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 $\upsilon = 1$ e $\mu = 1062$. Assim $p = 1063$ e $q = 1061$.

Factorização de Pollard

Gerando uma sequência pseudo-aleatória finita $\rho_0, \rho_1, \ldots \rho_n$ 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 ρ_i (com indicação de falha).

Método rho Os ρ_i são determinadospor um gerador quadrático

$$x_i = x_{i-1}^2 + 1 \pmod m \qquad y_i = (y_{i-1}^2 + 1)^2 + 1 \pmod m \qquad (\dagger)$$

$$\rho_i = x_i - y_i \pmod m$$

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 (†) 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 ρ_i são gerados por

$$X_1 = 2, \quad X_i = (X_{i-1})^i \pmod{m} \qquad i = 2, \ldots, B$$

$$\rho_i = X_i - 1$$

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 Computacional Algebraic Number Theory de H.Cohen.

Factorização Quadrática

90 FACTO

Se se verificar

$$x^2 = y^2 \pmod{m} \qquad \& \qquad x \neq \pm y \pmod{m} \tag{\dagger}$$

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* (†).

Seja $\mathcal{B} = \{p_1, \ldots, p_k\}$ uma **base de primos**; um **número-$\mathcal{B}$** é um inteiro cujos factores primos são todos elementos de $\mathcal{B}$. Se $a = p_1^{e1} \times \ldots \times p_k^{e_k}$, 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: $\langle e_1 \pmod 2, \ldots, e_k \pmod 2 \rangle$

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 $a_1, \ldots a_\mu$ tais que $\|a_1\| + \ldots + \|a_p\| = 0$ então o produto $a_1 \times \ldots \times a_\mu$ é um quadrado perfeito cuja raiz quadrada y é facilmente calculada já que as factorizações dos a_i são conhecidas; se for

$x_1 \times \ldots \times x_\mu \neq \pm y \pmod{m}$, como temos

$$(x_1 \times \ldots \times x_\mu)^2 = a_1 \times \ldots \times a_\mu = y^2 \pmod{m}$$

pode-se calcular $\gcd(x_1 \times \ldots \times x_\mu - y, m)$ 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 criptográficamente significante 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 $\log_g : G \longrightarrow \mathbb{Z}_{|G|}$ (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

DEFINIÇÃO 6
Dado um primo p ímpar, um algoritmo resolve $\textsc{LogDisc}(p)$ *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 $i = 0, \ldots, (p-2)$ e ordena-se pela segunda componente. Dado β 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 $\mathbb{Z}_p^*$

$$x_0 = 1, \qquad x_i = \alpha \times x_{i-1} \pmod{p} \quad i = 1, 2, \ldots, (p-2)$$

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 $j = 0, \ldots, n-1$.

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

$$\beta = \alpha^a \pmod{p} \qquad \text{sse} \qquad \beta \times (\alpha_n)^i = \alpha^j \pmod{p}$$

Para $i = 0, 1, \ldots, n-1$ calcula-se $\beta_i \equiv \beta \times (\alpha_n)^i \pmod{p}$ e procura-se, na segunda coluna da tabela, algum elemento igual a β_i ; se existir, com o respectivo índice j e com o índice i de β_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[27] 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

$$a = a_q \pmod{q} \qquad (\dagger)$$

então é possível calcular a usando o teorema chinês dos restos.

1º caso: *q é primo*

[27] Muitas técnicas criptográficas usam primos da forma $p = q\,2^t + 1$, 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 $(p-1) = q\,r$.

Donde $a\,r = r\,a_q \pmod{p-1}$ e , pelo isomorfismo $\mathbb{Z}_{p-1} \simeq \mathbb{Z}_p^*$, temos

$$(\alpha^a)^r = (\alpha^r)^{a_q} \pmod p \qquad \text{ou seja} \qquad \beta^r = (\gamma)^{a_q} \pmod p \qquad (‡)$$

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 γ e ordem q. Este problema é resolvido por um dos algoritmos anteriores.

2º caso: $q \equiv \mu^e$ *com* μ *primo.*

Representa-se a_q na base μ: temos $a_q \equiv x_1 + \mu x_2 + \cdots + \mu^{e-1} x_e$.
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 μ 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 $x_1, x_2, \ldots x_e$.

Complexidade: *proporcional ao maior divisor primo de* $(p-1)$

Método ρ-Pollard

Quando se pretende resolver o problema do logaritmos discreto num sub-grupo de ordem q de $\mathbb{Z}_p^*$ pode-se usar um método análogo ao método da factorização ρ 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 α em três conjuntos disjuntos S_0, S_1 e S_2 de tamanho aproximadamente igual e fácilmente computáveis.[28]
2. Define-se a sequência

$$x_0 = 1 \qquad x_{i+1} = \begin{cases} \beta\, x_i \pmod{p} & \text{se } x_i \in S_0 \\ x_i^2 \pmod{p} & \text{se } x_i \in S_1 \\ \alpha\, x_i \pmod{p} & \text{se } x_i \in S_2 \end{cases}$$

[28] Por exemplo, $S_i \equiv \{\, x \mid x = i \pmod 3 \,\} \quad i = 0, 1, 2$.

3. Tem-se $x_i = \alpha^{a_i}\,\beta^{b_i} = \alpha^{a_i+\mu\,b_i} \pmod p$ em que

$$(a_0, b_0) = (0,0) \qquad (a_{i+1}, b_{i+1}) = \begin{cases} (a_i, b_i+1) \pmod q & \text{se } x_i \in S_0 \\ (2a_i, 2b_i) \pmod q & \text{se } x_i \in S_1 \\ (a_i+1, b_i) \pmod q & \text{se } x_i \in S_2 \end{cases}$$

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 μ.

Complexidade computacional: *proporcional a* $\sqrt{q}$.

3.Funções Booleanas

O projecto e segurança de muitas técnicas criptográficas estão ligadas ao estudo das funções da forma $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$ ou $f : \mathbb{B}^n \rightarrow \mathbb{B}$ que tomam como argumento uma sequência finita de *bits* de comprimento fixo e devolvem uma sequência do mesmo tamanho ou então um simples bit.

Das várias representações possíveis para tais funções escolhemos algumas que são particularmente importantes:

1. Funções booleanas de n variáveis booleanas

$$f : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$$

2. Funções boolenas sobre o corpo de Galois de ordem n

$$f : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2)$$

3.1 Funções de argumento $\mathrm{GF}(2)^n$

Nas funções booleanas $f : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$ os argumentos x são vectores de n bits; as operações básicas sobre os argumentos são:

1. Selecção $\mathrm{sel} : \mathbb{Z}_n \times \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$,
$$\mathrm{sel}(i, x) = x_i$$
i.e. dado $i \in \mathbb{Z}_n$, seleciona a componente de ordem i do vector x.
2. Operações binárias $\oplus$, $* : \mathrm{GF}(2)^n \times \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)^n$
são as duas funções binárias que aplicam *xor* e *and* bit a bit.

$$(x \oplus y)_i \;=\; x_i + y_i \qquad (x * y)_i \;=\; x_i \cdot y_i$$

A noção de *índice* pode ser gereralizada; podemos tomar como índice um qualquer conjunto de inteiros $i \in \mathbb{Z}_n$; por exemplo, se tivermos $n = 8$, um índice será, por exemplo, $\{0, 3, 7\}$. O valor booleano seleccionado por este índice será

$$x_{0,3,7} \;\doteq\; x_0 \cdot x_3 \cdot x_7$$

Genéricamente um **índice** para funções boolenas de n argumentos booleanos é um sub-conjunto $u \subseteq \mathbb{Z}_n$; o conjunto desses índices representa-se por $\mathbb{U}_n$ e identifica-se com $\wp(\mathbb{Z}_n)$.

A função selecção $\text{sel} : \mathbb{U}_n \times \text{GF}(2)^n \longrightarrow \text{GF}(2)$ generaliza-se facilmente

$$\text{sel}(u, x) = \Pi_{i \in u}\, x_i \qquad , \qquad \text{sel}(\emptyset, x) = 1 \tag{49}$$

DEFINIÇÃO 7
Para $u \in \mathbb{U}_n$ *designa-se por* x_u *o monómio* $\text{sel}(u, x)$. *Designa-se por* $[x]_u$ *o polinómio* $\text{sel}(u, x) \cdot \Pi_{i \notin u}(1 + x_i)$.

Por exemplo, se for $n = 4$ e $u = \{0, 2\}$, temos

$$x_u \;=\; x_0 \cdot x_2 \qquad \text{e} \qquad [x]u = x_0 \cdot (1 + x_1) \cdot x_2 \cdot (1 + x_3)$$

Toda a função booleana de domínio $\text{GF}(2)^n$ pode ser escrita como um polinómio a n variáveis $x_0, x_1, \ldots, x_{n-1}$ dado por

$$f(x) \;=\; \sum_{u \in \mathbb{U}_n} a(u)\, x_u \tag{50}$$

para uma função $a : \mathbb{U}_n \longrightarrow \text{GF}(2)$.

Esta é a chamada **forma normal algébrica** de f e é completamente determinada pela função a dita *função de índices* ou *espectro de índices* ou, simplesmente, *espectro* se não existirem ambiguidades [29].

[29] Veremos adiante que é importante definir um outra forma de espectro de funções booleanas: o espectro de *Walsh-Hadamard*.

Uma forma equivalente de representar a mesma função consiste em considerar o conjunto $\mathcal{A} \subseteq \mathbb{U}_n$ de índices u para os quais $a(u) = 1$; essencialmente $\mathcal{A}$ é o conjunto que tem a função de coeficientes como função característica.

Nesse caso (50) escreve-se

$$f(x) = \sum_{u \in \mathcal{A}} x_u \tag{51}$$

Daqui se deduz que o número total de funções booleanas de argumento $\mathrm{GF}(2)^n$ é 2^{2^n}.

O **grau** de f é definido como $(\max_{u \in \mathcal{A}} |u|)$: i.e. a maior cardinalidade de um índice em $\mathcal{A}$.

Se o grau é 0 ou 1 a função diz-se **afim**.

Claramente que o número total de funções afins é 2^{n+1} metade das quais são **lineares**: i.e. funções f tais que $a(\emptyset) = 0$ ou, equivalentemente, $\emptyset \notin \mathcal{A}$.

EXEMPLO 18 :Considere-se as funções booleanas $f : \mathbb{B}^4 \to \mathbb{B}$ com 4 argumentos booleanos. Um exemplo de uma função na forma normal algébrica será

$$f(x) = 1 + x_0 + x_2 + x_1\,x_2 + x_1\,x_3 + x_0\,x_1\,x_3$$

O conjunto dos índices que correspondem a termos não nulos é

$$\mathcal{A} = \{\emptyset\ ,\ \{0\}\ ,\ \{2\}\ ,\ \{1,2\}\ ,\ \{1,3\}\ ,\ \{0,1,3\}\}$$

O maior deles tem 3 elementos; por isso f tem grau 3.

Esta função não é afim. Um exemplo de função afim será

$$g(x) = 1 + x_0 + x_2$$

que não é uma função linear devido à presença da constante 1. Um exemplo de função linear será

$$h(x) = x_0 + x_2$$

Em termos de espectros, o de g é $\{\emptyset\ ,\ \{0\}\ ,\ \{2\}\}$ enquanto que o de h é $\{\{0\}\ ,\ \{2\}\}$.

O **símbolo de Kronecker** de ordem n é a função $\delta : \mathsf{GF}(2)^n \to \mathsf{GF}(2)$ que verifica $\delta(x) = 1$ se e só se $x = (0, 0, \cdots, 0)$.

91 Facto
Para todo $x \in \mathsf{GF}(2)^n$ *verifica-se*

$$\delta(x) = (1 + x_0) \cdot (1 + x_1) \cdot (1 + x_2) \cdots (1 + x_{n-1}) = [x]_\emptyset$$

Para quaisquer $X, Y \subseteq \mathbb{U}_n$ define-se a sua **união disjunta** por

$$X \uplus Y \doteq (X \setminus Y) \cup (Y \setminus X)$$

e a sua **convolução** por

$$X \circledast Y \doteq \biguplus_{x \in X} \{ x \cup y \mid y \in Y \}$$

92 FACTO

Sejam $\mathcal{A}, \mathcal{B} \subseteq \mathbb{U}_n$ *os espectros de funções* f, g *respectivamente; i.e.*

$$f(x) = \sum_{u \in \mathcal{A}} x_u \qquad g(x) = \sum_{v \in \mathcal{B}} x_v$$

(i) Se f *é uma função constante então: se for* $f(x) = 1$, *o seu espectro é* $\{\emptyset\}$ *e, se for* $f(x) = 0$, *o seu espectro é* $\{\ \}$.

(ii) Se $\mathcal{A} = \mathbb{U}_n$ *então* f *coincide com o símbolo de Kronecker* δ. *Se* $\mathcal{A} = \mathbb{U}_n \setminus \emptyset$ *então* $f = 1 + \delta$ *(i.e.* $f(x) = 1$ *se e só se* $x \neq 0$*).*

(iii) $(f+g)$ *tem espectro* $\mathcal{A} \uplus \mathcal{B}$ *e* $(f \cdot g)$ *tem espectro* $\mathcal{A} \pitchfork \mathcal{B}$.

$$(f+g)(x) = \sum_{u \in \mathcal{A} \uplus \mathcal{B}} x_u \qquad (f \cdot g)(x) = \sum_{u \in \mathcal{A} \pitchfork \mathcal{B}} x_u$$

Corolário: $1+f$ *(a negação de* f *) tem espectro* $\mathcal{A} \uplus \emptyset$.

(iv) *Se* $g(x) = f(z * x)$, *para algum* $z \in \mathsf{GF}(2)^n$, *então*

$$\mathcal{B} = \{\, u \in \mathcal{A} \mid z_u = 1 \,\}$$

Comentários: O uso de tratamentos espectrais têm longa tradiação em Engenharia. Essencialmente procura-se uma representação alternativa para funções de tal forma que o estudo dessas funções possa ser feito (de maneira mais simples) na representação espectral.

Tradicionalmente procura-se analisar as relações entre propriedades no domínio das funções e propriedades nos domínios dos espectros e formas simples de determinar umas e outras.

Este resultado e os que se seguem vêm dentro dessa tradição. São apresentados várias formas particulares de construir funções e é analisada a forma equivalente no domínio dos espectros.

(i) As funções constantes são polinómioas de grau zero; o espectro de $f(x) = 1$ deriva directamente da definição de $x_\emptyset$. De forma semelhante temos o polinómio vazio que origina $f(x) = 0$.

(ii) Se tivermos $x = (0, 0, \ldots, 0)$ então temos $x_u = 1$ se e só se $u = \emptyset$. Sendo $f(x) = \sum_{x \in \mathbb{U}_n} x_u$ teremos claramente $f(x) = 1$.

Se for $x \neq (0,0,\ldots,0)$ então seja $\bar{x} \doteq \{\, i \mid x_i = 1 \,\}$; claramente $x_u = 1$ se e só se $u \subseteq \bar{x}$. O número de índices u tais que $x_u = 1$ é, assim, um número par (é dado por $2^{|\bar{x}|}$); consequentemente $f(x) = \sum_{x \in \mathbb{U}_n} x_u = 0$ já que é a a soma de um número par de elementos não nulos.

(iii) Os espectros de $(f+g)$ e $(f \cdot g)$ resultam directamente da expansão destas duas expressões substituindo $f(x)$ e $g(x)$ pelas respectivas somas.

(iv) Sendo $f(x) = \sum_{u \in \mathcal{A}} x_u$ então

$$g(x) = f(z * x) = \sum_{u \in \mathcal{A}} (z * x)u = \sum_{u \in \mathcal{A}} z_u \cdot x_u = \sum_{u \in \mathcal{A} \wedge z_u = 1} x_u$$

O conjunto $f^{-1}(1) = \{\, x \mid f(x) = 1 \,\}$ chama-se **suporte** de f e é representado por $\mathrm{supp}(f)$.
Chama-se **peso** de f (representado por $\mathrm{wt}(f)$) à cardinalidade desse conjunto – $\mathrm{wt}(f) = |f^{-1}(1)|$.
A função é **balanceada** quando, para metade dos seus argumentos, o valor for 1; isto é, $\mathrm{wt}(f) = 2^{n-1}$.

93 FACTO
Para um qualquer $z \in \mathsf{GF}(2)^n$, *seja* $\delta^{(z)}(x) \doteq \delta(x \oplus z)$ *(i.e.,* $\delta^{(z)}(x) = 1$ *se e só se* $x = z$*). Então:*

(i) $\delta^{(z)} \cdot \delta^{(w)} = \delta(z \oplus w) \cdot \delta^{(z)}$

(ii) *Qualquer função booleana* f *pode ser representada por*

$$f = \sum_{z \in \mathsf{supp}(f)} \delta^{(z)} \tag{52}$$

(iii) *O espectro de* $\delta^{(z)}$ *é o conjunto* $\uparrow \bar{z} \doteq \{\, u \mid \bar{z} \subseteq u \,\}$ *sendo* $\bar{z} \doteq \{\, i \mid z_i = 1 \,\}$ *(i.e.,* $\bar{z}$ *é o maior índice* u *tal que* $z_u = 1$ *).*

(iv) *Para todo* $z \in \mathsf{GF}(2)^n$ *tem-se* $[x]_{\bar{z}} = \delta^{(z)}(x)$.

Notas O primeiro resultado traduz apenas propriedades de $\delta^{(z)}$ que resultam directamente da definição.

O segundo resultado é consequência imediata do primeiro e prova-se considerando a expansão de $f(x)$ nos casos em que $x \in \mathsf{supp}(f)$ e $x \notin \mathsf{supp}(f)$. Tem muita importância computacional quando o suporte da função é tem baixa cardinalidade (f tem pouco peso) e pode ser calculado facilmente. Nestas circunstâncias (52) é uma forma conveniente de representar a função.

O resultado (iii) é importante nomeadamente porque sugere um ataque a uma função boolena $\mathcal{Q} : \mathbb{B}^n \to \mathbb{B}$ como a procura de um $k \in \mathbb{B}^n$ tal que $\mathcal{Q}(k) = 1$.

Suponhamos que tínhamos a certeza que existia só um k nestas condições mas que esse valor era, obviamente, desconhecido. Isso é equivalente a dizer que $\mathcal{Q}$ tem a forma $\delta^{(k)}$, para um k desconhecido. Suponhamos também (e isto é uma suposição muito forte) que existia um modo qualquer de determinar o espectro $\mathsf{Sp}(\mathcal{Q})$ de $\mathcal{Q}$.

O resultado anterior diz-nos que o espectro $\mathsf{Sp}(\mathcal{Q})$ é $\uparrow\bar{k}$. Portanto é um conjunto que tem um limite inferior $\bar{k}$ que pode ser calculado como

$$\bar{k} = \bigcap_{u \in \mathsf{Sp}(\mathcal{Q})} u$$

Supondo finalmente que esse limite era computável; então fica determinado $\bar{k}$ e, consequentemente, fica determinado k.

Para o provar considere-se a função f que tem $\uparrow\bar{z}$ por espectro. Notando que $x_u = 1$ se e só se $u \subseteq \bar{x}$, tem-se

$$f(x) = \sum_{u \supseteq \bar{z}} x_u = \sum_{u \supseteq \bar{z} \wedge u \subseteq \bar{x}} 1$$

Se $x = z$ existe apenas um índice u que satisfaz a condição $u \supseteq \bar{z} \wedge u \subseteq \bar{x}$ (nomeadamente $u = \bar{x} = \bar{z}$). Neste caso o somatório tem uma só parcela e o resultado é 1. Se $x \neq z$ então o conjunto de todos os índices u que satisfazem a condição ou é vazio (quando $\bar{x} \subset \bar{z}$) ou tem um número par de elementos; em qualquer dos casos o somatório é zero. Donde $f(x) = 1$ se e só se $x = z$; portanto, $f \equiv \delta^{(z)}$.

TEOREMA 29
O espectro $\mathsf{Sp}(f)$ *de uma função booleana* f *verifica*

$$\mathsf{Sp}(f) = \biguplus_{z \in \mathsf{supp}(f)} \uparrow\bar{z} \tag{53}$$

Seja $f^{(z)}$ *a função definida por* $f^{(z)}(x) = f(z \oplus x)$ *. Então*

$$\mathsf{Sp}(f^{(z)}) \;=\; \biguplus_{y \in \mathsf{supp}(f)} \uparrow(\bar{z} \uplus \bar{y}) \tag{54}$$

A prova do primeiro resultado é consequência imediata de (52) e do facto 92.

Para provar o segundo basta notar que

$$f^{(z)}(x) = \; f(z \oplus x) = \sum_{y \in \mathsf{supp}(f)} \delta^{(y)}(z \oplus x) = \sum_{y \in \mathsf{supp}(f)} \delta^{(y \oplus z)}(x)$$

Se atender-mos que $\overline{(y \oplus z)} = \{\, i \mid y_i + z_i = 1 \,\} = \bar{y} \uplus \bar{z}$ e que o espectro de $\delta^{(y \oplus z)}$ é $\uparrow\overline{y \oplus z} = \uparrow(\bar{y} \uplus \bar{z})$ obtém-se a expressão pretendida para o espectro.

A representação do espectro em termos do suporte da função f não é muito conveniente já que é difícil, quase sempre, calcular esse suporte. Por isso é necessária uma abordagem alternativa ao cálculo de $\mathsf{Sp}(f)$ e, para isso, é necessário introduzir uma representação de espectros inspirada nos *Binary Decision Diagrams* ou BDD's e uma interpretação desses diagrama análoga à usada no método de *Davis-Purtnam*.

Começa-mos por um exemplo.

EXEMPLO 19 : Consider-se a função booleana em $\mathrm{GF}(2)^4$

$$f(x_0, x_1, x_2, x_3) = 1 + x_0 \cdot x_1 + x_0 \cdot x_2 + x_1 \cdot x_2 \cdot x_3$$

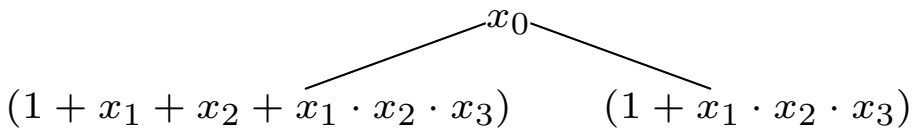
Pode-se sempre escrever

$$f(x_0, x_1, x_2, x_3) \;=\; x_0 \cdot f(1, x_1, x_2, x_3) + (1 + x_0) \cdot f(0, x_1, x_2, x_3)$$

ou seja, com alguma manipulação

$$= x_0 \cdot (1 + x_1 + x_2 + x_1 \cdot x_2 \cdot x_3) + (1 + x_0) \cdot (1 + x_1 \cdot x_2 \cdot x_3)$$

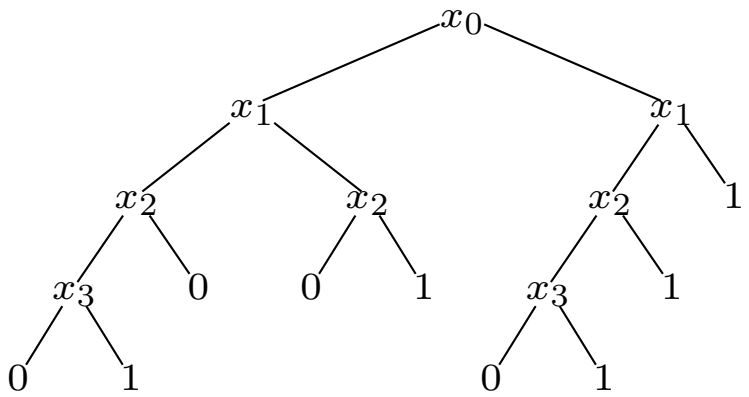
O que sugere uma representação arbórea



Repetindo o processo para as restantes variáveis, vemos que

$$\begin{aligned}(1 + x_1 + x_2 + x_1 \cdot x_2 \cdot x_3) &= x_1 \cdot x_2 \cdot (1 + x_3) + (1 + x_1) \cdot (1 + x_2) \\ (1 + x_1 \cdot x_2 \cdot x_3) &= x_1 \cdot (x_2 \cdot (1 + x_3) + (1 + x_2) \cdot 1) + (1 + x_1) \cdot 1\end{aligned}$$

O que sugere a seguinte árvore



Note-se que os percursos iniciados na raiz determinam valores de x e os respectivos valores $f(x)$ consoante a folha onde os percursos terminam. Os percursos são determinados pelas regras muito simples: $x_i = 1$ significa "*virar à esquerda* no nodo x_i", enquanto $x_i = 0$ significa "*virar à direita*.no nodo x_i".

Nesta árvore, por exemplo, os seguintes percursos terminam no valor 0

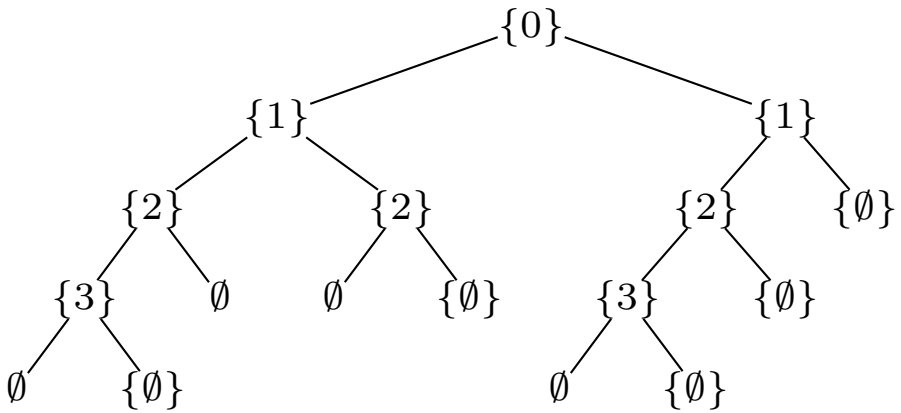
$$\{x_0 = 1\ ,\ x_1 = 0\ ,\ x_2 = 1\ ,\ x_3 = ?\} \quad , \quad \{x_0 = 1\ ,\ x_1 = 1\ ,\ x_2 = 0\ ,\ x_3 = ?\}$$
$$\{x_0 = 1\ ,\ x_1 = 1\ ,\ x_2 = 1\ ,\ x_3 = 1\} \quad , \quad \{x_0 = 0\ ,\ x_1 = 1\ ,\ x_2 = 1\ ,\ x_3 = 1\}$$

Todos os restantes percursos terminam no valor 1. Isto diz-nos que

$$f(1,0,1,0) = f(1,0,1,1) = f(1,1,0,0) =$$
$$= f(1,1,0,1) = f(1,1,1,1) = f(0,1,1,1) = 0$$

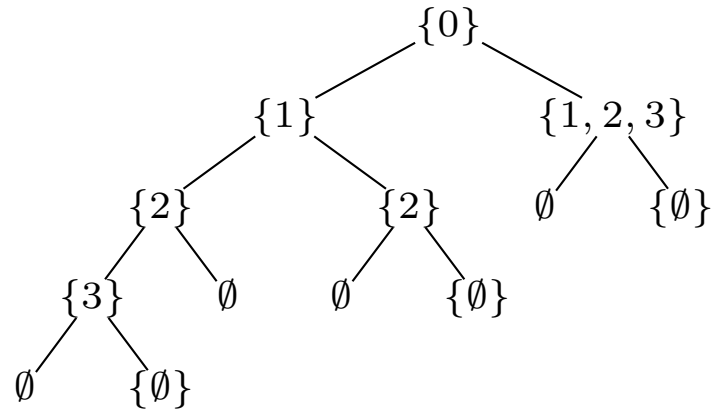
Uma árvore equivalente à anterior pode ser construída colocando nos nodos e nas folhas os espectros das expressões

que ocorrem na árvore anterior.



Admitindo que os nodos podem ter quaisquer índices (e não apenas índices singulares) a árvore pode-se simplificar

para



Esta árvore permite, agora, reconstruir o espectro da função inicial. De facto é fácil de verificar:

(i) Se a árvore for uma folha o espectro é o que a folha indica: $\emptyset$ ou $\{\emptyset\}$.

(ii) Se a árvore for um triplo $\alpha = \langle \sigma \ , \ \alpha^+ \ , \ \alpha^- \rangle$, o seu espectro $\mathcal{A}$ é

$$\mathcal{A} \ = \ \mathcal{A}^- \uplus \{\sigma\} \boxplus (\mathcal{A}^+ \uplus \mathcal{A}^-)$$

sendo $\mathcal{A}^+, \mathcal{A}^-$ os espectros representados pelas sub-árvores α^+ e α^- .

Para sistematizar esta construção seja $\mathbb{U}_{k,n} \doteq \wp(\mathbb{Z}_n \setminus \mathbb{Z}_k)$; isto é, o conjunto de todos os índices formado por elementos $k \leq i < n$.

DEFINIÇÃO 8
[30] *Dado um polinómio reduzido (sem monómios repetidos)* $f = \sum_{u \in \mathcal{A}} x_u$ *com* $\mathcal{A} \subseteq \mathbb{U}_{k,n}$ *o* **fraccionamento** *de* f *em* $\mathbb{U}_{k,n}$ *é:*

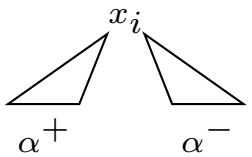
1. *Se* f *é uma função constante, o fraccionamento coincide com a própria função.*
2. *Se* f *não é constante (tem grau maior do que* 0*), sejam:*
   (i) $f^-(x_{k+1}, \cdots, x_{n-1})$ *o polinómio que se obtém eliminando de* f *todos os monómios que contenham* x_k*; seja* α^- *o seu fraccionamento em* $\mathbb{U}_{k+1,n}$ *determinado por este processo.*
   (ii) $f^+(x_{k+1}, \cdots, x_{n-1})$ *o polinómio que se obtém de* f *eliminando* x_k *de todos os seus monómios e reduzindo o resultado final através da eliminação de pares de monómios iguais; seja* α^+ *o seu fraccionamento em* $\mathbb{U}_{k+1,n}$ *.*

   Se $f^- = f^+$ *, o fraccionamento* α *de* f *coincide com* $\alpha^+ = \alpha^-$*; em caso contrário, é o triplo* $\alpha = \langle x_k \ , \ \alpha^+ \ , \ \alpha^- \rangle$ *.*

   A **ordem** *do fraccionamento é* 0 *se* f *for constante ou, em caso contrário, é* $(n - i)$ *sendo* i *o índice da variável que constitui o primeiro elemento deste triplo.*

[30] Baseada na noção de fraccionamento ("*split*")de Davis-Putnam.

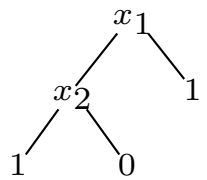
Esta definição sugere imediatamente uma representação arbórea para o fraccionamento de uma função cujo espectro está contido em $\mathbb{U}_{k,n}$.
As funções de grau zero serão as folhas da árvore. As funções não constantes têm fraccionamentos que são triplos $\langle x_i\ ,\ \alpha^+\ ,\ \alpha^-\rangle$ (com $i \geq k$) formados por uma variável e dois outros fraccionamentos.



Se a função não for constante existe pelo menos um monómio com uma ou mais variáveis. Não é necessário que contenha a variável x_k; por exemplo, para $k = 0$ e $n = 3$, considere-se a função $f(x_0, x_1, x_2) = 1 + x_1 + x_1 \cdot x_2$.

$f(x_0, x_1, x_2)$ tem 3 monómios nenhum dos quais contém x_0. O cálculo de f^+ e de f^-, com o fraccionamento feito em $\mathbb{U}_{0,3}$, não modifica f uma vez que não contém x_0; teremos, assim, $f = f^+ = f^-$.

Porém, quando se efectua o fraccionamento em $\mathbb{U}_{1,3}$, tem-se $f^- = 1$ e $f^+ = 1 + 1 + x_2$ que, após redução, se simplifica em x_2.
O fraccionamento final de f está representado ao lado e, como vemos, tem ordem 2.



TEOREMA 30
Nas condições da definição anterior.

1. *Os polinómios f^+ e f^- estão reduzidos.*

2. *A condição* $f^+ = f^-$ *verifica-se se e só se* f *não contém* x_k *em nenhum dos seus monómios. Neste caso tem-se* $f^+ = f^- = f$.
3. *Sendo* $\mathcal{A}^+, \mathcal{A}^- \in \mathbb{U}_{k+1,n}$ *os espectros de* f^+ *e* f^- *então*

$$\mathcal{A}^- = \{\, u \mid u \in \mathcal{A} \;\&\; k \notin u \,\}$$

$$\mathcal{A}^+ = \{\, u \setminus \{k\} \mid u \in \mathcal{A} \,\}$$

$$\mathcal{A} = \mathcal{A}^- \uplus (\mathcal{A}^+ \uplus \mathcal{A}^-) \oplus \{\{k\}\}$$

4. *Verifica-se*

$$f^-(x_{k+1}, \cdots, x_{n-1}) = f(0, x_{k+1}, \cdots, x_{n-1})$$

$$f^+(x_{k+1}, \cdots, x_{n-1}) = f(1, x_{k+1}, \cdots, x_{n-1})$$

$$f = x_k \cdot f^+ + (1 + x_k) \cdot f^-$$

Sejam f, g duas funções booleanas e α, β os respectivos fraccionamentos em algum $\mathbb{U}_{k,n}$. Representemos por $(\alpha + \beta)$, $(\alpha \cdot \beta)$, $\alpha^{(z)}$ os fraccionamentos, respectivamente, das funções $(f + g)$, $(f \cdot g)$, $f^{(z)}$.

94 FACTO
Verifica-se:

1. $(\alpha + 0) = (\alpha \cdot 1) = \alpha$, $(\alpha \cdot 0) = 0$, $0^{(z)} = 0$, $1^{(z)} = 1$, $\alpha \cdot \alpha = \alpha$ *e* $\alpha + \alpha = 0$.
2. *Redução:* $\langle x \ , \ \alpha \ , \ \alpha \rangle$ *simplifica em* α .
3. *Se for* $\alpha = \langle x_k \ , \ \alpha^+ \ , \ \alpha^- \rangle$ *e* β *é um fracionamento de ordem inferior à de* α*, então*

$$(\alpha + \beta) = \langle x_k \ , \ \alpha^+ + \beta \ , \ \alpha^- + \beta \rangle$$
$$(\alpha \cdot \beta) = \langle x_k \ , \ \alpha^+ \cdot \beta \ , \ \alpha^- \cdot \beta \rangle$$

4. *Se* α *e* β *têm a mesma ordem e se for* $\alpha = \langle x_k \ , \ \alpha^+ \ , \ \alpha^- \rangle$ *e* $\beta = \langle x_k \ , \ \beta^+ \ , \ \beta^- \rangle$ *, então*

$$(\alpha + \beta) = \langle x_k \ , \ \alpha^+ + \beta^+ \ , \ \alpha^- + \beta^- \rangle$$
$$(\alpha \cdot \beta) = \langle x_k \ , \ \alpha^+ \cdot \beta^+ \ , \ \alpha^- \cdot \beta^- \rangle$$

5. *Se for* $\alpha = \langle x_k \ , \ \alpha^+ \ , \ \alpha^- \rangle$ *, então*

$$\alpha^{(z)} = \begin{cases} \langle x_k \ , \ (\alpha^+)^{(z)} \ , \ (\alpha^-)^{(z)} \rangle & se \quad z_k = 0 \\ \langle x_k \ , \ (\alpha^-)^{(z)} \ , \ (\alpha^+)^{(z)} \rangle & se \quad z_k = 1 \end{cases}$$

□

Frequentemente as funções boolenas aparecem agrupadas em vectores. Uma função booleana vectorial $\mathbf{S} : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)^n$ pode ser descrita por um vector de n funções booleanas escalares

$$\mathbf{S}(x_1, x_2, \cdots, x_n) = \begin{bmatrix} h_1(x_1, x_2, \cdots, x_n) \\ h_2(x_1, x_2, \cdots, x_n) \\ \cdots \\ h_n(x_1, x_2, \cdots, x_n) \end{bmatrix}$$

Na terminologia criptográfica, uma tal função é designada por uma $n \times n$ **S-Box**.

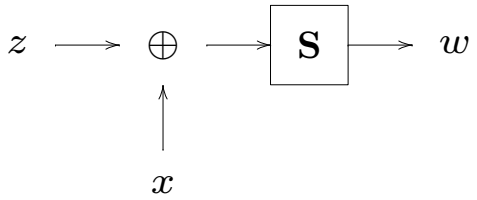
Facilmente se generaliza o conceito para S-Boxes não quadradas: uma $n \times m$ **S-Box** é uma função $\mathbf{S} : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)^m$ definida por um vector de m componentes em que, cada uma, é uma função $h_i : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$, com $i \in 0..m-1$.

EXEMPLO 20 : As três funções boolenas

$$\begin{bmatrix} h_0(x) = & 1 + x_0 \cdot x_1 \\ h_1(x) = & x_0 \cdot x_2 \cdot (1 + x_1) \\ h_2(x) = & 1 + x_0 + x_1 + x_2 \end{bmatrix}$$

definem uma SBox quadrada 3×3.

O exemplo mais corrente desta representação pode ser descrito pela figura seguinte



(55)

Pode-se ver x como uma chave, z como o texto de uma mensagem a cifrar e w como o criptograma resultante.

Problemas directos:

(I) determinar se existe algum valor de x que seja solução da equação

$$\mathbf{S}(z \oplus x) = w \tag{56}$$

(II) Caso exista gerar aleatoriamente **uma** solução

(III) Caso existam, enumerar **todas** as soluções.

O problema de tipo **I** procura saber, apenas, se existe uma chave, o problema de tipo **II** procura descobrir uma chave e o problema de tipo **III** procura enumerar todas as chaves.

EXEMPLO 21 : Recuperemos a **SBox** 3×3 do exemplo 20 e suponhamos o seguinte par *entrada-saída*

$$z = (1, 0, 1) \qquad w = (0, 1, 0)$$

A equação que resulta de (58) (com $w_0 = 0, w_1 = 1, w_2 = 0$) será

$$h_0(z \oplus x) \cdot (1 + h_1(z \oplus x)) \cdot h_2(z \oplus x) = 1$$

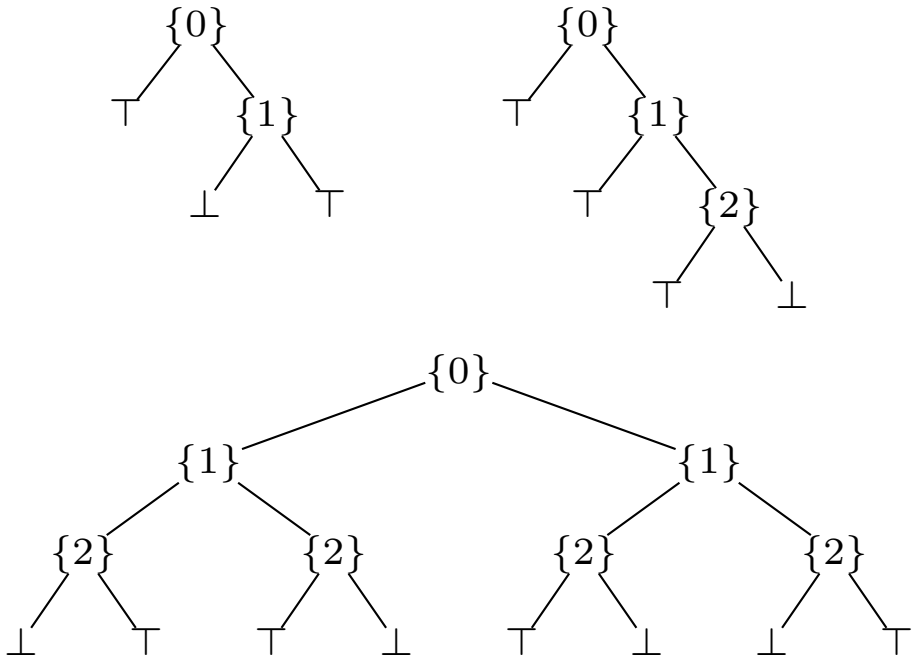
expandindo

$$(1 + (1 + x_0) \cdot x_1) \cdot (1 + (1 + x_0) \cdot (1 + x_1) \cdot (1 + x_2)) \cdot$$
$$\cdot (1 + (1 + x_0) + x_1 + (1 + x_2)) = 1$$

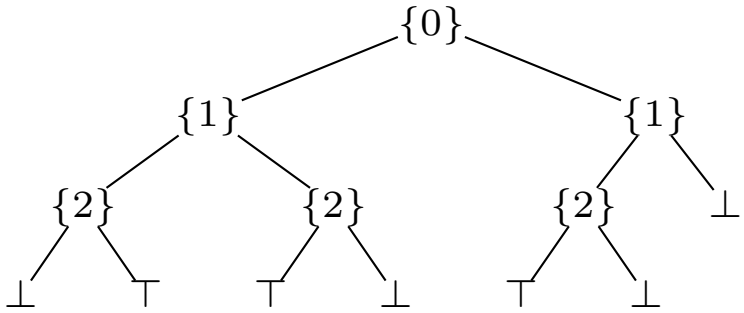
simplificando

$$(1 + x_1 + x_0 \cdot x_1) \cdot (1 + (1 + x_0) \cdot (1 + x_1) \cdot (1 + x_2)) \cdot (1 + x_0 + x_1 + x_2) = 1$$

Os espectros dos três factores nesta equação serão (abrev. $\top \equiv \{\emptyset\}$, $\perp \equiv \emptyset$),



Usando as regras no facto 94 o espectro resultante será



(57)

Esta árvore permite resolver os problemas (I), (II) e (III), neste caso, usando o resultado seguinte

95 Facto

Seja $f : \mathsf{GF}(2) \longrightarrow \mathsf{GF}(2)$ *uma função booleana e* α *o seu fraccionamento em* $\mathbb{U}_n$ *reduzido (não contém componentes da forma* $\langle u\ ,\ \beta\ ,\ \beta\rangle$ *) e construído segundo as regras do facto 94. Então:*

1. $\mathsf{supp}(f) = \emptyset$ *se e só se* $\alpha \equiv \bot$.
2. $x \in \mathsf{supp}(f)$ *se e só determina um caminho válido em* α *de acordo com as seguintes regras:*
   (a) *Numa folha* $\top$, x *determina o caminho válido vazio* ε *; não existe caminho válido numa folha* $\bot$.
   (b) *O caminho válido determinado por* x *em* $\langle u\ ,\ \alpha^+\ ,\ \alpha^-\rangle$ *(caso exista) é a sequência* $u^s\,\omega$, *em que* $s = +$, *se for* $x_u = 1$, *e* $s = -$, *se for* $x_u = 0$, *e* ω *é o caminho válido determinado por* x *em* α^s .

Notas: O que este resultado nos diz é que, basicamente, o fraccionamento α é uma representação do conjunto $\text{supp}(f)$ que é computacionalmente conveniente: as regras fornecem um algoritmo para determinar se o conjunto é ou não vazio e, caso não seja vazio, o valor lógico da relação $x \in \text{supp}(f)$.

Conhecido o fraccionamento α se se procurar soluções para os problemas básicos, teremos:

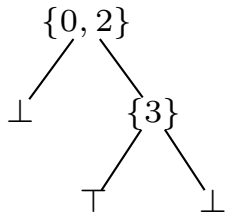
1. *Problema de tipo **I**:* saber se $\text{kwd}(f) = \emptyset$ reduz-se a saber se $\alpha = \bot$.
2. *Problema de tipo **II**:* se $\alpha \neq \bot$ encontrar um elemento arbitrário $x \in \text{kwd}(f)$ resolve-se gerando aleatoriamente um caminho válido em α. Isto significa criar um caminho que se inicie na raiz de α e siga, com igual probabilidade, por qualquer dos seus sub-fraccionamentos distintos de $\bot$.
3. *Problema de tipo **III**:* gerar todo o conjunto $\text{kwd}(f)$ é equivalente a gerar todos os caminhos válidos em α. Computacionalmente, devido ao resultado anterior, não existe distinção entre α e o suporte de f; por isso, gerar $\text{kwd}(f)$ é, essencialmente, construir o fraccionamento α.

É importante notar que diferentes valores de x podem determinar o mesmo caminho ω em α; isto acontece sempre que a árvore não está "cheia", i.e., quando o caminho não contém todos os possíveis índices $i \in 0..n-1$.

Tome-se, como exemplo, o seguinte fraccionamento em $\mathbb{U}_4$ (índices $0..3$) cujo o único caminho válido é

$$\{0,2\}^{-}\,\{3\}^{+}$$

Isto significa que qualquer x que pertença ao suporte da função tem de verificar $x_0 = x_2 = 0$ e $x_3 = 1$.



Note-se que o índice 1 não ocorre no caminho; isto significa que a componente x_1 não está fixa a nenhum dos valores 0 ou 1; representemos este facto pela relação $x_1 = ?$. O "pseudo- valor" $?$ é conhecido por *"do not care"* e, com esta notação, pode-se escrever o elemento x do suporte como $x = (0, ?, 0, 1)$.

Existe um outro "pseudo-valor" importante, representado pelo símbolo !, que indica que uma variável booleana está, simultaneamente, obrigada a ter um valor 0 e obrigada a ter um valor 1. Exemplo, $x = (!, 1, !, 1)$ indicaria que todas as componentes são fixas em 1 e, adicionamento, a primeira e terceira estão fixas em 0.

Numa máquina determinística típica isto é impossível e isso conduziria, naturalmente, a um resultado não-válido para uma computação que produzisse este pseudo-valor como resultado parcial. No entanto é possível ter modelos da computação onde um resultado ! seja perfeitamente válido; por exemplo, em modelos de computação quântica ou, genericamente, computação não-determinística.

Escrita de outro modo, a equação (56) é $\delta(w \oplus \mathbf{S}(z \oplus x)) = 1$, ou, equivalentemente,

$$\delta^{(w)}(\mathbf{S}^{(z)}(x)) = 1 \qquad \text{ou} \qquad [h^{(z)}]_{\bar{w}} = 1 \tag{58}$$

Podemos definir uma função boolena

$$\mathcal{Q}_{z,w}(x) \doteq \delta^{(w)}(\mathbf{S}^{(z)}(x)) \tag{59}$$

e a solução de um problema directo (procurar a chave x) como um ataque a $\mathcal{Q}_{z,w}$.

Uma computação tratável φ que produza uma solução para um problema directo do tipo **II** designa-se por **ataque directo** ao triplo $\langle \mathbf{S}, z, w \rangle$. A entropia de $\mathcal{Q}_{z,w}$ extendida aos ataques directos

$$\mathcal{H}(\mathcal{Q}_{z,w}) = \min_{\varphi} \mathcal{H}(\varphi) - \log_2\{\mathcal{Q}_{z,w}\}_{\varphi}$$

é a **entropia directa** de $\langle \mathbf{S}, z, w \rangle$.

Nota Recordemos que $\{\mathcal{Q}_{z,w}\}_{\varphi}$ representa a probabilidade da computação φ produzir um resultado x tal que $\mathcal{Q}_{z,w}(x) = \delta(w \oplus \mathbf{S}(z \oplus x)) = 1$.

Tendo agora em atenção o facto (95) vemos que a complexidade computacional para encontrar essa solução x nas sua duas componentes (construir o fraccionamento α e um caminho qualquer nesse fraccionamento) tem uma complexidade computacional que é determinada, essencialmente, pela primeira componente.

A construção do fraccionamento, na pior das circunstâncias, pode ser exponencial com o número de *bits* e, por isso, dificilmente será considerada "tratável". Se, para um determinado triplo $\langle \mathbf{S}, z, w \rangle$ o cálculo do fraccionamento for tratável, então a probabilidade $\{\mathcal{Q}\}_{\varphi}$ é igual a 1 e a entropia reduz-se ao cálculo da menor entropia da computação φ que produz esse fraccionamento.

Deve-se ter em atenção que um ataque directo ao triplo $\langle \mathbf{S}, z, w \rangle$, com um par (z, w) particular, fornece muito pouca informação sobre $\mathbf{S}$; nomeadamente sobre a chave x se ela estiver a ser usada com outros pares *entrada-saída*. Por isso faz sentido definir uma forma mais realista de ataque.

Ataque 3.1 .1 Seja μ um sub-conjunto de cardinalidade N do conjunto

$$\Omega \doteq \{ (z, w) \mid \mathbf{S}(z \oplus k) = w \} \tag{60}$$

Um **ataque directo** a $\mathbf{S}$ de dimensão N é uma computação tratável que, conhecidos $\mathbf{S}$ e μ, determina k.

Notas O ataque ao triplo $\langle \mathbf{S}, z, w \rangle$ fornece aleatoriamente uma solução x possível para k; se estiver em causa apenas um par (z, w) qualquer solução x serve. É uma ataque directo a $\mathbf{S}$ de dimensão 1.

Porém se este par for apenas um dos elementos de μ a solução x encontrada é apenas um dos valores possíveis para k. Sabe-se que k, solução do ataque directo, é um dos valores possíveis do suporte de $\mathcal{Q}_{z,w}$; mas não sabemos qual é. O valor x relaciona-se com k apenas pelo facto de serem ambos elementos desse suporte.

Pode-se construir uma função boolena, análoga a (59), que representa o facto de, para <u>todos</u> os pares $(z, w) \in \mu$, se verificar $\mathcal{Q}_{z,w}(x) = 1$

$$\begin{aligned} \mathcal{Q}_{\mu}(x) &\doteq \Pi_{(z,w)\in\mu}\, \mathcal{Q}_{z,w}(x) = \\ &= \Pi_{(z,w)\in\mu}\, \delta(w \oplus \mathbf{S}(z \oplus x)) \end{aligned} \tag{61}$$

Uma computação φ que encontre um x tal que $\mathcal{Q}_{\mu}(x) = 1$ não "acerta" necessariamente em k. A probabilidade de se ter $x = k$ é determinada pelo tamanho do suporte da função $\mathcal{Q}_{\mu}$; isto é, pelo *peso* dessa função.

Portanto a probabilidade de uma computação determinística ϕ, que tome o resultado de φ e o considere um resultado para k, é dado pela razão entre o número de possíveis k se tivéssemos toda a informação possível (isto é o peso da função $\mathcal{Q}_\Omega$) e o número de hipótese de resultados de φ dados pelo peso de $\mathcal{Q}_\mu$.

$$-\log_2\{\mathcal{Q}_\Omega : \mathcal{Q}_\mu\}_\phi \;=\; -\log_2 \mathrm{wt}(\mathcal{Q}_\Omega) + \log_2 \mathrm{wt}(\mathcal{Q}_\mu)$$

A entropia de ϕ é 0 porque é determinística; por isso, usando pode-se escrever a relação que dá a entropia do ataque directo à SBox $\mathbf{S}$

$$\mathcal{H}(\mathcal{Q}_\Omega) \;=\; \min_\mu \left(\mathcal{H}(\mathcal{Q}_\mu) + \log_2 \mathrm{wt}(\mathcal{Q}_\mu) - \log_2 \mathrm{wt}(\mathcal{Q}_\Omega)\right) \tag{62}$$

No caso particular em que só existe um k possível e a solução x para $\mathcal{Q}_\mu)(x) = 1$ pode ser encontrada deterministicamente por uma computação tratável (i.e. $\mathcal{H}(\mathcal{Q}_\mu) = 0$) ainda resta a componente $\log_2 \mathrm{wt}(\mathcal{Q}_\mu)$.

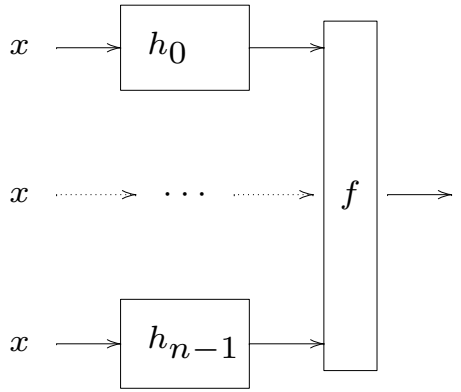
3.2 Composição de funções booleanas

Considere-se uma uma S-Box $k \times n$, $H : \mathrm{GF}(2)^k \longrightarrow \mathrm{GF}(2)^n$. Pode-se escrever $H = (h_0, \ldots, h_{n-1})$ em que os vários h_i são funções boolenas de k argumentos.

$$h_i : \mathrm{GF}(2)^k \longrightarrow \mathrm{GF}(2)$$

Seja $f : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$ uma outra função boolena. É possível construir uma computação em que os resultados das n funções h_i são usados como argumentos de f; i.e, uma computação da forma

$$f(h_0(x), \ldots, h_{n-1}(x))$$



Fica definida uma nova função boolena (com k argumentos) a que chamamos **composição** de f e H e que representamos por

$$f \circ H \quad \text{ou} \quad H; f$$

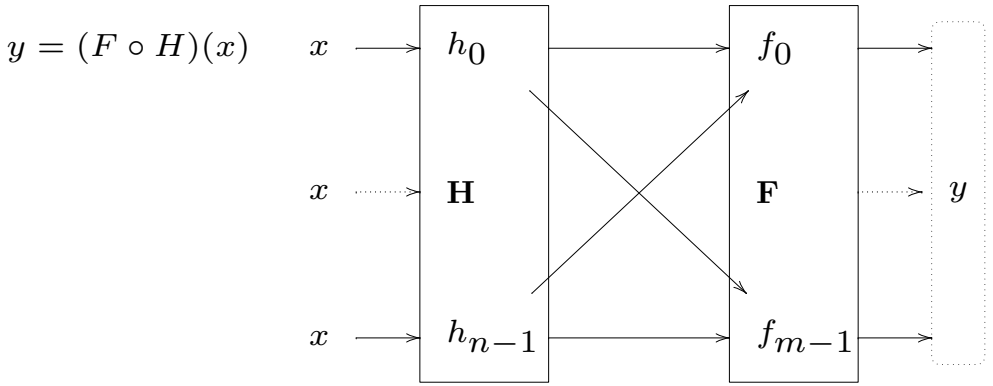
□

Considerando agora, não só uma função f, mas m funções deste tipo (formando uma S-Box de dimensão $n \times m$)

$$F = (f_0 \ , \ f_1 \ , \ \cdots \ , \ f_{m-1}) \qquad \text{com} \quad f_i : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$$

então, através da composição de cada um dos f_i com H, constrói-se o vector de funções boolenas que define a **composição** das S-Boxes F e H.

$$F \circ H \doteq (f_0 \circ H \ , \ f_1 \circ H \ , \ \cdots \ , \ f_{m-1} \circ H)$$



Para exprimir o espectro destas composições é necessário aplicação a noção se selecção apresentada na definição 7 (página 158) a S-Boxes.

DEFINIÇÃO 9

Seja H uma S-Box $n \times m$; para $u \in \mathbb{U}_m$, as funções booleanas H_u e $[H]_u$ definem-se por

$$H_u = \Pi_{i \in u} \, h_i \qquad [H]_u = H_u \cdot \Pi_{i \notin u} \, (1 + h_i)$$

Deste modo, se f tem espectro $\mathcal{A}$, será $f(y) = \sum_{u \in \mathcal{A}} y_u$. Se for $y = H(x)$ teremos

$$(f \circ H)(x) \;=\; f(y) \;=\; \sum_{u \in \mathcal{A}} H_u(x)$$

Representemos por $\mathcal{B}_i$ o espectro da função boolena h_i. Então o espectro de $H_u = \Pi_{i \in u}\, h_i$ será $\left(\bigoplus_{i \in u} \mathcal{B}_i\right)$; usando as regras atrás definidas para produtos de espectros, este cálculo é polinomial em n.

O espectro de $(f \circ H)$ pode, agora, ser calculado como

$$\biguplus_{u \in \mathcal{A}} \left(\bigoplus_{i \in u} \mathcal{B}_i \right) \tag{63}$$

Funções Lineares

Infelizmente, no caso geral, a fórmula (63) tem pouco interesse computacional porque obriga a percorrer todo u no espectro de f que, em princípio, cresce exponencialmente com n.

No entanto, para algumas formas particulares de f, o espectro $\mathcal{A}$ assume formas que tornam simples o cálculo (63). Nomeadamente quando f é uma **função linear**.

Qualquer função linear $f : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$ verifica

$$f(x \oplus y) = f(x) + f(y) \quad \text{para todo} \quad x, y \in \mathrm{GF}(2)^n$$

e o seu espectro é formado por conjuntos com um único índice.

Neste caso (63) resume-se a $\biguplus_{\{i\} \in \mathcal{A}} \mathcal{B}_i$ que pode ser calculado com complexidade polinomial.

Uma função linear importante é a **função traço** definida por

$$\mathrm{Tr}(x_0, x_1, \cdots, x_{n-1}) \doteq x_0 + x_1 + \cdots + x_{n-1} \tag{64}$$

cujo espectro é o conjunto $\{ \{i\} \mid i \in 0..n-1 \}$ formado por todos os índices singulares.

Esta função determina a **paridade** do vector $x = (x_0, \cdots, x_{n-1})$; i.e. a paridade do número de componmentes iguais a 1 no vector x.

Esta noção de paridade pode ser generalizada. Tomemos uma função linear f qualquer e o vector constante c^f que tem componentes a 1 exactamente para os índices onde f tem monómios; isto é,

$$(c^f)_u = 1 \qquad \text{se e só se} \quad u \in \mathrm{Sp}(f) \tag{65}$$

Designamos c^f por **máscara** ou **paridade** de f . Nessas circunstâncias

96 Facto

Se f *é linear e tem máscara* c^f *, então* $f(x) = \mathsf{Tr}(x * c^f)$ *para todo* $x \in \mathsf{GF}(2)^n$.

Exemplo: se for $f(x) = x_0 + x_2 + x_7$, com $x \in \mathsf{GF}(2)^8$, então a máscara é

$$c^f = (1, 0, 1, 0, 0, 0, 0, 1)$$

Note-se que $x * c^f$ é o vector $(x_0, 0, x_2, 0, 0, 0, 0, x_7)$; o seu traço constrói precisanente o valor de $f(x)$.

3.3 Diferenças e Linearidade

Considere-se de novo uma S-Box $\mathbf{S}$ e o problema definido em (55) e (56) de determinar a chave x tal que $\mathbf{S}(z \oplus x) = w$.

Vamos supor que $\mathbf{S}$ era tal que existia uma solução, pelo menos, para o seguinte problema:

Diferenças críticas: encontrar $a, b \in \mathrm{GF}(2)^n$ tais que,

$$\mathbf{S}(a \oplus y) \oplus \mathbf{S}(y) = b \qquad \text{para todo} \quad y \in \mathrm{GF}(2)^n \tag{66}$$

Os elementos (a, b) chamam-se **diferenças críticas**.

Se for possível encontrar um ou mais pares de difenças críticas (a, b) então qualquer cifra assente na complexidade computacional de inverter **S** perde entropia.

Por exemplo, o ataque directo a $\mathbf{S}$ tem a seguinte variante:

Ataque 3.3 .2 Criptoanálise Diferencial do Ataque Directo

1. Recolher pares (z_j, w_j) com $w_j = \mathbf{S}(z_j \oplus k)$; a chave k é a mesma em todos os pares e é a incógnita deste ataque.
2. Recolher pares de diferenças críticas (a_i, b_i) .
3. Encontrar x' tal que, para algum i e todos os j , se verifique $\mathbf{S}(z_j + x) = w_j \oplus b_i$.
4. Nestas circunstâncias tem-se $x \doteq a_i \oplus x'$ é uma solução eventual para k .

A justificação reside no facto de, sendo (a_i, b_i) uma par de diferenças críticas, verifica-se $\mathbf{S}(a_i \oplus (z_j \oplus x')) \oplus \mathbf{S}(z_j \oplus x') = b_i$.

Por construção tem-se, para todo j , $\mathbf{S}(z_j \oplus x') = w_j \oplus b_i$; escolhendo $x = x' \oplus a_i$ verifica-se, para todo j, $\mathbf{S}(z_j \oplus x) \oplus (w_j \oplus b_i) = b_i$; donde $\mathbf{S}(z_j \oplus x) = w_j$ para todo j ; isto significa que x é, eventualmente, k .

Note-se que:

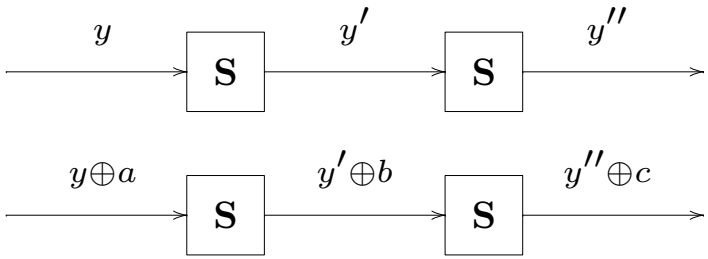
1. O passo (3) é, essencialmente, uma solução do problema inicial mas para valores diferente de *outupts*; é uma versão do problema inicial para pares $(z_j, w_j \oplus b_i)$.
   Aparentemente não se ganha nada com esta formalização dado que, como parte do problema inicial, temos de resolver um problema do mesmo tipo.
2. O ganho reside apenas no facto de ser possível adaptar o problema a uma escolha criteriosa de valores b_i caso seja possível calcular o respectivo a_i .

É possível resolver um ataque directo para diferentes conjuntos de pares $\mu_i = \{ (z_j, w_j \oplus b_i) \mid j \in 1.. \}$ e seleccionar a solução com menor entropia.

No entanto os ataques mais importantes resultam da aplicação das diferenças críticas à composição de S-Boxes.

Suponhamos um caso simples com a composição de $\mathbf{S}$ consigo próprio e vamos supor que (a, b) e (b, c) são dois pares de diferenças críticas.

Consider-se a seguinte situação em que a S-Box resultante é aplicada a duas *entradas* diferentes: y e $y \oplus a$.



(67)

Pelo propriedade das diferenças críticas, conhecidos os vários valores y, y' e y'' no primeiro caso, é muito simples determinar os valores respectivos quando a *entrada* muda de y para $y \oplus a$:

(i) O resultado intermédio muda de y' para $y' \oplus b$ porque o primeiro par de diferenças críticas me diz que $\mathbf{S}(y \oplus a) \oplus \mathbf{S}(y) = b$; logo, $\mathbf{S}(y \oplus a) = \mathbf{S}(y) \oplus b = y' \oplus b$.

(ii) O resultado final muda de y'' para $y'' \oplus c$ porque o par de diferenças críticas (b, c) diz-nos que $\mathbf{S}(y' \oplus b) \oplus \mathbf{S}(y') = c$.
A entrada do 2º $\mathbf{S}$ mudou de y' para $y' \oplus b$; a diferença crítica c repflecte-se, agora, na respectiva saída.

Por este facto se diz que

o par de diferenças críticas (a, b) **propaga** *a diferença* a, *na entrada, para a diferença* b *na saída,*

A propagação de diferenças diminui consideravelmente a entropia de uma concatenação de várias S-Boxes. O aumento de entropia que devia resultar dessa concatenção acaba por se não se manifestar dado que muita da complexidade se perde com a existência de diferenças críticas.

A propósito, o exemplo anterior demonstra o seguinte facto

97 FACTO
Se (a, b) *é um par de diferenças críticas para* $\mathbf{S}$ *e* (b, c) *é um par de diferenças críticas para* $\mathbf{R}$ *então* (a, c) *é um par de diferenças críticas para* $\mathbf{R} \circ \mathbf{S}$.

□

É possível exprimir diferenças críticas através de uma função booleana. Representemos por $\Delta^{\mathbf{S}}_{a,b}$ a função que, para todo o argumento y, verifica

$$\Delta^{\mathbf{S}}_{a,b}(y) \;=\; \delta(\mathbf{S}(a \oplus y) \oplus \mathbf{S}(y) \oplus b) \tag{68}$$

e representemos por

$$\rho_{\mathbf{S}}(a,b) \;\doteq\; 2^{-n} \cdot \mathsf{wt}(\Delta^{\mathbf{S}}_{a,b})$$

a razão entre número de argumentos y para os quais $\Delta^{\mathbf{S}}_{a,b}(y) = 1$ e o número total de argumentos possíveis.

Note-se que (a,b) é um par de diferenças críticas se e só se $\Delta^{\mathbf{S}}_{a,b}$ é a função constante 1. Ou seja, quando $\rho_{\mathbf{S}}(a,b) = 1$.

Porém pode acontecer que (a,b) não seja um par de diferenças críticas mas, ainda assim, exista a possibilidade de, para a maioria dos possíveis argumentos y, se verificar $\Delta^{\mathbf{S}}_{a,b}(y) = 1$; isto significa que será $\rho_{\mathbf{S}}(a,b) < 1$ mas, ainda assim, próximo do limite 1 .

Recordemos a noção de entropia; a notação $\{\Delta^{\mathbf{S}}_{a,b}\}_{\varphi}$ representa a probabilidade de a computação φ gerar um resultado y que verifique $\Delta^{\mathbf{S}}_{a,b}(y) = 1$.

Se (a,b) fosse um para de diferenças críticas esta probabilidade seria sempre 1 qualquer que seja a computação φ que produza resultados em $\mathsf{GF}(2)^n$.

É possível exprimir a entropia de $\Delta^{\mathbf{S}}_{a,b}$ relativo a φ.

$$\mathcal{H}(\Delta^{\mathbf{S}}_{a,b})_{\varphi} \doteq \mathcal{H}(\varphi) - \log_2\{\Delta^{\mathbf{S}}_{a,b}\}_{\varphi}$$

Se φ for determinística será $\mathcal{H}(\varphi) = 0$. Se se comportar como um gerador aleatório, a probabilidade de "acertar" num elemento so suporte de $\Delta^{\mathbf{S}}_{a,b}$ é $\rho_{\mathbf{S}}(a,b)$. Nestas circunstâncias particulares (para um tal φ) será

$$\mathcal{H}(\Delta^{\mathbf{S}}_{a,b})_{\varphi} = -\log_2 \rho_{\mathbf{S}}(a,b) \qquad \text{ou} \tag{69}$$

$$\rho_{\mathbf{S}}(a,b) = 2^{-\mathcal{H}(\Delta^{\mathbf{S}}_{a,b})_{\varphi}}$$

Estas relações definem uma aproximação para a perda de entropia introduzida por um candidato a par de diferenças críticas quando não existe informação sobre a função de diferenças $\Delta^{\mathbf{S}}_{a,b}$ para além do seu suporte.

Facilmente se generaliza o facto 97 para "candidatos" a pares de diferenças críticas

98 FACTO

$$\rho_{\mathbf{R}\circ\mathbf{S}}(a,c) \geq \rho_{\mathbf{S}}(a,b) \cdot \rho_{\mathbf{R}}(b,c)$$

□

A **distância de Hamming** entre duas funções $f, g : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ (representa-se por $d(f,g)$) é o peso de $f+g$; isto é, a $d(f,g)$ conta o número de argumentos nos quais as funções divergem. A **não-linearidade** de f é a menor das distâncias $d(f,g)$ quando g percorre todas as funções booleanas afins em $\mathrm{GF}(2)^n$.

A **correlação** entre f e g, é a função racional

$$\mathcal{C}(f,g) \doteq 1 - 2^{-(n-1)} \cdot d(f,g) \tag{70}$$

Nota:
A correlação tem um resultado compreendido entre -1 e 1; caso as funções sejam iguais temos $d(f,g)=0$ e $\mathcal{C}(f,g)=1$; se as funções forem totalmente distintas (i.e. $f=1+g$) então $d(f,g)=2^n$ e $\mathcal{C}(f,g)=-1$; se o número de argumentos x para os quais $f(x) \neq g(x)$ for metade do total, então $d(f,g)=2^{n-1}$ e $\mathcal{C}(f,g)=0$.

Convenção:
Vimos que cada função linear $\omega : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ é únicamente determinada por um elemento $c^\omega \in \mathrm{GF}(2)^n$ designado por máscara ou paridade ω. Se não surgirirem ambiguidades designaremos a função e a respectiva paridade pelo mesmo símbolo. Assim escrever $\omega \in \mathrm{GF}(2)^n$ e $\omega(x)$ são notações compatíveis: a primeira representa o vector paridade e a segunda faz referência à função linear que esse vector determina.

Alguns resultados intermédios

99 FACTO
Sejam ω, υ funções lineares e f uma outra função boolena qualquer. Então

(i) $\mathcal{C}(\omega, \upsilon) = \delta(\omega \oplus \upsilon)$

(ii) $\mathcal{C}(1 + f, \omega) = -\mathcal{C}(f, \omega)$

(iii) *Se* $g(x) = f(x) + \upsilon(x)$ *então* $\mathcal{C}(g, \omega) = \mathcal{C}(f, \omega + \upsilon)$

DEFINIÇÃO 10
Dada uma função boolena $f : \mathsf{GF}(2)^n \longrightarrow \mathsf{GF}(2)$ *e uma função linear* ω *, seja*

$$F(\omega) \doteq \mathcal{C}(f, \omega) \tag{71}$$

A função $F : \omega \mapsto F(w)$ *, fazendo* ω *percorrer todas as* 2^n *funções lineares realizáveis em* $\mathsf{GF}(2)^n$*, chama-se o* **espetro de Walsh** *de* f *.*

A transformação $\mathcal{W} : f \mapsto F$ *, que associa* f *ao seu espectro de Walsh, chama-se* **transformada de Walsh-Hadamard***.*

100 FACTO
Para todo $f, g \in \mathsf{GF}(2)^n \longrightarrow \mathsf{GF}(2)$ *verifica-se*

$$\mathcal{C}(f, g) = 2^{-n} \cdot \sum_x (-1)^{f(x)+g(x)} \tag{72}$$

e, se $F \doteq \mathcal{W}(f)$, *verifica-se para todo* $x \in \mathsf{GF}(2)^n$

$$(-1)^{f(x)} = \sum_{\omega} F(\omega)\,(-1)^{\omega(x)} \tag{73}$$

em que a primeira soma se estende a todos $x \in \mathsf{GF}(2)^n$ *e a segunda soma a todas as funções lineares* $\omega \in \mathsf{GF}(2)^n$.

Se $G \doteq \mathcal{W}(g)$ *verifica-se* $\mathcal{W}(f+g) = F \otimes G$ *em que a* **convolução** *de espectros de Walsh é definida por*

$$(F \otimes G)(\omega) = \sum_{\upsilon} F(\omega \oplus \upsilon) \cdot G(\upsilon) \tag{74}$$

DEFINIÇÃO 11
Seja $\mathbf{S}$ *uma SBox definida por um vector de funções boolenas* $(h_0, h_1, \cdots, h_{n-1})$. *Para quaisquer duas funções lineares* $\omega, \upsilon \in \mathsf{GF}(2)^n$ *seja*

$$\mathcal{C}^{\mathbf{S}}(\upsilon, \omega) \doteq \mathcal{C}(\upsilon \circ \mathbf{S}, \omega) \tag{75}$$

Vista como uma matriz $2^n \times 2^n$, *a função* $\mathcal{C}^{\mathbf{S}}$ *chama-se* **matriz de correlação** *de* $\mathbf{S}$.

101 FACTO
Nas condições da definição anterior,

(i) $\mathcal{W}(\upsilon \circ \mathbf{S}) = \bigotimes_{\upsilon_i=1} \mathcal{W}(h_i)$.

(ii) *Se* f *é uma função boolena de espectro* F *e* $g = f \circ \mathbf{S}$ *então o seu espectro* G *pode ser calculado como*

$$G \;=\; F \times \mathcal{C}^{\mathbf{S}}$$

em que G *e* F *são vistos como vectores-linha de* 2^n *componentes e a operação* $\times$ *é vista como a multiplicação de matrizes.*

(iii) *Se* $\mathbf{R}$ *é uma outra S-Box então a matriz correlação da composição* $\mathbf{R} \circ \mathbf{S}$ *é o produto (no sentido da multiplicação de matrizes) das duas matrizes de correlação.*

$$\mathcal{C}^{\mathbf{R}\circ\mathbf{S}} \;=\; \mathcal{C}^{\mathbf{R}} \times \mathcal{C}^{\mathbf{S}}$$

3.4 Funções de argumento GF(2^n)

As funções $f : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2^n)$ (SBoxes $n \times n$) têm argumentos que são vistos como elementos do corpo de Galois de ordem n. Como o domínio da função é finito ela pode ser sempre descrita[31] por um polinómio de grau inferior a $2^n - 1$

$$f(x) \;=\; \sum_{i=0}^{2^n-1} a_i\, x^i \;, \qquad a_i \in \mathrm{GF}(2^n) \tag{76}$$

Tendo em atenção que, para todo $x \neq 0$, se verifica

$$x^{2^n-1} = 1 \;,\; x^{2^n-2} = x^{-1} \;,\; \cdots \;,\; x^{2^n-1-i} = x^{-i}$$

então é possível escrever a representação anterior do seguinte modo:

$$f(x) \;=\; \begin{cases} a_0 & , \quad \text{se} \quad x = 0 \\ b_0 + \sum_{i=1}^{N} a_i\, x^i + b_i\, x^{-i} & , \quad \text{se} \quad x \neq 0 \end{cases} \tag{77}$$

[31] Qualquer função $f : \mathbb{F}_q \longrightarrow \mathbb{F}_q$, que passa por N pontos, pode ser aproximada por um polinómio de Lagrange de grau $N - 1$; o número total de argumentos de f é q e, por isso, existem, quanto muito, q pontos que a determinam; logo qualquer aproximação (incluindo a própria função) tem grau igual ou inferior a $q - 1$.

com

$$N = 2^{n-1} - 1 \ , \quad b_0 = a_0 + a_{(2^n-1)} \ , \quad b_i = a_{(2^n-1-i)} \quad i > 0$$

EXEMPLO 22 : Na cifra AES as operações sobre *bytes* são descritas por funções $f : \mathrm{GF}(2^8) \rightarrow \mathrm{GF}(2^8)$. O corpo de Galois é representado numa base polinomial do tipo 0 usando o polinómio característico $\mathbf{c} = y^8+y^4+y^3+y+1$.

A definição do **AES** especifica uma função `ByteSub` que é a composição de duas funções $\mathrm{ByteSub} = \mathrm{Inv} \circ \mathrm{Afim}$. A primeira das quais é a função auto-inversa

$$\mathrm{Inv}(x) \ = \ \begin{cases} 0 & \text{se} \quad x = 0 \\ x^{-1} & \text{se} \quad x \neq 0 \end{cases} \tag{78}$$

A segunda função é uma transformação afim definida não sobre $\mathrm{GF}(2^8)$ mas sim sobre sobre $\mathrm{GF}(2)^8$; tem a forma

$$\mathrm{Afim}(x) \ = \ c \oplus (\omega_0(x), \cdots, \omega_{n-1}(x))$$

$$\text{com} \quad c, \omega_i \in \mathrm{GF}(2)^8$$

(os valores concretos das constantes c, ω_i constam na especificação oficial do AES).

É importante ter uma representação de ambas as funções da mesma forma. Por isso é indispensável ver como converter uma representação linear ou afim genérica em $\mathrm{GF}(2)^n$ numa função de domínio $\mathrm{GF}(2^n)$. É o que faremos em seguida.

Alguns casos especiais de funções de domínio $\mathrm{GF}(2^n)$ merecem referência especial:

1. Os **automorfismos** $f : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2^n)$ (funções injectivas que preservam a estrutura do corpo) fixam necessariamente os elementos de $\mathrm{GF}(2)$.

   Vendo $\mathrm{GF}(2^n)$ como uma extensão de $\mathrm{GF}(2)$, o facto 84 determina que f tem a forma $\sigma^k : x \mapsto x^{2^k}$, para algum $k \in 0..n-1$. Só as potências do morfismo de Frobenius $\sigma : x \mapsto x^2$ são automorfismos neste corpo. É possível escrever os σ^k de uma forma vectorial de modo a permitir o uso de uma álgebra matricial para representar transformações lineares.
   Para tal define-se a função $(\cdot)_\sigma : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2^n)^n$ que transforma um elemento x do corpo de Galois no vector, sobre esse corpo, formado por todos $\sigma^k(x)$ (as imagens de x pelos vários automorfismos).

$$x_\sigma \doteq (x \ , \ \sigma(x) \ , \ \sigma^2(x) \ , \ \cdots \ , \ \sigma^{n-1}(x)) \tag{79}$$

2. As **funções boolenas** $f : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2)$ podem ser representadas pelas formas genéricas em (76) ou (77) tendo em atenção que o contradomínio $\mathrm{GF}(2)$ está imerso em $\mathrm{GF}(2^n)$.

EXEMPLO 23 :a função *traço* (ver definição 4 – página 123), é definida pelo polinómio $\mathrm{tr}(x) = x + x^2 + x^4 + \cdots + x^{2^{n-1}}$.

3. As **funções lineares** $f : \mathrm{GF}(2^n) \to \mathcal{K}$, sendo $\mathcal{K}$ uma qualquer extensão de $\mathrm{GF}(2)$, são funções que preservam somas e multiplicações por escalares:

$$f(x+y) = f(x) + f(y) \quad , \quad f(a\,x) = a\,f(x)$$

para todos $x, y \in \mathrm{GF}(2^n)$ e $a \in \mathrm{GF}(2)$.

A função linear f pode sempre ser escrita como um polinómio

$$f(x) \;=\; \sum_{k=0}^{n-1} a_k\,\sigma^k(x) \qquad \text{com} \quad a_k \in \mathcal{K} \tag{80}$$

Representanto por $\mathbf{a}$ o vector $(a_0, a_1, \cdots, a_{n-1})$, o polinómio (80) pode ser escrito como o produto escalar[32] de dois vectores

$$f(x) \;=\; \mathbf{a} \cdot x_\sigma \tag{81}$$

[32] Se $u, v \in X^n$ o produto escalar $u \cdot v$ é $u^t\,v = v^t\,u$, em que $(\cdot)^t$ representa a transposição de matrizes.

EXEMPLO 24 :Como caso particular temos construções da forma $\mathsf{tr}(x \cdot y)$; pela definição verifica-se imediatamente que, para todos $x, y \in \mathsf{GF}(2^n)$

$$\mathsf{tr}(x\, y) \;=\; x_\sigma \cdot y_\sigma$$

Voltando à representação em (81), seja $y = f(x) = \mathbf{a} \cdot x_\sigma$. Então y_σ pode ser calculado simplesmente como

$$y_\sigma \;=\; \mathbf{A}\, x_\sigma$$

sendo $\mathbf{A}$ a matriz cujo elemento genérico é

$$\mathbf{A}_{ij} = (a_k)^{2^i} \qquad \text{com} \quad k = j - i \pmod{n-1} \tag{82}$$

4. As **funções bilineares** são funções de dois argumentos

$$f : \mathsf{GF}(2^n) \times \mathsf{GF}(2^n) \longrightarrow \mathsf{GF}(2^n)$$

que são lineares em cada um dos seus argumentos.
Isto é, para todo $x, y, z \in \mathsf{GF}(2^n)$ e $a \in \mathsf{GF}(2)$

$$f(x, y+z) = f(x,y) + f(x,z) \quad , \quad f(x+z, y) = f(x,y) + f(z,y)$$

$$f(x, a \cdot y) = a \cdot f(x,y) = f(a \cdot x, y)$$

A forma mais geral para estas funções é dada por

$$f(x) \;=\; x_\sigma \cdot (\mathbf{A}\, y_\sigma) \tag{83}$$

sendo $\mathbf{A} \in \mathrm{GF}(2^n)^{n\times n}$ uma matriz com elementos em $\mathrm{GF}(2^n)$.

EXEMPLO 25 :
Exemplos de formas bilineares em $\mathrm{GF}(2^n)$, com $n > 3$

$$f(x,y) = x \cdot y \quad , \quad f(x,y) = x^2 \cdot y^4 + x^4 \cdot y^2 + c \cdot x^8 \cdot y^8$$

5. As **funções quadráticas** $g : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n)$ têm a forma $g(x) = f(x,x)$ sendo $f(\cdot,\cdot)$ uma função bilinear. Isto é,

$$g(x) \;=\; x_\sigma \cdot \mathbf{A}\, x_\sigma \tag{84}$$

para uma matriz apropriada $\mathbf{A} \in \mathrm{GF}(2^n)^{n\times n}$.

EXEMPLO 26 :
A função $g(x) = x^3$ é uma função quadrática já que pode ser escrita na forma $f(x,x)$ sendo $f(x,y) = x\, y^2$ que é, claramente, uma forma bilinear.

Pelo mesmo motivo qualquer monómio x^k é uma função quadrática se se puder escrever $k = 2^i + 2^j \pmod{2^n - 1}$ para i, j apropriados.

6. Uma base $\mathcal{B}$ de $\mathrm{GF}(2^n)$ determina sempre n **projecções**; i.e., funções booleanas $p_\mu : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2)$, uma para cada elemento $\mu \in \mathcal{B}$, de tal forma que qualquer $x \in \mathrm{GF}(2^n)$ pode ser reconstruído a partir dos elementos μ e dos valores $p_\mu(x)$

$$x \;=\; \sum_{\mu \in \mathcal{B}} p_\mu(x)\, \mu \tag{85}$$

Representemos por $(\cdot)^{\tilde{}} : \mathrm{GF}(2^n) \longrightarrow \mathrm{GF}(2)^n$ a função que associa x ao vector das suas projecções. Vectorialmente, (85) é

$$x \;=\; \mathcal{B} \cdot x^{\tilde{}} \tag{86}$$

Notas Algumas propriedades das projecções que derivam directamente de (85)

(i) Como a representação de x em $\mathcal{B}$ é única, as projecções são também únicas.

(ii) A projecção em $\mu \in \mathcal{B}$ de um outro elemento da base $\upsilon \in \mathcal{B}$ tem de ser zero porque, por definição de base, não é possível representar υ como uma soma de outros elementos da mesma base. Por isso

$$p_\mu(\upsilon) \;=\; \delta(\mu + \upsilon) \qquad \forall\, \mu, \upsilon \in \mathcal{B}$$

(iii) São sempre funções lineares que preservam a multiplicação escalar; i.e, verificam

$$p_\mu(0) = 0 \quad , \quad p_\mu(x+y) = p_\mu(x) + p_\mu(y) \quad , \quad p_\mu(a\,x) = a\,p_\mu(x)$$

para todos $x, y \in \mathrm{GF}(2^n)$ e $a \in \mathrm{GF}(2)$.

(iv) Para todo $x \in \mathsf{GF}(2^n)$ existe y tal que $p_\mu(x) \neq p_\mu(y)$; isto é, as projecções são funções sobrejectivas.

Se compararmos as duas últimas propriedades com as da função traço (página 123) vemos que elas coincidem; por isso é natural que existam semelhanças entre as duas noções. De facto é relativamente simples provar o seguinte resultado,

102 FACTO
Para qualquer elemento $\mu \in \mathcal{B}$ *de uma base de* $\mathsf{GF}(2^n)$ *existe um único elemento* $\mu' \in \mathsf{GF}(2^n)$, *designado por* **elemento dual** *de* μ, *tal que, para todo* $x \in \mathsf{GF}(2^n)$,

$$p_\mu(x) \;=\; \mathsf{tr}(\mu' x) \;=\; (\mu')_\sigma \cdot x_\sigma$$

O conjunto $\mathcal{B}' \doteq \{\, \mu' \mid \mu \in \mathcal{B} \,\}$ *de todos os elementos duais forma uma nova base de* $\mathsf{GF}(2^n)$ *que será designada por* **base dual** *de* $\mathcal{B}$.

Sugestão de prova: Construa-se a matriz $n \times n$ de elementos $\mathbf{T}_{\mu\upsilon} \doteq \mathsf{tr}(\mu\,\upsilon)$; as colunas da sua inversa $\mathbf{T}^{-1}$ determinam os elementos da base dual.

□

Ordenando os elementos de $\mathcal{B}$ num vector, e preservando a mesma ordem na base dual $\mathcal{B}'$, a prova do facto

anterior mostra que estes vactores estão relacionados por

$$\mathcal{B}' \;=\; \mathbf{T}^{-1}\,\mathcal{B} \tag{87}$$

em que $\mathbf{T}$ designa a matriz dos traços cruzados: $\mathbf{T}_{\mu\upsilon} = \mathrm{tr}(\mu\,\upsilon)$.

EXEMPLO 27 : Consideremos de novo $\mathrm{GF}(2^4)$ com uma base polinomial do tipo 0

$$\mathcal{B} = \{1\ ,\ \beta\ ,\ \beta^2\ ,\ \beta^3\}$$

Para calcular a matriz dos traços $\mathbf{T}_{ij} \doteq \mathrm{tr}(\beta^i \cdot \beta^j) = \mathrm{tr}(\beta^{i+j})$ basta calcular a lista dos traços das potências de β para expoentes entre 0 e 6.

Usando o polinómio característico $\mathbf{c}(X) = X^4 + X + 1$ facilmente se constrói a tabela

i	0	1	2	3	4	5	6
$\mathrm{tr}(\beta^i)$	0	0	0	1	0	0	1

A matriz $\mathbf{T}$ e a sua inversa $\mathbf{T}^{-1}$ são

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Portanto a base dual é formada pelos elementos (pela ordem dos respectivos duais em $\mathcal{B}$)

$$\mathcal{B}' = \{1 + \beta^3 \ , \ \beta^2 \ , \ \beta \ , \ 1\}$$

e as projecções respectivas serão

$$p_1(x) = \mathsf{tr}(x) + \mathsf{tr}(x\,\beta^3) \qquad , \qquad p_\beta(x) = \mathsf{tr}(x\,\beta^2)$$

$$p_{\beta 2}(x) = \mathsf{tr}(x\,\beta) \qquad , \qquad p_{\beta 3}(x) = \mathsf{tr}(x)$$

o que permite concluir a identidade

$$x = \mathsf{tr}(x) + \mathsf{tr}(x\,\beta^3) + \mathsf{tr}(x\,\beta^2)\,\beta + \mathsf{tr}(x\,\beta)\,\beta^2 + \mathsf{tr}(x)\,\beta^3$$

□

A noção de elemento dual pode ser extendida a qualquer $x \in \mathrm{GF}(2^n)$. O **elemento dual** de x na base $\mathcal{B}$, representado por x', é elemento de $\mathrm{GF}(2^n)$ que tem exactamente as mesmas componentes de x mas na base dual $\mathcal{B}'$; i.e. para todo $\mu \in \mathcal{B}$

$$p_\mu(x) = \mathrm{tr}(\mu' x) = \mathrm{tr}(\mu x') = p_{\mu'}(x') \tag{88}$$

Tomando como implícita a base $\mathcal{B}$, o operador $(\cdot)^\sim$ associa cada elemento de $\mathrm{GF}(2^n)$ ao vector das suas projecções nessa base. Então verifica-se

$$x'^\sim = \mathbf{T}^{-1} x^\sim \quad , \quad x' = \mathcal{B} \cdot x'^\sim = \mathcal{B}' \cdot x^\sim \tag{89}$$

□

O operador $(\cdot)_\sigma$ pode ser extendido para vectores de elementos $\mathrm{GF}(2^n)$. Dado $A \in \mathrm{GF}(2^n)^n$ a matriz $A_\sigma \in \mathrm{GF}(2^n)^{n\times n}$ tem, por linha de ordem i, o vector $(A_i)_\sigma$; isto é $(A_\sigma)_{ik} = \sigma^k(A_i)$.

Nestas circunstâncias

103 FACTO

Se $\mathcal{B}, \mathcal{B}'$ são um par de bases duais então:

1. $(\mathcal{B}')_\sigma = \mathbf{T}^{-1} \mathcal{B}_\sigma$

2. $x^{\tilde{}} = (\mathcal{B}')_\sigma \, x_\sigma \quad e \quad x_\sigma = (\mathcal{B}_\sigma)^t \, x^{\tilde{}}$
3. $(\mathcal{B}_\sigma)^t \, \mathcal{B}_\sigma = \mathbf{T} \quad e \quad (\mathcal{B}_\sigma)^t \, (\mathcal{B}')_\sigma = \mathbf{I}.$
4. $\mathcal{B}_\sigma \, (x')_\sigma = (\mathcal{B}')_\sigma \, x_\sigma.$

Prova O primeiro resultado é consequência da relação $\mathcal{B}' = \mathbf{T}^{-1} \, \mathcal{B}$, da linearidade dos morfismos σ^k e do facto de fixarem todos os elementos da matrix $\mathbf{T}^{-1}$.

O segundo resultado é consequência do anterior e do facto 102. Os últimos resultados são as definições da matriz dos traços cruzados $\mathbf{T}$, da base dual e elemento dual.

□

Todas estas noções são essenciais quando se pretende estudar funções booleanas que requerem "mudança de representação"; isto ocorre quando uma função é formada pela composição de uma sequência de funções que manipulam as palavras de *bits* alternadamente na representação $\mathrm{GF}(2^n)$ e na representação $\mathrm{GF}(2)^n$.

É o caso da SBox do AES (introduzida na exemplo 22) que é determinada pela composição da função $x \mapsto x^{-1}$ em $\mathrm{GF}(2^8)$ seguida de uma transformação afim em $\mathrm{GF}(2)^8$.

O problema essencial é o seguinte:

Dada uma base de representação $\mathcal{B}$ *em* $\mathsf{GF}(2^n)$ *, dada uma função* $f^{\tilde{}}$ *de domínio* $\mathsf{GF}(2)^n$ *, construir a função* f *de domínio* $\mathsf{GF}(2^n)$ *que verifica* $f^{\tilde{}}(x^{\tilde{}}) = f(x)^{\tilde{}}$ *para todo* x *.*
Ou, inversamente, dada a função f *, construir* $f^{\tilde{}}$ *.*

Note-se que: $f^{\tilde{}}(x^{\tilde{}})$ denota a função de palavras de bits $f^{\tilde{}}$ aplicada ao vector de bits que representa x (visto como elemento do corpo de Galois); $f(x)^{\tilde{}}$ é o vector de bits que representa o resultado $f(x)$; a relação entre as duas funções diz-nos que estes dois vectores são iguais.

Vamos procurar resolver este problema para algumas formas particulares de funções $f^{\tilde{}} : \mathsf{GF}(2)^n \longrightarrow \mathsf{GF}(2)^n$ ou $f^{\tilde{}} : \mathsf{GF}(2)^n \longrightarrow \mathsf{GF}(2)$.

$f^{\tilde{}}(x^{\tilde{}}) = x^{\tilde{}} \oplus c^{\tilde{}}$ com $c \in \mathsf{GF}(2^n)$.

Esta é a função *branqueamento* ("whitening") que ocorre quase sempre nos andares das cifras. A função de domínio $\mathsf{GF}(2^n)$ equivalente é

$$f(x) \;=\; x + c$$

$f^{\tilde{}}(x^{\tilde{}}) = c^{\tilde{}}(x^{\tilde{}}) = \mathsf{Tr}(c^{\tilde{}} * x^{\tilde{}})$ com $c \in \mathsf{GF}(2^n)$

Forma genérica da função booleana linear; a função de domínio $\mathsf{GF}(2^n)$ equivalente é

$$f(x) \;=\; \mathsf{tr}(c'\,x) \;=\; (c')_\sigma \cdot x_\sigma$$

Prova: Simples utilização das identidades no facto 103

$$f^{\tilde{}}(x^{\tilde{}}) = c^{\tilde{}} \cdot x^{\tilde{}} =$$

$$(\mathcal{B}_\sigma\,(c')_\sigma) \cdot (\mathcal{B}')_\sigma\, x_\sigma = (((\mathcal{B}')_\sigma)^t \mathcal{B}_\sigma\,(c')_\sigma) \cdot x_\sigma = (c')_\sigma \cdot x_\sigma$$

$f^{\tilde{}}(x^{\tilde{}}) = \mathbf{H}\, x^{\tilde{}}$ com $\mathbf{H} \in \mathrm{GF}(2)^{n\times n}$.

Esta é a forma genérica da SBox linear onde cada *bit* à saída é uma combinação linear dos *bits* de entrada.
A função de domínio $\mathrm{GF}(2^n)$ equivalente é o polinómio

$$f(x) = \quad \mathbf{h} \cdot x_\sigma \quad = \quad \sum_{k=0}^{n-1} \mathbf{h}_k\, x^{2^k} \tag{90}$$

em que $\mathbf{h} \in \mathrm{GF}(2^n)^n$ é o vector

$$\mathbf{h} = \quad (\mathcal{B}')_\sigma^t\, \mathbf{H}^t\, \mathcal{B}$$

Prova:

$$f(x) = \mathcal{B} \cdot f^{\sim}(x^{\sim}) =$$

$$\mathcal{B} \cdot (\mathbf{H}\,(\mathcal{B}')_{\sigma}\,x_{\sigma}) = ((\mathcal{B}')^{t}_{\sigma} \cdot \mathbf{H}^{t}\,\mathcal{B}) \cdot x_{\sigma}$$

É importante ter-se em conta que (90), apesar da forma aparentemente complexa, é uma função linear. porque tem a forma prevista em (80).

Pode-se interpretar $\mathbf{H}^{t}\,\mathcal{B}$ como o vector determinado pelas colunas de $\mathbf{H}$ vendo cada coluna como os coeficientes de um elemento de GF(2^{n}) na base $\mathcal{B}$.

Este vector (e, consequentemente, a matriz $\mathbf{H}$) pode ser recuperado de $\mathbf{h}$ por

$$\mathbf{H}^{t}\,\mathcal{B} \;=\; \mathcal{B}_{\sigma}\,\mathbf{h}$$

Isto permite fazer a conversão em sentido inverso: dada a função linear de domínio GF(2^{n}), obter a matriz que determina a função equivalente de domínio GF$(2)^{n}$.

A relação entre $\mathbf{H}$ e $\mathbf{h}$ pode ainde ser vista de outr forma; note-se que se podia escrever

$$\mathbf{h} \cdot x_{\sigma} = \;(\mathbf{H}^{t}\,\mathcal{B}) \cdot (\mathcal{B}')_{\sigma}\,x_{\sigma} = (\mathbf{H}^{t}\,\mathcal{B}) \cdot x^{\sim}$$

Finalmente pode-se ver

$$\mathbf{h} \;=\; (\mathbf{H}\,\mathcal{B}')^{t}_{\sigma}\,\mathcal{B}$$

$\mathbf{H}\,\mathcal{B}'$ representa ver as linhas da matriz como vectores representados na base $\mathcal{B}'$; isto é, se ω_{i} representar o elemento de GF(2^{n}) representado pela linha de ordem i da matriz **H**, então $\mathbf{H}\,\mathcal{B}' = (\omega'_{0}, \cdots, \omega'_{n-1})$ é o vector dos elementos duais.

Exemplo 28 : Regressando à cifra AES e ao exemplo 22, a transformação afim $g\tilde{}(y\tilde{}) = \mathbf{b} \oplus \mathbf{H}\, y\tilde{}$ é definida, na especificação da cifra, por

$$\mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

ou, representando cada *byte* em base hexadecimal e a marix como um vector de colunas,

$$\mathbf{b} = 63 \qquad \mathbf{H} = (\texttt{8F}\ ,\ \texttt{C7}\ ,\ \texttt{E3}\ ,\ \texttt{F1}\ ,\ \texttt{F8}\ ,\ \texttt{7C}\ ,\ \texttt{3E}\ ,\ \texttt{1F})$$

A especificação do AES define o polinómio caractarístico

$$\mathbf{c}[X] = 1 + X + X^3 + X^4 + X^8$$

Para uma base polinomial do tipo 0 gerada por este polinómio, e usando a mesma metodologia de representação,

temos a matriz dos traços cruzados, a sua inversa e $\mathcal{B}_\sigma$

$$\mathbf{T} = (\texttt{05} \ , \ \texttt{0A} \ , \ \texttt{15} \ , \ \texttt{2B} \ , \ \texttt{57} \ , \ \texttt{AE} \ , \ \texttt{5C} \ , \ \texttt{B9})$$

$$\mathbf{T}^{-1} = (\texttt{94} \ , \ \texttt{0D} \ , \ \texttt{1A} \ , \ \texttt{A0} \ , \ \texttt{65} \ , \ \texttt{CA} \ , \ \texttt{25} \ , \ \texttt{2A})$$

$$\mathcal{B}_\sigma = \begin{bmatrix} \texttt{01} & \texttt{01} & \texttt{01} & \texttt{01} & \texttt{01} & \texttt{01} & \texttt{01} & \texttt{01} \\ \texttt{02} & \texttt{04} & \texttt{10} & \texttt{1B} & \texttt{5E} & \texttt{E4} & \texttt{4D} & \texttt{FA} \\ \texttt{04} & \texttt{10} & \texttt{1B} & \texttt{5E} & \texttt{E4} & \texttt{4D} & \texttt{FA} & \texttt{02} \\ \texttt{08} & \texttt{40} & \texttt{AB} & \texttt{B3} & \texttt{E8} & \texttt{1D} & \texttt{4A} & \texttt{EF} \\ \texttt{10} & \texttt{1B} & \texttt{5E} & \texttt{E4} & \texttt{4D} & \texttt{FA} & \texttt{02} & \texttt{04} \\ \texttt{20} & \texttt{6C} & \texttt{97} & \texttt{94} & \texttt{91} & \texttt{80} & \texttt{9A} & \texttt{C5} \\ \texttt{40} & \texttt{AB} & \texttt{B3} & \texttt{E8} & \texttt{1D} & \texttt{4A} & \texttt{EF} & \texttt{08} \\ \texttt{80} & \texttt{9A} & \texttt{C5} & \texttt{20} & \texttt{6C} & \texttt{97} & \texttt{94} & \texttt{91} \end{bmatrix}$$

Os elementos de $\mathcal{B}_\sigma$ são polinómios; nesta representação são apresentados os seus coeficientes como vectores de bits, começando no grau mais elevado. Por exemplo $\texttt{E4} = \texttt{11100100}$ denota o polinómio $X^7 + X^6 + X^5 + X^2$.

Com estas três matrizes é possível computar todos os elementos necessários; por exemplo,

$$(\mathcal{B}')_\sigma = \mathbf{T}^{-1}\, \mathcal{B}_\sigma$$

que é a componente essencial para calcular $\mathbf{h}$ a partir de $\mathbf{H}$.

Feitas as contas conclui-se

$$\mathbf{h} = (\texttt{05}\ ,\ \texttt{09}\ ,\ \texttt{F9}\ ,\ \texttt{25}\ ,\ \texttt{F4}\ ,\ \texttt{01}\ ,\ \texttt{B5}\ ,\ \texttt{8F})$$

Conclui-se portanto que a transformação afim do AES é

$$\begin{aligned} g(y) = \quad & \texttt{63} + \texttt{05} \cdot y + \texttt{09} \cdot y^2 + \texttt{F9} \cdot y^4 + \texttt{25} \cdot y^8 + \\ & + \texttt{F4} \cdot y^{16} + \texttt{01} \cdot y^{32} + \texttt{B5} \cdot y^{64} + \texttt{8F} \cdot y^{128} \end{aligned}$$

A SBox do AES resulta da composição dessa função ao resultado da transformação

$$x \mapsto x^{254}$$

que mapeia 0 em si próprio e qualquer $x \neq 0$ em x^{-1}.

A transformação final abtém-se então substituindo, em $g(y)$, a variável y por x^{254} e reduzindo os expoentes módulo 255 (uma vez que $x^{255} = 1$ se $x \neq 0$).

Por exemplo

$$y^{128} \quad \mapsto \quad (x^{254})^{128} \quad \mapsto \quad x^{(254*128 \mod 255)} \quad \mapsto \quad x^{127}$$

Genericamente y^k reduz-se a x^{255-k}. O resultado final é

$$f(x) = \quad 63 + 05 \cdot x^{254} + 09 \cdot x^{253} + F9 \cdot x^{251} + 25 \cdot x^{247} + \\ + F4 \cdot x^{239} + 01 \cdot x^{223} + B5 \cdot x^{191} + 8F \cdot x^{127} \tag{91}$$

□

Finalmente vamos examinar a representação das **funções bilineares**

$$f(x, y) \ = \quad x_\sigma \cdot \mathbf{A}\, y_\sigma \tag{92}$$

ou equivalentemente, com $\boldsymbol{\alpha} \doteq \mathcal{B}_\sigma\, \mathbf{A}\, \mathcal{B}_\sigma^t$

$$f(x, y) \ = \quad x^{\tilde{}} \cdot \boldsymbol{\alpha}\, y^{\tilde{}} \tag{93}$$

Usando as identidades $x_\sigma = \mathcal{B}_\sigma^t\, x^{\tilde{}}$ e $\boldsymbol{\alpha} = \mathcal{B}_\sigma\, \mathbf{A}\, \mathcal{B}_\sigma^t$ tem-se

$$f(x, y) = (\mathcal{B}_\sigma^t\, x^{\tilde{}}) \cdot \mathbf{A}\, \mathcal{B}_\sigma^t\, y^{\tilde{}} = x^{\tilde{}} \cdot \mathcal{B}_\sigma\, \mathbf{A}\, \mathcal{B}_\sigma^t\, y^{\tilde{}} = x^{\tilde{}} \cdot \boldsymbol{\alpha}\, y^{\tilde{}}$$

Seja $\boldsymbol{\alpha}_k$ a matriz $\mathrm{GF}(2)^{n\times n}$ que selecciona a componente k de cada um dos elementos de $\boldsymbol{\alpha}$; seja z_k a componente correspondente de $f(x,y)$; então

$$z_k = x^{\tilde{}} \cdot \boldsymbol{\alpha}_k \, y^{\tilde{}}$$

Desta forma é possível calcular as componentes individuais de $f(x,y)$; expandindo obtém-se uma função booleana com monómios da forma $x_i \, y_j$.

$$z_k = \sum_{ij} \boldsymbol{\alpha}_k^{ij} \, x_i \, y_j$$

EXEMPLO 29 : Um exemplo de tal função, em $\mathrm{GF}(2)^4$ seria definida pelo sistema de equações

$$\begin{bmatrix} z_0 = & x_0\,y_0 + x_1\,y_0 + x_1\,y_1 \\ z_1 = & x_1\,y_0 + x_1\,y_2 + x_2\,y_0 \\ z_2 = & x_3\,y_3 \\ z_3 = & x_2\,y_0 + x_0\,y_2 \end{bmatrix}$$

As matrizes $\boldsymbol{\alpha}_k$ são

$$\boldsymbol{\alpha}_0 = \begin{bmatrix} 1000 \\ 1100 \\ 0000 \\ 0000 \end{bmatrix} \qquad \boldsymbol{\alpha}_1 = \begin{bmatrix} 0000 \\ 1010 \\ 1000 \\ 0000 \end{bmatrix} \qquad \boldsymbol{\alpha}_2 = \begin{bmatrix} 0000 \\ 0000 \\ 0000 \\ 0001 \end{bmatrix} \qquad \boldsymbol{\alpha}_3 = \begin{bmatrix} 0010 \\ 0000 \\ 1000 \\ 0000 \end{bmatrix}$$

A matriz $\boldsymbol{\alpha}$ congrega estas componentes individuais em vectores de *bits*

$$\boldsymbol{\alpha} = \begin{bmatrix} 1000 & 0000 & 0001 & 0000 \\ 1100 & 1000 & 0100 & 0000 \\ 0101 & 0000 & 0000 & 0000 \\ 0000 & 0000 & 0000 & 0010 \end{bmatrix}$$

Conhecida a matriz $\boldsymbol{\alpha}$, é possível recuperar a matriz $\mathbf{A}$,

$$\boldsymbol{\alpha} = \mathcal{B}_\sigma \, \mathbf{A} \, \mathcal{B}_\sigma^t \quad \Longrightarrow \quad \mathbf{A} = (\mathcal{B}')_\sigma^t \, \boldsymbol{\alpha} \, (\mathcal{B}')_\sigma \quad \text{com} \quad (\mathcal{B}')_\sigma = \mathbf{T}^{-1} \, \mathcal{B}_\sigma$$

Tomemos o polinómio característico $\mathbf{c}[X] = X^4 + X + 1$ e a base polinomial de tipo 0 apresentada no exemplo 27 (página 209).

A base dual calculada nesse exemplo foi

$$\mathcal{B}' = \{1 + \beta^3 \ , \ \beta^2 \ , \ \beta \ , \ 1\}$$

As matrizes $(\mathcal{B}')_\sigma$ e $\mathbf{A}$ que daí resultam são

$$(\mathcal{B}')_\sigma = \begin{bmatrix} 9 & 4 & 2 & 1 \\ D & 3 & 4 & 1 \\ E & 5 & 3 & 1 \\ B & 2 & 5 & 1 \end{bmatrix} \qquad \mathbf{A} = \begin{bmatrix} A & 5 & B & A \\ 3 & D & D & B \\ 2 & D & 7 & 1 \\ 5 & 5 & 7 & D \end{bmatrix}$$

As **funções quadráticas** têm uma representação que deriva directamente da representação das funções bilineares: se for $z = g(x) = x_\sigma \cdot \mathbf{A} x_\sigma$ então teremos

$$z \;=\; \tilde{x} \cdot \boldsymbol{\alpha}\, \tilde{x}$$

com $\boldsymbol{\alpha} = \mathcal{B}_\sigma\, \mathbf{A}\, \mathcal{B}_\sigma^t$. Deste modo, a componente z_k do resultado é

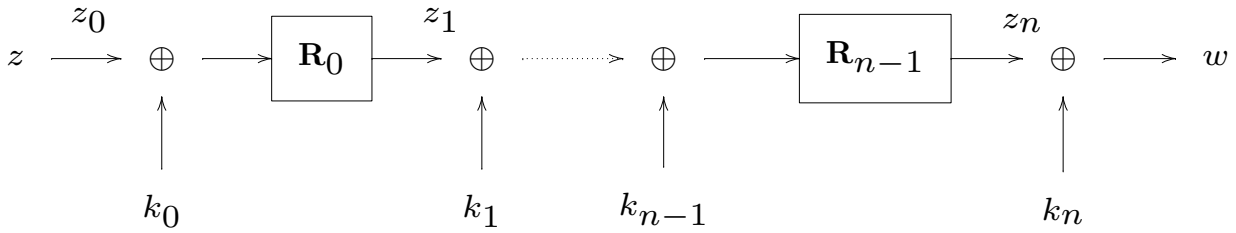
$$z_k \;=\; \sum_{ij} \boldsymbol{\alpha}_k^{ij}\, x_i\, x_j \tag{94}$$

em que $\boldsymbol{\alpha}_k^{ij}$ denota a componente de ordem k do elemento de índices i, j da matriz $\boldsymbol{\alpha}$.

O valor boolenao $\boldsymbol{\alpha}_k^{ij}$ determinam se o monómio $x_i\, x_j$ ocorre ou não na função boolena que calcula z_k. Assim, em termos de representações em $\mathrm{GF}(2)^n$, as formas quadráticas produzem (como seria de esperar) funções boolenas de grau 2; todos os monómios são da forma $x_i\, x_j$.

3.5 Criptoanálise Algébrica

As principais cifras simétricas[33] modernas têm a seguinte estrutura



(95)

A função **cifrar** ($w = f(k, z)$) processa-se de acordo com

1. As componentes essenciais são n SBoxes $\mathbf{R}_i$ invertíveis (os "**rounds**") e um **programador de chaves** (não representado na figura) que expande a **chave de controlo** k em $n+1$ **chaves de "round"** $k_0, k_1, .., k_n$.
2. Existe um **estado** z_i , inicializado com a entrada z , e que é recorrentemente alterado por duas transformações: uma soma $\oplus$ com a chave k_i e a aplicação de $\mathbf{R}_i$. Após o "round" final, a saída w obtém-se somando uma última chave k_n ao estado.
3. A relação de recorrência, fazendo variar $i \in 0..n-1$, é

$$z_0 = z \quad , \quad z_{i+1} = \mathbf{R}_i(z_i \oplus k_i) \qquad w = z_n \oplus k_n \tag{96}$$

[33] Cifras que usam a mesma chave para cifrar e decifrar.

A operação **decifrar** ($z = f^{-1}(k, w)$) tem a mesma estrutura com as alterações:

1. A novas chaves de "round" k'_i são as iniciais mas são apresentadas pela ordem inversa; isto é, $k'_i = k_{n-i}$
2. As SBoxs $\mathbf{R}_i$ são substitídas elas suas inversas algébricas e são apresentadas pela ordem inversa; isto é, $\mathbf{R}'_i = \mathbf{R}^{-1}_{n-i-1}$
3. O estado, representado por w_i , calcula-se fazendo variar $i \in 0..n-1$

$$w_0 = w \qquad w_{i+1} = \mathbf{R}'_i(w_i \oplus k'_i) \qquad z = w_n \oplus k'_n$$

Para ver que realmente estas duas funções são a inversa uma da outra basta notar que se preserva o invariante

$$w_i \oplus k'_i = z_{n-i} \qquad \text{ou} \qquad z_i \oplus k_i = w_{n-i}$$

Presupõe-se que uma mesma chave k é usada para cifrar um grande número de mensagens[34]. Presupõe-se também que são conhecidos alguns (poucos) pares mensagem+criptograma $\langle z_i, w_i \rangle$ gerados com essa chave.

Um ataque é uma algoritmo tratável que, a partir desta informação, descobre a chave k ou, equivalentemente, um algoritmo tratável que, a partir de um w arbitrário (distincto dos w_i conhecidos), descobre o elemento z

[34] Se a chave k fosse usada apenas uma vez então a cifra mais segura é simplesmente $w = z \oplus k$; esta é a chamada **segurança perfeita** de Shanon.

que cifrado com k reconstrói w. Numa cifra ideal a sua entropia está limitada pelo tamanho da chave em *bits*. Qualquer quebra em relação a este valor máximo é um ataque.

A segurança da cifra depende crucialmente do "design" dos "rounds" $\mathbf{R}_i$ e, normalmente, tem-se em vista dois objectivos principais:

- Não deve existir propagação de diferenças e, por isso, cada $\mathbf{R}_i$ deve manifestar um elevado grau de não-linearidade[35].

  Mais precisamente deve existir uma boa **mistura** entre a chave e a entrada: não deve ser possível "separar", à saída, os efeios individuais de cada um destes items.
- Não devem existir correlações entre visões particulares (paridades) da entrada e da saída. A existência de tais correlações implicaria a existência de relações priveligiadas entre alguns *bits* da entrada com alguns *bits* da saída, o que equivale a uma quebra na entropia da cifra.

  Deve existir uma boa **difusão** da influência de qualquer *bit* da entrada por toda a saída.

Idealmente os objectivos de "boa mistura" e "boa difusão" deveriam ser assegurados por um "design" único das SBoxes. No entanto, se atendermos que é necessário assegurar também boas propriedades computacionais numa função não-linear invertivel, este "design único" torna-se muito difícil.

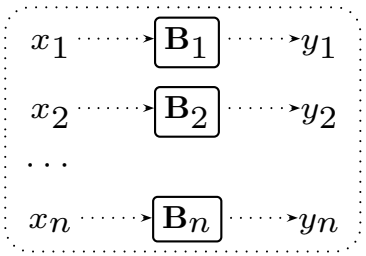
[35] Como cada "round" tem de ser uma função algebricamente invertível (para ser possível decifrar) se fosse linear as técnicas usuais de álgebra linear permitiriam um ataque trivial.

Por isso é usual decompor as SBox em componentes cada uma das quais com um objectivo próprio. Por exemplo é usual escolher uma componente linear com propriedades óptimas em termos de difusão mas muito má (por ser linear) em termos de mistura. Outro tipo de componente são as "bricklayer functions" (que veremos em seguida) que têm boas propriedades em termos de mistura e eficiência computacional mas que apresentam elevadas correlações e, por isso, são más em termos de difusão.

Funções "bricklayer"

A entrada x e a saída y são vectores de $n \times s$ *bits* agrupados em n blocos de s *bits*.

Cada bloco é transformado independentemente dos restantes por uma $s \times s$-SBox $\mathbf{B}_i$. A SBox global $\mathbf{B}$ (de tamanho $(n\,s) \times (n\,s)$) obtém-se fraccionando a entrada x em blocos x_i, aplicando a SBox $\mathbf{B}_i$ ao bloco x_i e agregando os resultados parciais y_i num único vector y.



$$y = \mathbf{B}(x) \qquad y, x \in (\mathbb{B}^s)^n$$
$$y = (y_1, \ldots, y_n) \qquad x = (x_1, \ldots, x_n)$$
$$y_i = \mathbf{B}_i(x_i) \qquad x_i, y_i \in \mathbb{B}^s$$

Se todas as SBoxes $\mathbf{B}_i$ forem invertíveis, também $\mathbf{B}$ é invertível. Adicionalmente $\mathbf{B}^{-1}$ é também uma função "bricklayer" determinada pelas n inversas $\mathbf{B}_i^{-1}$.

A vantagem destas funções reside na sua eficiência computacional; isto deriva do facto de lidar com SBox de uma dimensão s que é muito menor do que a simensão $(n\, s)$ exigida para a SBox global.

Para além de ausências de diferenças e correlações (ligadas aos objectivos de "boa mistura" e "boa difusão"), surge um outro tipo de objectivo que deriva da existência de uma outra forma de ataque.

Considere-se, de novo, as relações em (96) que traduzem as relações essenciais entre os vários items de informação que caracterizam a cifra. Delas resulta um sistema de equações com incógnitas z_i e k_i conhecida a entrada z e o criptograma w.

$$\begin{cases} z_0 & = z \\ z_n \oplus k_n & = w \\ z_{i+1} \oplus \mathbf{R}_i(z_i \oplus k_i) & = 0 \qquad i \in 0..n-1 \end{cases} \tag{97}$$

Uma tentativa de resolver estas equações esbarra com a forma complicada das SBoxes $\mathbf{R}_i$. No entanto é geralmente possível inserir estas equações numa estrutura algébrica adequada (num corpo finito, específicamente) de tal modo que as equações possam ser escritas

$$\begin{cases} z_0 & = z \\ z_n + k_n & = w \\ \mathbf{F}_i(z_{i+1}, z_i + k_i) & = 0 \qquad i \in 0..n-1 \end{cases} \tag{98}$$

em que as funções $\mathbf{F}_i(\cdot,\cdot)$ têm uma estrutura algebricamente muito mais símples do que os $\mathbf{R}_i(\cdot)$.

EXEMPLO 30 : O exemplo paradigmático é SBox AES determinada por $y = x^{254}$ ou $y = x^{-1}$ (quando $x \neq 0$) que, explicitamente, é uma função não-linear bastante complexa mas que, implicitamente, se escreve como uma forma bilinear

$$x \cdot y = 1 \quad , \; x \neq 0$$

A forma indicada em (92) e (93) (página 219) para as funções bileneares permite-nos ver esta forma gerada pela matrizes

$$\mathbf{A} = \begin{bmatrix} 10000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \end{bmatrix} \qquad \boldsymbol{\alpha} = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 \\ 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1B \\ 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 \\ 08 & 10 & 20 & 40 & 80 & 1B & 36 & 6C \\ 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 \\ 20 & 40 & 80 & 1B & 36 & 6C & D8 & AB \\ 40 & 80 & 1B & 36 & 6C & D8 & AB & 4D \\ 80 & 1B & 36 & 6C & D8 & AB & 4D & 9A \end{bmatrix}$$

Uma vez mais, o elementos genérico a em ambas as matrizes é descrito pela palavra de bits $a^{\tilde{}}$ em notação hexadecimal.

Recordemos, nesta representação da forma bilinear $x \cdot y$, a equação $1 = x \cdot y$ se descreve como

$$1 \;=\; \sum_{i,j} \boldsymbol{\alpha}_{ij} \cdot \; x_i \, y_j$$

É possível olhar para esta representação de uma outra forma: considera-mos, por momentos, os pares $x_i \, y_j$ como uma única incógnita e a matriz $\boldsymbol{\alpha}$ como um vector de coeficientes. Vamos chamar X ao vector destas "incógnitas duplas" e continuamos a representar por $\boldsymbol{\alpha}$ a forma linearizada da matriz.

Com $64 = 8 \times 8$ incógnitas booleanas a equação é escrita

$$1 \;=\; \boldsymbol{\alpha} \cdot X \tag{99}$$

Esta equação base dá origem e outras; nomeadamente

$$x = x^2 \, y, \quad x^2 = x^4 \, y^2, \quad x^4 = x^8 \, y^4, \quad \cdots x^{64} = x^{128} \, y^{64}$$

e a versão dual para y

$$y = x \, y^2, \quad y^2 = x^2 \, y^4, \quad \cdots \quad y^{64} = x^{64} \, y^{128}$$

Cada uma destas duas sequências é gerada a partir da equação base ($x = x^2 \cdot y$ ou $y = x \cdot y^2$) por aplicação sucessiva da transformação linear $\sigma : z \mapsto z^2$ a ambos os lados da equação; as equações resultantes são, portanto, linearmente dependentes.

De facto σ gera transformações lineares nos vectores $x\tilde{}$ e $y\tilde{}$ que permitem ir transformando uma equação na seguinte.

Portanto temos três equações em GF(2^8) (das quais resultam 3×8 equaçoes em GF(2)) que podem ou não ser linearmente independentes e que derivam de

$$x \cdot y = 1 \quad (x \neq 0) \qquad x^2\, y + x = 0 \qquad x\, y^2 + y = 0$$

Existem ainda outras formas bilineares; por exemplo, multiplicando ambos os lados da 2ª equação por x^2 e ambos os lados da 3ª equação por y^2 constrói-se

$$x^4\, y + x^3 = 0 \qquad x\, y^4 + y^3 = 0$$

As formas bilineares em todas estas 5 equações são por matrizes¡ $\mathbf{A}$ apropriadas que vão dar origem a matrizes $\boldsymbol{\alpha}$.

Os restnates constituintes dos lados esquerdos das equações (para além das formas bilineares) são monímios x , y , que são formas lineares, ou os monómios x^3 , y^3 , que são formas quadráticas.

As formas bilineares têm exactamente a mesma forma que (99) e dão origem a somas de monómios do tipo $x_i\, y_j$ calculadas por $x\tilde{} \cdot \boldsymbol{\alpha}\, y\tilde{} = \sum_{ij} \boldsymbol{\alpha}_{ij}\, x_i\, y_j$.

Para a forma $x^2\, y$ tem-se a matriz $\mathbf{A}'$ com $\mathbf{A}'_{10} = 1$ e $\mathbf{A}'_{ij} = 0$ para os restantes índices.

A forma $x\,y^2$ é determinada por uma matriz $(\mathbf{A}')^t$ que é transposta da anterior.

A forma $x^4\,y$ é definida pela matriz $\mathbf{A}''$ com $\mathbf{A}''_{20} = 1$ e $\mathbf{A}'_{ij} = 0$ para os restantes índices. A forma $x\,y^4$ é definida pela transposta desta matriz.

As matrizes $\boldsymbol{\alpha}$ correspondentes serão

$$\boldsymbol{\alpha}' = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 \\ 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 \\ 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 \\ 40 & 80 & 1B & 36 & 6C & D8 & AB & 4D \\ 1B & 36 & 6C & D8 & AB & 4D & 9A & 2F \\ 6C & D8 & AB & 4D & 9A & 2F & 5E & BC \\ AB & 4D & 9A & 2F & 5E & BC & 63 & C6 \\ 9A & 2F & 5E & BC & 63 & C6 & 97 & 35 \end{bmatrix}$$

$$\boldsymbol{\alpha}'' = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 \\ 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 \\ 1B & 36 & 6C & D8 & AB & 4D & 9A & 2F \\ AB & 4D & 9A & 2F & 5E & BC & 63 & C6 \\ 5E & BC & 63 & C6 & 97 & 35 & 6A & D4 \\ 97 & 35 & 6A & D4 & B3 & 7D & FA & EF \\ B3 & 7D & FA & EF & C5 & 91 & 39 & 72 \\ C5 & 91 & 39 & 72 & E4 & D3 & BD & 61 \end{bmatrix}$$

Juntanto todas estes vectors $\boldsymbol{\alpha}$ como linhas de uma matriz constrói-se uma nova matriz com $64 = 8 \times 8$ colunas

e $5 \times 8 = 40$ linhas. Daqui resulta um sistema de equações "aparentemente linear" nas incógnitas X

$$\begin{cases} 1 & = \boldsymbol{\alpha} \cdot X \\ x & = \boldsymbol{\alpha}' \cdot X \\ y & = (\boldsymbol{\alpha}')^t \cdot X \\ x^3 & = \boldsymbol{\alpha}'' \cdot X \\ y^3 & = (\boldsymbol{\alpha}'')^t \cdot X \end{cases} \quad \Longrightarrow \quad \begin{bmatrix} 1 \\ x \\ y \\ x^3 \\ y^3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}' \\ (\boldsymbol{\alpha}')^t \\ \boldsymbol{\alpha}'' \\ (\boldsymbol{\alpha}'')^t \end{bmatrix} X$$

Não se trata realmente de um sistema de equações lineares porque os $x\tilde{}, y\tilde{}$ aparecem por si no lado esquerdo das equações ao mesmo tempo que aparecem nas incógnitas X .

A questão essencial está no número de equações que são linearmente independentes e sobre a forma como estes sistemas podem ser resolvidos.

No exemplo anterior resultaram 5 formas bilineares

$$x \cdot y \qquad x^2 \cdot y \qquad x^4 \cdot y \qquad x \cdot y^2 \qquad x \cdot y^4$$

para as quais foi possível construir equações lineares no binómios $x_i y_j$. Levantam-se imediatamente duas questões:

1. As equações resultantes são linearmente independentes?

2. Será possível acrescentar outras formas (por exemplo, $x^8 \cdot y$) que gerem equações linearmente independentes das existentes?