Técnicas Criptográficas

José Manuel E. Valença *

16 de Novembro de 2009

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

1.Fundamentos da Computabilidade

...God made the integers, all else is the work of man...[1]

...God made the bit-strings, all else is the work of programmers...[2]

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

Vamos focar algumas noções de computabilidade e falaremos dos domínios relevantes ao processo computacional. 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.

[1] Leopold Kronecker- 1823/1891

[2] senso comum!

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 afir- . mação 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 3 representa $f(n) = \Theta(n\,g(n))$; ou seja,

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

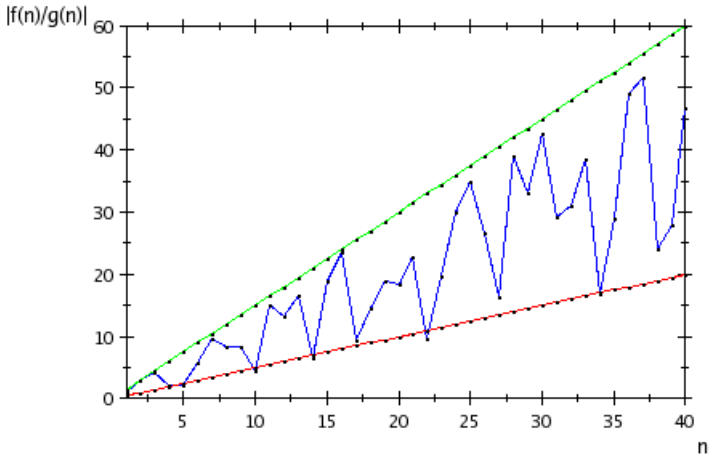
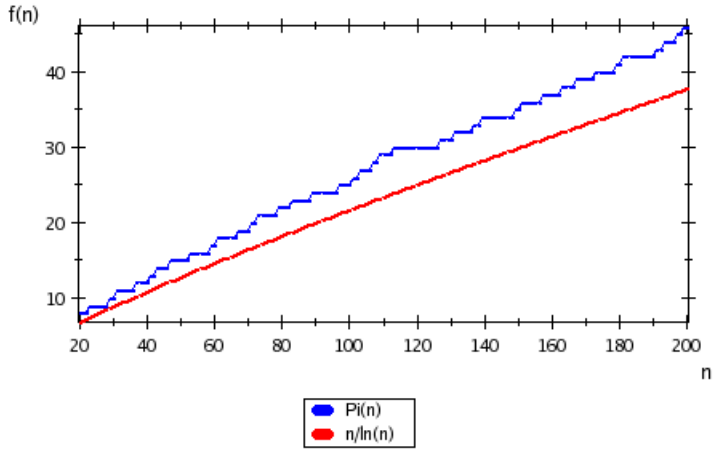


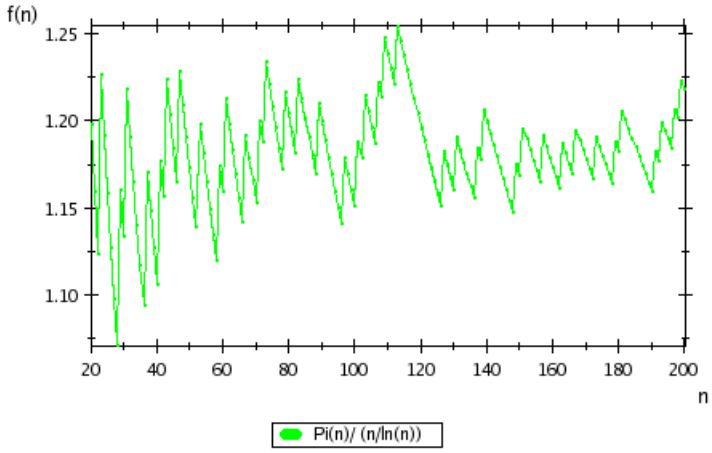
Figura 1: $f(n) = \Theta(n\,g(n))$.

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*. [3]

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* $p(n)$[4]*,*
- ***polinomial inversa*** *quando* $\tau(n)^{-1} = O(p(n))$ *, para um polinómio positivo* $p(n)$[5]*,*

[3] Em alternativa, a complexidade pode contar o número *máximo* ou o número *mínimo* de *slots* temporais ou unidades de memória.
[4] Como casos particulares temos as ordens **linear** ($\tau(n) = O(n)$), **quadrática** ($\tau(n) = O(n^2)$), **cúbica** ($\tau(n) = O(n^3)$), etc. . .
[5] 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).

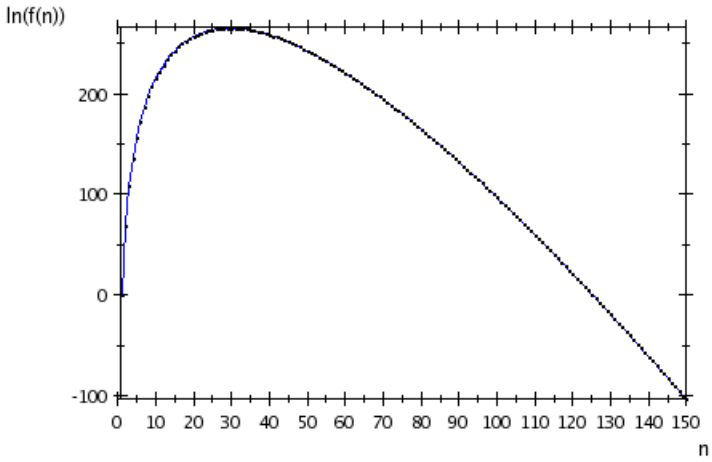
□

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 Domínios

Neste capítulo inicial procuraremos introduzir a notação matemática usada neste curso. Começamos por alguma notação sobre **domínios**, independentemente da estrutura algébrica que sobre eles colocar-mos.

Neste texto usaremos as seguintes notações

- $\mathbb{N}$ é o domínio dos **inteiros naturais** $\{0, 1, 2, \cdots\}$.
  - $\mathbb{N}_n$ (também representado por $\mathbb{Z}_n$) é o intervalo $[0..n-1]$ os primeiros n números naturais.
  - $\mathbb{N}^n$ é o domínio dos **vectores** de números naturais com exactamente n elementos,
  - $\mathbb{N}^* = \bigcup_{n=0}^{\infty} \mathbb{N}^n$ é o domínio das **sequências finitas** de números naturais.
  - $2^{<\mathbb{N}}$ é o domínio dos **conjuntos finitos** de números naturais.

- $\mathbb{B}$ é o domínio $\{0, 1\}$ dos **bits**; também representado como $\mathbb{Z}_2$.
  - $\mathbb{B}^n = \{0, 1\}^n = \mathbb{Z}_2^n$ representa a classe das **palavras** com exactamente n bits.
  - $\mathbb{B}^* = \bigcup_{n=0}^{\infty} \mathbb{B}^n$ representa o domínio das **palavras** (ou **sequências finitas**) de *bits*.

- $\mathbb{Q}$ representa o domínio dos **racionais**,
  Formalmente um número racional é definido por um par de inteiros escrito na notação *numerador/denominador* n/d (**fracções**) assumindo que representam o mesmo racional duas fracções n/d e n'/d' que verifiquem $n * d' = d * n'$.
  Por exemplo $3/6$ e $1/2$ são fracções distintas que denotam o mesmo racional.

Alguma notação com *strings* de bits.

- $|s|$ denota o **comprimento** de s; i.e. o número e elementos da sequência de bits.
- $(r\,s)$ denota a **concatenação** de duas *strings* r e s; i.e. $(r\,s)_i = r_i$, se $i \leq |r|$, e $(r\,s)_i = s_{i-|r|}$, se $i \geq |r|$.
- $r \leq s$ verifica-se, e diz-se que r é um **prefixo** de s, quando existe uma *string* u tal que $s = r\,u$.
- $s \restriction k$ é o maior prefixo de s cujo comprimento não excede k. $s \downharpoonright k$ é o maior sufixo de s cujo comprimento não excede $|s| - k$. Quando $|s| \geq k$, $s \restriction k$ é a *string* formada pelo primeiros k bits de s; $s \downharpoonright k$ é a string u tal que $s = (s \restriction k)\,u$.

□

Uma "coisa" é computável!. O que significa esta frase?

Ao longo deste curso, usaremos a designação de **computável** para classificar várias "coisas": *funções*, *domínios*, *relações*, *conjuntos*, *linguagens* e *sequências*. Por exemplo, a existência de "funções computáveis", que verifiquem certas propriedades, são extensivamente usadas na formalização dos critérios de segurança de técnicas criptográficas.

Trata-se, por isso, de uma noção central em Criptografia e, logicamente, vamos ter a oportunidade de discutir nos capítulos que se seguem as várias abordagens possíveis ao conceito de computabilidade. No entanto, convém avançar com algumas das motivações que justificam as escolhas aqui feitas.

Tome-se, por exemplo, a noção de **função computável**.

Em primeiro lugar parece óbvio que, para ser computável, uma função precisa de ser implementável num dispositivo computacional *standard* e que tal dispositivo tem de ser descrito por um modelo computacional de referência. Dado as *Máquinas de Turing* são o modelo computacional mais comum, faz sentido impor um primeiro princípio que diga algo do tipo:

> Só é computável uma função que for implementável por uma Máquina de Turing Universal.

Essencialmente, para que a função f seja computável, tem de existir um *programa* numa máquina de Turing de referência, que, face a um argumento arbitrário x, receba como *input* uma representação de x e produza um *output* que seja uma representação de $f(x)$. Esta condição pode ser necessária mas está longe de ser suficiente.

Nomeadamente não entra em conta com a *dificuldade de cálculo* inerente ao *output*. Claramente são necessárias outras condições se caracterizar completamente a noção de função computável e, para isso, existem várias abordagens que, não sendo completamente alternativas, representam, ainda assim, pontos de vista distintos.

1. **Orientada à Complexidade**
   A Teoria da Complexidade é uma área bem estabelecida das Ciências Computação que tem por objectivo a "contabilidade" dos recursos (normalmente tempo e espaço) necessários à "execução" de um programa.

Nesta perspectiva faz sentido considerar como computáveis as funções que são implementáveis por programas que consumam "recursos razoáveis" nas suas execuções.

Esta abordagem assume que o dispositivo computacional, e as restantes pré-requisitos da computação, são completamente determinados e que a computabilidade é algo que é inerente, exclusivamente, ao programa.

2. **Orientada à Probabilidade**

   Uma perspectiva diferente consiste em supor que o dispositivo pode não estar completamente determinado e, ao invés, estabelecer um conjunto de "critérios de sanidade" que devem ser satisfeitos por uma execução desejável dum dispositivo computacional típico.

   Assumindo uma selecção "aleatória" dum argumento x (seguindo uma determinada distribuição probabilística), procura-se determinar a probabilidade de existir uma execução que calcule $f(x)$ e que satisfaça os referidos critérios. Serão consideradas computáveis as funções onde essa probabilidade seja "suficientemente elevada".

3. **Orientada à Incerteza**

   A noção de "incerteza" traduz a dificuldade em prever a ocorrência de um determinado **evento**. Tipicamente interessam-nos eventos que descrevam a *"obtenção de um novo conhecimento"* por *"agentes"* caracterizados pelas seus *"privilégios"* (para computar, memorizar, adivinhar!, ...) e pelo seu *"conhecimento prévio"*.

   Fixemos, pelos seus privilégios, uma classe de agentes. *"A conhece x"* e *"A conhece $f(x)$"* podem representar dois eventos; se a dificuldade em prever o primeiro for substancialmente menor que a dificuldade em prever o segundo, independentemente de x ou do agente A, é razoável assumir que a função f *introduz incerteza*.

   Nesta perspectiva serão computáveis, precisamente, as funções que não introduzem incerteza.

Parece óbvio que a primeira abordagem é a que melhor permite uma definição formal mas isso não significa que essa

definição traduza uma visão realista do conceito. A abordagem orientada à incerteza é a mais geral e é aquela que melhor traduz uma noção de computabilidade que se reflecte a intuição que possuimos sobre as noções de segurança das técnicas criptográficas; porém é aquela que tem uma formalização mais complicada.

No entanto as abordagens não são alternativas e é possível converter umas noutras. Por exemplo, se restinguirmos os agentes a não terem memória e limitar-mos os seus privilégios à utilização de uma máquina que consuma "recursos razoáveis", então a abordagem orientada à incerteza colapsa na abordagem orientada à complexidade.

A abordagem orientada à probabilidade é algo intermédio entre as outras duas. Não têm as limitações expressivas da abordagem orientada à complexidade mas também não tem o poder expressiva da abordagem orientada à incerteza. Por outro lado é mais facilmente formalizável. Por estas razões é frequentemente adoptada (mais precisamente, as noções de segurança que dela derivam) nas chamadas "provas de segurança" das técnicas criptográficas.

□

De momento vamos-nos abstrair em relação a uma definição precisa de "função computável", vamos aceitar aquilo que a intuição nos diz ser uma visão razoável para este conceito. Nomeadamente vamos assumir que a composição de duas funções computáveis é também computável. Partindo destas noção básica, vamos ver como outros conceitos de desenvolvem.

Seja X um domínio. Uma função sobrejectiva computável $f : \mathbb{N} \to X$ é **unidireccional** (ou **"one-way"**)

quando não tem quasi-inversa (também designada "inversa fraca")[6] que seja computável; i.e., quando não existe $g : X \to \mathbb{N}$ computável tal que $(\forall\, n)\,[f(g(f(n))) \simeq f(n)]$.

Designaremos por **computacionalmente enumerável** (ou só *"enumerável"*) um domínio X, quando existe uma função sobrejectiva computável $f : \mathbb{N} \to X$. Assim, X é enumerável quando existe f computável tal que

$$(x \in X) \iff (\exists\, n \in \mathbb{N})\ [\, f(n) = x\,] \tag{1}$$

A função f designa-se por **enumeração** de X. Um valor n tal que $f(n) = x$ é uma *prova* da asserção $(x \in X)$

Se, adicionalmente, f tem uma quasi-inversa computável, o domínio X diz-se **codificável**. Se g for essa quasi-inversa, $g(x)$ computa uma prova de $(x \in X)$, que, neste caso, se designa por *código* de x.

Nota

A noção de domínio enumerável pretende descrever as situações onde um elemento $x \in X$ só pode ser determinado pelo facto de ocupar uma determinada ordem n numa enumeração f de X.

O domínio é determinado pela sua enumeração e a relação $(x \in X)$ é representada pela sua prova; nomeadamente a selecção de um elemento x do domínio só pode ser feita gerando uma prova n da relação $(x \in X)$.

[6] Qualquer função sobrejectiva $f : \mathbb{N} \to X$ tem, pelo menos, uma quasi-inversa se se admitir válido o *Axioma da Escolha*. Aliás esta afirmação não é mais do que uma das muitas formulações desse axioma. Porém a questão que se coloca, para saber se f é unidireccional ou não, é a computabilidade dessa quasi-inversa.

Mesmo a própria inclusão de domínios $X \subseteq Y$ tem de ser definida usando estes princípios. Diz-se que $X \subseteq Y$ se existem enumerações, f de X e g de Y, tais que $(\forall\, n)\,(\exists\, m)\,[f(n) = g(m)]$.

Genericamente, se f e g são enumerações escreve-se $f \subseteq g$ para designar a relação $(\forall\, n)\,(\exists\, m)\,[f(n) = g(m)]$.

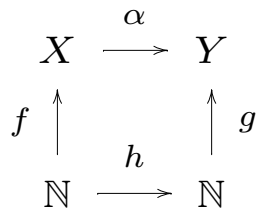
A enumeração f de X não é necessariamente única: o mesmo domínio pode ter várias enumerações. Se g for uma segunda enumeração de X, então (1) implica que têm de ser válida a asserção $g \subseteq f$ e $f \subseteq g$. As enumerações f e g são *equivalentes* se existe uma bijecção computável s tal que $(\forall\, n)\,[g(s(n)) = f(n)]$.

Temos $X \subseteq Y$ se existem enumerações, f de X e g de Y, tais que $f \subseteq g$. Neste caso $X \subseteq Y$ é **computável** se existe h computável tal que $(\forall\, n)\,[f(n) = g(h(n))]$.

Nota O axioma da escolha diz-nos que a validade de $(\forall\, n)\,(\exists\, m)\,[f(n) = g(m)]$ implica a existência de uma função h tal que $(\forall\, n)\,[f(n) = g(h(n))]$. No entanto não dá quaisquer garantias que tal h seja computável.

Generalisando, se X e Y são dois domínios enumeráveis, uma aplicação $\alpha\colon X \to Y$ diz-se **computável** se existem enumerações f, g e uma função computável h tal que

$$(\forall\, n)\,[\alpha(f(n)) = g(h(n))]$$



Se α for computável, é **unidireccional** quando em todos f, g, h que verifiquem a igualdade anterior, h é unidireccionl ou g é unidireccional.

Nota

Num domínio eneumerado X, cada elemento $x \in X$ só pode ser determinado por uma prova n dessa asserção. Por isso, a aplicação $\alpha(x)$ só pode ser computada a partir desse valor n.

Mas, pela mesma razão, $\alpha(x)$ só pode ser determinada pela ordem m que ocupa na enumeração de Y. Desta forma deve existir uma função h que compute um possível m a partir de n; isto é, tem de ser $m = h(n)$ e, consequentemente, $\alpha(f(n)) = \alpha(x) = g(m) = g(h(n))$.

Nesta abordagem α é unidireccional se g for unidireccional (i.e., não é computável, a partir de $y \in Y$, determinar uma ordem m tal que $g(m) = y$) ou então h é unidireccional (i.e., não é computável, mesmo que seja determinado m, computar n tal que $h(n) = m$).

□

Uma função parcial booleana $\xi : X \to \{0, 1\}$, que verifique

$$\xi(x) \simeq 1 \;\Rightarrow\; x \in X \qquad , \qquad \xi(x) \simeq 0 \;\Rightarrow\; x \notin X \tag{2}$$

designa-se por **função característica** do domínio X. O domínio X é **computável** quando tem uma função característica total e computável.

A igualdade (1) estabelece a relação fundamental entre uma função característica e a enumeração de um domínio enumerável. De facto existe uma noção mais genérica de que (1) é um caso particular.

Um domínio X é **semidecidível** se existe uma função booleana computável $\tau : \mathbb{N} \times X \to \mathbb{B}$ tal que

$$(x \in X) \quad \Longleftrightarrow \quad (\exists\, n)\; [\, \tau(n, x) = 1 \,] \tag{3}$$

Nota Se X verificar (3) é sempre possível construir uma função recursiva f que verifique (1). Porém não há garantias que esta função seja computável; por isso, toda o domínio computacionalmente enumerável é semidecidível mas o inverso não é verdade.

Exemplo 2: Seja $\mathbb{P}$ o domínio dos *números primos*. O seu complemento $\bar{\mathbb{P}} = \mathbb{N} \setminus \mathbb{P}$, o domínio dos *números compostos*, é semidecidível mas não é computacionalmente enumerável.

De facto pode-se definir uma função booleana computável τ tal que $\tau(n, x) = 1$ se e só se n é um divisor de x diferente de 1 e de x. Este predicado garante que $\bar{\mathbb{P}}$ é, pelo menos, semidecidível.

Porém não existe nenhuma maneira de encontrar o número composto de ordem n que não passe por computar todos os números compostos de ordem inferior a n. Claramente tal função é recursiva mas não é computável.

Essencialmente X é enumerável quando, se ocorrer $x \in X$, é sempre possível prová-lo; (1) diz-nos que basta percorrer todos os inteiros n e fazer o teste $f(n) = x$. Eventualmente o teste tem sucesso e o inteiro n é uma **prova** de que $x \in X$ ocorre.

Este procedimento, porém, não produz qualquer prova se x não pertence a X. De facto pode-se percorrer todos os n, sem encontrar um que satisfaça o teste e sem ter garantias que não vai aparecer futuramente um que o satisfaça.

Para ilustrar este ponto considere-se o seguinte programa (numa pseudo-linguagem semelhante ao C).

```
in (x)  ==>
     for (n=0 ; ; n++)
           if (f(n) == x) return 1;
```

Figura 2: Função parcial característica de um domínio enumerável.

Se X for enumerável, então o programa termina se e só se $x \in X$. Por isso, para provar $x \notin X$, é necessário outro conceito.

Um domínio enumerável $X \subseteq \mathbb{N}$ é **totalmente enumerável** se existir $g : \mathbb{N} \to \mathbb{N}$, computável, que verifica

$$(x \in X) \iff (\forall\, n \in \mathbb{N})\ [\, g(n) \neq x \,] \tag{4}$$

Isto é, X é totalmente enumerável se tanto X como o seu complemento $\mathbb{N} \setminus X$ forem enumeráveis.

Por analogia, diremos que um domínio é **computacionalmente decidível** se tanto X como $\mathbb{N} \setminus X$ forem semidecidíveis. Isto significa que tem de existir uma função booleana computável $\sigma : \mathbb{N} \times X \to \mathbb{B}$ tal que

$$(x \in X) \iff (\forall\, n \in \mathbb{N})\ [\, \sigma(n, x) = 1 \,] \tag{5}$$

Claramente, se X é um domínio computável (i.e., se a sua função característica for computável), então também é decidível e o seu complemento $\mathbb{N} \setminus X$ é também computável.

Quando um domínio X é totalmente enumerável, o seguinte programa termina sempre e implementa a função característica de X.

```
in (x) ==>
     for (n=0 ; ; n++)
           if (g(n) == x) return 0;
           if (f(n) == x) return 1;
```

Figura 3: Função característica do domínio totalmente enumerável por `f` e `g`.

Note-se que, mesmo sendo X decidível, esta implementação da função característica não introduz limites sobre o número de ciclos; tem-se a certeza que o programa termina, mas pode demorar 1 milhão de anos!. Portanto, a função característica na figura 3 não é, necessariamente, computável.

Este facto sugere um conceito mais forte do que a simples decidibilidade. Assim,

O domínio X diz-se **d-enumerável** quando existe uma enumeração f e uma função computável $d\colon X \to \mathbb{N}$, designada por **profundidade** (**"depth"**) de X, tal que

$$(x \in X) \iff (\exists\, n < d(x))\ [\, f(n) = x\,] \tag{6}$$

Um domínio d-enumerável é decidível[7] e o programa seguinte implementa a sua função característica.

```
in (x) ==>
     for (n=0 ; ; n++)
           if (n == d(x)) return 0;
           if (f(n) == x) return 1;
```

Figura 4: Função característica de um domínio d-enumerável.

Esta implementação da função característica $\xi(x)$ termina com um número de ciclos que não excede $d(x)$. Isto permite inferir um limite superior para a complexidade computacional de ξ (que entra em conta com a complexidade das computações $d(x)$ e $f(n)$) mas este facto pode não ser suficiente para garantir que seja computável.

[7] Basta escolher o predicado $\sigma(n,x) \equiv (d(x) = n) \wedge (f(n) \neq x)$ para determinar o seu complemento.

Em Criptografia, a complexidade computacional é importante mas é mais importante ter uma medida do grau de incerteza que temos sobre a obtenção de um resultado. Por isso convencionamos chamar computáveis às funções que não introduzem incerteza na expectativa de se obter um resultado.

Nesta perspectiva, abstrairmos-nos completamente em relação à contribuição das computações $d(x)$, $f(n)$ ou qualquer outra computação intermédia, e concentramos toda a incerteza nos limites $d(x)$.

Nomeadamente interessa-nos encontrar a ordem de d em função do comprimento de x. Isto é, interessa-nos uma função p tal que $(\forall_\infty x)\ d(x) \leq p(|x|)$. Se p for suficientemente limitada (por exemplo, polinomial de baixo grau), então o cálculo de $\xi(x)$ não introduz incerteza significativa quanto à expectativa de se obter um resultado. Neste caso fará sentido considerar o domínio X computável.

□

Passando agora para domínios que não são necessariamente sub-conjuntos de $\mathbb{N}$. Um domínio X é **decidível** (resp., **totalmente enumerável**) se existir uma bijecção computável entre X e um sub-domínio decidível (resp., ou totalmente enumerável) de $\mathbb{N}$.

O domínio X é **computacionalmente ϖ-equivalente** se existir uma bijecção computável entre X e $\mathbb{N}$.

O domínio $\mathbb{B}^*$ é computacionalmente ϖ-equivalente

Para validar esta afirmação, usamos começar por uma função auxiliar **bin**: $\mathbb{N} \longrightarrow \mathbb{B}^*$ que associa a cada inteiro n a sua menor representação binária.

$$\textbf{bin}(0) \;=\; \varepsilon \quad , \quad \textbf{bin}(n) = (n \mod 2)\,\textbf{bin}(n/2) \qquad \text{para} \quad n > 0 \tag{7}$$

Porque a *string* $\textbf{bin}(n)$ é a menor representação binária de n, termina sempre no dígito 1; por isso a função **bin** não é sobrejectiva já que nunca produz um resultado terminado em 0.

É, porém, injectiva; por isso, para definir uma função simultaneamente injectiva e sobrejectiva basta eliminar o último bit da representação binária de $n+1$. Assim a enumeração **bs**: $\mathbb{N} \longrightarrow \mathbb{B}^*$ das bit-strings define-se

$$\textbf{bs}(n) = s \restriction k \qquad \text{send} \quad s = \textbf{bin}(n+1) \quad \text{e} \quad k = |s| - 1 \tag{8}$$

Por exemplo, $\textbf{bs}(0) = \textbf{bin}(1) \restriction 0 = \text{"1"} \restriction 0 = \varepsilon$. Do mesmo modo tem-se

$$\textbf{bs}(1) = 0, \quad \textbf{bs}(2) = 1, \quad \textbf{bs}(3) = 00, \quad \textbf{bs}(4) = 10, \quad \textbf{bs}(5) = 01, \quad \textbf{bs}(6) = 11, \quad \textbf{bs}(7) = 000$$

A enumeração **bs** é bijectiva e a sua inversa **ord**: $\mathbb{B}^* \longrightarrow \mathbb{N}$ é a codificação

$$\textbf{ord}(s) \;=\; \textbf{bs}^{-1}(s) \quad = \quad \sum_{i=0}^{|s|-1} s_i \cdot 2^i + 2^{|s|} - 1 \tag{9}$$

Por exemplo,

$$\mathbf{ord}(01) \;=\; (0 \cdot 2^0 + 1 \cdot 2^1) + 2^2 - 1 \;=\; 5$$

O facto de f ser computável e sobrejectiva, assegura que $\mathbb{B}^*$ é um domínio enumerável. O facto de f ser injectiva e tanto f como a sua inversa serem computáveis, assegura que é, além disso, um domínio ϖ-equivalente.

Este isomorfismo entre os domínio $\mathbb{N}$ e $\mathbb{B}^*$ permite induz uma noção de **comprimento**[8] de um número natural n, representado por $|n|$, como o comprimento da *string* $\mathbf{bs}(n)$

Por exemplo, $|0| = 0$, $|5| = 2$, etc. É fácil verificar a igualdade

$$|n| \quad = \quad \lfloor \log_2(n+1) \rfloor \tag{10}$$

sendo $\lfloor\, r \,\rfloor$ o maior inteiro menor que o real r.

O domínio dos conjuntos finitos de naturais $2^{<\mathbb{N}}$ é ϖ-equivalente.

Se $A \subseteq \mathbb{N}$ é finito, a sua codificação, representada por (A), é o natural n dado por $n = \sum_{i \in A} 2^i$.

Inversamente, a enumeração é a função $\mathbf{fs}(n) = \{\, i \leq |s| \;\mid\; s_i = 1 \quad \text{sendo} \quad s = \mathbf{bin}(n) \,\}$.

[8] No capítulo seguinte veremos que esta noção de comprimento está incluída numa noção mais geral de comprimento de n.

Os domínios dos pares de inteiros ou strings *($\mathbb{N} \times \mathbb{N}$, $\mathbb{B}^* \times \mathbb{B}^*$, $\mathbb{N} \times \mathbb{B}$) e os domínios das sequências de inteiros ou* strings *($\mathbb{N}^*$, $(\mathbb{B}^*)^*$) são todos computacionalmente ϖ-equivalentes.*

Comecemos por analisar a equivalência entre $\mathbb{B}^*$ e os pares $\mathbb{B}^* \times \mathbb{B}^*$. Basta estabelecer uma bijecção computável $\mathcal{J} : \mathbb{B}^* \times \mathbb{B}^* \longrightarrow \mathbb{B}^*$ que tenha uma inversa computável. Defina-se

$$\mathcal{J}(s, r) \;=\; 1^{|s|}\, 0\, s\, r \tag{11}$$

A função inversa é construída a partir de duas funções π_1 e π_2 que recuperam, respectivamente, a primeira e a segunda componente. Tem-se

$$\pi_1(u) = (u \lfloor (l+1)) \rceil l \quad , \quad \pi_2(u) = u \lfloor (2\,l+1) \quad \text{sendo} \quad l = \max\{\, k \mid 1^k \leq u \,\} \tag{12}$$

Usando agora a bijecção computável entre $\mathbb{N}$ e $\mathbb{B}^*$, establece-se bijecções entre qualquer par dos domínios $\mathbb{N}$, $\mathbb{B}^*$, $\mathbb{N} \times \mathbb{N}$, $\mathbb{B}^* \times \mathbb{B}^*$ e $\mathbb{N} \times \mathbb{B}^*$.

A construção de pares $\mathcal{J}$ definida em (11) induz uma das várias instanciações possíveis da construção de pares de inteiros:

$$n \| m \quad = \quad \mathbf{ord}(\mathcal{J}(\mathbf{bs}(n), \mathbf{bs}(m))) \tag{13}$$

Assumindo essa construção, pode-se codificar sequências de naturais por uma função $\langle\cdot\rangle : \mathbb{N}^* \longrightarrow \mathbb{N}$ definida

recursivamente da seguinte forma

$$\langle \varepsilon \rangle \;=\; 0 \qquad , \qquad \langle a_0, a_1, \cdots, a_{k-1} \rangle \;=\; a_0 \,\|\, \langle a_1, \cdots, a_{k-1} \rangle \tag{14}$$

É simples verificar que esta codificação determina uma bijecção.

O domínio dos racionais $\mathbb{Q}$ é codificável

Neste contexto os pares de naturais $\mathbb{N} \times \mathbb{N}$ são designados por *fracções*, e representados por $\mathbb{F}$, e o seu elemento genérico é representado por n/m. Pelo que dissemos atrás, o espaço das fracções $\mathbb{F}$ é computacionalmente equivalente a $\mathbb{N}$

Recorde-se que o domínio $\mathbb{Q}$ dos racionais positivos é o espaço quociente definido no espaço das fracções pela relação de equivalência $p/q \sim n/m$ sse $n \cdot q = m \cdot p$.

Considere-se uma função **red** que toma como argumentos uma fracção n/m e produz uma nova fracção p/q dividindo n e m por todos os divisores comuns a ambos e maiores do que 1. Nessas circunstâncias p e q serão primos entre si e tem-se $n/m \sim p/q$.

Tomando **red**(n/m) como o representante da classe de equivalência (racional) que contém n/m, a redução representa a aplicação sobrejectiva canónica **red**: $\mathbb{F} \longrightarrow \mathbb{Q}$ que mapeia fracções em racionais

O seguinte programa usa uma implementação gcd do algoritmo de Euclides para calcular o máximo divisor comum de dois naturais. É razoável assumir que tanto gcd como a função que o programa implementa, são computáveis.

```
red(n,m) ==>
    if (m == 0) return (1,0)
    if (n == 0) return (0,1)
    for (l = gcd(n,m); l > 1; )
             n = n/l ;  m = m/l
    return (n,m)
```

Figura 5: Redução de uma fracção.

A enumeração **rac**: $\mathbb{N} \longrightarrow \mathbb{Q}$ define-se agora como $\mathbf{rac}(n) = \mathbf{red}(\pi_1(n), \pi_2(n))$. Uma quasi-inversa é a função que mapeia a fracção reduzida p/q no inteiro $\mathcal{J}(p, q)$.

As sequências de naturais $\mathbb{N}^{\mathbb{N}}$ não constituem um domínio enumerável

Sopunhamos, por momentos, que o domínio $\mathbb{N}^{\mathbb{N}}$ era enumerável.

Então existiria uma aplicação $\rho\colon \mathbb{N} \longrightarrow \mathbb{N}^{\mathbb{N}}$ que enumeraria todas as funções; isto é, qualquer função f teria associado um código e tal que $(\forall\, n)\, [\rho[e](n) = f(n)]$.

Considere-se a função $f(n) = 1 + \rho[n](n)$ e seja e o seu código. Pela definição de enumeração tem-se, para todo n, $\rho[e](n) = f(n) = 1 + \rho[n](n)$. Em particular, quando $n = e$, teremos $\rho[e](e) = 1 + \rho[e](e)$, o que é impossível.

Mesmo pondo de lado a questão da computabilidade, não existe nenhuma enumeração possível de $\mathbb{N}^{\mathbb{N}}$. Do mesmo modo não existe nenhuma enumeração possível de $2^{\mathbb{N}}$ (conjuntos de naturais) ou de $\mathbb{B}^{\infty}$ (strings infinitas de bits)[9]. Todos estes domínios são classificados numa classe distincta de objectos que não podem estar associados a conjuntos enumeráveis.

□

Voltando aos domínios X que são, no mínimo, semidecidíveis, coloca-se a questão de saber que a relação existe entre as várias noções que apresentámos nesta secção. Numa breve sinopse, recorde-se que X é

semidecidível	se existe τ booleana computável tal que	$(x \in X) \Leftrightarrow (\exists\, n)\,[\tau(x, n) = 1]$
enumerável	se existe f computável tal que	$(x \in X) \Leftrightarrow (\exists\, n)\,[f(n) = x]$
d-enumerável	se existem f, d computáveis tais que	$(x \in X) \Leftrightarrow (\exists\, n < d(x))\,[f(n) = x]$
computável	se existe ξ booleana computável tal que	$(x \in X) \Leftrightarrow (\xi(x) = 1)$

[9] O argumento usado para provar a não-enumeralidade de $\mathbb{N}^{\mathbb{N}}$ designa-se por *diagonalização* e pode ser reproduzido, com ligeiras variantes, em todas estas situações.

decidível	se tanto X como $\mathbb{N} \setminus X$ são semidecidíveis
totalmente enumerável	se tanto X como $\mathbb{N} \setminus X$ são enumeráveis
codificável	se é enumerável e a enumeração tem quasi-inversa computável
ϖ-equivalente	se é enumerável, a enumeração é bijectiva e a inversa é computável

Muitas relações entre noções desta tabela estabelecem-se independentemente do modelo de computabilidade usado. Por exemplo

$$d\text{-enumerável} \Rightarrow \text{enumerável} \wedge \text{decidivel} \quad , \quad \text{decidível} \Rightarrow \text{semidecidível}$$

$$\varpi\text{-equivalente} \Rightarrow \text{codificável} \quad , \quad \text{total. enumerável} \Rightarrow \text{enumerável} \wedge \text{decidível}$$

Outras implicações dependem do modelo de computabilidade usado. Por exemplo, numa interpretação muito lata da noção de "função computável" como significado de "função recursiva", então tem-se

$$\text{enumerável} \Leftrightarrow \text{semidecidível} \quad , \quad d\text{-enumerável} \Leftrightarrow \text{computável}$$

Codificação

Seja $f : \mathbb{N} \to X$ uma enumeração de X. Recordemos que, se f tem uma quasi-inversa computável, então X é codificável. Assim faz sentido definir **função codificação** $h : X \to \mathbb{N}$ como uma função computável que seja quasi-inversa de alguma enumeração de X.

Cada $x \in X$ define um conjunto $f^{-1}(x)$ formado pelos vários códigos n que são mapeados em x; a função de codificação h é apenas uma função computável que escolhe um desses códigos. Uma vez obtido um código $n = h(x)$, a enumeração $f(n)$ permite descodificar n e recuperar o valor x.

Vimos já como $\mathbb{B}^*$ é codificável com a representação bs definida por (8). Dado que esta função é bijectiva e a sua inversa ord $=$ bs^{-1} é computável (e é calculada por (9)) a função de codificação é ord.

Vimos também como pares de sequências de domínios codificáveis são codificáveis. Finalmente vimos que $\mathbb{Q}$ é codificável.

□

Outros exemplos de codificação são específicos de usos concretos. Por exemplo:

Base64 (ver `http://en.wikipedia.org/wiki/Base64`) é uma codificação que resulta da necessidade de transmitir palavras de bits arbitrárias (imagens, sons, documentos complexos) através de canais de informação (por exemplo, *e-mail*) que foram concebidos para transmitir apenas "texto imprimível".

O requisito "texto imprimível" implica sempre um texto formado por um conjunto limitado de caracteres. *Base64*, como o nome indica, usa 64 caracteres[10]. Independentemente da escolha de caracteres pode-se representar cada um deles por um inteiro no intervalo $0..63$ e definir a codificação referindo apenas a esses códigos.

Para descrever, de forma simplificada, o *Base64*, note-se que $64 = 2^6$.

O domínio X é, aqui, o espaço $\mathbb{B}^{24}$ das palavras de 24 bits organizadas em 4 palavras de 6 bits; isto é, cada $x \in X$ tem a forma $x = w_0\, w_1\, w_2\, w_3$, com $w_i \in \mathbb{B}^6$. Claramente, X é enumerado.

A função de enumeração $f : \mathbb{N} \to X$, constrói a palavras $f(n) = w_0\, w_1\, w_2\, w_3$ iterativamente;

$$n_i \leftarrow n \mod 64 \quad ; \quad n \leftarrow (n - n_i)/64 \quad ; \quad w_i \leftarrow \mathbf{bin}_6(n_i) \qquad i = 0..3$$

sendo $\mathbf{bin}_6 : \mathbb{N}_{64} \to \mathbb{B}^6$ a função bijectiva[11] que mapeia inteiros no intervalo $[0..63]$ na sua representação binária. A função de codificação será dada, simplesmente, por

$$h(w_0\, w_1\, w_2\, w_3) = \sum_{i=0}^{3} 64^i \cdot \mathbf{bin}^{-1}(w_i)$$

[10] A versão MIME Base64 usa os caracteres `A..Z`, `a..z`, `0..9` e dois outros caracteres que dependem das versões.

[11] Ver adiante.

□

O exemplo anterior sugere a resposta às questões essenciais: *que razões levam à codificação da informação? porque é que ela é necessária?*.

Para ser possível transmitir ou armazenar informação, de forma a que ela possa ser recuperada de forma segura e eficiente, o meio físico usado impõe normalmente formatos específicos. Daí a necessidade de codificar informação proveniente de um domínio genérico em informação num domínio padrão; normalmente esse domínio é $\mathbb{N}$, algum dos seus sub-domínios ou então domínios de palavras de bits.

Um modelo muito geral do mecanismo de codificação/descodificação é representado na figura seguinte.

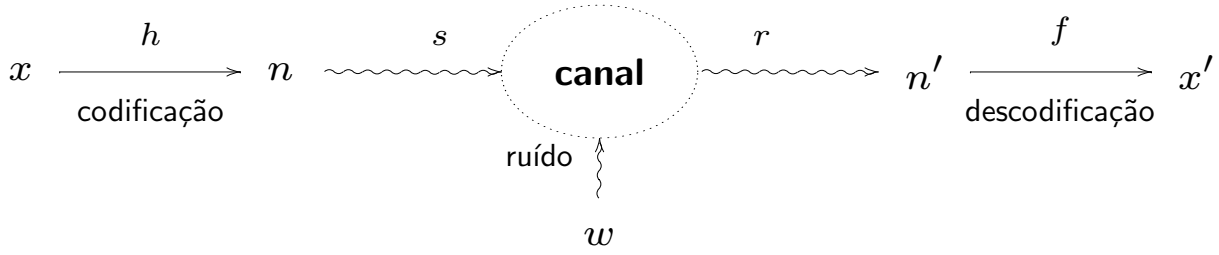


Figura 6: Modelo geral da codificação/descodificação de canal.

Na figura 6 o **canal** é um modelo matemático que representa as transformações de informação num processo *"send/receive"* ou num processo *"store/retrieve"*. Em ambos os casos procura-se que a informação final x' seja tão próximo quanto possível da informação inicial x; o mecanismo de codificação/descodificação existe para satisfazer esse objectivo.

Esse modelo tem de entrar em conta com factores externos não previsíveis, normalmente designados por *ruído* (quando os factores externos não resultam da intervenção activa de agentes hostis) ou *perturbação* ou *ameaça* (quando existe tal intervenção); ambas as situações são aqui representados pela código w.

Funcionalmente, a sequência de transformações é representada pelas seguintes funções,

$$n \leftarrow h(x) \quad , \quad \textbf{canal} \leftarrow s(\textbf{canal}, n) \quad , \quad n' \leftarrow r(\textbf{canal}, w) \quad , \quad x' \leftarrow f(n') \tag{15}$$

Genericamente, qualquer processamento de informação (por exemplo, com técnicas criptográficas) é definido por transformações matemáticos num domínio padrão. Nomeadamente as técnicas criptográficas são, inevitavelmente, expressas em termos de funções de inteiros ou de palavras de *bits*. Por isso, o uso destas técnicas em items de informação provenientes de um domínio genérico, requer uma codificação prévia para o domínio padrão.

Por exemplo, o uso de uma cifra para transmissão de informação sobre um canal sujeito a perturbações hostis, pode ser representada no modelo seguinte.

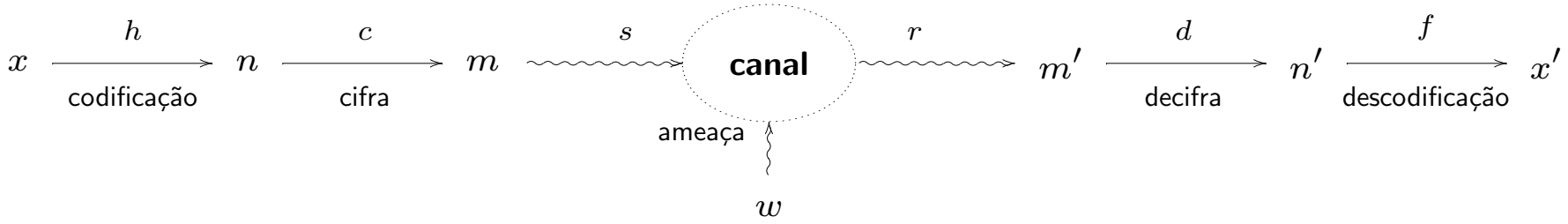


Figura 7: Modelo geral da codificação/cifra de canal.

1.3 Breve Sinopse da Teoria Clássica da Recursividade

Para contextualizar melhor as noções de computabilidade usadas neste curso de Criptografia, convém fazer uma referência (mesmo breve) ás noções clássicas de computabilidade.

Este capítulo deriva essencialmente, e com pequenas adaptações, da obra *Classical Recursion Theory*, *P.G. Odifreddi, North-Holland, Volume 1 (1992) e Volume 2 (1999)*.

Convém começar por alguma terminologia específica da Teoria Clássica da Recursividade que pode, em certas circunstâncias, introduzir noções que diferem das noções designadas de forma semelhante noutros contextos.

1. Na Teoria Clássica da Recursividade a noção de **"função"** designa apenas aplicações de domínio e contradomínio em $\mathbb{N}$. Quaisquer outras aplicações em domínios computacionalmente enumeráveis são implementadas via a respectiva codificação.

   Para representar aplicações de vários argumentos, assume-se que existe uma aplicação bijectiva $\langle\,\rangle$ que codifica sequências finitas de naturais $(a_0, a_1, \ldots, a_{k-1}) \in \mathbb{N}^*$ em naturais $\langle a_0, a_1, \ldots, a_{k-1}\rangle \in \mathbb{N}$.

   Neste contexto, a notação $f(a_0, \ldots, a_{k-1})$ é uma abreviatura de $f(\langle a_0, \langle a_1, \langle \ldots, \langle a_{k-1}\rangle\rangle\rangle\rangle)$.

   O natural 0 codifica a sequência nula (i.e. $\langle\,\rangle = 0$) e, para todo $i < k$, tem-se $\langle a_0, \cdots, a_{k-1}\rangle > a_i$.

Se considerarmos apenas sequências de bits (isto é, se todo $a_i \in 0, 1$) é sempre $\langle s \rangle = \sum_{s_i=1} 2^i$.

Define-se **comprimento** do código n, representado por $|n|$, como o comprimento da sequência que n codifica.
Sejam $n = \langle a_0, \cdots, a_{k-1} \rangle$ e $m = \langle b_0, \cdots, b_{l-1} \rangle$ dois códigos. A **concatenação** $n\, m$ é o código da sequência de comprimento $|n| + |m|$ que se obtém concatenando as sequências que n e m codificam; isto é

$$n\, m \quad = \quad \langle a_0, \cdots, a_{k-1}, b_0, \cdots b_{l-1} \rangle$$

Escreve-se $n \sqsubseteq m$ quando existe p tal que $m = n\, p$; diz-se que n é **prefixo** de m.
É bijectiva a codificação de pares $\| : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N} \setminus \{0\}$, definida por $n \| m = \langle n \rangle\, m$.
Existem *projecções* sobrejectivas $\pi_1, \pi_2 \colon \mathbb{N} \setminus \{0\} \longrightarrow \mathbb{N}$ tais que $\pi_1(a \| b) = a$ e $\pi_2(a \| b) = b$.

2. A designação **relação** aplica-se a qualquer função total (definida em todo o domínio $\mathbb{N}$) cuja imagem tenha apenas dois valores distintos. Sem perda de generalidade assume-se que a imagem é sempre $\{0, 1\}$.

   A designação de **"conjuntos"** aplica-se apenas a sub-domínios de $\mathbb{N}$.
   Cada conjunto X determina-se através de uma relação ξ que verifica $(x \in X) \Leftrightarrow \xi(x) = 1$; essa relação designa-se por **relação característica** de X.
   Por exemplo, as relações constantes 0 e 1 determinam, respectivamente, o conjunto vazio $\emptyset$ e o domínio total $\mathbb{N}$.

3. Colecções de funções (relações, conjuntos, etc..) são designadas por **classes**. A designação **objecto** engloba as noções de número natural (*objecto de ordem* 0), funções e conjuntos (*objectos de 1ª ordem*) e classes (*objectos de 2ª ordem*).

   São também objectos de ordem 0, todos os elementos de domínios isomórficos com $\mathbb{N}$ (sequências de *bits*, sequências ou tuplos tuplos de outros objectos de ordem 0, etc.).

   O domínio de todos os objectos de ordem 0 é representado por ϖ. A classe de todos os objectos de 1ª ordem é representada por 2^{ϖ} (equivalentemente ϖ^{ϖ}). Representa-se por $\varpi^n, \varpi^{<\varpi}$ os domínios de todos os n-tuplos e todas as sequências finitas de objectos de ordem 0.

4. Uma aplicação onde um dos argumentos ou o resultado são de 1ª ordem, designa-se por **funcional**.

   Por exemplo, a relação $\in$ é uma funcional; de facto pode-se ver $\in : \varpi \times 2^{\varpi} \to \varpi$ como a aplicação que toma, por argumentos, um número n e um conjunto A e dá um resultado 1 ou 0 consoante $(n \in A)$ se verifica ou não.

   Outro exemplo: cada função total f determina $f' : \varpi \to \varpi^{\varpi}$ definida por $f' : y \mapsto \{x \mapsto f(x, y)\}$.

A Teoria Clássica da Recursividade lida, essencialmente, com *funções parciais*; isto é, funções que só são definidas num sub-domínio de $\mathbb{N}$. Para lidar com a noção de parcialidade é necessário, também, alguma terminologia específica das funções.

1. $f(x) \simeq y$ é um **predicado** que indica que f é definida 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.
$f(x)\downarrow$ é predicato que é válido se e só se f é definida em x; $f(x)\uparrow$ é a sua negação.
$\mathrm{dom}(f) = \{\, x \mid f(x)\downarrow \,\}$ é o **domínio** de f.
$\mathrm{rng}(f) = \{\, y \mid (\exists x)\,[f(x) \simeq y] \,\}$ é a **imagem** de f.
$\mathcal{F}\, f$ é equivalente a $(\forall_\infty x)[f(x)\uparrow]$; ou seja, o domínio de f tem um número finito de elementos.

2. Funções parciais admitem uma ordem parcial $\lesssim$ definida por

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

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

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

$$f \;\simeq\; g \quad \text{sse} \quad (f \lesssim g) \;\wedge\; (g \lesssim f)$$

3. Funções **totais** (também designadas **sequências**) são funções que têm todo $\mathbb{N}$ como domínio; i.e. funções f que verificam $\forall x \cdot f(x)\downarrow$.
Seja f uma função total.
A **truncatura** de f é definida por $f \restriction k = \langle f(0), f(1), \cdots, f(k-1)\rangle$.

A **história** de f é a função $\downarrow f : k \mapsto f \restriction k$.
Escreve-se $y \sqsubseteq f$, e diz-se que y é um **prefixo** de f, quando $(\exists\, k)\, [\, f \restriction k \simeq y\,]$.

4. A classe $\uparrow y \;=\; \{\, f \;\mid\; y \sqsubseteq f \,\}$ designa-se por *upper cut* de y.
Uma classe $\mathcal{O} \subseteq \varpi^{\varpi}$ é **aberta** quando existe uma função α tal que

$$(f \in \mathcal{O}) \quad \Leftrightarrow \quad (\exists\, k)\, [\alpha(f \restriction k) \neq 0]$$

Uma classe $\mathcal{C} \subseteq \varpi^{\varpi}$ é **fechada** quando existe uma função β tal que

$$(f \in \mathcal{C}) \quad \Leftrightarrow \quad (\forall\, k)\, [\beta(f \restriction k) = 0]$$

Pode-se provar que todo o *upper cut* é uma classe aberta e, inversamente, toda a classe aberta é uma união contável de *upper cuts*. Finalmente, toda a classe fechada é uma intersecção contável de classes abertas.

5. Duas funções totais f e g são **separáveis**, e escreve-se $f \# g$, quando $(\exists\, x)\, [f(x) \neq g(x)]$.
Se as funções f, g são separáveis, a sua **distância** é $d(f,g) = 2^{-s}$, sendo $s = \min \{\, x \;\mid\; f(x) \neq g(x) \,\}$.
Se não forem separáveis, define-se $d(f,g) = 0$.
A distância $d(f,g)$ é uma **ultrametrica** na classe das funções totais; isto é, para todos f, g, h verifica $d(f,g) = d(g,f)$, $d(f,g) = 0 \;\Leftrightarrow\; f \simeq g$ e $d(f,g) \leq \max\,(d(f,h), d(h,g))$.

6. Recorde-se que $x \sqsubseteq y$ quando x codifica uma sub-sequência da sequência cdificada por y.
   Uma função parcial f é **livre de prefixos** se se verifica

$$f(x)\downarrow \wedge f(y)\downarrow \wedge x \sqsubseteq y \quad \Rightarrow \quad (x = y)$$

   Isto é, dados dois elementos distintos do domínio de f, nenhum deles pode ser prefixo do outro.

7. Cada função f determina uma **indexação** de funções $\{f_x\}_{x\in\varpi}$.
   A notação f_x representa a função $y \mapsto f(x, y)$. Adicionalmente, a notação $f_{x,y}$ representa $(f_x)_y$; isto é a função $z \mapsto f(x, y, z)$. Esta notação pode ser estendida para qualquer nível de índices.

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

$$r(x, y) \simeq 1 \quad \text{sse} \quad f(x) \simeq y \tag{16}$$

   Dado que cada relação determina um conjunto por compreensão, cada função f determina também um conjunto através do seu grafo: $\{ z \mid f(\pi_1(z)) \simeq \pi_2(z) \}$. Portanto, neste curso, a designação "grafo" refere-se (dependendo do contexto) quer à relação em (16) como o conjunto que lhe está associado.
9. A construção do grafo é uma construção que permite representar qualquer função f por uma relação r.
   Inversamente é possível construir uma função recursiva f, a partir de qualquer relação r, que verifique

$$f(x)\downarrow \quad \Longleftrightarrow \quad (\exists y)\,[\, r(x, y) = 1 \,] \tag{17}$$

Esta construção é designada por **valor mínimo** ou **recursividade-μ** e a função escreve-se $f(x) = \mu y \cdot r(x,y)$. Intuitivamente $f(x) = \mu y \cdot r(x,y)$ procura o menor valor de y que verifica $r(x,y) = 1$; a função f assim construída tem r como grafo.

10. A **recursividade-μ** aplica-se a qualquer função parcial h. Assim $(\mu y \cdot h(x,y))$ é a função parcial que verifica

$$\mu\, y \cdot h(x,y) \simeq z \iff (\forall y < z)\,[h(x,y) \simeq 0] \wedge h(x,z)\!\downarrow \wedge\, (h(x,z) > 0) \tag{18}$$

$(\mu y \cdot h(x,y))$ procura o menor valor z tal que $h(x,y)$ está defindo em todo $y \leq z$, e $h(x,z)$ não é zero.

11. Define-se **recursividade-μ limitada** associando h à função $\mu\, y < l \cdot h(x,y)$ dada por

$$(\mu\, y < l \cdot h(x,y)) = (\mu\, y \cdot h'(l,x,y)) \qquad \text{sendo} \quad h'(l,x,y) \simeq \begin{cases} h(x,y) & \text{se } y < l \\ 0 & \text{se } y \geq l \end{cases} \tag{19}$$

Com $(\mu\, y < l \cdot h(x,y))$ procura-se o menor valor $y < l$ tal que $h(x,z)$ esteja definida para todos os $z \leq y$ e que verifique $h(x,y) > 0$.

12. As funções constante 0, a função sucessor $\mathcal{S} : x \mapsto x+1$, as projecções π_1, π_2 e a codificação de pares $\|$, formam a colecção das **funções primitivas**.

13. Uma classe de funções $\mathcal{C}$ é
    (i) **enumerável** quando existe uma função $\varphi \in \mathcal{C}$ tal que para todo $f \in \mathcal{C}$ existe um x tal que $f \simeq \varphi_x$.
    (ii) **parametrizável** quando existe uma função total p (a "parametrização") tal que, para todos $f \in \mathcal{C}$ e x, y, z , tem-se $f_x \in \mathcal{C}$ e $f_{x,y}(z) \simeq f_{p(x,y)}(z)$.
    (iii) **fechada por composição** quando $f, g \in \mathcal{C}$ implica que $x \mapsto f(g(x))$ também pertence a $\mathcal{C}$,
    (iv) **fechada por recursividade primitiva** quando, dados $f, g \in \mathcal{C}$, pertence a $\mathcal{C}$ a função h que verifica

$$h(0, x) \simeq g(x) \quad , \quad h(\mathcal{S}(y), x) \simeq f(h(y, x), y, x)$$

    (v) **fechada por recursividade-μ limitada** quando, para todo $f \in \mathcal{C}$, existe um $l > 0$ tal que a função $x \mapsto \mu\, y < l \cdot f(x, y)$ pertence a $\mathcal{C}$.
    (vi) **fechada por recursividade-μ ilimitada** quando $x \mapsto \mu\, y \cdot f(x, y)$ pertence a $\mathcal{C}$ para todo $f \in \mathcal{C}$.

14. A classe $\mathcal{R}_p$ das funções **recursivas primitivas** é a menor classe de funções totais que contém as funções primitivas e é fechada por composição e recursividade primitiva.

    A classe $\mathcal{R}$ das funções **parciais recursivas** é a menor classe de funções parciais que contém as funções primitivas e é fechada por composição, recursividade primitiva e recursividade-μ ilimitada.
    As **funções recursivas** são as funções totais em $\mathcal{R}$.

    A classe $\mathcal{R}^f$ das funções **parciais recursivas com oráculo** f, é a menor classe de funções parciais que contém f, as funções primitivas e é fechada por composição, recursividade primitiva e recursividade-μ ilimitada.

15. Um conjunto A diz-se **enumerável** quando é o conjunto vazio ou então é a imagem de uma função total f.
Isto é, verifica $x \in A \Leftrightarrow (\exists\, y)\,[f(y) \simeq x]$.
Um conjunto A é **recursivamente enumerável** se é vazio ou é enumerável por uma função recursiva f.
O conjunto diz-se **recursivo** se tanto A como o seu complemento $\mathbb{N} \setminus A$ são recursivamente enumeráveis.
Um conjunto recursivamente enumerável e não-recursivo A é **simples** se o seu complemento $\mathbb{N} \setminus A$ é infinito e não contém nenhum sub-conjunto infinito que seja recursivamente enumerado.

16. Alguns resultados importantes:
    (i) Um conjunto A é recursivamente enumerável sse for o domínio de uma função parcial recursiva.
    (ii) Um conjunto A é recursivamente enumerável sse a sua função característica ξ_A é recursiva.
    (iii) Um conjunto não vazio A é recursivo se admitir uma enumeração f recursiva e não-decrescente.
    (iv) A é recursivamente enumerável (resp. recursivo) sse o grafo da sua relação característica for recursivamente enumerável (resp. recursivo).

17. Dado que a classe das funções parciais recursivas é definida indutivamente a partir de um número finito de constantes e um número finito de regras, é sempre possível enumerar a classe das funções parciais recursivas.

4 Teorema (Enumeração das Funções Parcias Recursivas)
A classe $\mathcal{R}$ das funções parciais recursivas é enumerável e parametrizável. Adicionalmente, se φ e ψ são duas enumerações distintas de $\mathcal{R}$, existe uma bijecção recursiva s tal que $\varphi_x \simeq \psi_{s(x)}$, para todo x.

No contexto das funções parciais recursivas, quando $f = \varphi_x$ diz-se que x é o **programa** de f para o **interpretador** φ. O teorema da enumeração permite-nos olhar para φ como um "interpretador universal"[12] que recebe o programa x e um argumento y e calcula $f(y)$.

18. Uma enumeração φ é uma função parcial recursiva e, por isso, pode ser construída por uma sequência arbitrária de composições, aplicações da composição, recursividade primitiva e recursividade-μ. O seguinte teorema mostra que existe uma forma normalizada de construir φ usando a recursividade-μ apenas uma vez.

5 TEOREMA (NORMALIZAÇÃO DAS FUNÇÕES PARCIAIS RECURSIVAS)
Se φ é uma enumeração das funções parciais recursivas, existem duas funções recursivas primitivas ρ, τ tais que, para todo x,

$$\varphi(x)\downarrow \;\Leftrightarrow\; (\exists\, y)\,[\tau(x,y) > 0] \qquad , \qquad \varphi(x)\downarrow \;\Rightarrow\; \rho(\mu\, y \cdot \tau(x,y)) \simeq \varphi(x) \tag{20}$$

19. A normalização das funções parciais recursivas sugere algumas extensões. Assim, as aproximações das funções parciais recursivas é normalizável. Assim, define-se a aproximação $\varphi'(l,x) \lesssim \varphi(x)$ através do seu domínio

$$\varphi'(l,x)\downarrow \;\Leftrightarrow\; (\exists\, y < l)\,[\tau(x,y) > 0] \tag{21}$$

[12] Os programadores de linguagens funcionais reconhecem em φ a função `eval` da linguagem Lisp, ou a função `$` da linguagem Haskell.

20. Seja f uma função total. A classe $\mathcal{R}^f$ das funções parciais recursivas que recorrem ao oráculo f é enumerável e parametrizável. Qualquer enumeração ψ^f destas classe é normalizável por duas funções recursivas primitivas ρ e τ, tais que

$$\psi^f(x)\downarrow \;\Leftrightarrow\; (\exists\, y)\; [\tau(x, y, f\restriction y) > 0] \qquad , \qquad \psi^f(x)\downarrow \;\Rightarrow\; \rho\,(\mu\, y \cdot \tau(x, y, f\restriction y)) \simeq \psi^f(x) \qquad (22)$$

A diferença essencial, em relação o teorema 5, está no facto de τ ter, como argumento extra, a história $f\restriction y$.

21. **Reducibilidade de Turing**

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

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

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

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

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

6 PROPOSIÇÃO

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

Seja φ uma enumeração universal das funções parciais recursivas. Seja $\mathcal{K} = \{\, x \mid \varphi(x,x)\downarrow \,\}$. Então

7 Teorema (Post)
$\mathcal{K}$ é reecursivamente enumerável e não é recursivo. Se A é um conjunto recursivamente enumerável então $A \leqslant_T \mathcal{K}$.

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

(i) o grau representado por $\mathbf{0}$ que é formado exclusivamente pelas funções recursivas (equivalentemente, pelos conjuntos recursivos). A proposição 6 diz-nos que qualquer grau $\boldsymbol{\alpha}$ contém $\mathbf{0}$.

(ii) o grau representado por $\mathbf{0'}$ que é o menor grau que contém o conjunto $\mathcal{K}$. Pelo teorema 7 qualquer grau que contenha um conjunto recursivo está contido em $\mathbf{0'}$.

22. A hierarquia de ordem 0, designada **hierarquia aritmética**, define duas sequência de classes $\left\{\Sigma_n^0\right\}$ e $\left\{\Pi_n^0\right\}$ definidas indutivamente pelas seguintes regras

(i) $\Sigma_0^0 \equiv \Pi_0^0 \equiv$ classe das funções recursivas,

(ii) Se $f \in \Sigma_n^0$, então:
- Qualquer função g que verifica $g(x)\downarrow \Leftrightarrow (\exists\, y)\,[f(x,y)\downarrow]$ também pertence a Σ_n^0.
- Qualquer função g que verifica $g(x)\downarrow \Leftrightarrow (\forall\, y)\,[f(x,y)\downarrow]$ pertence a Π_{n+1}^0.

(iii) Se $f \in \Pi_n^0$ então toda a função g que verifique $g(x)\uparrow \Leftrightarrow f(x)\downarrow$, pertence a Σ_n^0.

É simples verificar que Σ_1^0 coincide com a classe $\mathcal{R}$ das funções parciais recursivas.

Transpondo funções para os seus domínios, constrói-se uma hierarquia de conjuntos representada pelos mesmos símbolos. Isto é, $A \in \Sigma_n^0$ (resp. Π_n^0) quando é o domínio de uma função em Σ_n^0 (resp. Π_n^0).
Assim Σ_0^0 denota a classe dos conjuntos recursivos enquanto que Σ_1^0 denota a classe dos conjuntos recursivamente enumeráveis.

23. A hierarquia de ordem 1, designada **hierarquia analítica** é semelhante à hierarquia aritmética mas permite quantificação sobre funções.
Por exemplo, Π_1^1 é a classe dos conjuntos A tais que $(x \in A) \Leftrightarrow (\forall\, f)\,(\exists\, y)\, h(x, \bar{f}(y))$, para alguma função recursiva h.
Do mesmo modo, Σ_1^1 é a classe dos conjuntos A tais que $(x \in A) \Leftrightarrow (\exists\, f)\,(\forall\, y)\, h(x, \bar{f}(y))$, para alguma função recursiva h.

1.4 Computabilidade e Máquinas de Turing

O modelo clássico da computabilidade exprime-se em termos de **Máquinas de Turing**. Diferentes classes de máquinas de Turing cracterizam diferentes noções de computabilidade:

Máquinas de Turing Determinístas (DTM´s)

Um máquina de Turing determinística $M = (\mathcal{T}, \mathcal{Q}, \delta)$ é caracterizadas por uma *"memória longa" ("tape")* $\mathcal{T}$ (sequência duplamente infinita de células contendo bits ou marcas que representam "conteúdo"), uma *"posição"* (um inteiro positivo ou negativo), um conjunto de *"estados"* $\mathcal{Q}$ (um dos quais classificado como "inicial" e outro classificado como "final") e uma função ("tabela") de *"transição"* δ.

O triplo *estado*×*posição*×*conteúdo* designa-se por **configuração**. A transição δ recebe como argumento o estado, a posição e o conteúdo da célula selecionada pela posição e, como resultado, produz um novo estado, uma nova posição e altera o conteúdo apenas na célula seleccionada pela nova posição.

Um *passo* de máquina é um par de configurações (α, β) em que $\beta = \delta(\alpha)$. Uma configuração é *inicial* (ou *final*) se contém o estado inicial (ou final).

8 NOÇÃO

Uma **execução ("run")** *é uma sequência finita de configurações* $(\alpha_0, \cdots, \alpha_{n-1})$, *em que* (α_{k-1}, α_k) *é um passo de máquina, para todo* $0 < k < n$, *e em que* α_0 *é uma configuração inicial e* α_{n-1} *uma configuração final.*

Uma execução de M em x é uma execução cuja configuração inicial tem conteúdo x.

Uma máquina de Turing M **para** *em x, e escreve-se $M(x)\downarrow$, se existe uma execução de M em x.*

9 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 existe uma execução de M em x de comprimento limitado a $T(|x|)$, em que cada configuração não ocupa mais do que $S(|x|)$ células, e a configuração final tem 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 cada configuração poder transitar para um número finito de configurações. Desta forma, δ é uma relação mas não, necessariamente, uma função.

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".

10 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.

11 TEOREMA
Seja $\mathcal{C} = \{M_n\}$ uma enumeração de máquinas de Turing e seja $\{f_n\}$ a enumeração das funções parciais que estas máquinas computam: $M_n \simeq f_n$. 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.*

Uma máquina de Turing M diz-se **livre de prefixos** se o seu domínio $\{ x \mid M(x)\downarrow \}$ é uma linguagem livre de prefixos; isto é, $M(x)\downarrow \wedge M(y)\downarrow \wedge x \leq y \Rightarrow x = y$.

12 Teorema
Existe uma máquina universal livre de prefixos $\mathcal{U}$ *tal que, para qualquer outra máquina de Turing livre de prefixos* M*, existe um* $p \in \varpi$ *tal que* $M(x) \simeq y \Leftrightarrow \mathcal{U}(p\,x) \simeq y$.

O elemento p designa-se por **programa** de M e o seu comprimento $|p|$ é a *constante de codificação* de M.

Máquinas de Turing livres de prefixos têm uma importância fundamental na descrição da aleatoriedade de conjuntos, sequências ou funções. Além disso codificam quaisquer outras máquinas de Turing. De facto, se pensarmos na enumeração $\mathcal{C}$ do teorema 11, constrói-se uma máquina universal livre de prefixos fazendo $\mathcal{U}(1^n\,0\,x) \simeq M_n(x)$.

Na secção 6 analisaremos as consequências que derivam desta construção básica.

□

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.

13 Teorema (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.

No entanto, está dentro e é central a este curso as noções de **probabilidade** de sucesso da execução de programas em máquina de Truring e a noção de **incerteza**, como grau de imprevisibilidade de eventos.

Para responder à nossa questão inicial *quais são as funções que devemos considerar "computáveis"?* recorremos

à análise já referida anteriormente (ver pag. 2). Neste curso vamos seguir a abordagem normalmente assumida em Criptografia e, a menos que seja explicitamente estipulado em contrário, vamos convencionar uma de três abordagens:

Orientada à Complexidade
São **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.

Orientada à probabilidade
São **computáveis** as funções em qua cada execução numa qualquer máquina de Turing que a computa, tem elevada probabilidade (dirigida pela distribuição de probabilidade dos seus "inputs") de satisfazer determinados "critérios de sanidade".

Orientada à Incerteza
São **computáveis** as funções que não introduzem incerteza; isto é, funções onde o grau de incerteza no argumento da função não é amplificado no resultado.

1.5 Domínios de Baire

Nesta secção vamos introduzir novos domínios formados, normalmente, por objectos de 1^{a} ordem como funções, conjuntos, relações, sequências, etc. Vamos começar, no entanto, por um domínio formado por objectos de ordem 0.

$\mathbb{L}$, os **racionais de Lebesgue**, é o domínio das somas finitas de fracções da forma 2^{-i}, com $i > 0$.

Claramente, cada racional de Lebesgue q, está contido no intervalo $[0, 1)$; isto é, $0 \leq q < 1$. Adicionalmente cada string $s \in \mathbb{B}^*$ determina um $q \in \mathbb{L}$ dado por $q = \sum_{s_i=1} 2^{-i-1}$.

Inspirado na notação usada para as mantissas dos números reais, o racional de Lebesgue determinado pela string s é representado como $0.s$. Por exemplo

$$0.0101 \quad = \quad 2^{-2} + 2^{-4} \quad = \quad 5/16$$

É claro que a aplicação $s \mapsto 0.s$ não é injectiva mas é sobrejectiva. De facto as sequências $s\,0^{<\varpi}$, que se obtêm concatenando s com qualquer número finito de zeros, determinam exactamente o mesmo racional $0.s$ que s determina. No entanto todo o racional de Lebesgue tem a forma $0.s$, para algum s.

Pela definição, se for $s = r\,u$, então $0.s = 0.r + 2^{-|r|} \cdot 0.u$. Como consequência $r \leq s \;\Rightarrow\; 0.r \leq 0.s$.

Este exemplo ilustra também como nem todos os racionais $q < 1$ são racionais de Lebesgue. Considere-se por exemplo o racional $1/3$. Uma mantissa binária "equivalente" a $1/3$ teria de ser infinita. De facto tem-se

$$1/3 \quad = \quad 0.01010101\cdots \quad = \quad \sum_{i>0} 2^{-2\,i}$$

Considere-se a função parcial recursiva definida sobre racionais $q < 1$, da seguinte forma

$$\mathbf{m}(q) \quad = \quad \begin{cases} \varepsilon & \text{se} \quad q = 0 \\ 0\,\mathbf{m}(2\,q) & \text{se} \quad 0 < q < 1/2 \\ 1\,\mathbf{m}(2\,q-1) & \text{se} \quad q \geq 1/2 \end{cases} \tag{23}$$

Verifica-se $\mathbf{m}(q)\!\downarrow \Leftrightarrow (q \in \mathbb{L})$; isto é, a função é definida se e só o argumento q é um racional de Lebesgue. Adicionalmente verifica $\mathbf{m}(q) \simeq s \Leftrightarrow q = 0.s$.

Considere-se agora uma modificação da função "mantissa" introduzindo um "parâmetro de erro" n.

$$\mathbf{m}(n,q) \quad = \quad \begin{cases} \varepsilon & \text{se} \quad q \leq 2^{-n} \\ 0\,\mathbf{m}(2\,q) & \text{se} \quad 2^{-n} < q < 1/2 \\ 1\,\mathbf{m}(2\,q-1) & \text{se} \quad q \geq 1/2 \end{cases} \tag{24}$$

Agora, para todo n e todo o racional $q < 1$, $\mathbf{m}(n, q)$ converge sempre. Adicionalmente, se for $\mathbf{m}(n, q) \simeq s_n$, então tem-se sempre $|s_n| \leq n$ e $0.s_n \leq q < 0.s_n\,1$. Vê-se também que $s_n \leq s_{n+1}$, para todo n.

Se $n \in \mathbb{N}$, a notação $0.n$ denota o racional de Lebesgue determinado pela representação binária de n; isto é $0.n \equiv 0.s$ para qualquer s tal que $n = \langle s \rangle$. Por exemplo $0.5 \equiv 0.101 = 5/8$.

Facilmente se verifica que a função $n \mapsto 0.n$ é uma bijecção e, por isso, $\mathbb{L}$ é ϖ-equivalente.

□

Considere-se a colecção $\mathcal{N} = \varpi^{\varpi}$ das funções de naturais para naturais ou, genericamente, de qualquer ϖ-equivalente domínio para outro ϖ-equivalente domínio. Vimos na secção 2 que os elementos desta colecção são objectos de 1ª ordem e que a colecção deve ser designada como *classe*. Mais precisamente $\mathcal{N}$ é designada por **domínio de Baire**.

Vimos também que $\mathcal{N}$ não é enumerável mesmo no sentido mais lato do termo. De facto o domínio de Baire é uma instância de uma colecção mais vasta de domínios que designamos por $\mathcal{N}$-**equivalentes**. Alguns exemplos

Sequências

Seja X um domínio enumerável e σ uma sua enumeração. Qualquer função $f \in \mathcal{N}$ determina uma sequência $\{x_n\}$ definindo $x_n = f(\sigma(n))$. A sequência $\{x_n\}$ é recursiva (ou, computável) quando f é recursiva (ou, computável).

Conjuntos e sequências infinitas de bits

Pensando em ϖ como o paradigma do domínio enumerável ($\mathbb{N}$ e $\mathbb{B}^*$ são as suas instâncias mais importantes) vimos que o domínio dos conjuntos finitos $2^{<\varpi}$ é ϖ-equivalente. Porém a classe de todos os conjuntos, representada por 2^{ϖ}, não é ϖ-equivalente.

Também não pode ser ϖ-equivalente a classe das bit-strings infinitas. Strings de bits infinitas são funções $\alpha \in \mathcal{N}$ cuja imagem é $\{0,1\}$. Assim, cada conjunto $A \subseteq \varpi$ identifica-se naturalmente com uma bit-string infinita α, vendo-a como a relação característica de A. Isto é, $(n \in A) \Leftrightarrow \alpha(n) = 1$.

Esta identificação entre bit-strings infinitas e conjuntos permite transferir muita da notação de strings para conjuntos. Assim, por exemplo, define-se a truncatura de A, $A \restriction k$, como $\{\, n \in A \mid n < k \,\}$.

Bit strings infinitas identificam-se também com funções $s : \mathbb{N} \to \mathbb{B}^*$ que verifiquem $(\forall\, n)\,[s(n) < s(n+1)]$. Como a sequência $\{|s(n)|\}$ é estritamente crescente, para todo k existe n tal que $k < |s(n)|$; define-se então $\alpha(k) = s(n)_k$.

Como cada bit-string infinita (ou conjunto) é uma função, pode ser visto como um elemento de $\mathcal{N}$. Inversamente, dada uma função arbitrária $f \in \mathcal{N}$ é sempre possível associá-la a uma bit-strings infinita α que lhe seja, de alguma forma, equivalente. Essa equivalência estabelece-se através da construção de reais.

Reais

Uma sequência de racionais $\{q_n\}$ que seja monótona não-decrescente e limitada, é uma sequência de Cauchy e, por isso, determina um real $\alpha = \lim_n q_n$. Estes reais designam-se por *left reals*. Um *left real* é *computável* quando a sequência $\{q_n\}$ é computável.

Dado $\beta = \lim_n p_n$ outro *left real*, escreve-se $\beta \geq \alpha$ quando $(\forall\, n)\,(\exists\, m)\,[p_m \geq q_n]$; se existe uma função crescente recursiva s tal que $(\forall\, n)\,[p_{s(n)} \geq q_n]$, a asserção $\beta \geq \alpha$ diz-se *computável*. Tem-se $\beta = \alpha$ quando $\beta \leq \alpha$ e $\alpha \geq \beta$.

No que se segue assuma-se sempre **reais computáveis**; isto é, *left reals* $\alpha = \lim_n \{q_n\}$ determinados por sequências recursivas.

Escreve-se $\alpha \leq_S \beta$, e diz-se que α é **Solovay-redutível** a β, se existem s recursiva e constante c tais que

$$(\forall\, n)\,[\ \alpha - q_{s(n)} \ < \ 2^c\ (\beta - p_n)\ ] \tag{25}$$

Podemos ler esta definição como uma relação entre as "velocidades" de convergência das duas sequências: a convergência para α de $\{q_n\}$ é tão ou mais rápida que a convergência de $\{p_n\}$ para β. A outra questão é a relação entre a reducibilidade de Solovay e a ordem entre reais. Pode-se provar uma extensão da transitividade

14 TEOREMA
Se $\alpha \leq \alpha'$ *é recursiva, então* $\alpha' \leq_S \beta \Rightarrow \alpha \leq_S \beta$ *e* $\beta \leq_S \alpha \Rightarrow \beta \leq_S \alpha'$.

Dois reais α, β são **Solovay-equivalentes**, e escreve-se $\alpha \equiv_S \beta$, quando são mutuamente **Solovay-redutíveis**: $\alpha \equiv_S \beta$ sse $\alpha \leq_S \beta$ e $\beta \leq_S \alpha$. As classes de equivalência definidas pela relação $\equiv_S$ nos reais recursivas chamam-se **graus de Solovay**.

Quando $\alpha \not\leq_S \beta$ e $\beta \not\leq_S \alpha$ os reais α, β dizem-se **Solovay-incomparáveis** e escreve-se $\alpha \mid_S \beta$.

□

Reais ≡ bit-strings infinitas ≡ funções ≡ conjuntos

Seja $\alpha \in 2^{\varpi}$ uma bit-strings infinita e $\{s_n = \alpha \restriction n\}$ a sequência das suas sucessivas truncaturas. Defina-se a sequência $\{q_n = 0.s_n\}$. Uma vez que, para todo n, se verifica $s_n < s_{n+1}$, então teremos $q_n \leq q_{n+1}$; portanto q_n determina um *left real*. Tal real será representado por $0.\alpha$.

Inversamente, se $\lim_n \{p_n\}$ é um real recursivo, existe $\alpha \in 2^{\varpi}$ recursiva tal que $0.\alpha = \lim_n \{p_n\}$.

Justificação Defina-se $t(n)$ como o natural tal que $2^{-t(n)-1} < p_{n+1} - p_n \leq 2^{-t(n)}$. Seja $s_n = \mathbf{m}(t(n), p_n)$. Tem-se $0 \leq p_n - 0.s_n < 2^{-t(n)}$. Claramente que ambas as funções $n \mapsto t(n)$ e $n \mapsto s_n$ são recursivas. Daqui resulta que tem de se verificar $s_n \leq s_{n+1}$ e, por isso, a sequência $\{s_n\}$ determina uma string de bits infinita α.

Se A for um conjunto então $0.A$ designa o *left real* gerado pela sequência de racionais $\{0.A \restriction n\}$. Note-se que $0.A \restriction n = \sum_{k \leq n \wedge k \in A} 2^{-k}$.

Analogamente, dada uma função $f \in \mathcal{N}$, a sequência $\{q_n = 0.f \restriction n\}$ determina um real esquerdo que representamos por $0.f$.

A equivalência entre f, A e α estabelece-se através da igualdade dos reais que determinam; por exemplo, f e α serão equivalentes se $0.f = 0.\alpha$.

Nesta perspectiva todo o real computável se pode escrever na forma $0.A$ sendo A um conjunto computacionalmente decidível. Se, adicionalmente, A for computacionalmente enumerável, então $0.A$ é *fortemente computável*.

Se $0.A$ for fortemente computável, existe uma enumeração d de A que é crescente e recursiva. consequentemente

$$0.A = \sum_{k=0}^{\infty} 2^{-d(k)} \tag{26}$$

Genericamente, se $0.A$ é computável, então prova-se que existe sempre uma função recursiva d que verifica (26).

Um conjunto A é $\mathbb{L}$-**limitado** se a sequência de racionais de Lebesgue $q_n = \sum_{x \in A \wedge |x| \leq n} 2^{-|x|}$ é limitada. Se for, então esta sequência determina um real, designado **medida de Lebesgue** de A

$$\mu(A) = \sum_{x \in A} 2^{-|x|} \tag{27}$$

Designamos por **comprimento de Lebesgue** de A o elemento de $\mathbb{N}+\infty$, representado por $\lambda(A)$, e definido do modo seguinte: se $\mu(A)=0$ então $\lambda(A)=\infty$; se $\mu(A)>0$ então $\lambda(A)$ é o menor de todos os n tais que $\mu(A)>2^{-n}$; ou, equivalentemente, $\sum_{x\in A} 2^{n-|x|}>1$.

□

Linguagens livres de prefixos

Um conjunto L de bit-strings finitas que verifica a condição $(x\in L)\wedge(x<y)\ \Rightarrow\ (y\notin L)$, diz-se **livre de prefixos**. Linguagens livres de prefixos são outra forma de definir reais.

De facto, é possível definir um *left real* $\mu(L)=\sum_{x\in L} 2^{-|x|}\leq 1$, construindo uma sequência de racionais $q_n=\sum_{x\in L\wedge|x|\leq n} 2^{-|x|}$.

Para se verificar que $q_n\leq 1$, para todo n, defina-se a_n como o número de elementos de L de comprimento n.

Será $q_n=\sum_{k=0}^{n} a_k\,2^{-k}$. Por outro lado, como a linguagem é livre de prefixos, o número total 2^n de strings de comprimento n, tem de verificar $2^n\geq a_n+2\,a_{n-1}+2^2\,a_{n-2}+\cdots+2^n\,a_0$. Consequentemente, $q_n\leq 1$.

Operações Básicas

Interpretando $A,B\in 2^{\varpi}$ como sub-conjuntos de $\mathbb{N}$, temos as operações básicas que vêm da teoria dos conjuntos: união $A\cup B$, intersecção $A\cap B$, complemento $\bar{A}$ e diferença $A\setminus B$.

Interpretando os mesmos A, B como bit-strings infinitas temos duas operações:

- $A \cdot B$ é a string cujo bit i é o **and** dos bits A_i e B_i ;
- $A \oplus B$ é a string cujo bit i é o **xor** dos bits A_i e B_i .

Vendo $A, B \in 2^{\varpi}$ como conjuntos de naturais, a *união disjunta* $A \uplus B$ é

$$A \uplus B \quad = \quad \{\, 2\,a \mid a \in A \,\} \cup \{\, 2\,b+1 \mid b \in B \,\}$$

Vendo A e B como linguagens, então $A \uplus B$ é a linguagem

$$A \uplus B \quad = \quad \{\, 0\,a \mid a \in A \,\} \cup \{\, 1\,b \mid b \in B \,\}$$

Com esta notação, temos

15 FACTO

Uma linguagem L é livre de prefixos se e só se é a linguagem vazia $\emptyset$, ou é a linguagem singular $\mathbf{1} = \{\varepsilon\}$ ou então é a união disjunta $L \uplus L'$ de linguagens livres de prefixos.

EXEMPLO 3:

As linguagens singulares $\{0\}$ e $\{1\}$ são, respectivamente, $\mathbf{1} \uplus \emptyset$ e $\emptyset \uplus \mathbf{1}$.

1.6 Máquinas de Turing Livres de Prefixos

Recorde-se que uma máquina de Turing M é livre de prefixos quando o seu domínio é uma linguagem (necessariamente enumerável) que é livre de prefixos; isto é, $M(x)\downarrow \wedge M(y)\downarrow \wedge x \leq y \Rightarrow x = y$.

Recorde-se o teorema 12 que estabelece a universalidade das PFTM´s ("prefix-free Turing machines") afirma que existe uma máquina $\mathcal{U}$ tal que, qualquer máquina M é determinada por um programa p; isto é, verifica-se $\mathcal{U}(p\,x)\downarrow \Leftrightarrow M(x)\downarrow$ e $\mathcal{U}(p\,x) \simeq y \Leftrightarrow M(x) \simeq y$.

No que se segue $\mathcal{U}_p$ designa a PFTM determinada pelo programa p

A universalidade das PFTM´s não significa que $\mathcal{U}$ seja única. De facto existe uma ϖ-enumeração das PFTM´s universais. No entanto elas estão fortemente relacionadas; de facto prova-se que existe sempre uma correspondência bijectiva entre os programas aceites por uma máquina universal e os programas aceites por outra máquina universal.

16 Facto
Se $\mathcal{U}$ e $\mathcal{U}'$ são duas PFTM´s universais, então existe um bijecção recursiva s tal que $\mathcal{U}_p \simeq \mathcal{U}'_{s(p)}$.

Probabilidade de Paragem

Seja $M = \mathcal{U}_p$ uma PFTM; como o domínio de $\mathcal{U}_p$ é livre de prefixos, determina um *left real*

$$\Omega_p \;=\; \mu(\mathrm{dom}(\mathcal{U}_p)) \;=\; \sum_{\mathcal{U}(p\,x)\downarrow} 2^{-|x|} \tag{28}$$

O real Ω_p verifica $\Omega_p \leq 1$ (por ser gerado por uma linguagem livre de prefixos); assim faz sentido vê-lo como uma "probabilidade"; designa-se por **probabilidade de paragem** do programa p.

Nomeadamente o programa vazio determina a probabilidade de paragem da máquina universal, $\mathcal{U}$.

$$\Omega \;=\; \mu(\mathrm{dom}(\mathcal{U})) \;=\; \sum_{\mathcal{U}(x)\downarrow} 2^{-|x|} \tag{29}$$

Este real, designado por **real de Chaitin**, é essencial à modelação da aleatoriedade computacional.

As probabilidade de paragem, assim definidas, estão ligadas a uma máquina universal específica $\mathcal{U}$. Uma vez que a máquina universal $\mathcal{U}$ não é única, deveria, em rigor, ser designada por "probabilidades de paragem relativas a $\mathcal{U}$".

No entanto é simples provar, por simples expansão das definições, que

17 Teorema
Se Ω_p é a probabilidade de paragem da máquina M relativa à máquina universal $\mathcal{U}$, e se Ω'_q é a probabilidade de paragem da mesma máquina M relativa à máquina universal $\mathcal{U}'$, então Ω_p e Ω'_q são Solovay-equivalentes.

Isto significa que é possível abstrair em relação à máquina que define as probabilidades de paragem se se considerar que esses reais estão definidos a menos de uma equivalência de Solovay. Os graus de Solovay assumem, por isso, uma importância essencial: são eles que determinam as probabilidades de paragem das PFTM´s.

O real Ω satisfaz uma propriedade essencial, que está expressa no seguinte teorema

18 Teorema
Todo o real recursiva α é Solovay-redutível a Ω ; isto é, verifica $\alpha \leq_S \Omega$. Adicionalmente, se α verificar $\Omega \leq_S \alpha$, então existe uma PFTM universal $\mathcal{U}'$ de que α é a probabilidade de paragem.

Dito de outro modo

19 Corolário *O grau de Solovay que contém Ω é a classe formada pelas probabilidades de paragem das máquinas de Turing universais livres de prefixos.*

Os reais computáveis nestas circunstâncias (elementos do grau de Solovay que contém Ω) dizem-se Ω-**equivalentes**.

Teorema de Kraft-Chaitin

Acabámos de verificar que cada PFTM determina, via a sua probabilidade de paragem, um real recursiva. A relação é, porém, mais forte; de facto também se verifica que todo o real recursiva é a probabilidade de paragem de uma PFTM.

Para estabelecer-mos esta relação necessitamos de um dos teoremas mais importantes no estudo da computabilidade.

20 TEOREMA (KRAFT-CHAITIN)
Seja d *uma função recursiva que verifica* $\sum_{n\in\varpi} 2^{-d(n)} \leq 1$ *e* f *uma qualquer função recursiva. Então existe uma PFTM* M *e uma sequência recursiva* $\{\alpha(n)\}$ *tal que* $|\alpha(n)| = d(n)$ *e* $M(\alpha(n)) \simeq f(n)$.

EXEMPLO 4: Seja A um domínio decidível verificando $\mu(A) < 1$.

Seja c o menor natural tal que $1 - \mu(A) \geq 2^{1-c}$. Defina-se $d(x) = |x|$, se $x \in A$, e $d(x) = x + c$ se $x \notin A$. Desta forma, temos

$$\sum_x 2^{-d(x)} = \sum_{x\in A} 2^{-|x|} + \sum_{x\notin A} 2^{-c}\, 2^{-x} \leq \mu(A) + 2^{1-c} \leq 1$$

Considere-se a função recursiva característica de A; i.e. $f(x) = 1 \Leftrightarrow (x \in A)$. O teorema de Kraft-Chaitin diz-nos que existe uma PFTM, M e uma enumeração do seu domínio α tal que, para todo x, $M(\alpha(x)) \simeq f(x)$ e $|\alpha(x)| = d(x)$.

Como consequência, $M(y) \simeq 1$ sse $(\exists\, |x| = |y|)\, [\, y = \alpha(x)\,]$ e $M(y) \simeq 0$ sse $(\exists\, |x| < |y|)\, [\, y = \alpha(x))$.

EXEMPLO 5: Alterando o exemplo anterior, suponhamos que A é enumerado, infinito e $\mathbb{L}$-limitado. Seja c o menor inteiro tal que $\mu(A) < 2^c$ e seja $\{x(k)\}$ uma sequência recursiva que enumera A sem repetições. Logo $\sum_k 2^{-|x(k)|-c} < 1$; fazendo $d(k) = |x(k)| + c$, será $\sum_k 2^{-d(k)} < 1$.

Considere-se uma segunda enumeração recursiva sem repetições $\{y(n)\}$ do mesmo conjunto A, e vamos tentar resolver o seguinte problema: dado um qualquer $x(k)$, encontrar o $y(n)$ tal que $x(k) = y(n)$.

Define-se a função recursiva $n(k) = \min\{\, t \mid y(t) = x(k) \,\}$ que determina o índice n que procuramos. Usando o teorema de Kraf-Chaitin é possível construir uma PFTM M e uma sequência recursiva $\{\alpha(k)\}$ tais que $|\alpha(k)| = |x(k)| + c$ e $M(\alpha(k)) \simeq n(k)$.

O ponto interessante nesta conclusão resulta do facto de, num conjunto enumerado, os elementos não são necessariamente conhecidos mas apenas é conhecido o índice que determina esse elemento numa determinada enumeração. Por isso este resultado permite estabelecer, em primeiro lugar, que existe uma PFTM que computa esta "equivalência de índices" e, em segundo lugar, fixa limites ao tamanho dos argumentos reconhecíveis por essa máquina.

□

Os pares de funções (d, f) que verificam as condições do teorema 20 designam-se por **pares de Kraft-Chaitin** (também designados por *conjuntos de Kraft-Chaitin*, ou *KC-sets*). Se a máquina de Turing M for recursiva então o KC-set (d, f) diz-se recursiva.

No que se segue necessitamos apenas de uma versão mais fraca

21 COROLÁRIO *Se d uma função recursiva que verifica $\sum_{n\in\varpi} 2^{-d(n)} \leq 1$, existe uma máquina de Turing livre de prefixos M e uma enumeração α do seu domínio tal que $|\alpha(n)| = d(n)$, para todo n.*

Recorrendo à representação (26) e usando o corolário 21, conclui-se

22 TEOREMA
Um left-real *é computável se e só se é a probabilidade de paragem de uma máquina de Turing livre de prefixos.*

1.7 Geradores, Decisores e Indistinguibilidade Probabilística

Seja G uma PFTM, e $\bar{G}$ a TM que computa o "grafo" de G; isto é, $\bar{G}(y,x) \simeq 1 \Leftrightarrow G(y) \simeq x$.

Apesar de $\bar{G}$ não ser, normalmente, livre de prefixos, todas as indexações $\bar{G}_x$ são livres de prefixos (o domínio de $\bar{G}_x$ é um sub-domínio do domínio de G). Portanto faz sentido definir a probabilidade de paragem $\mu(\mathrm{dom}(\bar{G}_x))$.

Tem-se, portanto, $\mu(\mathrm{dom}(\bar{G}_x)) = \sum_{G(y)\simeq x} 2^{-|y|}$.

Para simplificar a notação representamos a probabilidade de paragem de $\bar{G}_x$ por $\mu[G = x]$ e a probabilidade de paragem de G por $\mu[G]$.

Note-se que os domínios $\mathrm{dom}(\bar{G}_x)$ são disjuntos e que se tem $\mathrm{dom}(G) = \bigcup_x \mathrm{dom}(\bar{G}_x)$. Isto significa que a colecção $\left\{\mathrm{dom}(\bar{G}_x)\right\}_x$ forma uma partição de $\mathrm{dom}(G)$. Consequentemente $\mu[G] = \sum_x \mu[G = x]$.

Genericamente, se A é um conjunto, representamos por $\mu[G \in A]$ a probabilidade $\sum_{x\in A} \mu[G = x]$.

23 NOÇÃO

Um conjunto A é um ***decisor probabilístico*** *para G quando*

$$\sum_{x\in A} 2^{|x|}\, \mu[G = x] < \infty \tag{30}$$

A quantidade em (30) *representa-se por* $\mathcal{D}(A|G)$.

Pode-se ver $\mathcal{D}(A|G)$ como um "integral", estendito ao domínio A, da dimensão de incerteza $2^{|x|}$ usando como medida a probabilidade de G gerar cada x.

Se A for finito este "integral" é, trivialmente, finito. Quando A é infinito, o "integral" só pode ser finito se a função $\lim_{|x|\to\infty} 2^{|x|}\,\mu[G=x]$ tender muito rapidamente para zero.

A notação das probabilidades pode ser modificada para comprimentos. Recorde-se que $\lambda(L)$ é $+\infty$ quando $\mu(L)=0$, e é $\min\{\, n \mid \mu(L) > 2^{-n} \,\}$, quando $\mu(L)>0$. Analogamente define-se $\lambda[G=x]$, $\lambda[G]$, etc.

Usando comprimentos, temos $\mathcal{D}(A|G) \sim \sum_{x\in A} 2^{|x|-\lambda[G=x]}$. Para que $\mathcal{D}(A|G)$ seja finito quando A é infinito, tem de se verificar $(\forall_\infty\, x \in A)\,[\,|x| < \lambda[G=x]\,]$.

Note-se que esta condição é necessária mas não suficiente; por exemplo, se for $\lambda[G=x] = 2\,|x|$, a condição $(\forall\, x \in A)\,[\,|x| < \lambda[G=x]\,]$ verifica-se, mas $\sum_{x\in A} 2^{-|x|}$ pode não ser finito.

Neste contexto a condição (30) vem modificada. Suponhamos que $\lambda[G=x] < \infty$ para todo $x \in A$. Nestas circunstâncias pode-se dizer que A é um decisor probabilístico para G, quando para algum $c>0$,

$$\sum_{x\in A} 2^{|x|-\lambda[G=x]} \;\leq\; \mathcal{D}(A|G) \;\leq\; 2^c$$

Suponhamos que A é infinito e que $\{x(k)\}$ é uma enumeração recursiva sem repetições de A. Seja d a função recursiva definida $d(k) = \lambda[G = x(k)] + c - |x(k)|$. Pode acontecer que $d(k)$ seja computável (ou estimável) mesmo que $x(k)$ não o seja. Se A for um decisor para G, então $\sum_k 2^{-d(k)} \leq 1$.

Usando o teorema de Kraft-Chaitin com o par (d, x), existe uma PFTM M e uma sequência recursiva $\{y(k)\}$ que verifica $|y(k)| = d(k) = \lambda[G = x(k)] + c - |x(k)|$ e também $M(y(k)) \simeq x(k)$.

Esta conclusão pode ser interpretada da seguinte forma: existe uma máquina M que encontra um elemento arbitrário de A usando "inputs" de tamanho limitado a $d(k) = \lambda[G = x(k)] + c - |x(k)|$. Assim $d(k)$ mede a maior ou menor dificuldade de M em encontrar $x(k)$.

Se G gerar $x(k)$ com elevada probabilidade, então o comprimento $\lambda[G = x(k)]$ é baixo, o que significa que $d(k)$ é também baixo. Se $x(k)$ for gerado por G com baixa probabilidade, isso implica que comprimento $\lambda[G = x]$ é elevado; consoante a diferença $\lambda[G = x(k)] - |x(k)|$ temos uma maior ou menor dificuldade em encontrar $x(k)$.

EXEMPLO 6:
Nos exemplos seguintes considere-se sempre um gerador G que produz *outputs* inteiros num domínio D associando a cada um uma probabilidade não-nula; i.e., para todo $x \in D$, tem-se $\lambda[G = x] < \infty$.

1. Vamos supor que G verifica $\lambda[G = x] > 2|x|$. Este gerador "favorece os pequenos *outputs*" dando uma probabilidade muito baixa à ocorrência de *outputs* de grande comprimento. Para qualquer A tem-se $\sum_{x \in A} 2^{|x| - \lambda[G=x]} < \sum_{x \in A} 2^{-|x|} = \mu(A)$. Consequentemente qualquer A que seja $\mathbb{L}$-limitada é um decisor para este gerador G.

2. Vamos supor que $A(n) = A \cap \mathbb{B}^n$ é o conjunto das bit-strings de comprimento n na linguagem A. Cada $A(n)$ é obviamente finito. Temos $\sum_{x \in A(n)} 2^n \, \mu[G = x] = 2^n \, \mu[G \in A(n)]$ que é sempre finito. Por isso qualquer $A(n)$ é, trivialmente, um decisor para um qualquer gerador G. Tem-se $\sum_{x \in A} 2^{|x|} \, \mu[G = x] = \sum_n 2^n \, \mu[G \in A(n)]$. Logo, para um gerador G arbitrário, A será decisor para G se esta "expectativa" $\sum_n 2^n \, \mu[G \in A(n)]$ é finita.

3. Seja A um conjunto (visto como uma bit-string infinita) e $\downarrow A = \{A \restriction k\}_k$ a linguagem formada pelos prefixos de A. Neste caso tem-se $\mathcal{D}(\downarrow A | G) = \sum_k 2^k \, \mu[G = A \restriction k]$; isto significa que as probabilidades de G gerar prefixos sucessivos de A, determina se A é ou não um decisor para G.

A noção de decisor para um gerador permite definir uma forma de comparação de geradores.

24 NOÇÃO
Se M é uma PFTM, escreve-se $G \leq_P M$ quando todo o decisor probabilístico computável para G é também um decisor probabilístico computável para M. Quando, simultaneamente, $G \leq_P M$ e $M \leq_P G$ escreve-se $G =_P M$ e diz-se que G e M são ***probabilisticamente equivalentes****.*

É crucial ter em atenção que a definição da ordem $\leq_P$ e da equivalência $=_P$ está relativizada pela noção de computabilidade que associamos aos decisores A.

A definição de decisor, por vezes, não é suficientemente forte para as necessidades impostas pelas técnicas

criptográficas. Uma noção mais forte considera sequências efectivamente enumeradas de conjuntos decisores e geradores.

Um gerador G define uma enumeração efectivamente recursiva de geradores $\{G_n\}$ dados por $G_n(y) \simeq x$ sse $G(y,n) \simeq x$. Da mesma forma cada conjunto recursivo A determina uma enumeração efectiva de conjuntos $\{A_n\}$ por enumeração das suas funções características $A_n(x) = A(x,n)$. Nestas circunstâncias

25 NOÇÃO
Um conjunto decidível A é um **decisor probabilístico fraco** *para o gerador G quando, para todo n, A_n é um decisor probabilístico para G_n .*

Se A for um decisor para G também é um decisor fraco para G. De facto, da definição de A_n e G_n verifica-se que $\sum_{x\in A_n} 2^{|x|}\,\mu[G_n = x] \leq \sum_{y\in A} 2^{|y|}\,\mu[G = y]$. No entanto, o inverso não é necessariamente verdade: basta que se verifique $\lim_n \mathcal{D}(A_n|G_n) = \infty$ para que A não seja decisor para G.

No que se segue, a menos que algo seja dito em contrário, vamos considerar apenas **geradores monótonos**; isto é, geradores G que verificam $G_n(y) \simeq x$ só se $|x| \geq n$. Mesmo G não seja monótono, é sempre possível construir um gerador monótono G' que gere a mesma informação: basta fazer $G'_n(y) \simeq 1^n\,0\,x$ quando $G_n(y) \simeq x$.

26 NOÇÃO
Dois geradores G e H são **probabilísticamente comparáveis** *quando todo A computável que seja decisor fraco para um deles é decisor fraco para o outro. Se G e H forem p-comparáveis, então*

(i) *Escreve-se* $G \leq_{\mathsf{ind}} H$ *quando, para todo* A *computável, existe um polinómio* $p(n)$ *tal que*

$$(\forall_\infty n)\ \mathcal{D}(A_n|G_n) \ \leq\ \mathcal{D}(A_n|H_n) \ +\ p(n) \tag{31}$$

Equivalentemente, $\mathcal{D}(A_n|G_n) - \mathcal{D}(A_n|H_n) = O(p(n))$.

(ii) *Escreve-se* $G =_{\mathsf{ind}} H$ *e diz-se que* G *e* M *são* **probablisticamente indistinguíveis** *quando* $G \leq_{\mathsf{ind}} H$ *e* $H \leq_{\mathsf{ind}} G$.

Os geradores G *e* H *são* **estritamente distinguíveis**, *e escreve-se* $G \# H$, *se existe um* A *computável que é decisor fraco para um dos geradores e não é decisor fraco para o outro.*

Mais uma vez é necessário ter em atenção que a noção de indistinguibilidade está relativizada à noção de "conjunto computável". Seguindo a tese de Church, um conjunto computável é, no mínimo, recursivo.

Tenha-se também em atenção que a ordem dos quantificadores faz com que a noção de distinguibildade estrita seja mais forte do que não-indistinguibilidade.

Na definição de distinguibilidade estrita chama-se a atenção para o papel dos decisores A. Tem de existir um A que "separe" os dois geradores no sentido em que é decisor fraco para um deles e não é para o outro; nomeadamente G e H nem sequer são comparáveis. Isto significa que tem de existir um A tal que, $(\exists\, n)\,[\,\mathcal{D}(A_n|G_n) = \infty\,]$ e $(\forall\, n)\,[\,\mathcal{D}(A_n|H_n) < \infty\,]$ ou, então, o inverso $(\exists\, n)\,[\,\mathcal{D}(A_n|H_n) = \infty\,]$ e $(\forall\, n)\,[\,\mathcal{D}(A_n|G_n) < \infty\,]$.

27 TEOREMA

G, H são indistinguíveis se e só se, para todo A existe um polinómio $p(n)$ tal que

$$(\forall_\infty n) \quad \sum_{x \in A_n} 2^{|x|} \, |\mu[G_n = x] - \mu[M_n = x]| \;\leq\; p(n) \tag{32}$$

Esboço de prova Representemos por $\Delta_n(G, M, x)$ a quantidade $2^{|x|} \, (\mu[G_n = x] - \mu[M_n = x])$. Prova-se facilmente este resultado decompondo a soma $\sum_{x \in A_n} |\Delta_n(G, M, x)|$ em duas parcelas positivas: uma relativa aos elementos $x \in A_n$ onde $\mu[G_n = x] \geq \mu[M_n = x]$ e outra relativa aos elementos $x \in A_n$ onde $\mu[M_n = x] > \mu[G_n = x]$. Sendo G, M indistinguíveis ambas serão limitadas e a sua soma será limitada.

No sentido inverso, se ocorrer $\sum_{x \in A_n} |\Delta_n(G, M, x)| < p(n)$, tem-se simultâneamente $\mathcal{D}(A_n|G_n) \leq \mathcal{D}(A_n|M_n) + p(n)$ e $\mathcal{D}(A_n|M_n) \leq \mathcal{D}(A_n|G_n) + p(n)$.

Note-se que, se $G =_{\text{pind}} M$ forem monótonos, então, para todo A computável, (32) pode-se simplificar em

$$\sum_{x \in A_n} |\mu[G_n = x] - \mu[M_n = x]| \;=\; O(p(n) \, 2^{-n})$$

Isto significa que o "integral" em A_n da diferença de probabilidadess entre G_n e M_n é, em função de n, desprezável. Note-se que $O(p(n) \, 2^{-n})$ converge para zero mais depressa do que o inverso de qualquer polinómio.

1.8 Complexidade e Aleatoriedade Computacional

Um dos problemas standard da computabilidade é o da *compressibilidade* bit-strings finitas ou, genericamente, de quaisquer elementos de um domínio ϖ-equivalente.

Fixada uma PFTM universal $\mathcal{U}$, um $x \in \varpi$ é **compressível** se existe uma máquina de Turing livre de prefixos $\mathcal{U}_p$ determinada por um programa p, que verifica $|p| < |x|$ e $\mathcal{U}_p(0) \simeq x$ (equivalentemente, se $\mathcal{U}(p) \simeq x$).

Faz sentido definir uma medida da compressibilidade de x determinando o tamanho mínimo dos programas que geram x. Assim, define-se **complexidade de Kolmogorov**[13] de x relativa à PFTM universal $\mathcal{U}$, como

$$K(x\,;\mathcal{U}) \;=\; \min\{\,|p| \;\mid\; \mathcal{U}(p) \simeq x\,\} \tag{33}$$

O argumento $\mathcal{U}$ em K, assim como a caveat "relativa à PFTM $\mathcal{U}$", pode ser abandonado se admitirmos que a complexidade de Kolmogorov está definida a menos de uma constante aditiva. Isto é resultado do seguinte facto,

28 FACTO

A complexidade de Kolmogorov é sub-aditiva; isto é,

$$K(p\,x\,;\mathcal{U}) \;<\; K(p\,;\mathcal{U}) + K(x\,;\mathcal{U}) \pm O(1)$$

[13] Estritamente, a definição aqui apresentada é designada por "complexidade livre de prefixos de Kolmogorov", para a distinguir de uma noção semelhante definida em TM genéricas. No entanto, neste curso, esta é a versão usada.

Nomeadamente, se $\mathcal{U}, \mathcal{V}$ são duas PFTM universais, então,

$$K(x\,;\,\mathcal{U}) \;<\; K(x\,;\,\mathcal{V}) \pm O(1)$$

A segunda parte da afirmação resulta da primeira e do facto que $\mathcal{V}$ é descrita por um programa p em $\mathcal{U}$.

Como consequência escreve-se sempre $K(x) = \min\{\,|p| \mid \mathcal{U}(p) \simeq x\,\}$, abstraindo-nos em relação à máquina universal $\mathcal{U}$, assumindo que K é sempre definido a menos de uma constante.

29 Facto
Para qualquer linguagem L verifica-se $\sum_{x\in L} 2^{-K(x)} \leq 1$.

Prova
Seja, para todo $x \in L$, $p(x)$ tal que $|p(x)| = K(x)$ e $\mathcal{U}(p(x)) \simeq x$. A linguagem $\{p(x)\}_{x\in L}$ é livre de prefixos porque está contida no domínio de $\mathcal{U}$. Consequentemente $1 \geq \sum_{x\in L} 2^{-|p(x)|} = \sum_{x\in L} 2^{-K(x)}$.

Aleatoriedade de Kolmogorov-Levin-Chaintin

30 Noção
A classe $\mathcal{T}_K = \{B_n\}$, formada pelos conjuntos $B_n = \{\,x \mid K(x) + n < |x|\,\}$, designa-se por ***Teste de Kolmogorov-Levin-Chaitin (KLC-test)***.

Como K é definida a menos de uma constante, duas definições de K conduzem a conjuntos que diferem apenas nos índices. Isto é, se for $K(x) < K'(x) \pm O(1)$, então com $n' = n \pm O(1)$ tem-se

$$\{ x \mid K'(x) + n' < |x| \} \quad \subseteq \quad \{ x \mid K(x) + n < |x| \}$$

Por isso $\mathcal{T}_K = \{B_n\}$ está definida independentemente da constante aditiva em K.

Note-se que a classe $\mathcal{T}_K$ é decrescente; isto é, verifica $B_{n+1} \subseteq B_n$. Adicionalmente é efectivamente enumerada; basta ter em atenção uma função parcial recursiva τ que verifique $\tau(n,x)\downarrow \Leftrightarrow (x \in B_n)$. Nomeadamente basta fazer $\tau(n,x)\downarrow \Leftrightarrow (\exists\, y)\,[\,|y| < |x| - n \;\wedge\; \mathcal{U}(y) \simeq x\,]$. Finalmente,

$$\mu(B_n) \;=\; \sum_{x \in B_n} 2^{-|x|} \;<\; \sum_{x \in B_n} 2^{-K(x)-n} \;\leq\; 2^{-n}$$

Portanto cada B_n é $\mathbb{L}$-limitado e verifica $\mu(B_n) < 2^{-n}$.

□

A classe $\mathcal{T}_K$ vai servir para definir um critério de aleatoriedade de conjuntos (ou bit-strings infinitas) baseadas na noção de compressibilidade. Procura-se capturar a intuição que nos diz não serem aleatórias as strings compressíveis.

Como estamos a lidar com conjuntos e bit-strings infinitas temos de lidar com truncaturas. Seria viável uma condição da forma $K(A \restriction k) < k$ para testar a compressibilidade de uma truncatura de A; porém, como a complexidade de

Kolmogorov, é definida menos de uma constante, faz mais sentido usar testes da forma $K(A\restriction k)+n<k$; ou seja, faz mais sentido testar os sucessivos predicados $A\restriction k\in B_n$.

31 NOÇÃO

Diz-se que o conjunto A ***satisfaz o teste KLC*** *quando, para todo n, existe uma truncatura $A\restriction k$ contida em B_n. Os conjuntos que não satisfazem o teste KLC dizem-se* ***KLC-aleatórios***.

Dito doutro modo, A não é KLC-aleatório quando $(\forall\, n)\,(\exists\, k)\,[\,K(A\restriction k)+n<k\,]$. Da mesma forma, A é KLC-aleatório quando $(\exists\, n)\,(\forall\, k)\,[A\restriction k\notin B_n]$ ou, equivalentemente, $(\exists\, n)\,(\forall\, k)\,[K(A\restriction k)+n\geq k]$.

□

O teste KLC e a noção de aleatoriedade KLC podem também ser apresentados numa perspectiva topológica.

Recorde-se que uma classe $\mathcal{O}$ é aberta sse existe uma função f tal que $A\in\mathcal{O}\Leftrightarrow(\exists\, k)\,[f(A\restriction k)\neq 0]$. Da mesma forma, uma classe $\mathcal{C}$ é fechada sse existe uma função f tal que $A\in\mathcal{C}\Leftrightarrow(\forall\, k)\,[f(A\restriction k)=0]$.

Fazendo f percorrer as funções características dos vários B_n, define-se uma enumeração de classes abertas $\{\mathcal{O}_n\}$ e uma enumeração de classes fechadas $\{\mathcal{C}_n\}$, através de

$$\begin{array}{lllll} A\in\mathcal{O}_n & \Leftrightarrow & (\exists\, k)\,[A\restriction k\in B_n] & \Leftrightarrow & (\exists\, k)\,[\,K(A\restriction k)+n<k\,] \\ A\in\mathcal{C}_n & \Leftrightarrow & (\forall\, k)\,[A\restriction k\notin B_n] & \Leftrightarrow & (\forall\, k)\,[\,K(A\restriction k)+n\geq k\,] \end{array} \tag{34}$$

Agora a aleatoriedade é muito simples de exprimir.

32 FACTO

A classe dos conjuntos que não são KLC-aleatórios é $\bigcap_n \mathcal{O}_n$. *Analogamente* $\bigcup_n \mathcal{C}_n$ *é a classe dos conjuntos KLC-aleatórios.*

As definições de aleatoriedade, aqui expressas em termos de conjuntos, podem-se estender a funções totais ou sequências. Basta substituir nessas definições, todas as referências a truncaturas de conjuntos $A \upharpoonright k$ por truncaturas de funções $f \upharpoonright k$. Por exemplo, diremos a função total f é KLC-aleatória quando $(\exists\, n)\,(\forall\, k)\ [K(f \upharpoonright k) + n \geq k]$. Como sequências são funções totais, uma sequência é KLC-aleatória se a função que a define for KLC-aleatória.

□

A complexidade de Kolmogorov também pode ser usada para definir uma forma de redução entre funções, sequências ou conjuntos.

Diz-se que f é **K-redutível** a g, e escreve-se $f \leq_K g$, quando, para todo k

$$K(f \upharpoonright k) \ \leq \ K(g \upharpoonright k) \ \pm O(1) \tag{35}$$

Naturalmente, f é **K-equivalente** a g, e escreve-se $f \equiv_K g$, quando $f \leq_K g \wedge g \leq_K f$; f é **K-incomparável** a g, e escreve-se $f|_K g$, quando $f \not\leq_K g \wedge g \not\leq_K f$. Pode-se provar que duas funções recursivas são sempre K-equivalentes; uma função f que seja K-equivalente a uma função recursiva diz-se **K-trivial**.

A K-redutibilidade é importante porque serve para comparar a aleatoriedade de duas funções ou conjuntos. De facto, duas consequências desta relação, são

33 Facto
Sejam $A, B \in 2^{\varpi}$ *que verificam* $A \leq_K B$ *. Então:*

(i) *se* A *é KLC-aleatório,* B *também é KLC-aleatório,*

(ii) $A \uplus B$ *não é KLC-aleatório.*

O resultado (i) é um consequência directa da definição. O resultado (ii) foi provado por vários autores nomeadamente Miller & Yu. Solovay provou, adicional,mente, que existe uma relação entre a reducibilidade K e a reducibilidade S,

34 Teorema (Solovay)
Se α, β *são reais computáveis, então* $\alpha \leq_S \beta$ *implica* $\alpha \leq_K \beta$ *.*

A noção de KLC-aleatoriedade e a comparação de aleatoriedade via a K-redução não permite saber se realmente existe algum conjunto, função ou sequência que seja KLC-aleatório. Até ao momento, não mostrámos evidência que a definição que apresentámos não é vazia.

De facto, provar que existe um objecto que seja aleatório, pode aparecer uma contradição; isto porque é necessário construir tal objecto e, aparentemente, "algo que é construido não pode ser aleatório".

No entanto a nossa definição de aleatoriedade é construtiva e, coube a Solovay mostrar que é mesmo possível construir um objecto aleatório. De facto prova-se

35 Facto
Ω é KLC-aleatório.

Como consequência deste resultado e ainda do teorema 34 e do facto 33, tem-se

36 Corolário *Todo real computável α que seja Ω-equivalente (isto é, verifique $\Omega \leq_S \alpha$) é KLC-aleatório.*

Nomeadamente, pelo teorema 18, as probabilidades de paragem das PFTM universais são reais KLC-aleatórios. Finalmente Kučera & Slaman provaram

37 Teorema
Todo o real computável que seja KLC-aleatório é Ω-equivalente.

Aleatoriedade Relativa

Suponhamos que se modificava a definição de complexidade de Kolmogorov, considerando um oráculo arbitrário T, e se definia

$$K^T(x) \;=\; \min\{\,|p| \;\mid\; \mathcal{U}^T(p) \simeq x\,\} \tag{36}$$

Essencialmente, à PFTM universal $\mathcal{U}$ é dada permissão para consultar o oráculo T.

Como consequência pode-se relativizar todos os conceitos inerentes à complexidade de Kolmogorov, ao oráculo T. Nomeadamente pode-se dizer que f é T-aleatório quando $(\exists\, n)\ (\forall_\infty k)\ [K^T(f \restriction k) + n \geq k]$.

Se um conjunto for aleatório se e só se for T-aleatório, então diz-se que T é **K-low**. A relação entre estes conceitos e o a K-reducibilidade pode ser expressa no seguinte teorema

38 Teorema (Nies, et al.)
Se Ω é T-aleatório e T é enumerável, então T é K-trivial. O conjunto T é K-trivial se e só se for é K-low.

Será que os conjuntos K-triviais se resumem aos conjuntos recursivos? De facto, a resposta é negativa

39 Teorema (Kučera & Terwijn)
Existe um conjunto enumerado e K-low que não é decidível.

Aleatoriedade de Miller e Decisores

Toda a sequência anterior de resultados estabelece uma ligação muito forte entre a redução de Solovay, a equivalência de Solovay, a redução de Kolmogorov e a noção de aleatoriedade KLC. Existe também uma relação entre a aleatoriedade KLC e a noção de decisores.

A seguir usa-se um resultado de Miller & Yu, para provar que existem objectos KLC-aleatórios,

40 Teorema (Miller & Yu)
A função σ verifica $\sum_k 2^{-\sigma(k)} < \infty$ se e só se existe uma função KLC-aleatória f tal que,

$$(\forall_\infty k)\ [\, K(f \upharpoonright k) \ \leq\ k + \sigma(k) \pm O(1)\,]$$

Como consequência

41 Teorema
Se $\downarrow A$ é um decisor para o gerador G, então existe uma função KLC-aleatória f que verifica

$$(\forall_\infty k)\ K(f \upharpoonright k) \ \leq\ \lambda[G = A \upharpoonright k]$$

Adicionalmente, se $\downarrow A$ for um decisor para a PFTM universal $\mathcal{U}$, então A é KLC-aleatório.

Prova
Seja A tal que a sua história $\downarrow A$ é um decisor para a PFTM G. Seja $\sigma(k) = \lambda[G = A \upharpoonright k] - |A \upharpoonright k|$. Pela hipótese, tem-se $\sum_k 2^{-\sigma(k)} < \infty$. Pelo teorema 40 existe uma função aleatória f que verifica para todo k, com um eventual número finito de excepções, $K(f \upharpoonright k) \leq k + \sigma(k) = \lambda[G = A \upharpoonright k] \pm O(1)$. Seja agora $G = \mathcal{U}$; como, para todo x, $\lambda[\mathcal{U} = x] \leq K(x)$, tem-se $(\forall_\infty k)\,[K(f \upharpoonright k) \leq K(A \upharpoonright k) \pm O(1)]$. Portanto $f \leq_K A$ e, dado que f é aleatório, o resultado (i) no facto 33 permite concluir que A é KLC-aleatório.

O inverso deste resultado seria: todo o A que seja KLC-aleatório tem uma história $\downarrow A$ que é decisor de $\mathcal{U}$. Este é um problema em aberto.

□

O resultado 41 abre caminho a uma outra forma de redução,

42 Noção
Escreve-se $A \leq_D B$ *, e diz-se que* A *é* **D-redutível** *a* B*, quando, para todo o gerador* G*, se* $\downarrow A$ *é um decisor para* G *também* $\downarrow B$ *é um decisor para* G*.*

Testes Computáveis e Aleatoriedade

Voltemos à definição do teste de Kolmogorov-Levin-Chaitin $\mathcal{T}_K$ que apresentámos na noção 30 (ver página 76). Pode-se generalizar esta noção de "teste" através dessas propriedades da sua interpretação topológica.

Recordemos que cada classe aberta $\mathcal{O}$ é determinada por uma função total α tal $f \in \mathcal{O} \Leftrightarrow (\exists\, k)\,[\alpha(f \restriction k) > 0]$. Sem perda de generalidade, a função α pode ser escolhida de forma que seja "livre de prefixos"; isto é, a linguagem $\{\, x \mid \alpha(x) > 0 \,\}$ seja livre de prefixos. Nestas circunstancias faz sentido definir a **medida** de Lebesgue de α (ou, equivalentemente, de $\mathcal{O}$) como $\mu(\mathcal{O}) \equiv \mu(\alpha) = \sum_{\alpha(x)>0} 2^{-|x|}$.

Como α é livre de prefixos, tem-se sempre $\mu(\alpha) \leq 1$.

43 Noção
Um **teste computável** *é enumeração computável* $\beta = \{\alpha_n\}$ *de funções computáveis livres de prefixos tal que, para todo* n*,* $\mu(\alpha_n) < 2^{-n}$ *e, sendo* $\mathcal{O}_n$ *a classe aberta determinada por* α_n*, verifica-se* $\mathcal{O}_{n+1} \subseteq \mathcal{O}_n$*.*

Uma classe $\mathcal{A}$ ***satisfaz*** *o teste* β *(ou é,* β***-nula***) *quando se verifica* $\mathcal{A} \subseteq \bigcap_n \mathcal{O}_n$.

Para testar a inclusão $\mathcal{A} \subseteq \bigcap_n \mathcal{O}_n$ basta verificar se, dado um qualquer $f \in \mathcal{A}$, é possível encontrar um índice n, tal que $f \in \mathcal{O}_n$. Tipicamente a classe $\mathcal{A}$ é enumerada por uma função apropriada φ; por isso, um f arbitrário pode ser escrito como $f = \varphi_m$. A verificação da satisfação é, agora, equivalente a $(\forall\, m)\,(\exists\, n)\,[\,\varphi_m \in \mathcal{O}_n\,]$.

A verificação de satisfação $\mathcal{A} \subseteq \bigcap_n \mathcal{O}_n$ é **computável** se $\mathcal{A}$ é computavelmente enumerável e existe uma função computável s tal que $(\forall\, m)\,[\,\varphi_m \in \mathcal{O}_{s(m)}\,]$.

O seguinte resultado é consequência de um teorema de Schnorr & Chaitin.

44 FACTO
Se f *é uma função KLC-aleatória, então não existe qualquer teste computável que a classe singular* $\{f\}$ *satisfaça.*

Quando, na definição de teste, se entende "computável" como "recursivo", os testes dizem-se **Martin-Löf**. A não existência de um teste de Martin-Löf que seja satisfeito pela classe singular $\{f\}$ é, precisamente, a definição de **aleatoriedade de Martin-Löf**.

O que o teorema de Schnorr & Chaitin prova é que estas duas noções de aleatoriedade, KLC e ML, são equivalentes: f é KLC-aleatório se e só se $\{f\}$ não satisfaz qualquer teste de Martin-Löf.

Neste curso estamos interessados em testes computáveis mais fortes: testes que sejam, no mínimo, testes de Martin-Löf mas verifiquem, para além disso, restrições que advêm de uma noção mais exigente de computabilidade. Adicionalmente quere-se que a verificação do teste seja, ela própria, computável.

Isto implica que a relação entre satisfação de testes e aleatoriedade segue apenas um sentido: não é KLC-aleatória um função f em $\{f\}$ satisfaz algum teste computável, mas o inverso não é, normalmente, válido.

1.9 Computabilidade Probabilística

Algumas noções fundamentais podem-se definir à custa da relação de indistinguibilidade; iremos referir a duas delas: a noção de **gerador pseudo-uniforme**[14], a noção de **função probabilisticamente computável** e a noção de **"bit-leak"**.

Antes porém é necessário entender para que são necessários geradores e porque os vemos como máquinas de Turing.

Geradores aparecem constantemente em Criptografia quer na implementação de técnicas criptográficas quer ainda nas provas de correcção ou de segurança dessas técnicas.

A ideia essencial é que um gerador é uma máquina cujo "input" é desconhecido e se consegue apenas observar o "output". Desse modo o "output" não pode ser detertminado com precisão e a única forma de o caracterizar assume a forma de probabilidades.

O paradigma que se subentende é que os "inputs" são apresentados de forma "aleatória" (veremos adiante o que isso significa) e utilização correcta dos "outputs" não depende do seu valor concreto mas apenas das probabilidades que associamos a esses valores. O paradigma dos geradores lida com a incerteza agregando inputs em distribuições de probabilidade.

[14] Preferimos esta designação à mais mais comum "pseudo-aleatório" porque que pensamos que esta última é inapropriada.

A definição de indistinguibilidade é uma instância típica deste paradigma: comparam-se geradores vendo as diferenças entre as distribuições de probabilidade e integrando essas diferenças de forma apropriada.

Gerador Uniforme e Geradores Pseudo-Uniformes

Seja l um polinómio que verifica $l(n) > n$, para todo n; tais polinómios serão aqui designados por **extensões**. Considera-se a PFTM U^l definida por

$$U^l(y,n)\downarrow \;\Leftrightarrow\; |y| = l(n) \qquad \text{e} \qquad U^l(y,n)\downarrow \;\Rightarrow\; U^l(y,n) \simeq y \tag{37}$$

Isto significa que U^l_n só aceita "inputs" de comprimento $l(n)$ e, nesse caso, copia o "input" para o "output".

O gerador U^l designa-se **gerador uniforme para a extensão** l. Este gerador é semelhante a uma sequência de "filtros" $\left\{U^l_n\right\}$ que deixam passar os "inputs" que têm o comprimento apropriado $l(n)$. Asumindo que os "inputs" são, de alguma forma, aleatórios, os "outputs" serão "aleatórios" com o comprimento adequado.

A forma deste gerador permite, também, determinar facilmente dimensões $\mathcal{D}(A|U^l_n)$.

Quando $|x| = l(n)$ tem-se $\lambda[U^l_n = x] = l(n)$; se $|x| \neq l(n)$, tem-se $\lambda[U^l_n = x] = \infty$. Logo, para todo A,

$$\mathcal{D}(A|U^l_n) \;=\; \sum_{x\in A} 2^{|x|-\lambda[U_n=x]} \;=\; \sum_{x\in A\wedge|x|=l(n)} 2^{l(n)-l(n)} \;=\; |A\cap\mathbb{B}^{l(n)}|$$

45 NOÇÃO

Um gerador G é **pseudo-uniforme** *para a extensão l se é computável, é indistinguível de U^l e existe uma extensão $h \ll l$ e uma máquina de Turing computável tais que $G(y) = M(U^h(y))$.*

Num gerador pseudo-uniforme, a máquina M recebe "inputs" de comprimento $h(n)$ e produz "outputs" de comprimento $l(n)$; a relação $h \ll l$ implica que, para todo n, $h(n) \ll l(n)$; por isso o "input" é substancialmente mais curto do que o "output". Finalmente $G = M \circ U^h$ é indistinguível de U^l.

Funções como Geradores e Funções Probabilisticamente Computáveis

Seja f uma qualquer função recursiva monótona (i.e. $|f(u,n)| > n$), para todo n e seja $\bar{f}$ uma qualquer PFTM monótona que verifique, para todos índice u, n

$$\mu[\bar{f}_{u,n} = x] = \begin{cases} 1 & \text{se } f(u,n) = x \\ 0 & \text{se } f(u,n) \neq x \end{cases} \tag{38}$$

É sempre possível construir uma tal máquina; um exemplo simples é a máquina definida pela enumeração $\{\bar{f}_{u,n}\}$ que verifica $\bar{f}_{u,n}(y)\downarrow \Leftrightarrow |y| = n$ e $\bar{f}_{u,n}(y)\downarrow \Rightarrow \bar{f}_{u,n}(y) \simeq f(u)$.

Considere-se agora uma classe $\mathcal{C}$ de PFTM que consideramos "computáveis" numa perspectiva de consumirem "recursos razoáveis". Por exemplo, a classe das PFTM que são PPT ("probabilistic polynomial time").

No que se segue a relação de indistinguibilidade está sempre referida à classe $\mathcal{C}$; nomeadamente os conjuntos A computáveis que ocorrem nas definições 25 e 26 estão limitados à classe $\mathcal{C}$.

46 NOÇÃO

Uma função f é **probabilistcamente computável** *(ou* **p-computável***) se existe um gerador $G \in \mathcal{C}$ e uma prova computável em $\mathcal{C}$ de que $(\forall\, u)\,[G_u =_{\mathsf{ind}} \bar{f}_u]$ é válido.*

Uma função f é **estritamente probabilisticamente computável** *(ou* **estritamente p-computável***) se existe um gerador $G \in \mathcal{C}$ que seja indistinguível de $\bar{f}$.*

Uma função f é **p-incomputável** *se, para todo o gerador computável G, existe uma prova computável de $(\forall\, u)\,[G_u \,\#\, \bar{f}_u]$.*

Uma função f é **p-unidireccional** *se é p-computável e toda a quasi-inversa de f é p-incomputável.*

Formalmente estas definições só podem ser estabelecidas para funções monótonas o que, à partida, exclui classes importantes de funções; nomeadamente exclui as funções características de conjuntos. Este facto torna, aparentemente, pouco úteis estas definições.

No entanto é simples converter uma função g, não monótona, numa função monótona f com a mesma informação; basta definir $f(u,n) = 1^n\,0\,g(u,n)$.

A segunda definição é, geralmente, mais forte que a primeira. Para vermos a diferença convém ver a sequência de quantificadores que se obtém expandindo cada uma das definições e a definição de indistinguibilidade. Tem-se

p-computável

$$(\exists\, G \in \mathcal{C})\ (\forall\, u)\ (\forall\, A \in \mathcal{C})\ (\exists\, p)\ (\forall_\infty\, n)\ \Delta_{u,n}(A_n) < p(n) \tag{39}$$

estritamente p-computável

$$(\exists\, G \in \mathcal{C})\ (\forall\, A \in \mathcal{C})\ (\exists\, p)\ (\forall_\infty\, n)\ (\forall\, u)\ \Delta_{u,n}(A_n) < p(n) \tag{40}$$

em que, em ambos os casos, $\Delta_{u,n}(A_n)$ representa o integral da diferença de probabilidades

$$\Delta_{u,n}(A_n) \quad = \quad \sum_{x \in A_n} 2^{|x|}\, |\mu[G_{u,n} = x] - \mu[\bar{f}_{u,n} = x]|$$

A distinção básica está na posição relativa dos quantificadores $(\forall\, u)$ e $(\exists\, p)$. Enquanto no primeiro caso, pode-se escolher o polinómio $p(n)$ em função do argumento u da função, no segundo caso o mesmo $p(n)$ tem de servir para todos os eventuais argumentos da função.

Uma diferença mais subtil está na alteração da posição relativa de $(\forall_\infty\, n)$ e $(\forall\, u)$. Normalmente a ordem relativa de dois quantificadores do mesmo tipo (dois existenciais ou dois universais) é irrelevante; por exemplo,

$(\forall\, u)\,(\forall\, A \in \mathcal{C})$ é completamente equivalente a $(\forall\, A \in \mathcal{C})\,(\forall\, u)$. Porém a ordem relativa de $(\forall_\infty\, n)$ e $(\forall\, u)$ não é irrelevante porque $(\forall_\infty\, n)$ tem uma componente de quantificador existencial; ao introduzir a possibilidade de existência de um número finito de excepções à quantificação $(\forall\, n)$, a ordem entre quantificadores $(\forall_\infty\, n)\,(\forall u)$ indica que as excepções em n servem para todo u, enquanto que a ordem $(\forall u)\,(\forall_\infty\, n)$ abre a possibilidade de essas excepções serem função de u.

Obviamente, se não forem admissíveis excepções (i.e., a quantificação em n for sempre $(\forall\, n)$) ou o domínio de f for finito, então a ordem entre os quantificadores em u e n é irrelevante.

Alguns importantes conceitos derivam naturalmente da definição de p-computabilidade.

Com esta definição de computabilidade de funções, temos imediatamente uma noção induzida de computabilidade de conjuntos: o conjunto A é **p-computável** (ou **estritamente p-computável**) se e só se a sua função característica é p-computável. No modo análogo definimos as noções mais fracas de conjunto *p-computacionalmente enumerável, totalmente enumerável, semidecidível e decidível*.

Unidireccionalide no contexto da p-computabilidade

São particularmente importantes em Criptografia as funções que, além de monótonas, preservam o comprimento do "input" no "output". Funções que verificam $f(u,n) = x$ implica $|x| = |u| > n$ dizem-se que *preservam comprimentos*. Neste caso, se cada f_n for uma bijecção, a função f é uma **permutação de bits** (ou, simplesmente, **permutação**). Por exemplo, todas as cifras têm na sua génese uma permutação de bits.

Recordemos que um dos primeiros conceitos que introduzimos foi o de *função unidireccional*: uma função que é computável em que toda a eventual quasi-inversa é incomputável. Interpretando "computável" como "p-computável" (ou, ainda, "estritamente p-computável") somos conduzidos à noção de **função p-unidireccional** (ver noção 46). No contexto da p-computabilidade (ou, estrita computabilidade) como se exprime a unidireccionalidade de uma função f que preserva comprimentos?

Em primeiro lugar temos de fixar a classe de geradores que consideramos computáveis. Normalmente fixa-se uma classe de máquinas de Turing livres de prefixos limitadas em recursos; por exemplo, as máquinas probabilísticas que correm em tempo polinomial com o comprimento do "input". Nomeadamente, os conjuntos A, usados usados no cáculo das decisóes $\mathcal{D}(A|G)$, são referidos como "computáveis" quando a sua função característica é implementada por um gerador computável.

A função f é p-computável (tem de satisfazer as condições da noção 46) e toda a sua quasi-inversa tem de ser p-cincomputável. Seja h uma qualquer quasi-inversa h de f; por analogia com (38), seja $\bar{h}$ um gerador que verifica

$$\mu[\bar{h}_{x,n} = u] \quad = \quad \mu[\bar{f}_{u,n} = x] \quad = \quad \begin{cases} 1 & \text{se } f(u,n) = x \\ 0 & \text{se } f(u,n) \neq x \end{cases} \tag{41}$$

Tem-se, para um qualquer conjunto computável B,

$$\mathcal{D}(B_n|\bar{h}_{x,n}) \quad = \quad \sum_{u \in B_n} 2^{|u|}\, \mu[\bar{h}_{x,n} = u] \quad = \quad \sum_{u \in B_n \wedge f(u,n)=x} 2^{|u|}$$

O integral da diferença de probabilidades $\sum_{u \in B_n} 2^{|u|} \, |\mu[\bar{h}_{x,n} = u] - \mu[H_{x,n} = u]|$ será

$$\sum_{u \in B_n \wedge f(u,n) = x} 2^{|u|} \, (1 - \mu[H_{x,n} = u]) \; + \sum_{u \in B_n \wedge f(u,n) \neq x} 2^{|u|} \, \mu[H_{x,n} = u] \qquad (42)$$

Para que h seja computável, tem de existir um gerador H tal que, para todo conjunto computável B existe uma função computável que mapeia cada x numa prova de que (42) é polinomialmente limitada. Em particular, se se quiser provar que f não é p-unidireccional (mesmo sendo p-computável) basta construir um gerador H tal que, para todo x e todo B computável, ambas as parcelas em (42) sejam polinomialmente limitadas.

Ao se afirmar que *toda a eventual quasi-inversa de f não é p-computável*, está-se a afirmar que,

> *para todo o gerador computável H, existe um x e um conjunto computável B, tais que para todo polinómio $p(n)$ pelo menos uma das parcelas em* (42) *não é superiormente limitada por $p(n)$.*

Isto porém, não é suficiente para garantir que f é p-unidireccional; é preciso exigir *toda a eventual quasi-inversa de f é p-incomputável*. Para isso a condição tem de ser mais forte:

> *para todo gerador computável H, para todo x e todo o polinómio $p(n)$, existe um B tal que pelo menos uma das parcelas em* (42) *é inferiormente limitada por $p(n)$.*

Correndo o risco de perder a ligação directa entre unidireccionalidade, p-computabilidade e p-incomputabilidade, pode-se fortalecer on enfraquecer este conceito, de forma a capturar alguma intruição extra. Alguns exemplos

1. A condição de p-incomputabilidade da quasi-inversa h exige que, para todo o gerador computável H e para todo x, H_x seja estritamente distinguível de $\bar{h}_x$. Essencialmente, para qualquer valor de x não é possível calcular u e n tal que $f(u,n) = x$.

   Esta condição pode ser enfraquecida admitindo a eventualidade de um número finito de excepções em x desde que, à partida, não seja previsível conhecer quais são essas excepções.

2. As duas parcelas em (42) têm interpretações diferentes:

   A parcela $\sum_{u \in B_n \wedge f(u,n)=x} 2^{|u|}\, \mu[H_{x,n} \neq u]$ representa a probabilidade de o gerador H com input x produzir um resultado $u' \in \mathbb{B}_n$ distinto do resultado correcto $u \in \mathbb{B}_n$. Um limite inferior nesta probabilidade implica que a probabilidade de H gerar o valor correcto u é pequena. Isto, porém, não invalida a hipótese de que, ainda assim, se tenha $f(u',n) = x$.

   Ao invés, a parcela $\sum_{u \in B_n \wedge f(u,n)\neq x} 2^{|u|}\, \mu[H_{x,n} = u]$ representa a probabilidade de H gerar um u "errado": isto é, um u tal que $f(u,n) \neq x$. Um limite inferiro nesta probabilidade implica que a probabilidade de H gerar um valor errado é elevada.

   A incomputabilidade exige apenas que uma destas parcelas seja inferiormente limitada. Nomedamente pode ser só a primeira parcela que seja inferiormente limitada. No entanto, como a primeira parcela abre a possibilidade de ser $f(u',n) = x$, uma condição de p-unidireccionalidade mais forte impõe a limitação exclusivamente na segunda parcela.

   Assim, exige-se que para tod H, todo o polinómio p, todo x (com um eventual número finito de excepções),

existe um B tal que

$$(\forall_\infty n) \sum_{u \in B_n \wedge f(u,n) \neq x} 2^{|u|}\, \mu[H_{x,n} = u] \;>\; p(n) \tag{43}$$

"Bit-leaks" e Predicados "Hardcore"

O facto de uma função ser unidireccional não implica, necessariamente, que seja impossível extrair do "output" $x = f(u)$ informação sobre o argumento u.

Seja C ("coin") um gerador monótono definido, para todo n, por $C(0,n) \simeq 1^n\,0$, $C(1,n) \simeq 1^{n+1}$ e $C(y,n)\!\uparrow$ se $|y| \neq 1$.

Consequentemente $\mu[C_n = x] = 1/2$, se $x = 1^n\,0$ ou se $x = 1^{n+1}$, e $\mu[C_n = x] = 0$ para todos os restantes valores de x. Para todo A, a decisão $\mathcal{D}(A_n, C_n)$ pode, agora, ser calculada como

$$\mathcal{D}(A_n, C_n) = \sum_{x \in A_n} 2^{|x|}\, \mu[C_n = x] \quad = \quad 2^n \times \begin{cases} 0 & \text{se} \quad 1^n\,0 \notin A_n \;\wedge\; 1^{n+1} \notin A_n \\ 2 & \text{se} \quad 1^n\,0 \in A_n \;\wedge\; 1^{n+1} \in A_n \\ 1 & \text{nos outros casos} \end{cases} \tag{44}$$

47 NOÇÃO

Um conjunto p-computável P *é* **1-Bit** *se existe um gerador* G, *indistinguível de* C, *tal que* $y \in P_n$ *se e só se* $G_n(y) \simeq 1^{n+1}$.

Seja f *uma função p-unidireccional. O conjunto 1-Bit* P *é um* **p-hardcore predicate** *se não existe qualquer conjunto p-computável* Q *que verifique* $(u \in P)$ *se e só se* $(f(u) \in Q)$ *; isto é,* $Q = f(P)$.

Se P *não é "hardcore predicate" para* f *(i.e., se* $f(P)$ *for computável), então designa-se por* **"bit-leak"** *para* f, *e a sua existência representa-se por* $P \xrightarrow{f} f(P)$,

Tendo em atenção que cada conjunto 1-Bit "divide a meio" o domínio de uma função p-direccional f, a existência de um "bit-leak" $P \xrightarrow{f} Q$ permite extrair informação sobre o "input" u a partir do conhecimento do "output" x, usando o seguinte algoritmo:

Objectivo: dado x encontrar um u tal que $f(u) = x$.

(1) Testa-se $x \in Q$
(2) Se o teste tem sucesso sabe-se a solução u está na "metade" do domínio contida em P.
(3) Se o teste falha, não existem soluções na "metade" do domínio contida em P

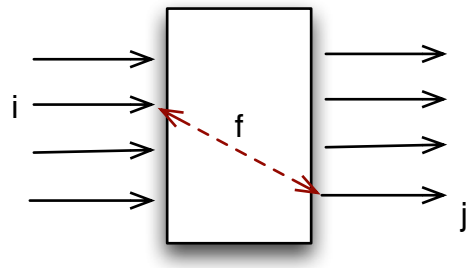
Em qualquer dos casos fica reduzida a incerteza sobre uma quasi-inversa de f aplicada a x: ou é um elemento que não está em P se o teste falha, ou então, se o teste tem sucesso, é um dos elementos de P.

Dois "bit-leaks" $P \xrightarrow{f} Q$ e $Q \xrightarrow{g} R$ são "combináveis" se Q também for um conjunto 1-Bit; isto é, o par P, R é um "bit-leak" para a composição de funções $f \circ g$. De facto isto ocorre já que $Q = f(P)$ e $R = g(Q)$.

Generalizando, suponhamos que é possível decompor uma função p-unidireccional f como a composição de uma sequência de n funções unidireccionais $f = f_0 \circ f_1 \circ \cdots \circ f_{n-1}$. Uma **bit-leak chain** para f é uma sequência de conjuntos 1-Bit, $S = P_0, P_1, \cdots, P_n$, tais que $P_{i+1} = f_i(P_i)$, com $i = 0..n-1$. Faz sentido representar tal cadeia por, $P_0 \xrightarrow{S} P_n = P_0 \xrightarrow{f_0} P_1 \xrightarrow{f_1} P_2 \cdots \xrightarrow{f_{n-1}} P_n$.

A existência de uma "bit-leak chain" é uma séria falha para uma função que se pretende p-unidireccional uma vez que cada passo na cadeia diminui em 1 bit a incerteza na quasi-inversa de f. Assumindo, como exemplo, que o domínio de f tem n bits de incerteza, então uma "bit-leak chain" com n passos reduz a zero essa incerteza.

A designação "bit-leak" (a palavra "leak" intrepretada como "fuga" ou "furo") tem origem numa interpretação clássica: *fuga de informação* entre um bit da entrada e um bit da saída numa *função com incerteza* representada por uma permutação de bits. Nesta interpretação, uma função é uma *"caixa preta"* que relaciona um determinada número n de bits à entrada com o mesmo número de bits à saída.

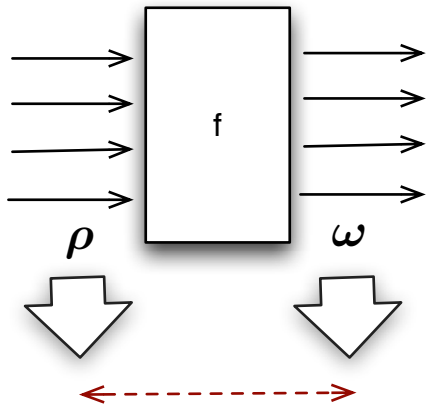


Supostamente a função é *segura* se preserva a incerteza: isto é, se todos os bits à saída dependem "igualmente" de todos os bits de entrada.

Um "bit-leak" é, aqui, uma violação de segurança que resulta de uma "ligação directa" entre um bit j da saída com um bit específico i da entrada,

o valor do bit j determina, "com elevada probabilidade", o valor do bit i

Este conceito tem uma extensão com a noção de **paridade**: mesmo que não seja possível estabelecer esta ligação directa entre bits, pode ainda ocorrer uma ligação directa entre combinações lineares de bits à entrada e à saída.



Uma **paridade** sobre palavras x de n bits, é uma função $\boldsymbol{\omega}$ da forma $\boldsymbol{\omega}(x) = \bigoplus_{i=0}^{|x|-1} x_i \cdot a_i$, em que os a_i são bits, o produto $\cdot$ é interpretado como **and** e a soma $\oplus$ é interpretada como **xor**. Qualquer paridade pode escrever-se como $\boldsymbol{\omega}(x) = x \cdot A$, para algum conjunto A .
Um "bit-leak" agora, será definido por duas paridades $\boldsymbol{\rho}$ e $\boldsymbol{\omega}$ e a condição:

com "elevada probabilidade", determinada por todos os eventuais argumentos u de f, o valor de $\boldsymbol{\omega}(f(u))$ determina o valor de $\boldsymbol{\rho}(u)$.

Só faz sentido falar de "fuga de informação" se, conhecido um "output", houver uma diminuição substantiva na incerteza do "input". Enquanto que no primeiro caso essa diminuição de incerteza era óbvia (o bit i do "input" ficava determinado imediatamente pelo conhecimento do "output"), no segundo modelo a diminuição de incerteza não é óbvia.

Falta responder à pergunta fundamental: *em que medida o conhecimento do bit de paridade* $\boldsymbol{\rho}(u)$ *diminui a incerteza sobre* u *?* Intuitivamente, essa "diminuição substantiva de incerteza" será pelo menos 1 bit, se não mais do que metade dos possíveis valores u tiverem esse bit de paridade a 1.

Por exemplo, a paridade $\boldsymbol{\rho}(u) = \bigoplus_{i=0}^{|u|-1} 0 \cdot u_i$ não produz uma diminuição de incerteza porque, trivialmente, se tem $\boldsymbol{\rho}(u) = 0$, para todo u.

A noção de "bit-leak" apresentada na noção 47 generaliza e formaliza este conceito de várias maneiras: em primeiro lugar as funções não estão limitadas a permutações de bits e podem ser quaisquer funções p-unidireccional; em segundo lugar, $\boldsymbol{\omega}$ e $\boldsymbol{\rho}$ não estão limitadas a paridades e podem ser predicados computáveis; finalmente formaliza o que se entende por "elevada probabilidade" e caracteriza com detalhe os predicados que detectam efectivas diminuições de incerteza.

Voltando às paridades, vamos usar um pouco de Algebra Linear.

Seja $\mathbf{I}$ a matriz identidade, e seja $\mathbf{I}_i$ a coluna de ordem i dessa matriz[15]. Então a componente de ordem i da

[15] Os vectores $\mathbf{I}_i$ formam **base** do espaço vectorial das strings de bits.

palavra de bits u, pode ser calculada como $u_i = u \cdot \mathbf{I}_i$.

Suponhamos que, sendo o valor de u desconhecido, são no entanto conhecidos os bits $u \cdot A$ e $u \cdot (A \oplus \mathbf{I}_i)$; então u_i pode ser calculado: basta constatar que $u \cdot (A \oplus \mathbf{I}_i) = (u \cdot A) \oplus (u \cdot \mathbf{I}_i) = (u \cdot A) \oplus u_i$.

Pode-se representar uma paridade por um gerador de duas formas:

(i) Fixado um conjunto A define-se $\boldsymbol{\omega}_A(y, n) = y \cdot A$ se $|y| = n$ e $\boldsymbol{\omega}_A(y, n)\uparrow$ se $|y| \neq n$.

(ii) Alternativamente pode-se ver $x \cdot a$ como uma função do par $y = x\|a$. Mais precisamente define-se o gerador $\boldsymbol{\omega}(x, a, n) = x \cdot a$, se $|a| \geq |x| = n$, e é indefinido se esta condição não se verifica.

48 LEMA $\mu[\boldsymbol{\omega}_{A,n} = 1] = 0$ *se* $n = 0$ *ou* $a_i = 0$, *para todo* $i \leq n$; $\mu[\boldsymbol{\omega}_{A,n} = 1] = \frac{1}{2}$ *em caso contrário. Adicionalmente* $\mu[\boldsymbol{\omega}_n = 1] = \frac{1}{2} - \left(\frac{1}{2}\right)^{n+1}$.

Prova

Como $x \cdot a = 0$ quando $|x| = 0$ tem-se $\mu[\boldsymbol{\omega}_0 = 1] = \mu[\boldsymbol{\omega}_0 = 1] = 0$.

Por simplicidade use-se a abreviatura $S_n = \bigoplus_{i=0}^{n-1} x_i \cdot a_i$. A indução usa a igualdade $S_{n+1} = (x_n \cdot a_n) \oplus S_n$. Daqui conclui-se, atendendo que $\mu[S_n = 0] = 1 - \mu[S_n = 1]$,

$$\begin{aligned}\mu[S_{n+1} = 1] &= \mu[S_n = 0] \cdot \mu[x_n = 1, a_n = 1] + \mu[S_n = 1] \cdot (1 - \mu[x_n = 1, a_n = 1]) \\ &= \mu[x_n = 1, a_n = 1] + \mu[S_n = 1] \cdot (1 - 2 \cdot \mu[x_n = 1, a_n = 1])\end{aligned}$$

Duas hipóteses:

1. *Os a_i são constantes:*
   Neste caso $\mu[S_n = 1]$ representa $\mu[\boldsymbol{\omega}_{A,n} = 1]$; tem-se $\mu[a_n = 1] = a_n$ e $\mu[x_n = 1] = 1/2$. A relação de recorrência transforma-se em $\mu[\boldsymbol{\omega}_{A,n+1} = 1] = a_n/2 + (1 - a_n) \cdot \mu[\boldsymbol{\omega}_{A,n} = 1]$. Se $a_n = 1$ será $\mu[\boldsymbol{\omega}_{A,n+1} = 1] = 1/2$; se $a_n = 0$ será $\mu[\boldsymbol{\omega}_{A,n+1} = 1] = \mu[\boldsymbol{\omega}_{A,n} = 1]$. Logo $\mu[\boldsymbol{\omega}_{A,n} = 1] = 1/2$ se existe $i \leq n$ com $a_i = 1$ e é zero em caso contrário.
2. *Os a_i são parte do argumento.*
   Neste caso $\mu[S_n = 1]$ representa $\mu[\boldsymbol{\omega}_n = 1]$; tem-se $\mu[a_n = 1] = \mu[x_n = 1] = 1/2$. A relação de recorrência é, agora, $\mu[\boldsymbol{\omega}_{n+1} = 1] = 1/4 + (1/2) \cdot \mu[\boldsymbol{\omega}_n = 1]$. Com a condição inicial $\mu[\boldsymbol{\omega}_0 = 1] = 0$, a solução desta recorrência é $\mu[\boldsymbol{\omega}_n = 1] = (1/2) + (1/2)^{n+1}$.

A conclusão imediata do lema 48 é

49 PROPOSIÇÃO
A paridade $\boldsymbol{\omega}$ determina um conjunto 1-Bit. A paridade $\boldsymbol{\omega}_A$ determina um conjunto 1-Bit sse $A \neq \emptyset$.

Prova Vamos provar apenas para $\boldsymbol{\omega}$ uma vez que a prova para $\boldsymbol{\omega}_A$ é análoga. Seja $G_n(y, a) = 1^n\, \boldsymbol{\omega}(y, a, n)$.
G_n só está definido quando $|y| = |a| = n$. Pelo lema 48, $\mu[G_n = 1^{n+1}] = 1/2 + 2^{-(n+1)}$, $\mu[G_n = 1^n\, 0] = 1/2 - 2^{-(n+1)}$, e $\mu[G_n = x] = 0$ para quaisquer outros valores de x. Fazendo a diferença em relação às probabilidades da "coin" C

$$\sum_{x \in A_n} 2^{|x|}\, |\mu[C_n = x] - \mu[G_n = x]| \quad = \quad 2^n \times \begin{cases} 0 & \text{se } 1^n\, 0 \in A_n \Leftrightarrow 1^{n+1} \in A_n \\ 2^{-n} & \text{nos dois restantes casos} \end{cases}$$

Logo G é indistinguível de C.

O conjunto que G determina é $\bigcup_n P_n = \{\, y\|a \mid (\exists\, n)\, G_n(y,a) = 1^{n+1} \,\} = \{\, y\|a \mid (\exists\, n)\, \boldsymbol{\omega}(y,a,n) = 1 \,\}$ que coincide com o conjunto definido por $\boldsymbol{\omega}$.

□

O estudo das propriedades de $\boldsymbol{\omega}$ tem por finalidade provar o seguinte teorema.

50 TEOREMA
Seja f uma função p-unidireccional; então a função $g(u,a) = f(u)\|a$, em que $|u| = |a|$, é unidireccional. Adicionalmente g é uma permutação de bits se e só se f for uma permutação de bits. Finalmente, $\boldsymbol{\omega}$ é "hardcore" para g.

A prova deste teorema é complexa e não a vamos aqui realizar. No entanto pode-se fazer notar algumas noções intuitivas.

Começamos por verificar que g é uma permutação de bits se e só se f for uma permutação de bits. De facto, assumindo que $x = f(u)$ implica $|x| = |u$, tem-se também que $|x\|a| = |u\|a|$. Por outro lado g é uma bijecção se e só se f é uma bijecção.

Se $g(u,a)$ tivesse uma inversa g' que fosse p-computável, então também f teria uma quasi-inversa p-computável.
De facto, se tivermos $x = f(u)$, será $x\|a = g(u\|a)$. Assumindo que existe uma quasi-inversa g^{-1}, seja

$u'\|a' = g^{-1}(x\|a)$. Pela definição de quasi-inversa tem-se $g(u'\|a') = f(u')\|a' = x\|a$; como $f(u') = x$, construimos uma quasi-inversa computável para de f. Como, por hipótese, f é unidireccional, não pode existir uma quasi-inversa computável para g.

De facto pode-se provar um resultado mais forte

51 LEMA *Se g tem uma quasi-inversa que não é p-incomputável, também f tem uma quasi-inversa que não é p-incomputável.*

Suponhamos, agora, que existia uma função computável h que, a partir do conhecimento de um qualquer par $x\|a$ calculava correctamente o valor $u \cdot a$; mais concretamente,

$$(\exists\, h \in \mathcal{C})\, [\, f(u,n) = x \;\Rightarrow\; h(x,a,n) = \boldsymbol{\omega}(u,a,n)\,] \tag{45}$$

Assumindo que $|u| = |x| = n$ (i.e. que f preserva comprimentos) pode-se fazer a percorrer as n colunas $\mathbf{I}_i$ da identidade $\mathbf{I} \in \mathbb{B}^{n\times n}$, e calcular os diversos $u_i = h(x, \mathbf{I}_i, n)$. Como, por hipótese, $u_i = \omega(u, \mathbf{I}_i, a)$, os bits u_i determinam as componentes do vector u. Assim (45) implica que f não é unidireccional, o que viola a nossa hipótese; portanto (45) não pode ocorrer.

Note-se que (45) é uma forma de afirmar que, se tal h existir, então h define um "bit-leak" para g. A conclusão que chegamos foi que, o facto de f ser unidireccional impede que apareçam "bit-leaks" para g da forma (45).

Não impede que existam outros "bit-leaks" que usem a paridade $\boldsymbol{\omega}$ no "input". Formalizemos:

Seja P o conjunto monótono determinado por $\boldsymbol{\omega}$; isto é, $P_n(u,a) = 1^n\,\boldsymbol{\omega}(u,a,n)$. Seja Q_n a sua imagem por g; isto é, $Q_n = g(P_n) = \{\, x\|a \mid (\exists\, u)\, f(u) = x \wedge \boldsymbol{\omega}(u,a,n) = 1 \,\}$. Como função monótona será $Q_n(x,a) = 1^n b$ em que b é o bit que denota o valor booleano de $(\exists\, u)\ [f(u) = x \wedge \boldsymbol{\omega}(u,a,n) = 1]$.

Sabemos (ver proposição 49) que P é um conjunto 1-Bit. Coloca-se agora a questão de saber se Q é ou não p-computável. De facto pode-se provar que

52 LEMA *Se Q for computável (equivalentemente, se $P \xrightarrow{g} Q$ é um "bit-leak") então f não é p-incomputável.*

A prova do teorema 50 resulta imediatamente do lema 51 e do lema 52.

□

Construção de geradores pseudo-uniformes

Seja f uma permutação de bits que é unidireccional e p a função característica de um conjunto "hardcore" para f. Como f é uma permutação de bits vamos ter $f_n(y) = x \Rightarrow |y| = |x| = n$.

53 PROPOSIÇÃO
O gerador $G_n(y) = f_n(y)\, p_n(y)$ é pseudo-uniforme com a extensão $l(n) = n + 1$.

Note-se que, pelo teorema 50, dada uma qualquer permutação de bits unidireccional f', é sempre possível construir outra permutação de bits unidireccional que tenha um conjunto "hardcore"; basta fazer $y = u\|a$, $f(y) = f'(u)\|a$ e usar o predicado $\boldsymbol{\omega}$.

A intuição por detrás desta proposição é a seguinte: como p é "hardcore" para f, o valor do bit $p_n(y)$ não pode ser calculado a partir do conhecimento de $f_n(y)$. Por isso a distribuição dos valores de $f_n(y)\, p_n(y)$ pode ser considerada uniforme; como o seu comprimento é $|f_n(y)| + 1$ e se tem $|f_n(y)| = |y| = n$, a função extensão é $l(n) = n + 1$.

□

Pode parecer que não se ganha muito: afinal converte-se n bits uniformemente distribuídos, representados no argumento y, por apenas $n + 1$ bits.

No entanto este 1 bit extra é suficiente para ser possível, gerar recursivamente uma sequência infinita de bits usando f e p como oráculos. Sejam

$$s_n(y, 0) = y \quad , \qquad s_n(y, k+1) = f_n(s_n(y, k)) \quad , \qquad A_n(y, k) = p_n(s_n(y, k)) \tag{46}$$

A partir do valor inicial y (o "seed") geram-se sucessivamente estados $s_n(y, k+1)$ aplicando f_n ao estado anterior $s_n(y, k)$. De cada estado, usa-se o predicado "hardcore" p_n para obter um bit que é incorporado numa string $A_n(y)$. Nestas condições

54 Proposição
Para todo o polinómio $l(n) > n$ *, o gerador* $G_n(y) = A_n(y) \restriction l(n)$ *é pseudo-uniforme.*

□

Construção de funções unidireccionais

Vimos como construir um gerador pseudo-uniforme de uma permutação de bits unidireccional; o inverso também é possível.

Concretamente vamos verificar que, dado um gerador pseudo-uniforme $G_n(y)$ para uma extensão $l(n) = 2\,n$, pode-se construir uma permutação de n bits que é unidireccional. Para isso, defina-se

$$G^{(0)}(y,n) = G(y,n) \restriction n \quad , \quad G^{(1)}(y,n) = G(y,n) \downharpoonright n \tag{47}$$

Isto significa que $G^{(0)}(y,n)$ captura os n bits mais à esquerda de $G(y,n)$ enquanto que $G^{(1)}(y,n)$ captura os n bits mais à direita.

Por recursividade primitiva, usando G_n como oráculo, define-se a função $f_n(y,k)$ como

$$f(0,y,n) \;=\; G^{(0)}(y,n) \tag{48}$$

$$f(2\,k,y,n) \;=\; G_n^{(0)}(f(k,y,n)) \quad , \quad f(2\,k+1,y,n) \;=\; G_n^{(1)}(f(k,y,n)) \tag{49}$$

A função $f_{n,y}(k)$ produz sempre resultados com n bits. Pode ser vista como uma função comandada por uma chave y que, ao receber um quelquer input k, vai percorrendo os bits da representação binária de k e aproveitando ora a metade esquerda de G_n ou a metade direita de G_n, em que G_n está aplicada ao anterior valor computado.

Desta forma $f_{n,y}$ pode ser aplicado a qualquer k e, por isso, normalmente, não preserva bits nem é uma permutação. No entanto

55 Teorema
Para cada y a função $k \mapsto f(k, y, n)$, com $|k| = n$, é unidireccional.

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 7: $\mathbb{Z}_5$

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

56 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). [16]

[16] 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**.

57 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 8:

$\mathbb{Z}_9^*$ 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

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

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

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

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

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

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

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

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

Tomando sempre congruências $\pmod p$, 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, $x = y$.

62 DEFINIÇÃO

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

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 9:

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

63 DEFINIÇÃO
O menor $t > 0$ *tal que* $a^t = 1 \pmod{m}$ *é a* ***ordem*** *do elemento* $a \in \mathbb{Z}_m^*$. *Se* $t \equiv \phi(m)$, a *diz-se* ***elemento primitivo*** *de* $\mathbb{Z}_m^*$

64 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.

65 DEFINIÇÃO
*Seja $p > 2$ um primo. O **símbolo de Legendre** $\left(\frac{a}{p}\right)$ é definido como sendo 0 se $p|a$, é igual a 1 se a é um resíduo quadrático módulo p e é igual a (-1) em caso contrário.*

*Se $m > 1$ tem uma factorização $p_1 p_2 \ldots p_n$ 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).

66 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*[17]*:*

- *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.*

[17] 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:

67 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 $v > 0$ tal que $\overbrace{1+1+\cdots+1}^{v \text{ vezes}} = 0$.

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

68 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***[18] $\mathbf{c} \in \mathbb{Z}_p[X]$ *de grau n tal que*

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

[18] 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*[19]*; seja* β *uma raíz de* $\mathbf{c}$ *em* K *que não seja raíz de nenhum polinómio* $(X^k - 1)$ *com* $k < m - 1$ [20]*.*

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* $n = \text{grau}(\mathbf{c})$ *, o elemento genérico desse corpo tem a forma*

$$a_0 + a_1\,\beta + a_2\,\beta^2 + \cdots + a_{n-1}\,\beta^{n-1} \quad \text{com} \quad a_i \in \mathbb{Z}_p \tag{50}$$

(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

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

[20] 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 (50), 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 (50).

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 10: 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[21]; 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 $1 - \beta$.

[21] 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 11: 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[22] 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 $\underbrace{\sigma \cdot \sigma \cdots \sigma}_{k \text{ vezes}}$.

69 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 $\mathbb{F}_p$.

[22] 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 (50), 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

70 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 $\mathrm{Gal}(L/K)$ 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 $\mathrm{Gal}(L/K)$ 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 $x \mapsto x^p$).

71 Definição

Sejam L e K corpos finitos como no facto 70. 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.

72 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.

73 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 12: O corpo finito $\mathbf{GF}(2^4)$ determinado pelo polinómio característico $\mathbf{c}[X] = (X^4 + X + 1)$ está representado na tabela 8 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 8: $\mathrm{GF}(2^4)$

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 71 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 (50) no facto 68

$$x_0 + x_1 \cdot \beta + \cdots + x_{n-1} \cdot \beta^{n-1} \qquad x_i \in \mathrm{GF}(2) \tag{51}$$

indica que o conjunto

$$\mathcal{B}_{\beta,0} = \left\{1 \ , \ \beta \ , \ \beta^2 \ , \ \cdots \ , \ \beta^{n-1}\right\} \tag{52}$$

forma uma base polinomial do tipo 0.

Um vector de bits $x^{\tilde{}} \in \mathrm{GF}(2)^n$ identifica (de forma única) um elemento $x \in \mathrm{GF}(2^n)$ através da representação (51).

Genericamente o operador $(\cdot)^{\tilde{}}$ *associa um emento* $x \in \mathrm{GF}(2^n)$ *à sua representação* $x^{\tilde{}} \in \mathrm{GF}(2)^n$ *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 $c = (c_0, c_1, \cdots, c_{n-1}) \in \mathrm{GF}(2)^n$ formado pelos coeficientes do polinómio característico para termos de ordem $< n$.

No exemplo 12 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

74 DEFINIÇÃO

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

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

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 12 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

75 FACTO

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

$$(y \cdot x)^{\tilde{}} \;=\; \bigoplus_{y_i=1} \mathbf{A}^i\, x^{\tilde{}} \tag{53}$$

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 (53), 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

$$\mathcal{B}_{\beta,1} = \left\{\beta \ , \ \beta^2 \ , \ \cdots \ , \ \beta^{n-1} \ , \ \beta^n\right\} \tag{54}$$

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

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

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

$$x = x_0 \cdot \rho + x_1 \cdot \rho^2 + x_2 \cdot \rho^4 + \cdots + x_{n-1} \cdot \rho^{2^{n-1}} \tag{55}$$

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 (55), 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 $\overset{\circlearrowright}{x^{\sim}}$.

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

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

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

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

A multiplicação

$$x \cdot y = \sum_{x_i=1} \sum_{y_j=1} \rho^{2^i} \cdot \rho^{2^j} \tag{56}$$

é bastante mais complicada e, genericamente, mais complexa nas bases normais do que nas bases polinomiais. A dificuldade reside no facto de (56) 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**.

76 FACTO

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

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

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

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

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, $\rho^p = 1$.

77 FACTO

Nas condições do facto 76 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*

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

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 (56): 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 13: 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 (56) 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 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 14: (**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 15: 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

78 FACTO

Se se verificar

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

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 $\log_g : G \to \mathbb{Z}_{|G|}$ (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

79 Definição
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[23] 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*

[23] 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.[24]
2. Define-se a sequência

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

[24] 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* $\sqrt{q}$.