Cours d’option : CODAGE

Eric Wegrzynowski
Caroline Fontaine

11 octobre 2002

Introduction

L'*informatique* est la science du traitement automatique de l'information. Ceci implique :

1. un traitement : interprétation des données, calcul (manipulation formelle), communication (transport d'information : réseau, satellite, téléphone portable, mais aussi CD) ;
2. automatique : sans intervention humaine ;
3. une représentation adéquate de l'information : par des symboles (lettres – alphabets latin et arabe – et chiffres – arabes et romains – par exemple).

Cette représentation est communément appelée *codage*, mais ses objectifs sont différents selon les contextes :

codage de source : son but est d'optimiser la place prise par les données, et débouche sur la notion de compression ;

codes correcteurs d'erreurs : leur but est de permettre la restitution des données après leur transmission (au sens large), et ce malgré les modifications qu'elles ont pu subir pendant cette transmission ; ils détectent et corrigent les erreurs.

cryptographie : confidentialité, authentification, signature …

On s'intéressera ici aux deux premiers points seulement.

Chapitre 1

Représentation des nombres

On regardera dans un premier temps le codage des nombres.

I Quelques systèmes de représentation des entiers naturels

Qu'est-ce qu'un nombre ? Il désigne une quantité. On peut lui associer diverses représentations à l'aide de *symboles*, appartenant à des *alphabets*.

Exemple 1.1 dix, X *et* 10 *sont trois représentations du nombre* 10 *par des symboles.*

dix *: avec l'alphabet latin* $\mathcal{A} = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$ *;*

X *: avec l'alphabet des chiffres romains* $\mathcal{A} = \{I, V, X, L, C, M, D\}$ *;*

10 *: avec l'alphabet des chiffres arabes* $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Ces représentations sont des mots *(des suites finies de symboles) définis sur les alphabets* $\mathcal{A}$*. De ces trois écritures, la dernière est la plus pratique pour effectuer des calculs.*

II Entiers naturels : bases de numération

1) Système décimal

C'est celui auquel on est le plus habitués : il utilise un alphabet de 10 chiffres et prend en compte la position de ces chiffres.

Exemple 1.2 *Le mot* 2387 *désigne le nombre* $2 \times 10^3 + 3 \times 10^2 + 8 \times 10 + 7$.

Tous les entiers naturels (positifs ou nuls) peuvent s'écrire de cette façon. Si, de plus, on s'interdit le chiffre 0 à gauche, alors tout entier naturel a une écriture unique.

2) Système de base b quelconque

Soit un entier $b > 1$.

Théorème 1.1 *Pour chaque entier $n \in \mathbb{N}$, $n \neq 0$, il existe un entier $p \geq 0$ et des entiers $a_0, a_1, \ldots, a_p$ compris entre 0 et $b-1$ (avec $a_p \neq 0$) tels que*

$$n = \sum_{k=0}^{p} a_k b^k = a_p b^p + a_{p-1} b^{p-1} + \ldots + a_2 b^2 + a_1 b + a_0.$$

De plus, cette somme est unique si on suppose $a_p \neq 0$.

preuve : (de l'existence de la décomposition)
On procède par récurrence sur l'entier n.
1) pour $1 \leq n \leq b-1$, on prend $p = 0$ et $a_0 = n$.
2) soit $n \geq b-1$, et supposons que le théorème est vérifié pour tout entier $1 \leq k \leq n$. Montrons qu'il est alors vérifié pour $n+1$. La division euclidienne de $n+1$ par b nous donne :

$$n + 1 = bq + r \qquad \text{avec} \qquad 0 \leq r < b.$$

Comme $b > 1$, on a $q < n+1$. D'autre part, $n \geq b-1$ entraîne $q > 0$. On a alors $1 \leq q \leq n$ et on peut appliquer l'hypothèse de récurrence sur q :

$$\text{il existe un entier } p' \text{ et des entiers } a'_0, a'_1, \ldots, a'_{p'} \text{ tels que } q = \sum_{k=0}^{p'} a'_k b^k.$$

On a alors

$$n + 1 = b \left(\sum_{k=0}^{p'} a'_k b^k \right) + r = \sum_{k=0}^{p'} a'_k b^{k+1} + r = \sum_{k=1}^{p'+1} a'_{k-1} b^k + r$$

On pose ensuite $p = p' + 1$ et

$$\begin{cases} a_0 = r \\ a_k = a'_{k-1} \text{ pour tout } 1 \leq k \leq p \end{cases}$$

On a alors la décomposition

$$n + 1 = \sum_{k=0}^{p} a_k b^k$$

ce qui clôt la démonstration. ♢

Exemple 1.3 *Prenons $b = 2$ et $n = 135$. On a*

135	2
$a_0 = 1$	67
	$a_1 = 1$

67	2
1	33
	$a_2 = 1$

33	2
1	16
	$a_3 = 0$

16	2
0	8
	$a_4 = 0$

8	2
0	4
	$a_5 = 0$

4	2
0	2
	$a_6 = 0$

2	2
0	$1 < 2$

Ceci signifie que

$$135 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0.$$

Si on convient de prendre tous les entiers compris entre 0 et $b-1$ comme symboles (chiffres), le nombre n s'écrit $a_p a_{p-1} \dots a_1 a_0$.

Exemple 1.4 *Le nombre 135 s'écrit en base 2 : 10000111. Pour éviter la confusion entres les écritures dans des bases différentes, on écrit :* $135_{10} = 10000111_2$

Algorithme de conversion :

`paramètres` : un entier $n \geq 0$ (à convertir)
un entier $b > 1$ (la base)
`résultat` : écriture de n dans la base b

Si $n < b$,	alors $[n]$	
	sinon	diviser n par b : $n = bq + r$ (avec $0 \leq r < b$) mettre r à droite de la représentation en base b de q

3) Taille du codage des entiers en base b

On souhaite déterminer le nombre de chiffres (symboles) permettant de représenter n en base b. On le note $|n|_b$.

Théorème 1.2 *Le nombre de chiffres dans l'écriture (propre) d'un entier $n \geq 1$ en base b est*

$$|n|_b = \lfloor \log_b n \rfloor + 1$$

Rappelons que $\lfloor x \rfloor$ désigne la valeur entière inférieure de x :

$$\lfloor x \rfloor = \max\{i \in \mathbb{Z}, i \leq x\},$$

et que le logarithme en base b de n, noté $\log_b n$ est défini par la relation :

$$\log_b n = x \iff n = b^x \ .$$

preuve : Tous les entiers n compris entre b^p (inclus) et b^{p+1} (exclus) sont représentés à l'aide d'exactement $p+1$ chiffres. Ceci signifie :

$$\forall\, n, b^p \leq n < b^{p+1} \Longleftrightarrow |n|_b = p+1 \tag{1.1}$$

Or on a également :

$$\begin{aligned} \forall\, n, b^p \leq n < b^{p+1} &\iff p\,\ln b \leq \ln n < (p+1)\ln b \\ &\iff p \leq \frac{\ln n}{\ln b} < p+1 \qquad \text{puisque } b > 1 \end{aligned}$$

Puisque $\log_b n = \frac{\ln n}{\ln b}$, on a

$$p \leq \frac{\ln n}{\ln b} < p+1 \Longrightarrow p = \lfloor \log_b n \rfloor \tag{1.2}$$

Des points (1.1) et (1.2) on déduit que $|n|_b = \lfloor \log_b n \rfloor + 1$.

♢

4) Bases les plus utilisées en informatique

binaire : base 2 $\rightarrow$ alphabet $\{0,1\}$, les symboles sont appelés *bits* (binary digits).

hexadécimal : base 16 $\rightarrow$ alphabet $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$ ($A=10, B=11, C=12, D=13, E=14, F=15$).

III Autres codages des entiers naturels

1) Décimal codé binaire (calculatrices de poche) : BCD (Binary Coded Decimal)

Le principe en est très simple : chaque chiffre décimal est représenté en binaire sur 4 bits :

chiffre décimal	codage binaire sur 4 bits
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Pour coder un nombre en BCD, il suffit de juxtaposer le codage en binaire de tous les chiffres de son écriture décimale.

Exemple 1.5 $1997_{10} = 0001\ 1001\ 1001\ 0111$

avantage : la conversion en décimal (usuel) est très facile.

inconvénient : on gaspille de la place ; sur l'exemple précédent, 1997 occupe 16 bits en BCD, alors qu'il n'en prend que 11 en binaire classique.

2) Biquinaire (1956 sur IBM 605)

Dans ce système, chaque chiffre décimal est codé sur 7 bits, dont 1 seul des 2 premiers vaut 1, et 1 seul des 5 derniers vaut 1 :

chiffre décimal	codage biquinaire
0	01 00001
1	01 00010
2	01 00100
3	01 01000
4	01 10000
5	10 00001
6	10 00010
7	10 00100
8	10 01000
9	10 10000

avantage : la conversion en décimal (usuel) est très facile, et on peut détecter les erreurs.

inconvénient : on gaspille de la place ; sur l'exemple précédent, 1997 occupe 28 bits en biquinaire, alors qu'il n'en prend que 11 en binaire classique.

IV Représentation des autres nombres

1) Entiers relatifs (signés)

Il existe plusieurs représentations.

a) Signe + valeur absolue

On prend la représentation binaire classique de la valeur absolue du nombre, et on rajoute devant un symbole pour coder son signe (0 pour le $+$, 1 pour le $-$).

Exemple 1.6 *codage de* $+23$, -23 *:* $0\ 1011_2$, $1\ 1011_2$.

inconvénient : le 0 est codé deux fois : $+0$ et -0.

b) Complément à b

Cette représentation s'effectue sur un nombre fixé n de symboles (ces symboles sont pris dans l'alphabet $\{0, \ldots, b-1\}$). Si le nombre à coder est positif ou nul, on le code simplement en base b. S'il est strictement négatif, on écrit la valeur absolue en base b classique, sur n symboles. Ensuite, chacun des symboles est remplacé par son complément à b (x est le complément à b de y si et seulement si $x + y = b - 1$). Enfin, on ajoute 1 à la représentation globale.

avantage : une seule représentation de 0.

Exemple 1.7 *1. on code -1890 en complément à 10 sur des nombres de 4 chiffres décimaux :*

	$1\ 8\ 9\ 0$
complément	
	$8\ 1\ 0\ 9$
ajout de 1	
	$8\ 1\ 1\ 0$

2. on code -6 en complément à 2 sur des nombres de 4 chiffres binaires :

	$0\ 1\ 1\ 0$
complément	
	$1\ 0\ 0\ 1$
ajout de 1	
	$1\ 0\ 1\ 0$

Remarque : lors du codage de la valeur absolue en binaire, le bit le plus à gauche (le plus significatif) doit être 0. Quand un entier relatif est codé avec ce principe (positif $\Rightarrow$ binaire, négatif $\Rightarrow$ complément à deux), ce bit désigne son signe : si c'est un 0 alors l'entier est positif et il suffit de lire la valeur en binaire, si c'est un 1 alors le nombre est négatif et il faut décoder le complément à deux.

2) Nombres fractionnaires (réels)

a) Virgule fixe

On utilise toujours le même nombre de chiffres après la virgule.

inconvénient : on ne peut pas coder de réels très proches de 0.

b) Virgule flottante

On représente les nombres de la manière suivante :

$$n = M \times B^E$$

où M (mantisse) et E (exposant) sont des entiers, et B (base) est un entier strictement positif.

Exemple 1.8 *Prenons l'exemple de la norme IEEE 754 :*

SM	E	M

– *simple précision :*

$$32\ \textit{bits} \begin{cases} SM & 1 & \textit{bits} & \\ E & 8 & \textit{bits} & -128\ ..\ 127 \\ M & 23 & \textit{bits} & 0\ ..\ 2^{23}-1 \end{cases}$$

– *double précision :*

$$64\ \textit{bits} \begin{cases} SM & 1 & \textit{bits} & \\ E & 11 & \textit{bits} & -1024\ ..\ 1023 \\ M & 52 & \textit{bits} & 0\ ..\ 2^{52}-1 \end{cases}$$

Chapitre 2

Codes, codages et décodages

Après la représentation des nombres, nous nous intéressons ici à la notion de codage, au sens général du terme : nous allons voir comment représenter un suite de symboles de manière adéquate, selon l'utilisation que l'on souhaite en faire.

En effet, on peut vouloir la coder pour :
- la transmettre directement à quelqu'un d'autre : on choisira alors une représentation utilisant les lettres de l'alphabet usuel, dans une langue commune (comme par exemple le français ou l'anglais) ;
- la stocker sur un ordinateur : on essaiera de minimiser la place n'écessaire à ce stockage ;
- la transmettre d'un ordinateur à un autre : on essaiera alors de minimiser les risques de perte d'information lors de cette transmission, en introduisant de la redondance.

Ces cas ont un point commun : ils reposent tous sur la donnée d'un alphabet $\mathcal{A}$. On définit ensuite un code qui permet de représenter l'information à l'aide des lettres (au sens large) de l'alphabet $\mathcal{A}$.

Voici, à titre d'exemples, quelques codes célèbres :

1837 : code Morse pour le télégraphe ; alphabet $\{\cdot, -, silence\}$
$A : \cdot -$ $B : - \cdot\cdot\cdot$ $C : - \cdot - \cdot$ $D : - \cdot\cdot$ $E : \cdot$ etc $S : \cdot\cdot\cdot$ etc

1917 : code Baudot pour le télex, réseau télégraphique commuté (CCITT n°2) ; code sur 5 bits $\Rightarrow$ 32 mots binaires ; mais 2 jeux de caractères sont codés à la fois (lettre, figure), et on dispose de 2 caractères permettant de commuter entre les deux types de caractères ; on peut donc coder au total 60 caractères différents.
A et $-$: 00011, B et $?$: 11001, ...

1963 : code ASCII (American Standard Code for Information Interchange) (ISO 646, CCITT n°5) ; code sur 7 bits $\Rightarrow$ 128 caractères
$A : 1000001$, $B : 1000010$, $C : 1000011$, ...
Le code ASCII a été étendu à plusieurs codes sur 8 bits $\Rightarrow$ 256 caractères. Par exemple ISO8859 : $A : 0100001$, ...

1991 - . . . : UNICODE Actuellement, on est en train de définir l'UNICODE[1] (ISO 10646), sur 16 bits $\Rightarrow 65536$ caractères possibles (on en a défini environ 50000 pour l'instant).

I Alphabets et mots

Pour représenter l'information, on utilise des symboles ou lettres dont l'ensemble forme un alphabet.

1) Alphabet

Définition 2.1 *Un* alphabet *est un ensemble fini non vide. Ses éléments sont nommés* lettres *ou* symboles.

Exemples :

1. L'alphabet latin composé des 26 lettres usuelles $\mathcal{A} = \{a, b, c, \ldots, z\}$.
2. L'alphabet des 10 chiffres décimaux $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
3. L'alphabet des deux symboles binaires $\mathcal{A} = \{0, 1\}$.
4. L'alphabet des 16 chiffres hexadécimaux $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$.
5. L'alphabet des chiffres romains $\mathcal{A} = \{I, V, X, L, C, D, M\}$.
6. L'alphabet du Morse $\mathcal{A} = \{., -, \text{silence}\}$ [2].

2) Mots

Les lettres d'un alphabet permettent de construire des *mots*.

Définition 2.2 *On appelle* mot *sur un alphabet $\mathcal{A}$ toute suite finie de lettres de $\mathcal{A}$. Le mot n'ayant aucune lettre est nommé* mot vide*, et noté ε.*

Exemples :

1. `codage` et `qwixusd` sont deux mots construits avec les lettres de l'alphabet latin.
2. L'écriture décimale d'un nombre entier est un mot construit avec les chiffres décimaux. `2002` est un mot.

[1] http ://www.unicode.org

[2] Pourquoi faut–il considérer qu'il y a un troisième symbole (souvent oublié) dans le Morse ?

3) Longueur d'un mot

Définition 2.3 *La* longueur *d'un mot est le nombre de lettres qui le composent.*

On note $|u|$ la longueur d'un mot u.

Exemples

1. $|\varepsilon| = 0$
2. $|\texttt{qwixusd}| = 7$
3. $|\texttt{2002}| = 4$

4) Ensembles de mots construits sur un alphabet

Définition 2.4 *Étant donné un alphabet $\mathcal{A}$ et un entier naturel n, on définit :*

$\mathcal{A}^n$ = *ensemble des mots de longueur n construits avec les lettres de $\mathcal{A}$*
$\mathcal{A}^+$ = *ensemble des mots non vides*
$\mathcal{A}^*$ = *ensemble de tous les mots*

Autrement dit,

$$\mathcal{A}^+ = \bigcup_{n\geq 1} \mathcal{A}^n \qquad \text{et} \qquad \mathcal{A}^* = \bigcup_{n\geq 0} \mathcal{A}^n = \mathcal{A}^+ \bigcup \{\varepsilon\}$$

Représentation arborescente de $\mathcal{A}$: On peut donner une représentation arborescente d'un ensemble de mots, bien utile par la suite. La figure 2.1 montre une telle représentation dans le cas où l'alphabet n'a qu'une seule lettre `a` (figure de gauche) et dans le cas où il a deux lettres `0,1` (figure de droite).

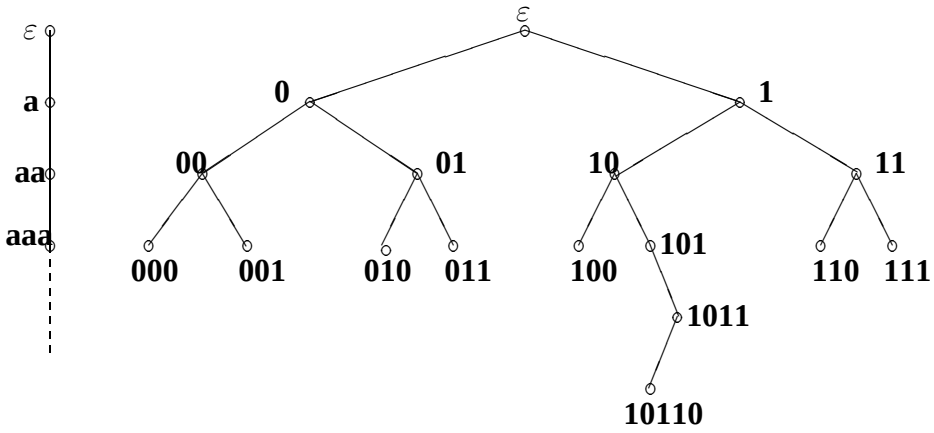


FIG. 2.1 – Arbres des mots pour un alphabet à une et à deux lettres

Nombre de mots : Le nombre de mots sur un alphabet est évidemment infini. Mais pour une longueur donnée, on se convainc aisément que pour un alphabet $\mathcal{A}$ ayant q lettres on a

$$\mathrm{card}(\mathcal{A}^n) = q^n$$

Ainsi, il y a 2^n mots sur un alphabet binaire.

5) Langages

Définition 2.5 *Étant donné un alphabet $\mathcal{A}$, on appelle* langage *toute partie de $\mathcal{A}^*$.*

6) Opération de concaténation

On peut définir une opération de *concaténation* sur les mots. Cette opération consiste à mettre bout à bout les deux mots.

Définition 2.6 *Etant donnés deux mots u et v sur un alphabet $\mathcal{A}$, on définit le mot* concaténé*, noté $u.v$, par*

$$u.v = a_1 \dots a_p b_1 \dots b_q$$

si les deux mots u et v s'écrivent

$$\begin{array}{rcll} u & = & a_1 \dots a_p & a_i \in \mathcal{A} \\ v & = & b_1 \dots b_q & b_j \in \mathcal{A} \end{array}$$

Exemples

1. si $u =$ `timo` et $v =$ `leon`, alors $u.v =$ `timoleon`.
2. si $u =$ `10` et $v =$ `110`, alors $u.v =$ `10110`.

Définition 2.7 *Lorsqu'un mot u peut s'écrire sous forme d'une concaténation de mots u_i ($u = u_1.u_2.\dots.u_n$), on dit que les u_i sont des* facteurs *de u et que $u_1.u_2.\dots.u_n$ est une* factorisation *de u.*

Propriétés de l'opération de concaténation

- associativité : $\forall u, v, w \in \mathcal{A}^*,\ (u.v).w = u.(v.w)$
- élément neutre ε : $\forall u \in \mathcal{A}^*,\ \varepsilon.u = u.\varepsilon = u$
- longueur : $\forall u, v \in \mathcal{A}^*,\ |u.v| = |u| + |v|$

Remarque : L'opération de concaténation confère à l'ensemble des mots sur un alphabet une structure algébrique nommée *monoïde*.

7) Préfixe d'un mot

Définition 2.8 *On dit que v est un* préfixe *(ou* facteur gauche*) de u s'il existe un mot w tel que $u = v.w$; on désigne par $Pre(u)$ l'ensembe des préfixes de u.*

Exemple : les préfixes du mot $u = \texttt{10110}$ sont au nombre de 6 :

$$Pre(u) = \{\varepsilon, \texttt{1}, \texttt{10}, \texttt{101}, \texttt{1011}, \texttt{10110}\}$$

L'ensemble des préfixes d'un mot est l'ensemble des mots que l'on rencontre le long de la branche de l'arbre des mots menant de la racine au mot considéré.

On remarquera que l'on a toujours $\mathrm{card}(Pre(u)) = |u| + 1$.

II Codage

Abordons maintenant la notion centrale de ce chapitre, celle de codage. Intuitivement un codage est l'association de chaque symbole d'un alphabet de départ ou alphabet *source* à un mot d'un alphabet *cible*.

On notera dans tout ce qui suit $\mathcal{S}$ l'alphabet source, et $\mathcal{A}$ l'alphabet cible.

Exemples :

1. le code Morse associe à chaque lettre de l'alphabet latin ($\mathcal{S}$) un mot construit sur l'alphabet $\mathcal{A} = \{., -, \mathrm{silence}\}$.
2. le code ASCII associe à chaque caractère ASCII un mot binaire de 8 bits.

Mais il ne suffit pas de convenit d'une association d'un mot sur l'alphabet cible à chaque lettre de l'alphabet source pour obtenir un codage.

Exemple Prenons $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$, $\mathcal{A} = \{0, 1\}$ et considérons les 2 tables suivantes :

x	$c_1(x)$	$c_2(x)$
s_1	`1`	`1`
s_2	`00`	`00`
s_3	`01`	`01`
s_4	`01`	`10`

Il est évident que l'association c_1 ne peut convenir pour un codage étant donné qu'aux deux lettres s_3 et s_4 est associé un même mot : `01`.

Moins évident est le cas de l'association c_2. Néanmoins, on peut facilement vérifier que les deux mots $s_1s_2s_1$ et s_4s_3 seraient "codés" en le même mot : `1001`. Comment alors "décoder" ce mot ? Par conséquent c_2 ne peut convenir pour un codage.

1) Morphisme

Dans le dernier exemple ci–dessus, il a été implicitement admis qu'une fois établis les mots u_1 et u_2 associés à deux lettres x_1 et x_2, alors le mot associé au mot x_1x_2 est $u_1.u_2$. Cela nous amène à considérer une classe particulière de transformations de mots constituée des *morphismes*.

Définition 2.9 *Une application $c : \mathcal{S}^* \longrightarrow \mathcal{A}^*$, qui à un mot de $\mathcal{S}^*$ associe un mot de $\mathcal{A}^*$, est un* morphisme *si elle vérifie les deux conditions suivantes*

1. $c(\varepsilon) = \varepsilon$
2. $\forall u, v \in \mathcal{S}^* \qquad c(u.v) = c(u).c(v)$

Autrement dit un morphisme est une application qui conserve l'opération de concaténation : l'image de la concaténée de deux mots est la concaténée de leurs images.

Un morphisme est entièrement déterminé si on connaît l'image de chacune des lettres de l'alphabet source.

Proposition 2.1 *Étant donnés un alphabet source $\mathcal{S} = \{s_1, \ldots, s_n\}$ et n mots $u_1, \ldots, u_n$ sur l'alphabet cible $\mathcal{A}$, il existe un unique morphisme $c : \mathcal{S}^* \longrightarrow \mathcal{A}^*$ tel que*

$$\forall i \in \{1, \ldots, n\} \qquad c(s_i) = u_i$$

2) Codage

Définition 2.10 *Un* codage *des mots sur un alphabet source $\mathcal{S}$ en des mots d'un alphabet cible $\mathcal{A}$ est un morphisme $c : \mathcal{S}^* \longrightarrow \mathcal{A}^*$ injectif. C'est–à–dire*

$$\forall u, v \in \mathcal{S}^* \qquad u \neq v \Rightarrow c(u) \neq c(v)$$

Un codage étant un morphisme, il est entièrement déterminé par le langage des mots associés aux lettres de l'alphabet source. Ce langage est appelé *code* associé.

Tout langage n'est pas forcément un code.

Définition 2.11 *Un langage C est un* code*, s'il n'existe pas de mot ayant deux factorisations distinctes avec des mots de C.*

Exemples :

1. le langage vide $C = \emptyset$ est un code ! (mais peu utile pour coder de l'information).
2. $C_2 = \{\mathtt{1}, \mathtt{00}, \mathtt{10}, \mathtt{01}\}$ n'est pas un code.
3. Tout langage ne contenant que des mots de même longueur est un code qualifié de code de *longueur fixe*.

Remarques :

1. Un code ne peut pas contenir le mot vide.
2. Tout sous-ensemble d'un code est un code.
3. Cette définition des codes n'impose pas qu'un code soit un ensemble fini. Par exemple le langage $C = \{\mathtt{01}^n \mid n \in \mathbb{N}\}$ est un code infini. Mais nous ne considèrerons que des codes finis dans ce cours.

3) Décodage

Coder un mot u, c'est donc trouver le mot $v = c(u)$ qui lui est associé par un codage c.

Inversement, *décoder* un mot v, c'est retrouver le mot de départ u, i.e. son antécédant par le codage c. Pour cela il suffit de trouver une factorisation de v avec des mots du code $C = c(\mathcal{S})$. Deux cas peuvent se présenter :

- v se factorise et alors cette factorisation est unique, puisque C est un code.
- v ne se factorise pas dans C. Cela signifie que v n'est pas un mot obtenu par codage d'un autre mot.

L'opération de décodage est une opération plus ou moins simple selon la nature des codes utilisés.

III Cas particuliers de codes

On l'a vu, n'importe quel langage n'est pas un code. La question naturelle qui se pose est alors : existe-t-il un algorithme qui permet de décider si oui ou non un langage est un code ?

L'algorithme de Sardinas-Patterson donne une réponse affirmative à cette question, comme nous le verrons plus loin.

Pour l'instant nous allons examiner quelques cas particuliers de codes facilement identifiables.

1) Codes de longueur fixe

On l'a déjà signalé, tout langage de mots ayant tous la même longueur est un code. Par exemple, le langage $C_3 = \{\texttt{010}, \texttt{100}, \texttt{110}, \texttt{101}\}$ est un code. Il permet de coder tout alphabet source de 4 lettres. Il est facile de voir qu'avec des mots binaires de longueur 3, on peut coder tout alphabet source dont le nombre de lettres n'excède pas 8.

Le nombre total de mots de longueur n sur un alphabet à q lettres est q^n. Pour un alphabet cible à q lettres, il est donc possible de coder les lettres d'un alphabet source $\mathcal{S}$ avec des mots de longueur n, à condition que $\mathrm{card}(\mathcal{S}) \leq q^n$.

Exemples : De nombreux codages utilisés en informatique utilisent des codes de longueur fixe : le code ASCII, l'UNICODE, les codages bitmap d'images.

2) Codes à "virgule"

Considérons le langage binaire $C_4 = \{\texttt{0}, \texttt{01}, \texttt{011}, \texttt{0111}\}$. Ces quatre mots partagent la propriété remarquable de tous commencer par la lettre `0`, et de ne la contenir q'une seule fois. Ils ne se distinguent que par le nombre de `1` qui suivent. Il s'ensuit que

toute concaténation de ces quatre mots ne peut être obtenue que d'une seule manière, et donc C_3 est un code.

Pour les mêmes raisons le langage $C'_4\{\mathtt{0}, \mathtt{10}, \mathtt{110}, \mathtt{1110}\}$ est un code.

Dans ces deux exemples, le rôle joué par la lettre `0` est celui d'un délimiteur : soit un marqueur de début, soit un marqueur de fin de mot du code.

De manière générale, on peut transformer tout langage en un code en rajoutant en tête ou en queue de chaque mot une nouvelle lettre n'apparaissant dans aucun des mots.

Proposition 2.2 *Soient $\mathcal{A}$ un alphabet, $\sharp$ une lettre n'appartenant pas à $\mathcal{A}$ et $\mathcal{A}' = \mathcal{A} \cup \{\sharp\}$. Pour tout langage $C \subset \mathcal{A}^*$, $C.\sharp$ est un code sur l'alphabet $\mathcal{A}'$.*

De tels codes sont appelés codes *à virgule*.

Exemples :

1. $C_2\sharp = \{\mathtt{1}\sharp, \mathtt{00}\sharp, \mathtt{10}\sharp, \mathtt{01}\sharp\}$ est un code.
2. le Morse est un code à virgule, les silences en faisant office.

3) Codes préfixes

Étudions maintenant le langage $C_5 = \{\mathtt{0}, \mathtt{11}, \mathtt{100}, \mathtt{101}\}$. Il n'est ni de longueur fixe, et aucun symbole ne sert de délimiteur.

Pourtant il jouit d'une bonne propriété : aucun mot n'est un préfixe d'un autre mot de l'ensemble. De tels ensembles sont qualifiés de *préfixes*.

Définition 2.12 *Un langage $C \subset \mathcal{A}^*$ est dit* préfixe *s'il satisfait à la condition (dite du préfixe) :*

$$\forall\, u, v \in C, u \neq v, u \notin Pre(v) \text{ et } v \notin Pre(u)$$

Exemple 2.1 *C_5 est préfixe.*

Proposition 2.3 *Tout langage préfixe de mots est un code.*

preuve : Il suffit de prouver qu'un langage qui n'est pas un code ne peut pas être préfixe.

Considérons donc un langage C qui n'est pas un code et un mot $w \in \mathcal{A}^*$ ayant deux factorisations dans C :

$$\begin{array}{rcll} w & = & u_1.u_2 \ldots u_p & \text{avec } u_1, \ldots, u_p \in C \\ & = & v_1.v_2 \ldots v_q & \text{avec } v_1, \ldots, v_q \in C \end{array}$$

On peut ici supposer sans perte de généralité que $u_1 \neq v_1$ (si ce n'est pas le cas, soit ℓ le premier indice pour lequel $u_\ell \neq v_\ell$; on supprime des deux factorisations les premiers termes $u_1 = v_1$, …, $u_{\ell-1} = v_{\ell-1}$).

Alors, comme $u_1 \neq v_1$, on a $|u_1| \neq |v_1|$, et donc

$$\begin{array}{lll} \text{soit} & |u_1| < |v_1| & \text{et alors } u_1 \text{ est préfixe de } v_1 \\ \text{soit} & |u_1| > |v_1| & \text{et alors } v_1 \text{ est préfixe de } u_1 \end{array}$$

et on peut conclure que le code n'est pas préfixe. ♢

On peut remarquer que la réciproque de cette proposition n'est pas vraie (voir le code C_4).

IV Algorithme de Sardinas-Patterson

Il existe des codes qui ne sont pas de longueur fixe, à virgule ou préfixe. Il en est ainsi de $C_6 = \{\mathtt{00}, \mathtt{01}, \mathtt{110}, \mathtt{001}\}$. Ce langage n'entre pas dans l'une des trois catégories citées, et pourtant c'est un code. S'il ne l'était pas, on pourrait trouver un mot w ayant deux factorisations avec des mots de C_6 :

$$w = u_1.u_2.\ldots.u_n = v_1.v_2.\ldots.v_m$$

où n et m sont deux entiers (non nuls) et les u_i et v_i des mots de C.

Si $u_1 \neq v_1$, alors nécessairement l'un des deux est un préfixe de l'autre. Supposons que c'est u_1 qui est un préfixe de v_1. En observant les mots de C_6, on trouve alors que $u_1 = \mathtt{00}$ et $v_1 = \mathtt{001}$.

La dernière lettre de v_1 étant un $\mathtt{1}$, u_2 doit commencer par cette lettre et donc $u_2 = \mathtt{110}$. Finalement, le mot v_2 doit commencer par $\mathtt{10}$ et C_6 ne contient pas de tel mot.

En conséquence les deux mots u_1 et v_1 sont nécessairement égaux. De la même manière on doit avoir $n = m$ et $u_i = v_i$ pour tout i, ce qui prouve que w ne peut pas avoir deux factorisations distinctes.

Il n'est pas toujours aussi facile de traiter "à la main" l'analyse d'un langage pour décider s'il est ou non un code. La suite de cette section établit un algorithme pour le faire.

1) Résiduel

Définition 2.13 *Soit $L \subset \mathcal{A}^*$ un langage et $u \in \mathcal{A}^*$ un mot. On appelle* langage résiduel à gauche *de L par u, le langage noté $u^{-1}.L$ défini par*

$$u^{-1}.L = \{v \in \mathcal{A}^* \mid \exists w \in L \text{ tq } w = u.v\}$$

Le résiduel (à gauche) d'un langage L par u est donc constitué des mots de L dont u est un préfixe, et desquels on a supprimé ce préfixe.

Exemples : Avec $L = \{1, 001, 1001, 1011\}$, $u_1 = 01$, $u_2 = 10$ et $u_3 = 1$, on a

1. $u_1^{-1}.L = \emptyset$ car aucun mot de L ne commence par u_1.
2. $u_2^{-1}.L = \{01, 11\}$ car u_2 est le début des deux mots 1001 et 1011.
3. $u_3^{-1}.L = \{\varepsilon, 001, 011\}$ car u_3 est le début des trois mots 1, 1001 et 1001.

Remarques :

1. $\varepsilon \in u^{-1}.L \iff u \in L$
2. si u n'est le préfixe d'aucun mot de L, $u^{-1}.L$ est vide.

2) Langage quotient

Définition 2.14 *Soit L et M deux langages. On appelle quotient à gauche de L par M le langage noté $M^{-1}.L$ et défini par*

$$M^{-1}.L = \bigcup_{u \in M} u^{-1}.L$$

Autrement dit $M^{-1}.L$ est la réunion de tous les résiduels à gauche de L par les mots de M. $M^{-1}.L$ désigne donc en quelque sorte les suffixes (la fin) des mots de L qui ont comme préfixe un mot de M.

Exemple : en reprenant l'exemple précédent, et en posant $M = \{u_1, u_2, u_3\}$, on a

$$M^{-1}.L = \{\varepsilon, 01, 11, 001, 011\}$$

Remarques :

1. $M^{-1}.L$ est vide si et seulement si aucun mot de M n'est un préfixe d'un mot de L. En particulier, si L est un langage préfixe, $L^{-1}.L$ est vide.
2. $\varepsilon \in M^{-1}.L$ si et seulement si L et M contiennent un même mot. En particulier, pour tout langage non vide L, on a $\varepsilon \in L^{-1}.L$.

3) L'algorithme

données :	un langage L sur un alphabet $\mathcal{A}$
but :	décider si L est un code ou non.

On calcule les ensembles L_n suivants :

$$\begin{array}{lcll} L_0 & = & L & \\ L_1 & = & L^{-1}.L \setminus \{\varepsilon\} & \\ L_n & = & L^{-1}.L_{n-1} \cup L_{n-1}^{-1}.L & \text{pour } n \geq 2 \end{array}$$

jusqu'à ce que l'on retombe sur un L_n déjà calculé précédemment, ou que le mot vide ε appartienne au L_n que l'on vient de calculer.
Si ε appartient à l'un des L_i, alors le code n'est pas décodable de manière unique, sinon c'est que le code est décodable de manière unique.

Cet algorithme s'arrête, car le nombre d'ensembles L_i distincts que l'on peut calculer est fini (chacun d'eux est une partie de l'ensemble des suffixes de L, qui lui-même est fini).

Exemples : Considérons l'alphabet $\mathcal{A} = \{0, 1\}$.

1. Prenons le langage C_2 défini précédemment :

   $$L = C_2 = \{\mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}\}$$

   On a :

   $$\begin{array}{lcl} L_0 & = & \{\mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}\} \\ L_1 & = & \{\mathtt{0}\} \\ L_2 & = & \{\mathtt{0}, \mathtt{1}\} \\ L_3 & = & \{\varepsilon, \mathtt{0}, \mathtt{1}\} \end{array}$$

   On s'arrête car $\varepsilon \in L_3$. On en conclut que C_2 n'est pas un code (ex : 101 peut être interprété comme $10\ 1$ ou $1\ 01$).

2. Prenons le langage C_5 défini précédemment :

   $$L = C_5 = \{\mathtt{0}, \mathtt{11}, \mathtt{100}, \mathtt{101}\}$$

   On a :

   $$\begin{array}{lcll} L_0 & = & \{0, 11, 100, 101\} & \\ L_1 & = & \emptyset & \\ L_n & = & \emptyset & \forall n \geq 2 \end{array}$$

   On s'arrête car $L_2 = L_1$. Aucun des ensembles L_i ne contient ε, et on en conclut que C_3 est un code.

3. Prenons le langage :

   $$L = \{\mathtt{000}, \mathtt{010}, \mathtt{011}, \mathtt{01001}\}$$

   On a :

   $$\begin{array}{lcl} L_0 & = & \{\mathtt{000}, \mathtt{010}, \mathtt{011}, \mathtt{01001}\} \\ L_1 & = & \{\mathtt{01}\} \\ L_2 & = & \{\mathtt{0}, \mathtt{1}, \mathtt{001}\} \\ L_3 & = & \{\mathtt{00}, \mathtt{10}, \mathtt{11}, \mathtt{1001}\} \\ L_4 & = & \{\mathtt{0}\} \\ L_5 & = & \{\mathtt{00}, \mathtt{10}, \mathtt{11}, \mathtt{1001}\} \end{array}$$

On s'arrête car $L_5 = L_3$. Aucun des ensembles L_i ne contient ε, et on en conclut que L est un code.

V Inégalité de Kraft

La longueur des mots d'un code peut–elle être arbitrairement petite ? Évidemment non. Un code ne peut pas contenir "trop" de "petits" mots. Par exemple, un code ne peut pas contenir toutes les mots d'une lettre et d'autres mots.

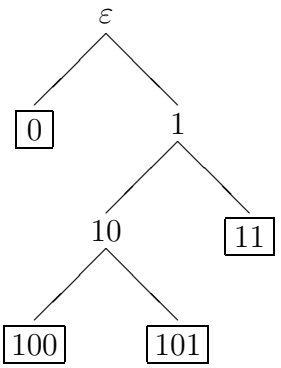
Les deux théorèmes qui suivent donnent une réponse précise à cette question.

1) Représentation graphique d'un code préfixe

Un code préfixe est représentable par un arbre dont les feuilles sont les mots du code.

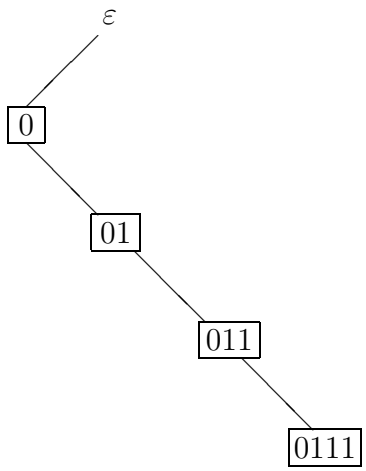
Exemples :

1. le code préfixe C_5 est représenté par l'arbre suivant :



   Il y a au maximum un mot du code (mots encadrés) par branche issue de la racine de l'arbre.

2. en revanche pour le code C_4, qui n'est pas préfixe, on s'aperçoit que certaines branches de l'arbre contiennent plusieurs mots du code



2) Inégalité de Kraft

Théorème 2.1 (Inégalité de Kraft) *Soit un alphabet $\mathcal{A}$ de q lettres et n un entier naturel non nul. Il existe un code préfixe contenant n mots de $\mathcal{A}^*$, de longueurs respectives $\ell_1, \ell_2, \ldots, \ell_n$, si et seulement si l'inégalité suivante, appelée* inégalité de Kraft, *est vérifiée :*

$$\sum_{i=1}^{n} q^{-\ell_i} \leq 1$$

preuve : on peut supposer sans perte de généralité (quitte à les considérer dans un autre ordre) que $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_n$.

1 - Condition nécessaire : il existe un code préfixe $\Rightarrow$ inégalité.

Le mot le plus long est de longueur ℓ_n. L'arbre de tous les mots de $\mathcal{A}^*$ de longueur $\leq \ell_n$ a une hauteur ℓ_n, et contient q^{ℓ_n} feuilles.

Soit C un code préfixe. Les mots de C sont des nœuds de cet arbre.

Comme C est préfixe, il n'y a pas deux mots de C sur la même branche de l'arbre. Ainsi, quand on a situé un mot de C dans l'arbre, on sait que tous les feuilles situées dessous ne feront pas partie de l'arbre de C. Si ce mot est de longueur ℓ_i, on supprime ainsi $q^{\ell_N - \ell_i}$ feuilles de l'arbre global. Au total, on en élimine donc ainsi $\sum_{i=1}^{n} q^{\ell_N - \ell_i}$.

Ce nombre de feuilles éliminées ne peut pas être plus grand que le nombre total de feuilles, q^{ℓ_N}. Ceci signifie qu'on a forcément

$$\sum_{i=1}^{n} q^{\ell_N - \ell_i} \leq q^{\ell_N}$$

et donc, en divisant pas q^{ℓ_N},

$$\sum_{i=1}^{n} q^{-\ell_i} \leq 1$$

2 - Condition suffisante : inégalité $\Rightarrow$ il existe un code préfixe.

On considère l'arbre de $\mathcal{A}^*$. On choisit un nœud à la profondeur ℓ_1, et on élimine toutes les feuilles situés en-dessous. Comme l'inégalité du théorème est vérifiée, on a $q^{-\ell_1} < 1$, et donc $q^{\ell_N - \ell_1} < q^{\ell_N}$.

Il reste donc des chemins de longueur ℓ_N. On choisit un nœud à la profondeur ℓ_2, et on recommence ... *etc*. ♢

Ce résultat donne une condition nécessaire et suffisante pour l'existence d'un code préfixe, mais il faut être prudent : ce n'est pas parce que les longueurs des mots d'un ensemble donné satisfont l'inégalité de Kraft que cet ensemble vérifie la propriété d'être préfixe.

La CNS énoncée est en fait une CNS pour l'existence d'un code (non nécessairement préfixe) de n mots de longueurs $\ell_1, \ell_2, \ldots, \ell_N$. Ce résultat sera admis ici et a été formulé par Mac Millan :

Théorème 2.2 (MacMillan) *Soit un alphabet $\mathcal{A}$ de q lettres et n un entier naturel non nul. Il existe un code contenant n mots de $\mathcal{A}^*$, de longueurs respectives $\ell_1, \ell_2, \ldots, \ell_n$, si et seulement si l'inégalité suivante est vérifiée :*

$$\sum_{i=1}^{n} q^{-\ell_i} \leq 1$$

La seule différence existant entre l'énoncé de ce théorème et celui du théorème précédent est dans la nature du code considéré : code préfixe dans le premier théorème, code quelconque dans celui–ci.

Là encore, ce théorème donne une CNS sur l'*existence* d'un code ayant des mots de longueurs données. Il permet simplement de dire, pour un langage donné, que :

- si les longueurs des mots ne satisfont pas l'inégalité, alors ce langage n'est pas un code.
- si les longueurs des mots satisfont l'inégalité, alors ce langage peut être un code, mais n'en est pas nécessairement un. Il faut alors utiliser l'algorithme de Sardinas-Patterson pour trancher.

Exemple : Soit le langage $C = \{0, 10, 1101, 1110, 1011, 110110\}$ défini sur $\mathcal{A} = \{0, 1\}$. On a $q = 2, n = 6, \ell_1 = 1, \ell_2 = 2, \ell_3 = \ell_4 = \ell_5 = 4, \ell_6 = 6$. Donc on a

$$\sum_{i=1}^{n} q^{-\ell_i} = \frac{1}{2} + \frac{1}{4} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16} + \frac{1}{64} = \frac{61}{64} < 1$$

Ce langage satisfait don l'inégalité de Kraft, et d'après le théorème de Mc-Millan il peut être un code. En est–il un ? On applique l'algorithme de Sardinas-Patterson :

$$\begin{aligned} L_0 &= \{0, 10, 1101, 1110, 1011, 110110\} \\ L_1 &= \{11, 10\} \\ L_2 &= \{\varepsilon, \ldots\} \end{aligned}$$

Par conséquent, L n'est pas un code. On peut voir, en analysant le déroulement de cet algorithme que l'ambiguïté concerne des suites de lettres du type $101101\ldots$

Chapitre 3

Codage optimal

Nous allons nous intéresser ici à l'otimisation de l'espace occupé par les données codées. Nous allons donc étudier la longueur des mots de code, et comprendre comment construire des codes efficaces (qui minimisent la longueur moyenne de ces mots).

Dans ce chapitre, et dans toute la suite du cours, on ne s'intéressera qu'aux codes décodables. En effet, quel est l'intérêt d'utiliser un codage non décodable ? On peut d'ailleurs remarquer que dans la littérature la notion de "code" sous-entend "décodable".

Une première remarque, à la lecture du théorème de MacMillan, est que si on utilise des mots de code longs, l'inégalité est plus facilement vérifiée ; donc, il y a plus de chance qu'un code décodable de manière unique existe si l'on autorise l'utilisation de mots de code longs. D'un autre côté, si on utilise trop de mots longs, alors la longueur du message codé va être très grande, et son stockage ne sera pas aisé. On sent donc qu'il faut trouver un compromis : utiliser des mots longs, mais pas trop.

Mais comment faire pour évaluer ce compromis, et le réaliser au mieux ? On remarque que les symboles de la source n'apparaissent pas tous avec la même fréquence pour un ensemble donné de messages (par exemple si on considère les textes écrits en français). On est donc tenté d'utiliser des mots de code courts pour coder les symboles les plus fréquents, diminuant ainsi la longueur moyenne des mots de code, et améliorant l'efficacité du code.

Nous allons ici construire une certaine classe de codes optimaux.

I Longueur moyenne

On associe à chaque symbole de la source $\mathcal{S}$ une *fréquence* d'apparition dans un message source. Il s'agit en fait d'une probabilité d'apparition.

Définition 3.1 *Une* mesure de probabilité, *ou* distribution de fréquences, *est une application de la forme*

$$\begin{array}{rccc} f: & \mathcal{S} & \longrightarrow & [0,1] \subset \mathbb{R} \\ & x & \longmapsto & f(x) \end{array}$$

avec la propriété que

$$\sum_{x \in \mathcal{S}} f(x) = 1 \, .$$

Connaissant la distribution des fréquences d'apparition des symboles de la source $\mathcal{S}$, et étant donné un codage C défini sur $\mathcal{S}$, on peut se poser la question : ¡¡Quelle est en moyenne, la longueur du codage d'un symbole de la donnée source ?¿¿.

Définition 3.2 *Soit C un code, $C : \mathcal{S} \longrightarrow \mathcal{A}^*$. La* longueur moyene *d'un mot de code, relativement à la distribution f de la source, est*

$$\overline{n} = \sum_{x \in \mathcal{S}} f(x) \, |C(x)| \, .$$

Remarquons qu'en toute rigueur cette longueur moyenne dépend du code considéré et de la distribution des fréquences de la source ; on devrait donc utiliser une notation du type $\overline{n_{C,f}}$. Cependant, on utilise en général une notation allégée lorsqu'il n'y a pas d'ambiguïté au sujet du code ou de la distribution des fréqences considérés.

Exemple 3.1 *Soit la source $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$, avec la distribution de fréquences suivante :*

x	$f(x)$
s_1	$0,3$
s_2	$0,3$
s_3	$0,25$
s_4	$0,15$

On a de manière générale pour un codage C défini sur $\mathcal{S}$:

$$\overline{n} = 0,3 \, |C(s_1)| + 0,3 \, |C(s_2)| + 0,25 \, |C(s_3)| + 0,15 \, |C(s_4)|.$$

Reprenons maintenant trois des codages binaires définis précédemment :

x	$C_2(x)$	$C_3(x)$	$C_5(x)$
s_1	1	0	00
s_2	00	11	01
s_3	01	100	001
s_4	10	101	110

Comparons les longueurs moyennes qui leur sont asociées :

$$\begin{aligned}
\overline{n}_{C_2} &= 0,3 \times 1 + 0,3 \times 2 + 0,25 \times 2 + 0,15 \times 2 = 1,7 \\
\overline{n}_{C_3} &= 0,3 \times 1 + 0,3 \times 2 + 0,25 \times 3 + 0,15 \times 3 = 2,1 \\
\overline{n}_{C_5} &= 0,3 \times 2 + 0,3 \times 2 + 0,25 \times 2 + 0,15 \times 2 = 2
\end{aligned}$$

Donc, en moyenne, le codage d'une information avec C_2 est plus court que le codage avec C_5, qui est lui-même plus court que le codage avec C_3.

Exemple 3.2 *Considérons maintenant la source constituée des lettres de l'alphabet latin et de l'espace : $\mathcal{S} = \{A, B, \ldots, Z, espace\}$. On va considérer ces lettres sans distinction du fait que ce sont des majuscules ou des minuscules. Le tableau suivant donne les fréquences d'appartition de ces lettres en français (ces mesures s'appuient sur des données statistiques) ; les trois dernières colonnes présentent des codages pour cet ensemble de symboles.*

Symbole	*fréquences*	*Longueur fixe*	*Code "virgule"*	*Code de Huffman*
espace	*0,1835*	*00000*	*0*	*111*
E	*0,1486*	*00001*	*10*	*110*
S	*0,0697*	*00010*	*110*	*1011*
A	*0,0640*	*00011*	*1110*	*1010*
N	*0,0623*	*00100*	*...*	*1001*
T	*0,0572*	*00101*		*0110*
I	*0,0591*	*00110*		*1000*
R	*0,0555*	*00111*		*0101*
U	*0,0506*	*01000*		*0011*
L	*0,0465*	*01001*		*0001*
O	*0,0459*	*01010*		*0000*
D	*0,0260*	*01011*		*01110*
C	*0,0259*	*01100*		*01001*
P	*0,0256*	*01101*		*01000*
M	*0,0245*	*01110*		*00101*
V	*0,0100*	*01111*		*001000*
G	*0,0083*	*10000*		*0111110*
Q	*0,0081*	*10001*		*0111101*
F	*0,0078*	*10010*		*0111100*
B	*0,0064*	*10011*		*0010011*
H	*0,0061*	*10100*		*0010010*
X	*0,0031*	*10101*		*01111110*
J	*0,0023*	*10110*		*011111110*
Y	*0,0021*	*10111*		*0111111111*
Z	*0,0008*	*11000*	*...*	*01111111101*
K	*0,0001*	*11001*	*111...110*	*011111111001*
W	*0,0000*	*11010*	*111...111*	*011111111000*

longueur fixe : *si on souhaite coder ces symbole à l'aide d'un code de longueur fixe n, la plus petite longueur n envisageable est 5 car $Card(\mathcal{S}) = 27$ et $2^4 < 27 \leq 2^5$. La longueur moyenne est ici $\overline{n} = 5$.*

code "virgule" : *ce type de code consiste en l'utilisation d'un symbole de séparation entre les mots, ce qui lève toute ambiguïté lors du décodage. L'exemple présenté ici nous donne $\overline{n} = 6,3034$. Ce code n'est pas très efficace ici car les probabilités d'apparitions des symboles de la source sont assez proches les unes des*

autres. Il le serait beaucoup plus si plus de symboles apparaissaient vraiment très souvent, et plus quasiment jamais.

code de Huffman : *en fait, le codage de Huffman est optimal, comme on le verra plus loin. On a ici* $\overline{n} = 3,4929$.

Remarque : on peut aller plus loin en considérant non pas les symboles de la source un à un, mais en exploitant le fait que certains se suivents plus que d'autres. Par exemple, en français, un "q" est très fréquemment suivi d'un "u". Ce phénomène ne sera pas pris ici en considération, et on travaillera sous l'hypothèse que les symboles de la sources sont indépendants les uns des autres.

II Codage optimal : définitions et propriétés

Définition 3.3 *Un codage* C *de* $\mathcal{S}$ *est dit* optimal *au regard de la distribution* f *s'il est décodable de manière unique et qu'il n'existe auncun autre codage décodable* C' *tel que* $\overline{n_{C'}} < \overline{n_C}$.

On va donc étudier quelles propriétés vérifie un tel codage existe, et comment en construire.

Une première remarque est qu'un code optimal n'est, a priori, pas unique.

Exemple 3.3 *Voici par exemple deux codes optimaux pour une source composée de trois symboles de même fréquence :*

s	$f(s)$	$C_1(s)$	$C_2(s)$
s_1	$\frac{1}{3}$	1	0
s_2	$\frac{1}{3}$	00	10
s_3	$\frac{1}{3}$	01	11

Proposition 3.1 *Soit* $\mathcal{S}$ *une source, et* f *la distribution des fréquences de ses symboles. Soit* C *un codage optimal pour la distribution* f*. Alors il existe un codage préfixe* C' *qui est optimal pour* f *et tel que pour tout symbole* x *de la source :* $|C'(x)| = |C(x)|$.

preuve : Ceci provient du théorème de MacMillan et de l'inégalité de Kraft. Supposons qu'il existe un codage optimal ; ses paramètres (longueur des mots, ...) satisfont l'inégalité de Kraft (cf. MacMillan). Et donc il existe un codage préfixe ayant ces mêmes paramètres (cf. Kraft). ♢

Ce résultat implique qu'on peut restreindre l'étude aux codes préfixes ; on sera alors certains de travailler sur des codes décodables.

Proposition 3.2 *La longueur des codes des lettres dans un codage optimal* $C : \mathcal{S} \longrightarrow \mathcal{A}^*$ *est une fonction décroissante de leur fréquence :*

$$\forall\, x, y \in \mathcal{S},\ f(x) > f(y) \Longrightarrow |C(x)| \leq |C(y)|.$$

preuve : Nous allons démontrer la contraposée : s'il existe deux symboles x et y tels que $f(x) > f(y)$ et $|C(x)| > |C(y)|$ alors C n'est pas un codage optimal.

On suppose donc qu'il existe deux symboles x et y tels que $f(x) > f(y)$ et $|C(x)| > |C(y)|$. Soit C' le codage défini par

$$\begin{array}{rccl} C' : & \mathcal{S} & \longrightarrow & \mathcal{A}^* \\ & z & \longmapsto & \left\{ \begin{array}{ll} C'(z) = C(z) & \text{si } z \neq x, y \\ C'(y) = C(x) & \\ C'(x) = C(y) & \end{array} \right. \end{array}$$

Comparons maintenant les longueurs moyennes de C et C' ; on a

$$\begin{aligned} \overline{n_{C'}} &= \overline{n_C} - f(x)\,|C(x)| - f(y)\,|C(y)| + f(x)\,|C'(x)| + f(y)\,|C'(y)| \\ &= \overline{n_C} - f(x)\,|C(x)| - f(y)\,|C(y)| + f(x)\,|C(y)| + f(y)\,|C(x)| \\ &= \overline{n_C} - (f(x) - f(y))\,(|C(x)| - |C(y)|) \end{aligned}$$

Or, on sait que x et y sont tels que $(f(x) - f(y))\,(|C(x)| - |C(y)|) > 0$; on a donc $\overline{n_{C'}} < \overline{n_C}$, et comme C' est décodable (mêmes mots de code que C), ceci implique que C n'est pas optimal. $\diamondsuit$

Attention, la réciproque n'est pas vraie : ce n'est pas parce qu'un code vérifie cette propriété des longueurs de mots qu'il est optimal.

Exemple 3.4 *Voici par exemple deux codes pour une source composée de trois symboles de même fréquence :*

s	$f(s)$	$C_1(s)$	$C_2(s)$
s_1	$\frac{1}{3}$	00	0
s_2	$\frac{1}{3}$	01	10
s_3	$\frac{1}{3}$	10	11

Le code C_1 vérifie

$$\forall\, x, y \in \mathcal{S},\ f(x) > f(y) \Longrightarrow |C(x)| \leq |C(y)|$$

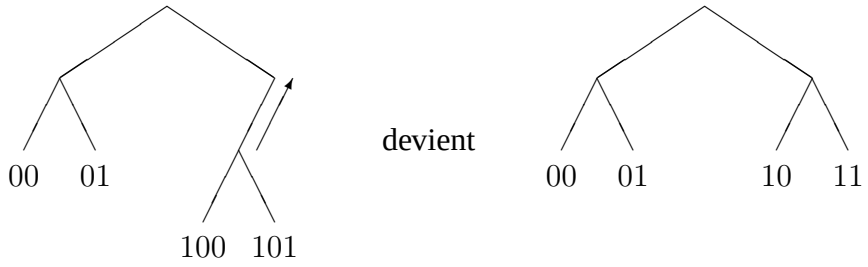
mais n'est pas optimal : $\overline{n_{C_1}} = 2 > \overline{n_{C_2}} = \frac{5}{3}$.

Ce résultat confirme l'idée que les symboles les plus fréquents sont codés par les mots les plus courts.

Désormais, on se restreint aux codages binaires : $\mathcal{A} = \{0, 1\}$.

Proposition 3.3 *Soit $C : (\mathcal{S}, f) \to \{0,1\}^*$ un codage binaire préfixe optimal pour la distribution f. L'arbre de C est alors complet : chaque nœud (interne) de l'arbre de C a exactement deux fils. Autrement dit, pour tout $x \in \mathcal{S}$, tous les préfixes de $C(x)$ autres que lui-même sont préfixes d'autres mots du code.*

preuve : Raisonnons par l'absurde : on suppose qu'il existe un code binaire préfixe optimal dont l'arbre n'est pas complet. Il existe dans cet arbre un nœud ν qui n'a qu'un seul fils. Le code étant préfixe, ses mots sont des feuilles de l'arbre ; donc si on remplace le nœud ν par le sous-arbre qu'il génère (on remonte ce sous-arbre d'un cran : voir le schéma), on obtient l'arbre d'un autre code préfixe, dont les mots sont plus courts. C n'est alors pas optimal, ce qui contredit l'hypothèse de départ.



On peut formaliser ce raisonnement de la manière suivante : on suppose que le code préfixe optimal C est tel qu'il existe un $x \in \mathcal{S}$ tel qu'un de ses préfixes, disons u, n'est préfixe d'aucun $C(y), y \in \mathcal{S}, y \neq x$. Soit C' un deuxième code défini par :

$$C' : \begin{array}{ccc} \mathcal{S} & \longrightarrow & \{0,1\}^* \\ y & \longmapsto & C'(y) = \left\{ \begin{array}{ll} C(y) & \text{si } C(y) \text{ n'admet pas } u \text{ comme préfixe} \\ u \cdot v' & \text{si } C(y) = u \cdot v, v' \text{ désignant } v \text{ privé de sa 1ère lettre} \end{array} \right. \end{array}$$

Comme C est préfixe, C' l'est aussi. De plus, on a $\overline{n_{C'}} < \overline{n_C}$ (puisque $|C'(y)| < |C(y)|$ dès que u est préfixe de $C(y)$, et $|C'(y)| = |C(y)|$ pour les autres). Ceci rentre en contradiction avec le fait que C est un code optimal. ♢

Corollaire 3.1 *Soit $C : (\mathcal{S}, f) \rightarrow \{0,1\}^*$ un codage binaire préfixe. Alors les deux symboles les moins fréquents de $\mathcal{S}$ sont codés par deux mots de même longueur qui ne diffèrent que par leur dernier bit.*

Pour les codages binaires optimaux, l'inégalité de Kraft devient une égalité.

Proposition 3.4 *Soit C un codage binaire optimal de la source $(\mathcal{S}, f)$, et soient $n_1, \ldots, n_N$ les longueurs respectives des mots de code. Alors l'inégalité de Kraft devient une égalité :*

$$\sum_{i=1}^{N} 2^{-n_i} = 1$$

preuve : laissée en exercice. ♢

Attention, un code satisfaisant l'égalité n'est pas forcément optimal : il faut prendre en compte l'association des symboles de la source aux mots de code (si on associe un mot long à un symbole fréquent, le codage ne sera pas optimal).

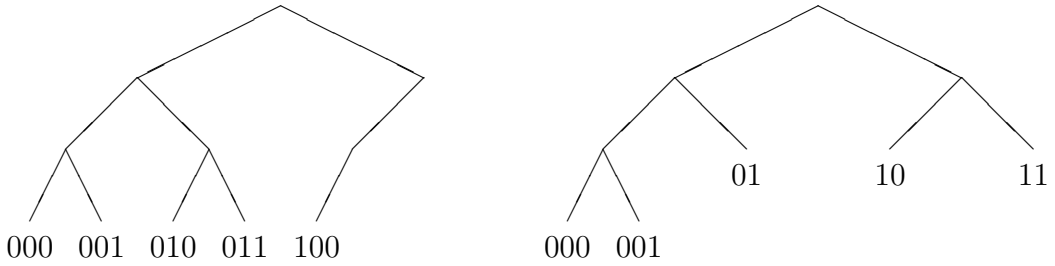
III Codage optimal : constructions

On se restreint dans tout ce qui va suivre aux codages binaires : $\mathcal{A} = \{0, 1\}$.

1) Cas de la distribution uniforme

Soit N le nombre de symboles de la source ; on part d'un code de longueur fixe minimale p, avec $2^{p-1} < N \leq 2^p$. Si $N = 2^p$, alors on ne peut pas faire mieux. Par contre, si $N < 2^p$, on peut remplacer $2^p - N$ mots par des mots de longueur $p - 1$.

Exemple 3.5 *Soit $N = 5$ et pour tout symbole x de la source, $f(x) = 1/5$. On construit un code de longueur fixe minimale : cette longueur est $p = 3$ puisque $2^2 < 5 < 2^3$. On obtient par exemple le code $\{000, 001, 010, 011, 100\}$. On peut remplace $8 - 5 = 3$ mots par des mots de longueur $p - 1 = 2$. Voici les deux arbres correspondant à ces codes : à gauche le code de longueur fixe, à droite le code optimal.*



2) Cas d'une distribution de la forme $f(x) = \frac{1}{2^k}$

Soit $\mathcal{S} = \{s_1, \ldots, s_N\}$ la source à coder, et $f_i = f(s_i)$ les fréquences associées aux symboles. On suppose dans ce paragraphe que ces fréquences sont de la forme $\frac{1}{2^k}$, avec $k \geq 1$, et que les symboles sont "ordonnés", de manière à avoir $f_1 \geq f_2 \geq \ldots \geq f_N$ (on peut le supposer sans perte de généralité). Rappelons que par définition on a $\sum_i f_i = 1$.

La proposition suivante établit qu'à partir de tout codage optimal d'une source ayant au moins deux symboles, on peut découper cette source en deux et extrire du codage optimal deux codages optimaux pour chacun des deux morceaux de la source.

Proposition 3.5 *Soit C un codage binaire préfixe optimal pour la distribution f. Les sous-arbres gauche et droit induisent des codages optimaux C_1 et C_2 pour les sources $\mathcal{S}_1$ (des lettres dont le code commence par un 0) et $\mathcal{S}_2$ (des lettres dont le code commence par un 1), avec les distributions f_1 et f_2 qui sont les restrictions respectives de f à $\mathcal{S}_1$ et $\mathcal{S}_2$:*

$$f_i(s) = \frac{f(s)}{\sum_{x \in \mathcal{S}_i} f(x)} \qquad \textit{pour } i = 1, 2$$

Voici une méthode proposée par R.M. Fano pour construire un code optimal :

- si $N = 1$, alors on associe le mot vide ε au symbole.
- si $N = 2$, alors on a forcément $f_1 = f_2 = \frac{1}{2}$ et le code optimal est le code de 2 mots de longueur 1 : au premier symbole on associe la lettre 0, et au deuxième la lettre 1.
- si $N > 2$, on procède récursivement :

  1. on peut regrouper les symboles de la source en deux ensembles $\mathcal{S}_1$ et $\mathcal{S}_2$ tels que
  $$\sum_{s \in \mathcal{S}_1} f(s) = \sum_{s \in \mathcal{S}_2} f(s) = \frac{1}{2}$$
  2. on attribue la lettre 0 aux symboles de $\mathcal{S}_1$, et la lettre 1 aux symboles de $\mathcal{S}_2$.
  3. pour chacun de ces ensembles, on réapplique séparément la procédure en pondérant les fréquences comme indiqué à la proposition précédente (ici, on multiplie les fréquences par 2) : le codage d'un symbole de la source de départ sera obtenu par concaténation des lettres attribuées à ce symbole lors de la procédure.

Exemple 3.6 *Prenons $\mathcal{S} = \{s_1, \ldots, s_8\}$ avec la distribution des fréquences :*

$$\begin{aligned} f(s_1) &= \frac{1}{4} \\ f(s_2) &= \ldots = f(s_6) = \frac{1}{8} \\ f(s_7) &= f(s_8) = \frac{1}{16} \end{aligned}$$

Ceci donne lieu à la construction suivante :

symboles	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
fréquences	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{16}$
lettres	0	0	0	1	1	1	1	1
fréquences	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$
lettres	0	1	1	0	0	1	1	1
fréquences	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$
lettres		0	1	0	1	0	1	1
fréquences		1	1	1	1	1	$\frac{1}{2}$	$\frac{1}{2}$
lettres							0	1
fréquences							1	1
lettres								
codages	00	010	011	100	101	110	1110	1111

Un codage construit par la méthode de Fano est optimal si les fréquences sont de la forme $\frac{1}{2^k}$.

On peut remarquer que cette technique peut également s'appliquer même lorsque les fréquences ne sont pas de la forme $\frac{1}{2^k}$; on sépare alors les symboles en deux ensembles les plus "équitables" possible (pour ce qui est des fréquences). Le codage obtenu n'est alors pas forcément optimal.

3) Cas général : codage de Huffman

Il s'agit de la généralisation du principe de Fano, et s'appuie sur le résultat suivant :

Proposition 3.6 *Soit une source $\mathcal{S} = \{s_1, \ldots, s_N\}$ munie de la distribution des fréquences f ; on suppose que les symboles sont ordonnés de la manière suivante : $f(s_1) \geq \ldots \geq f(s_N)$; soit une deuxième source*

$$\mathcal{S}' = S \setminus \{s_{N-1}, s_N\} \cup \{s_{N-1,N}\}$$

munie de la distribution des fréquences donnée par

$$\begin{cases} f'(s) & = & f(s) & \text{si } s \neq s_{N-1,N} \\ f'(s_{N-1,N}) & = & f(s_{N-1}) + f(s_N) & \end{cases}$$

Alors, si C' est un codage binaire optimal pour $\mathcal{S}'$, le code C défini par :

$$\begin{array}{rccl} C: & \mathcal{S} & \longrightarrow & \{0,1\}^* \\ & s & \longmapsto & \begin{cases} C(s) & = & C'(s) & \text{si } s \neq s_{N-1}, s_N \\ C(s_{N-1}) & = & C'(s_{N-1,N}) \cdot 0 & \\ C(s_N) & = & C'(s_{N-1,N}) \cdot 1 & \end{cases} \end{array}$$

est un codage optimal pour la source $\mathcal{S}$.

L'idée proposée par Huffman est donc de tirer parti de cette propriété et de construire un codage optimal en considérant les deux symboles de plus basse fréquence de $\mathcal{S}$, en leur attribuant à chacun les lettres 0 et 1, puis en considérant ces symboles comme un seul symbole de la source $\mathcal{S}'$. On recommence en considérant $\mathcal{S}'$ comme la nouvelle source $\mathcal{S}$. Cette construction peut se faire sous forme d'arbre (voir les transparents du cours) ou de tableau (voir l'exemple page suivante).

Proposition 3.7 *Un code obtenu par la méthode de Huffman est optimal.*

On dispose ainsi d'une technique permettant de construire, pour toute source $(\mathcal{S}, f)$, un codage optimal. On peut donc énoncer le résultat suivant :

Théorème 3.1 *Pour toute source $\mathcal{S}$ et distribution f il existe un codage optimal.*

Exemple 3.7 *Soit la source*

s	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$f(s)$	$0,4$	$0,15$	$0,15$	$0,1$	$0,1$	$0,06$	$0,02$	$0,02$

La construction d'un codage de Huffman pour cette source est :

symboles	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
fréquences	$0,4$	$0,15$	$0,15$	$0,1$	$0,1$	$0,06$	$0,02$	$0,02$
lettres							0	1
symboles	s_1	s_2	s_3	s_4	s_5	s_6	s_{78}	
fréquences	$0,4$	$0,15$	$0,15$	$0,1$	$0,1$	$0,06$	$0,04$	
lettres						0	1	
symboles	s_1	s_2	s_3	s_4	s_5	s_{678}		
fréquences	$0,4$	$0,15$	$0,15$	$0,1$	$0,1$	$0,1$		
lettres					0	1		
symboles	s_1	s_{5678}	s_2	s_3	s_4			
fréquences	$0,4$	$0,2$	$0,15$	$0,15$	$0,1$			
lettres				0	1			
symboles	s_1	s_{34}	s_{5678}	s_2				
fréquences	$0,4$	$0,25$	$0,2$	$0,15$				
lettres			0	1				
symboles	s_1	s_{25678}	s_{34}					
fréquences	$0,4$	$0,35$	$0,25$					
lettres		0	1					
symboles	$s_{2345678}$	s_1						
fréquences	$0,6$	$0,4$						
lettres	0	1						

Il suffit ensuite de reconstruire les mots de code :

symboles	$s_{2345678}$	s_1						
codages	0	1						
symboles	s_1	s_{25678}	s_{34}					
codages	1	00	01					
symboles	s_1	s_{34}	s_{5678}	s_2				
codages	1	01	000	0001				
symboles	s_1	s_{5678}	s_2	s_3	s_4			
codages	1	000	001	010	011			
symboles	s_1	s_2	s_3	s_4	s_5	s_{678}		
codages	1	001	010	011	0000	0001		
symboles	s_1	s_2	s_3	s_4	s_5	s_6	s_{78}	
codages	1	001	010	011	0000	00010	00011	
symboles	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
codages	1	001	010	011	0000	00010	000110	000111

IV Notion d'entropie

Les résultats présentés ci-dessus restent ciblés (alphabet binaire, ...). Il est néanmoins intéressant de comprendre sur quel concepts ils reposent. La théorie de l'information, qui a pour objet d'étudier le codage de l'information en toute généralité (tout type d'information, mais aussi tout type d'utilisation de cette information : stockage, transmission ...) a été fondée par Claude Shannon en 1948. Elle repose sur la théorie des probabilités. Sans rentrer dans les détails mathématiques de cette théorie, il est important de dégager les idées maîtresses de Shannon.

1) Quantité d'information

Si un matin, en prenant votre petit déjeuner, vous lisez dans le journal : "le soleil se couchera ce soir", vous n'apprendrez pas grand chose. En revanche, si vous lisez : "il fera nuit ce midi", vous aurez appris quelque chose. La quantité d'information est nulle dans le premier cas, elle est très grande dans le second.

Intuitivement une information est d'autant plus importante qu'elle révèle quelque chose d'exceptionnel, c'est–à–dire de peu fréquent.

C'est cela (plus d'autres arguments que nous ne développerons pas) qui a conduit Claude Shannon à poser la définition suivante :

Définition 3.4 *Soit une source $\mathcal{S}$ avec une distribution de fréquences f. On appelle* quantité d'information *du symbole $x \in \mathcal{S}$ le nombre :*

$$I(x) = \log_2 \frac{1}{f(x)} = -\log_2 f(x)$$

Une quantité d'information se mesure en bits.

On voit que cette définition reflète bien notre intuition : $I(x)$ est un nombre positif d'autant plus grand que $f(x)$ est petit. Dans les cas extrêmes on a :

- $I(x) = 0$ si $f(x) = 1$, c'est–à–dire si le symbole x apparaît dans $100\ \%$ des cas,
- $\lim_{f(x)\to 0}(x) = +\infty$, c'est–à–dire si le symbole x n'apparaît presque jamais.

Exemple 3.8 *Reprenons la source du dernier exemple :*

x	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$f(x)$	$0,4$	$0,15$	$0,15$	$0,1$	$0,1$	$0,06$	$0,02$	$0,02$
$I(x)$	$1,32$	$2,74$	$2,74$	$3,32$	$3,32$	$4,06$	$5,64$	$5,64$

2) Entropie d'une source

C'est la quantité d'information moyenne par symbole d'une source.

Définition 3.5 *On appelle* entropie *de la source $\mathcal{S}$ avec la distribution des fréquences f la quantité moyenne d'information par symbole :*

$$H(\mathcal{S}, f) = \sum_{x \in \mathcal{S}} f(x)\ I(x) = -\sum_{x \in \mathcal{S}} f(x)\ \log_2(f(x))$$

Exemple 3.9 *Reprenons la source du dernier exemple :*

x	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$f(x)$	$0,4$	$0,15$	$0,15$	$0,1$	$0,1$	$0,06$	$0,02$	$0,02$

$$\begin{aligned} H(\mathcal{S}, f) &= 0,4 \times 1,32 + 0,15 \times 2,74 + 0,15 \times 2,74 + 0,1 \times 3,32 \\ &\quad +0,1 \times 3,32 + 0,06 \times 4,06 + 0,02 \times 5,64 + 0,02 \times 5,64 \\ &= 2,48 \end{aligned}$$

Lemme 3.1 *Soit $\{f_1, f_2, \ldots, f_N\}$ une distribution de fréquences et soit $R = \{r_1, r_2, \ldots, r_N\}$ un ensemble de N réels positifs non nuls dont la somme est inférieure ou égale à 1. On a alors l'inégalité :*

$$H(\mathcal{S}, f) = \sum_{i=1}^{N} f_i\ \log_2 \frac{1}{f_i} \leq \sum_{i=1}^{N} f_i\ \log_2 \frac{1}{r_i}$$

l'égalité ayant lieu lorsqu $f_i = r_i$ pour tout i.

preuve : laissée en exercice ♢

Proposition 3.8 *Pour toute source $\mathcal{S}$ ayant N symboles, et toute distribution de fréquences f, on a*

$$0 \leq H(\mathcal{S}, f) \leq \log_2 N$$

L'égalité $H(\mathcal{S}, f) = 0$ n'ayant lieu que si $\mathcal{S}$ contient un symbole de fréquence 1, et l'égalité $H(\mathcal{S}, f) = \log_2 N$ n'ayant lieu que si la distribution f est uniforme.

preuve : L'inégalité $0 \leq H(\mathcal{S}, f)$ est immédiate.
L'autre découle du lemme précédent avec en prenant $r_i = 1/N$ pour tout i. ♢

3) Théorème du codage sans bruit

Il existe une relation entre la longueur moyenne d'un codage d'une source $\mathcal{S}$ et l'entropie de cette source. Le théorème du codage sans bruit de Shannon établit cette relation :

Théorème 3.2 (Shannon) *Soit une source $\mathcal{S}$ de distribution f, et un codage binaire C défini sur cette source. Alors on a :*

$$H(\mathcal{S}, f) \leq \overline{n_C}.$$

preuve : Soit C un codage (décodable) de $\mathcal{S}$, et soient $l_1, l_2, \ldots, l_N$ les longueurs des mots du code résultant représentant les symboles $x_1, x_2, \ldots, x_N$ de la source. On pose, pour tout i, $r_i = \frac{1}{2^{l_i}}$.

D'après l'inégalité de Kraft on a

$$\sum_{i=1}^{N} r_i \leq 1$$

et on peut alors appliquer le lemme précédent :

$$H(\mathcal{S}, f) \leq \sum_{i=1}^{N} f_i \ \log_2 \frac{1}{r_i}$$

Compte tenu de la définition des r_i, la somme du membre droit de cette inégalité vaut :

$$\sum_{i=1}^{N} f(x_i) \log_2 2^{l_i} = \sum_{i=1}^{N} f(x_i) \ l_i \ \log_2 2 = \sum_{i=1}^{N} f(x_i) \ l_i = \overline{n_C}$$

Le théorème est ainsi démontré. ♢

Corollaire 3.2 *La longueur moyenne d'un codage binaire optimal ne peut pas être inférieure à l'entropie de la source.*

Proposition 3.9 *Soit une source $\mathcal{S}$ de distribution f. Il existe un codage binaire préfixe C tel que :*

$$H(\mathcal{S}, f) \leq \overline{n_C} < H(\mathcal{S}, f) + 1.$$

La théorie de l'information permet d'étudier divers types de sources : indépendantes (comme ici : chaque symbole est considéré séparément), markoviennes (on prend en compte les suites de symboles qui sont très fréquentes, exploitant l'information mutuelle entre les symboles).

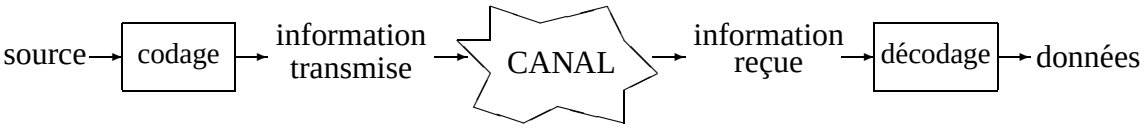
Chapitre 4

Codes correcteurs

Nous nous intéresserons dans ce chapitre aux erreurs qui peuvent survenir lors du stockage ou de la transmission des données. Nous nous placerons uniquemement dans le contexte binaire : $\mathcal{A} = \{0, 1\}$. Nous aborderons leur détection ainsi que leur éventuelle correction.

I Contexte et motivations

Les données peuvent être altérées lors de leur stockage (influence des champs électromagnétiques sur les disques durs, CD rayé, ...) ou de leur transmission (téléphone, télévision, réseau, satellites, ...). On représente ces perturbations de la manière suivante :



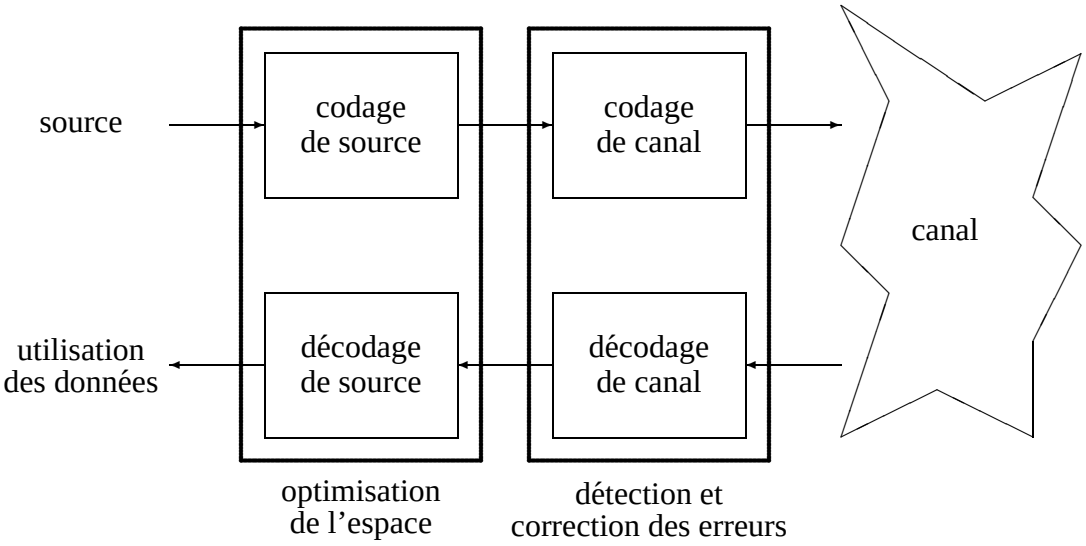
De telles erreurs arrivent très fréquemment, et il est essentiel de pouvoir les détecter et d'essayer de les corriger. Lors de la relecteur ou de la réception de données, plusieurs cas de figure peuvent se présenter :

1. les données sont indécodables : il y a clairement un problème ;
2. les données sont décodables, mais on se rend compte qu'il y a eu un problème : il s'agit par exemple d'un texte et on voit que certaines lettres sont incohérentes ;
3. tout a l'air normal : on décode correctement et les données obtenues ont un sens : on ne sait pas s'il y a eu des erreurs ou non (une erreur peut passer inaperçue, par exemple lorsqu'un nombre est remplacé par un autre dans une somme à payer).

Dans tous ces cas, il faut pouvoir décider où se situent les éventuelles erreurs, pour pouvoir rectifier les données.

II Le modèle

La situation se modélise plus précisément de la manière suivante :

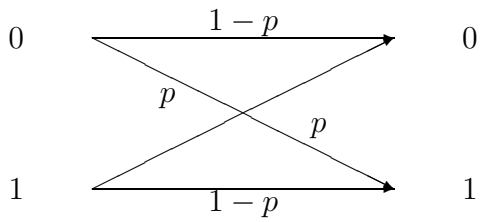


Les données sont traitées en deux temps :

1. codage/décodage de source : on s'attache à traduire le document source sous forme binaire ; c'est là qu'on essaie de minimiser l'espace occupé par les données en binaire (codages optimaux) ;
2. codage/décodage de canal : on s'attache à la détection et à la correction des erreurs (codes correcteurs : on rajoute lors du codage une information pertinente qui sera exploitée lors du décodage).

Le canal est une estimation mathématique (probabiliste) des conditions réelles de stockage ou de transmission. C. Shannon a proposé différents types de canaux, permettant d'approcher les situations concrètes. Nous n'étudierons ici que le plus simple : le *canal binaire symétrique sans mémoire*.

Définition 4.1 *Le* canal binaire symétrique sans mémoire *est un canal dans lequel chaque bit est transmis indépendamment des autres, et a une probabilité p d'être modifié lors de la transmission :*



Rappelons qu'une mesure de probabilité sur $\mathcal{X}$ est une application de la forme

$$\begin{array}{rccc} P: & \mathcal{X} & \longrightarrow & [0,1] \subset \mathbb{R} \\ & x & \longmapsto & P(x) \end{array}$$

avec la propriété que

$$\sum_{x \in \mathcal{X}} P(x) = 1 \, .$$

Dans le contexte des probabilités, $\mathcal{X}$ désigne l'ensemble de tous les événements qui peuvent se produire, et $x \in \mathcal{X}$ désigne donc un événement. Par extension, si A désigne un ensemble d'événements, on définit $P(A) = \sum_{x \in A} P(x)$.

Une des propriétés de base est que si A et B sont deux ensembles d'événements, on a $P(A \cup B) = P(A) + P(B) - P(A \cap B)$; si A et B sont incompatibles ($A \cap B = \emptyset$), alors on a $P(A \cup B) = P(A) + P(B)$; ainsi, si A et B sont incompatibles et tels que $A \cup B = \mathcal{X}$, alors $P(A) + P(B) = 1$.

Ici, à l'échelle de la transmission d'un bit, on a $\mathcal{X} = \{0 \to 0, 1 \to 1, 0 \to 1, 1 \to 0\}$; considérons deux ensembles d'événements : $A = \{0 \to 0, 1 \to 1\}$ (le bit est correctement transmis), et $B = \{0 \to 1, 1 \to 0\}$ (le bit est mal transmis). Ces ensembles sont bien disjoint, et leur union correspond bien à toutes les situations possibles. Ainsi,

$$P(\text{ ¡¡le bit est correctement tranmis¿¿ }) + P(\text{ ¡¡le bit est mal transmis¿¿ }) = 1.$$

On appelle communément p la probabilité qu'il y ait une erreur lors de la transmission du bit ; la probabilité que la transmission se passe bien est donc $1 - p$.

On peut, à partir de cette donnée de base, calculer les probabilités de bonne ou de mauvaise transmission d'une suite de bits. Ce calcul est simplifié ici car on a supposé que la transmission des bits était indépendante (canal sans mémoire) : il suffit alors de multiplier les probabilités concernant chacun des bits pour obtenir le résultat sur la suite entière.

Exemple 4.1 *Prenons une suite de deux bits. On a :*

- *la probabilité que les deux bits soient mal transmis est :* $P_1 = p^2$ *;*
- *la probabilité que les deux bits soient bien transmis est :* $P_2 = (1-p)^2$ *;*
- *la probabilité que le premier soit bien transmis et que le deuxième soit mal transmis est :* $P_3 = (1-p)p$ *;*
- *la probabilité que le premier soit mal transmis et que le deuxième soit bien transmis est :* $P_4 = p(1-p)$ *;*
- *la probabilité qu'il y ait exactement une erreur est la somme des probabilités de tous les cas où il y a une seule erreur :* $P_3 + P_4 = 2p(1-p)$ *;*
- *la probabilité qu'il y ait au moins une erreur lors de la transmission est :* $1 - P_2 = 1 - (1-p)^2$ *; on peut également la calculer de la manière suivante :* $P_1 + P_3 + P_4$ *(on somme, pour chaque nombre d'erreurs possible, les probabilités qui les concernent).*

Exemple 4.2 *Prenons $p = 10^{-3} = 0,001$, et considérons la transmission de 3 bits dans le canal. Supposons que la suite à transmettre est 001. On a alors :*

- *la probabilité que la suite reçue soit 001 est $(1-p)^3 = (0,999)^3 = 0,997$;*
- *la probabilité que la suite reçue soit 011 est : $(1-p)p(1-p) = (0,999)^2 \times 10^{-3} = 0,998 \times 10^{-3}$;*
- *la probabilité que la suite reçue soit 010 est : $(1-p)p^2 = 0,999 \times 10^{-6}$;*
- *la probabilité qu'il y ait au moins une erreur lors de la transmission est : $1-(1-p)^3 = 0,003$.*

Proposition 4.1 *On transmet une suite de N bits sur un canal binaire symétrique sans mémoire.*

- *Soient r bits choisis parmi les N bits reçus. La probabilité que ces r bits soient erronés et que les autres aient été bien transmis est*

$$p^r(1-p)^{N-r}$$

- *La probabilité que parmi les N bits reçus il y ait exactment r bits erronés est*

$$C_N^r\, p^r(1-p)^{N-r}$$

preuve : Le premier résultat est évident : on sait quels sont les bits bien ou mal transmis, et on multiplie les probabilités correspondantes. Pour le deuxième résultat, il faut prendre en compte les C_N^r choix possibles des positions des r bits erronés. ♢

III Détection et correction des erreurs : principes

La personne qui reçoit les données observe un train binaire, autrement dit une suite de bits. Elle doit pouvoir la découper en mots puis essayer de voir si ces mots sont valides, c'est-à-dire si ce sont des mots du code. Ainsi l'utilisation des codes de longueur variable est très hasardeuse, puisqu'une erreur dans le train binaire peut totalement désynchroniser le découpage en mots.

Exemple 4.3 *Considérons par exemple le code de Huffman défini pour le codage des textes en français (exemple 3.2 du chapitre précédent). On code le message "CETTE" :*

$$0100111001100110110$$

Lors de la transmission, une erreur se produit au 4ème bit ; le destinateur reçoit :

$$0101111001100110110$$

et lors du décodage il obtient "R UUT", ce qui est très différent du message de départ. On peut remarquer qu'il se rendra peut-être compte qu'il y a un problème car ce message n'a pas de sens, mais il peut se produire des cas où le message concerve un sens, qui n'est pas le bon (par exemple si les données sont numériques).

Ainsi, on utilise plutôt des codes de longueur fixe car ils nous assurent un bon découpage. Il reste ensuite le problème de la détection et de la correction des erreurs, mais on pourra traiter chacun des mots séparément.

On supposera dans la suite que me message à coder est binaire (c'est souvent dans la pratique le résultat d'un codage de Huffman binaire de la source) et on considérera uniquement les codages binaires de longueur fixe, également connus sous le nom de *codes en blocs*.

La procédure de codage est :

1. on découpe le message en blocs de k bits ;
2. chacun des blocs de k bits est codé séparément en un bloc de n bits ;
3. on concatène les blocs de n bits ainsi obtenus pour former le message qui sera transmis.

Et la procédure de décodage :

1. on découpe le message reçu en blocs de n bits ;
2. pour chacun de ces blocs de n bits, on cherche le mot du code le plus proche (cette notion est développée plus loin), et on extrait les k bits d'information (on retrouve le bloc de k bits qui est l'antécédent du mot de code en question) ;
3. on concatène les blocs de k bits ainsi obtenus pour reconstituer le message.

Définition 4.2 *Un* code en blocs binaire de longueur n *transforme chacun des* 2^k *blocs de* k *bits en des blocs de* n *bits. La quantité* k *désigne le nombre de bits d'information, et* $n - k$ *correspond à la redondance ajoutée. Le* rendement *du code, ou* taux d'information *est donné par le quotient* k/n.

Exemple 4.4 *Si on code* 1000 *bits avec un code dont les paramètres sont* $k = 5$ *et* $n = 8$*, la taille du message transmis sera* $1000 \times 8/5 = 1600$.

Voici maintenant quelques exemples élémentaires de codes en blocs :

Exemple 4.5 (le bit de parité) *Le principe en est très simple : on rajoute à chaque bloc de taille* k *un bit, dit de parité, de sorte que sur le nouveau bloc (de taille* $k + 1$*) le nombre de* 1 *est pair.*

Considérons par exemple $k = 4$*. L'ensemble des mots de* k *bits est*

$$\begin{array}{cccc} 0000 & 0001 & 0010 & 0011 \\ 0100 & 0101 & 0110 & 0111 \\ 1000 & 1001 & 1010 & 1011 \\ 1100 & 1101 & 1110 & 1111 \end{array}$$

A chacun d'entre eux on associe un mot de code de $k + 1 = 5$ *bits ; ces mots sont :*

$$\begin{array}{cccc} 0000\,0 & 0001\,1 & 0010\,1 & 0011\,0 \\ 0100\,1 & 0101\,0 & 0110\,0 & 0111\,1 \\ 1000\,1 & 1001\,0 & 1010\,0 & 1011\,1 \\ 1100\,0 & 1101\,1 & 1110\,1 & 1111\,0 \end{array}$$

Donc, si le message de départ est 0010101101010001*, on le découpe en blocs de* 4 *bits et on code chacun de ces blocs :*

$$\begin{array}{cccc} 0010 & 1011 & 0101 & 0001 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 00101 & 10111 & 01010 & 00011 \end{array}$$

Le message transmis est donc 00101101110101000011*.*

Ce code permet de détecter un nombre impair d'erreurs dans un bloc, mais pas un nombre pair. Et il ne permet en aucun cas de corriger ces erreurs.

Exemple 4.6 (le code à répétitions) *Un dernier exemple simple est le code à répétitions : il consiste en la répétition de chaque bit* n *fois. Ainsi ses paramètres sont* $k = 1$ *et* n *(*n *est indépendant de* k*). Prenons par exemple* $n = 4$ *: le bit* 0 *est codé par* 0000 *et le bit* 1 *par* 1111*. Donc, si le message de départ est* 0010101101010001*, on le découpe en bits et on code chacun de ces bits séparément :*

$$\begin{array}{cccccccccccccccc} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0000 & 0000 & 1111 & 0000 & 1111 & 0000 & 1111 & 1111 & 0000 & 1111 & 0000 & 1111 & 0000 & 0000 & 0000 & 1111 \end{array}$$

Le message transmis est donc constitué de $4 \times 16 = 64$ *bits :*
$0000000011110000111100001111111100001111000011110000000000001111$*.*

Ce type de codage permet de détecter et de corriger, pour chaque bloc de n *bits jusqu'à* $\frac{n-1}{2}$ *erreurs si* n *est impair, et* $\frac{n-2}{2}$ *si* n *est pair ; au-delà, et jusqu'à* $n-1$ *erreurs par blocs, on va détecter les erreurs, mais on ne les corrigera pas.*

IV Métrique de Hamming

L'ensemble des objets manipulés ici et dans la suite est constitué des mots binaires de longueur n, et d'opérations définies sur ces mots : distance, addition, ...En mathématiques, la rigueur nécessite de définir proprement ceci, selon des notions algébriques de groupes, anneaux, corps, espaces vectoriels[1]. On n'insistera pas ici sur ces notions, et on introduira seulement les opérations et propriétés utilisées dans la suite.

L'idée principale de la correction des erreurs est, pour chaque bloc de n bits du message reçu, de retrouver le mot de code qui en est le plus proche. Reste à définir cette notion de ¡¡proche¿¿. On utilisera ici une distance, introduite par Hamming :

[1] pour chaque ensemble d'objets, on peut définir des opérations ; ce sont les propriétés vérifiées par ces opérations qui engendrent ces structures ; à titre d'exemples, $\mathbb{Z}$ muni de l'addition standard des entiers est un groupe, $\mathbb{Z}$ muni des addition et multiplication standard sur les entiers est un anneau, $\mathbb{R}$ muni des addition et multiplication standard est un corps, $\mathbb{R}^n$ est un espace vectoriel

Définition 4.3 *La* distance de Hamming *entre deux mots binaires de longueur n est le nombre de composantes pour lesquelles ces mots diffèrent :*

$$\forall\, u, v \in \{0,1\}^n,\ d(u,v) = Card(\{i, u_i \neq v_i\})$$

Exemple 4.7 *Soit $n = 5$; $d(01001, 01011) = 1$, $d(01111, 01001) = 2$.*

Notons que toute application de la forme $d : \{0,1\}^n \times \{0,1\}^n \to \mathbb{R}$ n'est pas nécessairement une distance ; elle doit vérifier un certain nombre de propriétés :

- pour tout $u, v \in \{0,1\}^n, d(u,v) = 0 \Leftrightarrow u = v$;
- pour tous $u, v \in \{0,1\}^n, d(u,v) = d(v,u)$;
- inégalité triangulaire : pour tous $u, v, w \in \{0,1\}^n, d(u,v) \leq d(u,w) + d(v,w)$.

L'application définie plus haut satisfait ces conditions et peut, à juste titre, être appelée distance. Elle définit une métrique sur l'ensemble des mots binaires de longueur n.

Définition 4.4 *Soit C un code de longueur n, et $u \in \{0,1\}^n$. On définit la* distance de u à C *comme la plus petite distance entre u et un mot de C :*

$$d(u, C) = \min_{v \in C} d(u,v)$$

Définition 4.5 *Soit $u \in \{0,1\}^n$, la* boule de centre u et rayon r *désigne l'ensemble des mots binaires qui sont à une distance au plus r de u :*

$$B(u,r) = \{v \in \{0,1\}^n, d(u,v) \leq r\}.$$

Exemple 4.8 *Soit $n = 4$; on a :*

$$\begin{aligned}
B(1001,0) &= \{1001\} \\
B(1001,1) &= B(1001,0) \cup \{0001, 1101, 1011, 1000\} \\
B(1001,2) &= B(1001,1) \cup \{0101, 1111, 1010, 0000, 0011, 1100\} \\
B(1001,3) &= B(1001,2) \cup \{0111, 1110, 0010, 0100\} \\
B(1001,4) &= B(1001,3) \cup \{0110\} = \{0,1\}^4
\end{aligned}$$

Définition 4.6 *On définit l'*addition *de deux bits de la manière suivante :*

$$\begin{aligned}
0 \oplus 0 &= 0 \\
1 \oplus 1 &= 0 \\
1 \oplus 0 &= 1 \\
0 \oplus 1 &= 1
\end{aligned}$$

Elle correspond à l'opération logique connue sous le nom de ¡¡ou exculsif¿¿ (XOR en anglais).

Cette opération vérifie les propriétés suivantes[2] :
- commutativité : pour tous $u, v \in \{0,1\}, u \oplus v = v \oplus u$;
- associativité : pour tous $u, v, w \in \{0,1\}, (u \oplus v) \oplus w = u \oplus (v \oplus w)$;
- existence d'un élément neutre e tel que pour tout $u \in \{0,1\}, u \oplus e = e \oplus u = u$; cet élément est 0 ;
- opposés : tout $u \in \{0,1\}$ admet un opposé noté en général $-u$ (qui appartient à $\{0,1\}$) tel que $u \oplus (-u) = (-u) \oplus u = e = 0$; on remarquera ici que l'opposé de u est u lui-même.

La définition d'une opération d'addition implique implicitement celle d'une soustraction : on définit $u \ominus v$ comme $u \oplus (-v)$; pour l'addition définie ici, on obtient $u \ominus v = u \oplus v$. Ainsi, soustraire deux bits revient à les additionner.

Définition 4.7 *L'*addition de deux mots binaires de longueur n *est le mot binaire de longueur n résultant de leur addition bit-à-bit. Soient $u, v \in \{0,1\}^n$; on a :*

$$w = u \oplus v \iff w_i = u_i \oplus v_i \qquad \forall\, i = 1, \ldots, n$$

Ainsi, $u \oplus v$ possède des 1 là où u et v diffèrent, et des 0 là où u et v coïncident.

Les remarques concernant l'addition de deux bits s'étendent à l'addition de deux mots binaires :
- commutativité : pour tous $u, v \in \{0,1\}^n, u \oplus v = v \oplus u$;
- associativité : pour tous $u, v, w \in \{0,1\}^n, (u \oplus v) \oplus w = u \oplus (v \oplus w)$;
- existence d'un élément neutre e tel que pour tout $u \in \{0,1\}^n, u \oplus e = e \oplus u = u$; cet élément est $0000 \ldots 000$; on le notera 0 ;
- opposés : tout $u \in \{0,1\}^n$ admet un opposé, lui-même : $u \oplus u = 0$.

Il en découle que, pour tous $u, v, w \in \{0,1\}^n$,

$$u \oplus v = w \iff u \oplus w = v \iff v \oplus w = u$$

Définition 4.8 *Soit $u \in \{0,1\}^n$; on appelle* poids de u *le nombre de ses composantes non nulles :*

$$wt(u) = d(u, 0).$$

Cette notion peut, comme les précédentes, être définie pour tout alphabet ; dans le cas binaire, le poids d'un mot est le nombre de ¡¡1¿¿ dans ce mot. On a de plus la propriété suivante :

$$\forall\, u, v \in \{0,1\}^n, \qquad wt(u \oplus v) = d(u, v).$$

Ceci implique notamment que

$$\forall\, u, v, w \in \{0,1\}^n, \qquad d(u \oplus w, v \oplus w) = d(u, v).$$

On peut maintenant préciser ce que l'on entend par erreur lors de la transmission d'un mot u : on dira que u a subi des erreurs si le mot reçu v est différent de u. Cette différence peut être exprimée comme un mot non nul e tel que $v = u \oplus e$; $wt(e) = t$ désigne le nombre d'erreurs. Supposons maintenant que u est un mot d'un code C, le mot v se trouve alors à une distance inférieure ou égale à t de C.

[2] $\{0,1\}$ muni de cette addition est un groupe

V Détection et correction des erreurs : résultats

Un paramètre essentiel de la détection et de la correction des erreurs est le suivant :

Définition 4.9 *Soit C un code. Sa* distance minimale *est donnée par :*

$$\begin{aligned} d &= \min_{\substack{u,v \in C \\ u \neq v}} d(u,v) \\ &= \min_{\substack{u,v \in C \\ u \neq v}} wt(u \oplus v) \end{aligned}$$

Définition 4.10 *Soit C un code de longueur n, et soit $v \in \{0,1\}^n$; on dit que C détecte v si $v \notin C$. C est dit t-*détecteur *s'il peut détecter tous les mots comportant de 1 à t erreurs, c'est-à-dire tous les mots situés à une distance non nulle inférieure ou égale à t de C.*

Proposition 4.2 *Pour qu'un code de distance minimale d soit t-détecteur, il faut et il suffit que $d > t$.*

preuve : Un code C est t-détecteur si et seulement si pour tout mot $u \in C$ aucune des versions erronées de u, située à une distance au plus t de C, n'est un mot de C, autrement dit si et seulement si

$$\forall\, u \in C, \qquad B(u,t) \cap C = \{u\}.$$

Ainsi, si on considère toutes les boules de rayon t centrées en les mots de C, chacune ne contient qu'un seul mot de C, son centre (d'autres mots peuvent appartenir à plusieurs boules, mais ce ne seront pas des mots de C).

Soit C un code t-détecteur ; si u et v sont des mots de C tels que $d(u,v) = d$, ces deux mots ne peuvent appartenir à une même boule : $d(u,v) > t$: on a donc $d > t$.

La réciproque est évidente : si u et v sont des mots de C tels que $d(u,v) = d$, et si $d > t$, alors $d(u,v) > t$. Comme pour tous les autres couples (u',v') de mots de C, $d(u',v') \geq d(u,v)$, on a $d(u',v') > t$ et les boules centrées en les mots de C ne contiennent chacune qu'un seul mot de C : C est bien t-détecteur. $\diamondsuit$

Définition 4.11 *Soit C un code de longueur n, $u \in C$, e un mot de poids t tel que $v = u \oplus e \notin C$. On dit que C corrige v si*

1. *il n'existe pas de mot du code dont la distance à v soit strictement inférieure à t ;*
2. *u est le seul mot du code à distance t de v.*

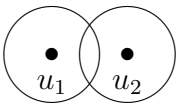
En d'autres ternes, la boule $B(v,t)$ ne contient qu'un seul mot du code : u.

Définition 4.12 *Un code est dit t-*correcteur *s'il peut corriger tous les mots ayant subi de 1 à t erreurs, c'est-à-dire tous les mots situés à une distance non nulle inférieure ou égale à t de C.*

On peut remarquer que pour qu'un code soit t-correcteur, il est nécessaire qu'il soit t-détecteur, et donc que sa distance minimale soit supérieure à t.

Proposition 4.3 *Soit C un code t-correcteur. Alors toutes les boules de rayon t, centrées en les mots de C sont disjointes.*

preuve : On va montrer que si deux boules de rayon t, centrées en deux mots distincts u_1 et u_2 de C, ne sont pas disjointes, alors C n'est pas t-correcteur.



Appelons d la distance séparant u_1 de u_2 : $d = d(u_1, u_2)$. Remarquons qu'en vertu de l'inégalité triangulaire, $d \leq 2t$, et que compte tenu de la remarque ci-dessus, on peut supposer $d > t$.

Il existe une suite de mots $v_1 = u_1, v_2, v_3, \ldots, v_{d+1} = u_2$ tels que la distance de $d(v_i, v_{i+1}) = 1$ (comme il y a $d + 1$ mots, cela signifie que $d(v_i, u_1) + d(v_i, u_2) = d$).

Le mot v_t se trouve à une distance t de u_1, et à une distance $d - t$ de u_2.

On distingue deux cas :

- si $d < 2t$, alors $d - t < t$ et v_t est plus proche de u_2 que de u_1 ;
- si $d = 2t$, alros $d - t = t$ et v_t est à égale distance de u_1 et u_2. On distingue deux cas :
  - soit u_1 et u_2 sont les mots du code les plus proches de v_t ;
  - soit il existe un mot du code plus proche encore de v_t.

Dans tous ces cas, C ne peut pas corriger v_t. C n'est donc pas t-correcteur. ♢

La réciproque est vraie :

Proposition 4.4 *Soit C un code. Alors si toutes les boules de rayon t, centrées en les mots de C sont disjointes, C est t-correcteur.*

preuve : laissée en exercice. ♢

Corollaire 4.1 *Pour qu'un code de distance minimale d soit t-correcteur, il faut et il suffit que $d \geq 2t + 1$.*

preuve : laissée en exercice. ♢

Ainsi, si d est la distance minimale du code C, ce code sait détecter jusqu'à $d - 1$ erreurs et corriger jusqu'à $\lfloor \frac{d-1}{2} \rfloor$ erreurs.

VI Codes linéaires

Nous allons nous intéresser ici à une famille de codes : les codes linéaires (binaires).

1) Définition et premières propriétés

Définition 4.13 *Un code binaire C est dit* linéaire *si $C \neq \emptyset$ et que toute somme de mots de C est encore un mot de C :*

$$\forall\, u, v \in C, \qquad u \oplus v \in C.$$

Proposition 4.5 *Soit C un code linéaire. Alors $0 \in C$.*

preuve : Soit $u \in C$; comme C est linéaire, $u \oplus u = 0$ est aussi un mot de C. $\diamondsuit$

Exemple 4.9 *Les codes suivants sont linéaires : $\{0\}$ (le mot 0 de longueur n : c'est le plus petit code linéaire de longueur n), $\{0,1\}^n$ (l'espace tout entier : c'est le plus grand code linéaire de longueur n), le code obtenu par ajout d'un bit de parité est linéaire, le code à répétition, ...*

Proposition 4.6 *Soit C un code linéaire. Sa distance minimale satisfait*

$$d = \min_{\substack{u \in C \\ u \neq 0}} wt(u)$$

preuve : Soitent $u, v \in C$ tels que $d(u,v) = wt(u \oplus v) = d$; alors le mot $z = u \oplus v$ appartient à C (linéarité) et $d = wt(z)$. De plus, tout autre mot z' de C est tel que $wt(z') \geq wt(z)$: en effet si il existait un z' tel que $wt(z') < wt(z)$, alors cela signifierait que $d(z',0) < d(z,0) = d$ ce qui est impossible (la distance minimale du code est d, et ne peut donc pas lui être inférieure). $\diamondsuit$

Définition 4.14 *Soit C un code linéaire de longueur n, et $u \in \{0,1\}^n$. On appelle* translaté de C engendré par u *le code :*

$$u \oplus C = \{u \oplus v, v \in C\}$$

Proposition 4.7 *Soit C un code linéaire de longueur n, et $u \in \{0,1\}^n$. Alors $(u \oplus C) \cup C$ est un code linéaire.*

preuve : Soient $v, w \in (u \oplus C) \cup C$; on va montrer que $v \oplus w \in (u \oplus C) \cup C$. Trois cas se présentent :

– si $v, w \in C$: la linéarité de C permet de conclure que $v \oplus w \in C$;

- si $v, w \in u \oplus C$: on va montrer que $v \oplus w \in C$. Il existe $v', w' \in C$ tels que $v = u \oplus v'$ et $w = u \oplus w'$; ainsi, $v \oplus w = u \oplus v' \oplus u \oplus w' = v' \oplus w'$. Or v' et w' sont deux mots de C qui est un code linéaire ; on a donc $v' \oplus w' \in C$, et finalement $v \oplus w \in C$;
- si $v \in C$ et $w \in u \oplus C$: soit $w = u \oplus z$, $z \in C$, alors on a $v \oplus w = v \oplus u \oplus z = u \oplus v \oplus z$; comme $v, z \in C$, on a $v \oplus z \in C$ et donc $v \oplus w \in u \oplus C$.

♢

Proposition 4.8 *Soit C un code linéaire de longueur n, et $u \in \{0,1\}^n$. Alors $u \oplus C$ et C ont même cardinal.*

preuve : Considérons maintenant l'application

$$\begin{array}{rccc} f_u : & \{0,1\}^n & \longrightarrow & \{0,1\}^n \\ & v & \longmapsto & v \oplus u \end{array}$$

Cette application est injective (deux mots distincts ont des images distinctes), *i.e.*

$$\forall\, v, w \in \{0,1\}^n, f_u(v) = f_u(w) \Longrightarrow v = w$$

En effet, si on suppose que $f_u(v) = f_u(w)$, alors on a $u \oplus v = u \oplus w$, et donc $u \oplus u \oplus v = w$, ce qui donne $v = w$.

Ainsi, $f_u(\{0,1\}^n) = \{f_u(v), v \in \{0,1\}^n\}$ contient autant d'éléments que $\{0,1\}^n$ et f_u est bijective.

Donc, si C est un code de longueur n on a $C \subseteq \{0,1\}^n$, et $u \oplus C = f_u(C)$ contient autant d'éléments que C. ♢

Proposition 4.9 *Soit C un code linéaire de longueur n, et $u \in \{0,1\}^n$. Alors on a :*

$$\left\{ \begin{array}{rcl} u \in C & \Longrightarrow & u \oplus C = C \\ u \notin C & \Longrightarrow & (u \oplus C) \cap C = \emptyset \end{array} \right.$$

preuve : 1) soit $u \in C$: on va montrer que $u \oplus C \subseteq C$. Soit $v \in u \oplus C$, $v = u \oplus w$ avec $w \in C$. Comme u et w sont deux mots de C et que C est linéaire, $v = u \oplus w \in C$. Tout mot de $u \oplus C$ appartient donc à C et on a bien $u \oplus C \subseteq C$. Comme on sait (par la proposition précédente) que $u \oplus C$ contient le même nombre d'éléments que C, on a $u \oplus C = C$.
2) soit $u \notin C$; raisonnons par l'absurde et supposons que $(u \oplus C) \cap C \neq \emptyset$. Soit $w \in (u \oplus C) \cap C$; il existe $v \in C$ tel que $w = u \oplus v$ et donc on a $u = v \oplus w$; or $v, w \in C$ et donc on a $u \in C$, ce qui contredit l'hypothèse de départ. ♢

Remarque : on peut en déduire que si $u \notin C$, alors $u \oplus C$ n'est pas un code linéaire (il ne contient pas le mot nul).

2) Constructions

L'élément de base de la construction d'un code linéaire est l'addition des mots. Regardons sur quelques exemples ce que cela implique quant à la représentation du code.

Exemple 4.10 *Soit* $C = \{0000, 0011, 0101, 1001, 0110, 1100, 1010, 1111\}$ *le code de parité de longueur* 4*, qui est un code linéaire. On va le reconstruire petit à petit, en ajoutant un par un des mots et en complétant par addition. Considérons l'ensemble*

$$\mathcal{E} = \{\}$$

On va reconstruire le code C *à partir de cet ensemble : prenons un mot de* C *qui n'est pas dans* $\mathcal{E}$*, par exemple* 0011*, et rajoutons-le :* $\mathcal{E} = \{0011\}$*. Comme on veut que* $\mathcal{E}$ *soit un code linéaire, on rajoute automatiquement aussi les mots obtenus par addition ; on a donc finalement :*

$$\mathcal{E} = \{0000, 0011\}$$

Prenons maintenant un autre mot de C *qui n'est pas dans* $\mathcal{E}$*, par exemple* 0101*, et rajoutons-le, ainsi que les mots obtenus par addition de* 0101 *avec les mots de* $\mathcal{E}$ *; on obtient :*

$$\mathcal{E} = \{0000, 0011, 0101, 0110\}$$

Prenons maintenant un autre mot de C *qui n'est pas dans* $\mathcal{E}$*, par exemple* 1100*, et rajoutons-le, ainsi que les mots obtenus par addition de* 1100 *avec les mots de* $\mathcal{E}$ *; on obtient :*

$$\mathcal{E} = \{0000, 0011, 0101, 0110, 1100, 1111, 1001, 1010\}$$

On a entièrement reconstitué le code C *en ajoutant* $0011, 0101, 1100$ *et en complétant à chaque par addition. Ceci signifie que le code paut être défini entièrement par ces* 3 *mots-là.*

Cet exemple nous montre qu'on peut essayer de repésenter un code linéaire par un sous-ensemble de ses mots.

Définition 4.15 *Soit* C *un code linéaire. On appelle* partie génératrice de C *tout ensemble de mots de* C *qui, lorsqu'on le complète par addition, redonne le code tout entier. On dit qu'elle est* minimale *si lorsau'on lui enlève un élément elle n'est plus génératrice du code.*

Une première remarque s'impose ici : tout code linéaire C possède au moins une partie génératrice, lui-meme. La question que l'on peut alors se poser est : ¡¡peut-on trouver une partie génératrice plus petite ? ¿¿

Exemple 4.11 *Soit* $C = \{0000, 0011, 0101, 1001, 0110, 1100, 1010, 1111\}$ *le code de parité de longueur* 4*, qui est un code linéaire. Soit*

$$\mathcal{E} = \{0000, 0011, 0101, 1001, 0110, 1100, 1010, 1111\}$$

On va essayer de réduire $\mathcal{E}$ *tout en conservant la propriété que les mots de* $\mathcal{E}$ *engendrent bien tout le code* C*. Commençons par enlever* 0000*, qui peut être obtenu avec n'importer quel mot (en l'additionnant à lui-même) :*

$$\mathcal{E} = \{0011, 0101, 1001, 0110, 1100, 1010, 1111\}$$

Essayons maintenant de supprimer 1111 *; on obtient alors*

$$\mathcal{E} = \{0011, 0101, 1001, 0110, 1100, 1010\}$$

Cet ensemble engendre bien C*, puisque* $0011 \oplus 1100 = 1111$*. On continue, et on enlève* 1010 *; on obtient*

$$\mathcal{E} = \{0011, 0101, 1001, 0110, 1100\}$$

Cet ensemble engendre bien C*, puisque* $0110 \oplus 1100 = 1010$ *et* $0011 \oplus 1100 = 1111$*. On continue, et on enlève* 1100 *; on obtient*

$$\mathcal{E} = \{0011, 0101, 1001, 0110\}$$

Cet ensemble engendre bien C*, puique* $0101 \oplus 1001 = 1100$*,* $0110 \oplus 1100 = 1010$ *et* $0011 \oplus 1100 = 1111$*. On continue, et on enlève* 0110 *; on obtient*

$$\mathcal{E} = \{0011, 0101, 1001\}$$

Cet ensemble engendre bien C*, puisque* $0011 \oplus 0101 = 0110$*,* $0101 \oplus 1001 = 1100$*,* $0110 \oplus 1100 = 1010$ *et* $0011 \oplus 1100 = 1111$*. On continue, et on enlève* 1001 *; on obtient*

$$\mathcal{E} = \{0011, 0101\}$$

Cet ensemble engendre $\{0011, 0101, 0110, 0000\} \neq C$*. Donc on ne peut pas enlever* 1001*. On essaie alors d'enlever* 0101 *:*

$$\mathcal{E} = \{0011, 1001\}$$

Cet ensemble engendre $\{0011, 1001, 1010, 0000\} \neq C$*. Donc on ne peut pas enlever* 0101*. On essaie alors d'enlever* 0011 *:*

$$\mathcal{E} = \{0101, 1001\}$$

Cet ensemble engendre $\{0101, 1001, 1100, 0000\} \neq C$*. Donc on ne peut pas enlever* 0011*. On ne peut pas réduire*

$$\mathcal{E} = \{0011, 0101, 1001\}$$

On a trouvé une partie de 3 *mots qui engendre* C*.*

Suite à ces exemples, on peut se poser pluseurs questions :
- ¡¡ Tout code linéaire admet-il une partie génératrice minimale ? ¿¿ La réponse est oui, car le code possédant un nombre fini d'éléments ; ce résultat sera abordé dans le cours de mathématiques du second semestre (algèbre linéaire) ;
- ¡¡ Une partie génératrice minimale est-elle unique ? ¿¿ La réponse st non : dans les deux exemples précédents, on a trouvé deux partie génératrices minimales différentes pour le code de parité de longuer 4 ;
- ¡¡ Les parties génératrices minimales ont-elles toutes le même cardinal ? ¿¿ La réponse est oui, et nous allons le démontrer maintenant.

Proposition 4.10 *Soit C un code linéaire. Alors le nombre de mots qu'il contient est de la forme 2^k. Ceci signifie que ses parties génératrices minimales contiennent toutes k mots.*

preuve : On va construire le code C petit à petit, en comptant, pour chaque mot rajouté le nombre de mots engendrés. On va procéder comme dans le premier exemple de ce paragraphe : on définit deux ensembles $\mathcal{E}_1$ et $\mathcal{E}_2$ qui seront respectivement égaux, au début à $\{0\}$ (le mot ne contenant que des 0) et $C \setminus \{0\}$.

Lors de la construction, $\mathcal{E}_1$ sera toujours un code linéaire ; on l'initialise à $\{0\}$ qui est le plus petit code linéaire possible.

On utilise une variable $taille$ qui va mesurer le nombre d'éléments de $\mathcal{E}_1$; au débaut, $taille = 1$.

Tant que $\mathcal{E}_2 \neq \emptyset$, faire :

$\mathcal{E}_{temp} = u \oplus \mathcal{E}_1$ (les mots obtenus par addition de u)	$\xleftarrow{u}$	choisir $u \in \mathcal{E}_2$
$\mathcal{E}_1 = \mathcal{E}_1 \cup \mathcal{E}_{temp}$ (proposition 4.6 : $\mathcal{E}_1$ de vient un nouveau code linéaire : on a rajouté le générateur u)		$\mathcal{E}_2 = \mathcal{E}_2 \setminus \mathcal{E}_{temp}$ (on enlève de $\mathcal{E}_2$ tous les mots ajoutés à $\mathcal{E}_1$ grâce à u)
$taille = 2 \times taille$ (voir les propositions 4.7 et 4.8)		

Ce procédé s'arrêtera forcément puisque C ne contient qu'un nombre fini d'éléments ; ce nombre sera la valeur de $taille$ à la fin et sera bien de la forme 2^k, où k sera le nombre de mots u qu'on aura rajoutés. ♢

Définition 4.16 *Soit C un code linéaire contenant 2^k mots. Le paramètre k, cardinal de toute partie génératrice minimale, s'appelle la* dimension de C.

Table des matières

Introduction **3**

1 Représentation des nombres **5**
I Quelques systèmes de représentation des entiers naturels 5
II Entiers naturels : bases de numération 5
1) Système décimal 5
2) Système de base b quelconque 6
3) Taille du codage des entiers en base b 7
4) Bases les plus utilisées en informatique 8
III Autres codages des entiers naturels 8
1) Décimal codé binaire (calculatrices de poche) : BCD (Binary Coded Decimal) 8
2) Biquinaire (1956 sur IBM 605) 9
IV Représentation des autres nombres 9
1) Entiers relatifs (signés) 9
2) Nombres fractionnaires (réels) 10

2 Codes, codages et décodages **13**
I Alphabets et mots 14
1) Alphabet 14
2) Mots 14
3) Longueur d'un mot 15
4) Ensembles de mots construits sur un alphabet 15
5) Langages 16
6) Opération de concaténation 16
7) Préfixe d'un mot 16
II Codage 17
1) Morphisme 17
2) Codage 18
3) Décodage 19
III Cas particuliers de codes 19
1) Codes de longueur fixe 19
2) Codes à "virgule" 19

3) Codes préfixes 20
IV Algorithme de Sardinas-Patterson 21
1) Résiduel 21
2) Langage quotient 22
3) L'algorithme 22
V Inégalité de Kraft 24
1) Représentation graphique d'un code préfixe 24
2) Inégalité de Kraft 25

3 Codage optimal **29**
I Longueur moyenne 29
II Codage optimal : définitions et propriétés 32
III Codage optimal : constructions 35
1) Cas de la distribution uniforme 35
2) Cas d'une distribution de la forme $f(x) = \frac{1}{2^k}$ 35
3) Cas général : codage de Huffman 37
IV Notion d'entropie 39
1) Quantité d'information 39
2) Entropie d'une source 39
3) Théorème du codage sans bruit 40

4 Codes correcteurs **43**
I Contexte et motivations 43
II Le modèle 44
III Détection et correction des erreurs : principes 46
IV Métrique de Hamming 48
V Détection et correction des erreurs : résultats 51
VI Codes linéaires 53
1) Définition et premières propriétés 53
2) Constructions 55