PF1 — Principes de Fonctionnement des machines binaires

Jean-Baptiste Yunès
Jean.Baptiste.Yunes@univ-paris-diderot.fr

Version 1.0

Q : Rions un peu :

à quoi ressemble le circuit (combinatoire) qui détermine si un nombre est premier ou non ?

prime checker

On va se limiter aux nombres <16, <32, <64 et <128

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127

Q: réaliser le circuit de la fonction ¬((a ∨ b) ∧ c) en utilisant que des portes et et non

Q: réaliser le circuit de la fonction ¬((a ∨ b) ∧ c) en utilisant que des portes non-ou

Puisqu'il existe différentes réalisations, on peut se poser la question du coût ?

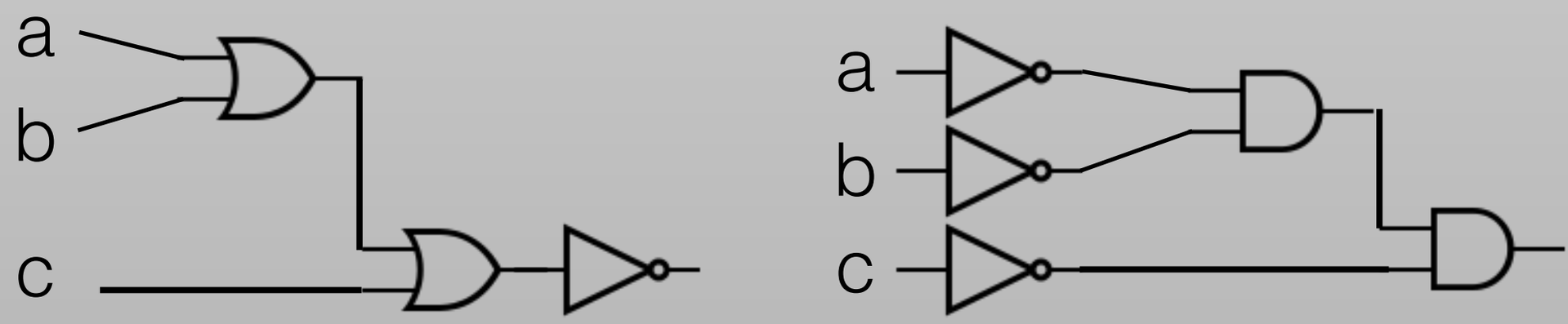
Par exemple, peut-on minimiser le nombre de portes ?

Cela revient au problème de la simplification des expressions logiques…

Nous avons déjà donné quelques propriétés des connecteurs comme par exemple une des lois de De Morgan :

$$\neg(a \vee b) \Leftrightarrow \neg a \wedge \neg b$$

cette loi permet de simplifier un circuit, comparons les deux :



En appliquant les propriétés déjà données on peut donc simplifier une expression (et donc un circuit)

Mais dans quel ordre doit-on appliquer les règles ? Le problème n'est pas simple…

Il faut de la méthode…

La méthode de Karnaugh permet de simplifier les expressions lorsque la fonction ne possède pas trop de variables

C'est une méthode visuelle/graphique…

L’idée de base est d’utiliser les propriétés $a \vee (\neg a) \Leftrightarrow \top$ et $a \wedge b + \neg a \wedge b \Leftrightarrow b$ dans les mintermes de la FND

cela conduit à éliminer ces instances de la variable a dans l’expression et donc diminuer la taille du circuit…

Il faut fabriquer un tableau de **Karnaugh**

Maurice Karnaugh

C’est la représentation d’une fonction booléenne dans laquelle les valuations sont ordonnées de sorte que d’une ligne/colonne à l’autre une seule variable change de valeur

C’est possible ça ?

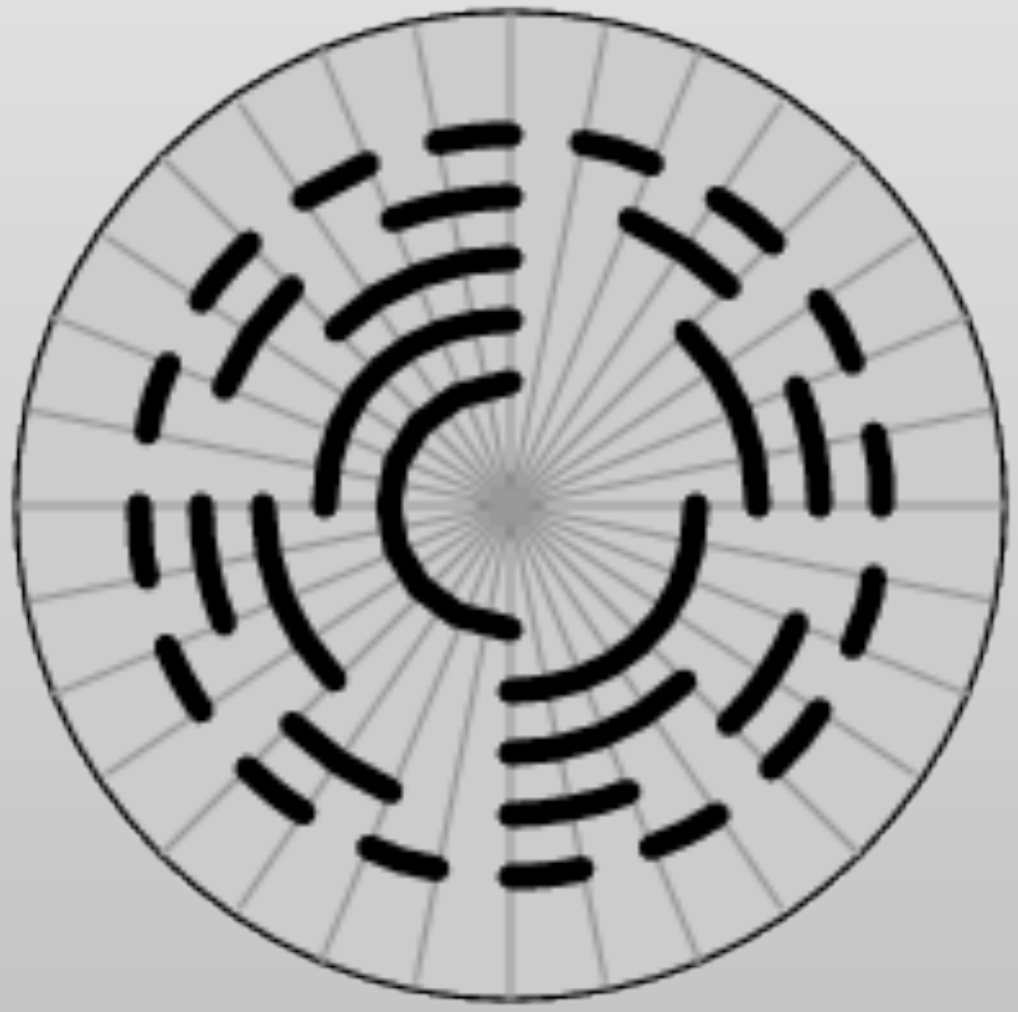
Oui, c’est le **code de Gray** ou **code binaire réfléchi**…

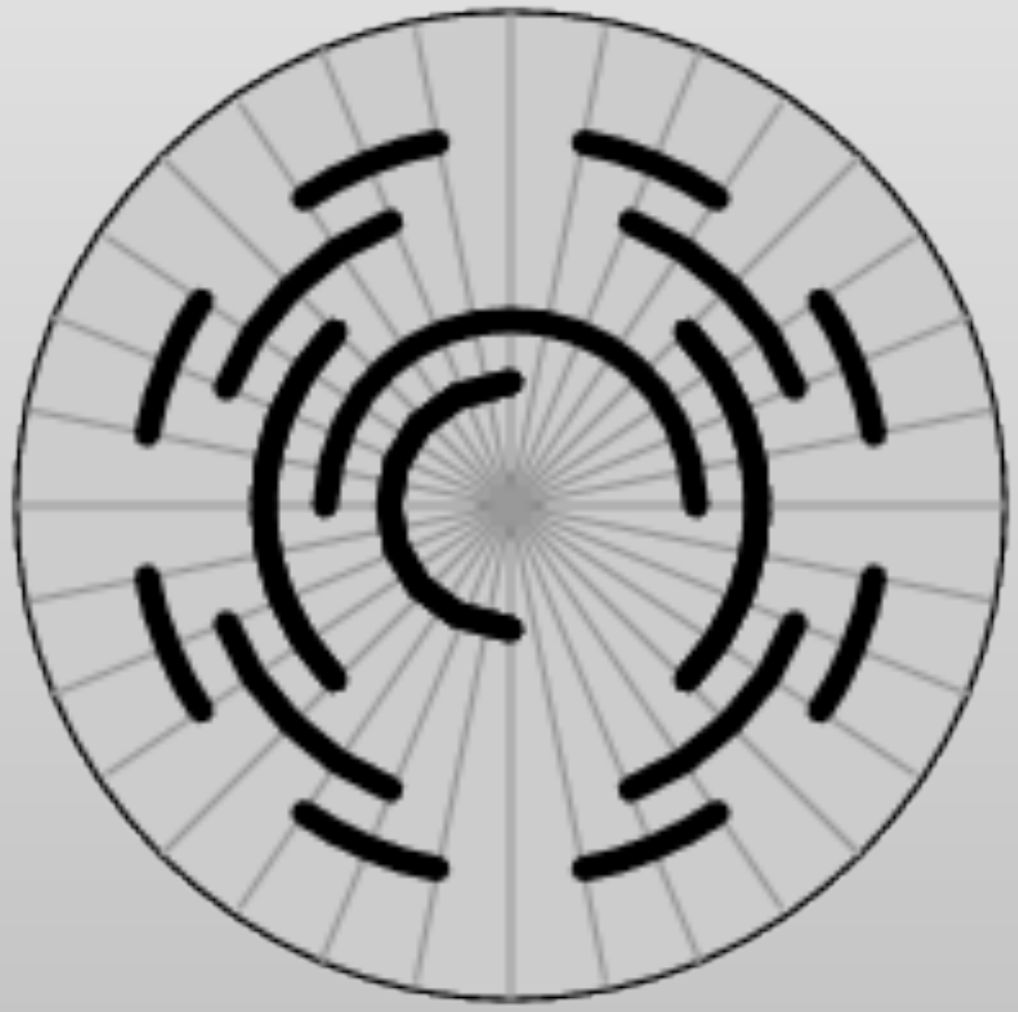
C’est très utile notamment en circuiterie afin d’éviter les états transitoires…

code de Gray pour les mots de 1 bits : 0, 1
code de Gray pour les mots de 2 bits : 0**0**, 0**1**, 1**1**, 1**0**
code de Gray pour les mots de 3 bits : 0**00**, 0**01**, 0**11**, 0**10**, 1**10**, 1**11**, 1**01**, 1**00**

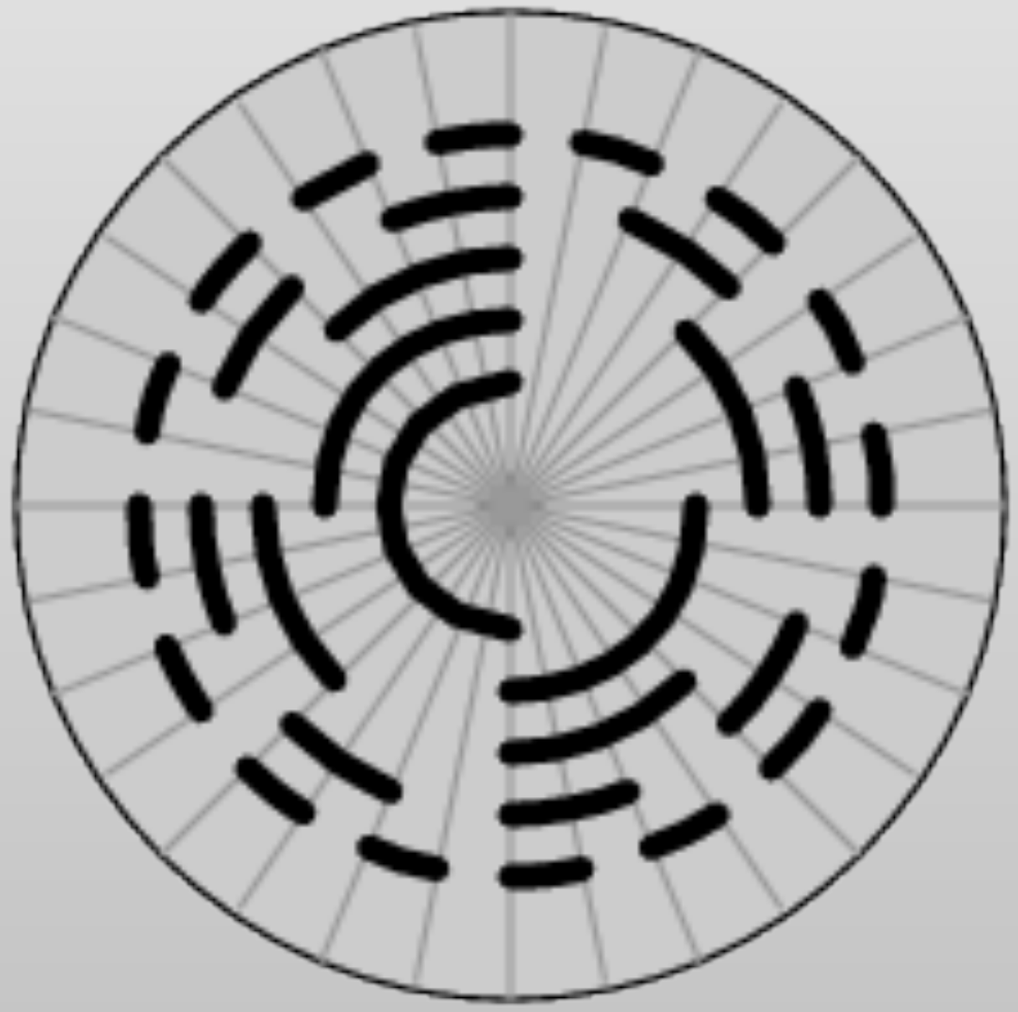
0	00	000
1	01	001
	11	011
	10	010
		110
		111
		101
		100

on remarque que d'une ligne à l'autre un seul bit est modifié et ceci est valable aussi en passant de la dernière ligne à la première
le code de Gray est un tore

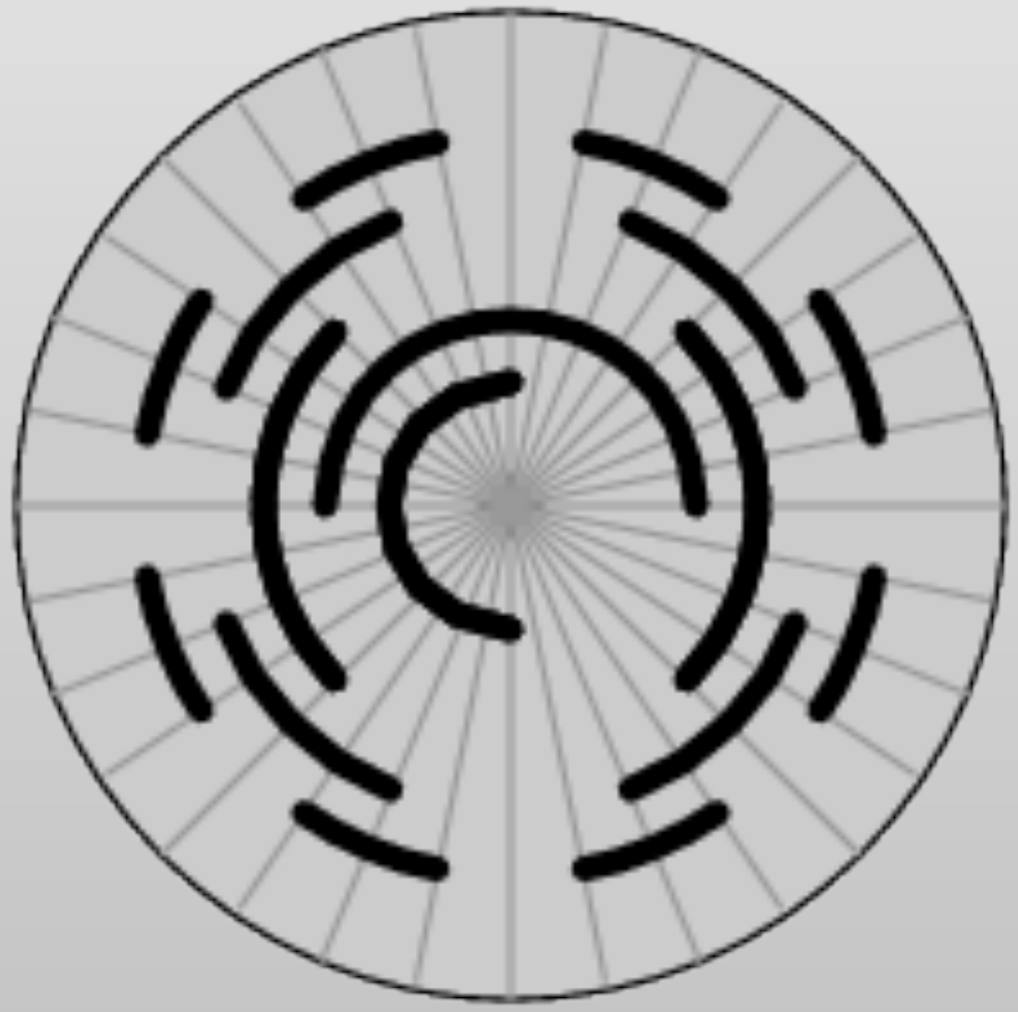




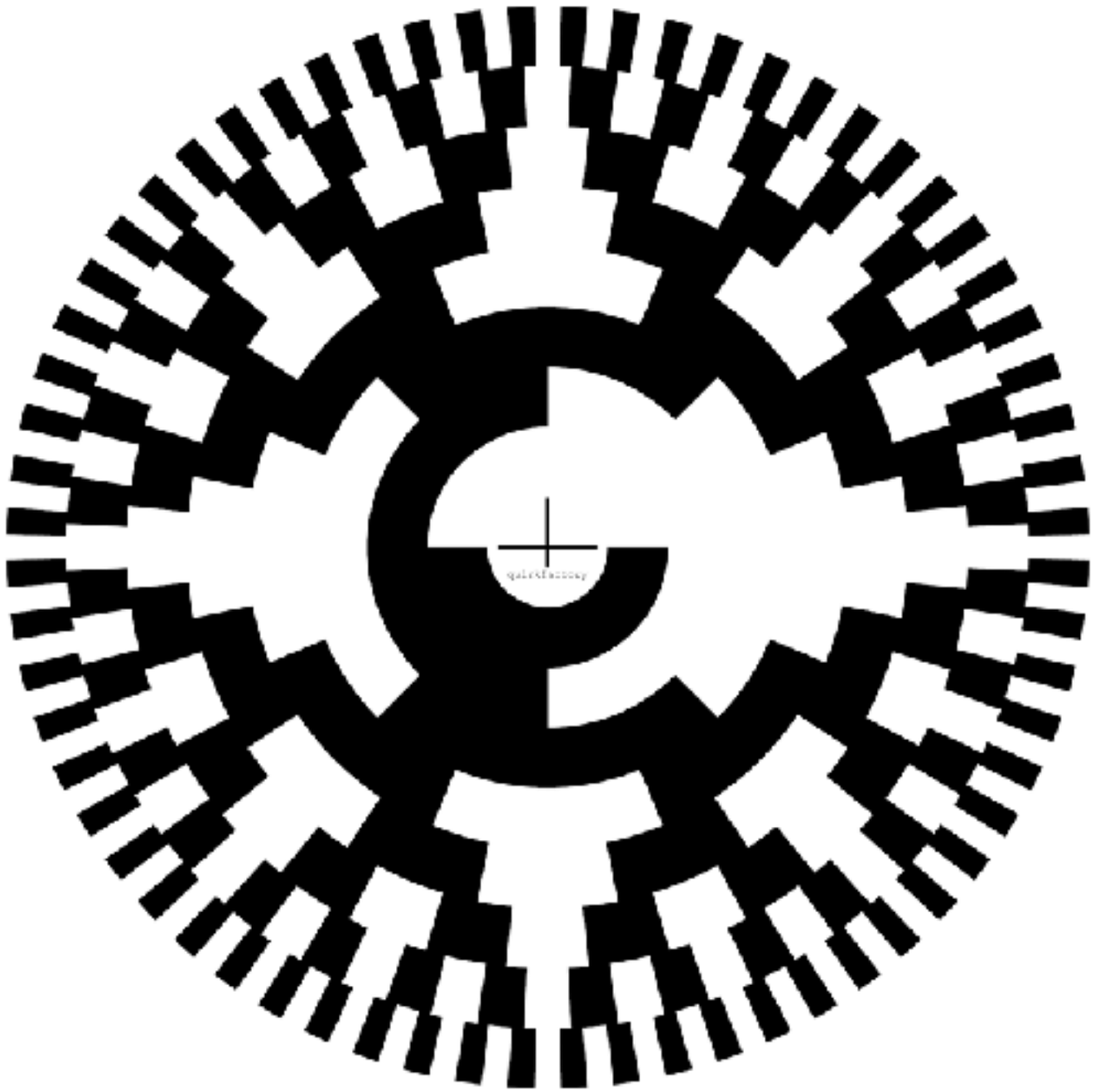
Quelle roue correspond à un codage Binaire ? de Gray ?



Binaire



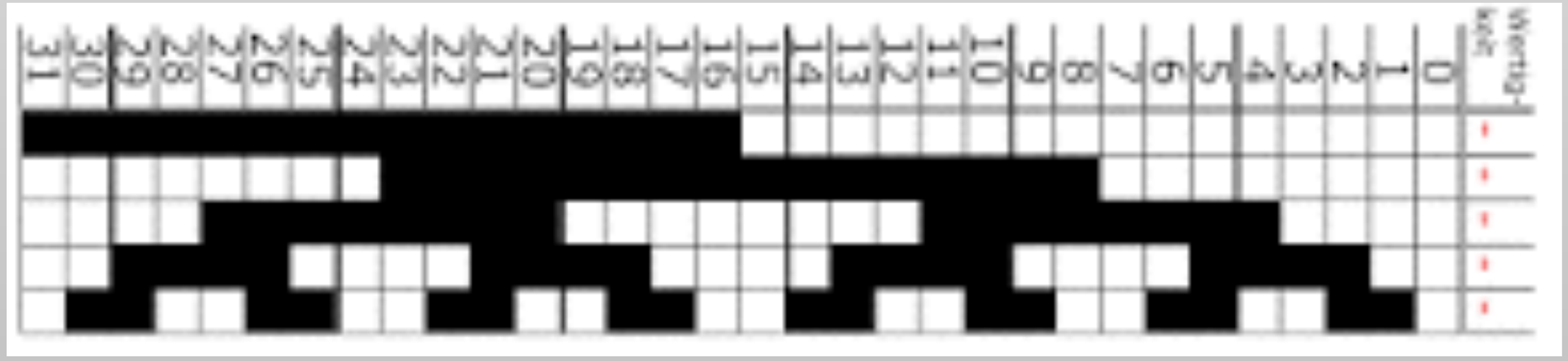
Gray



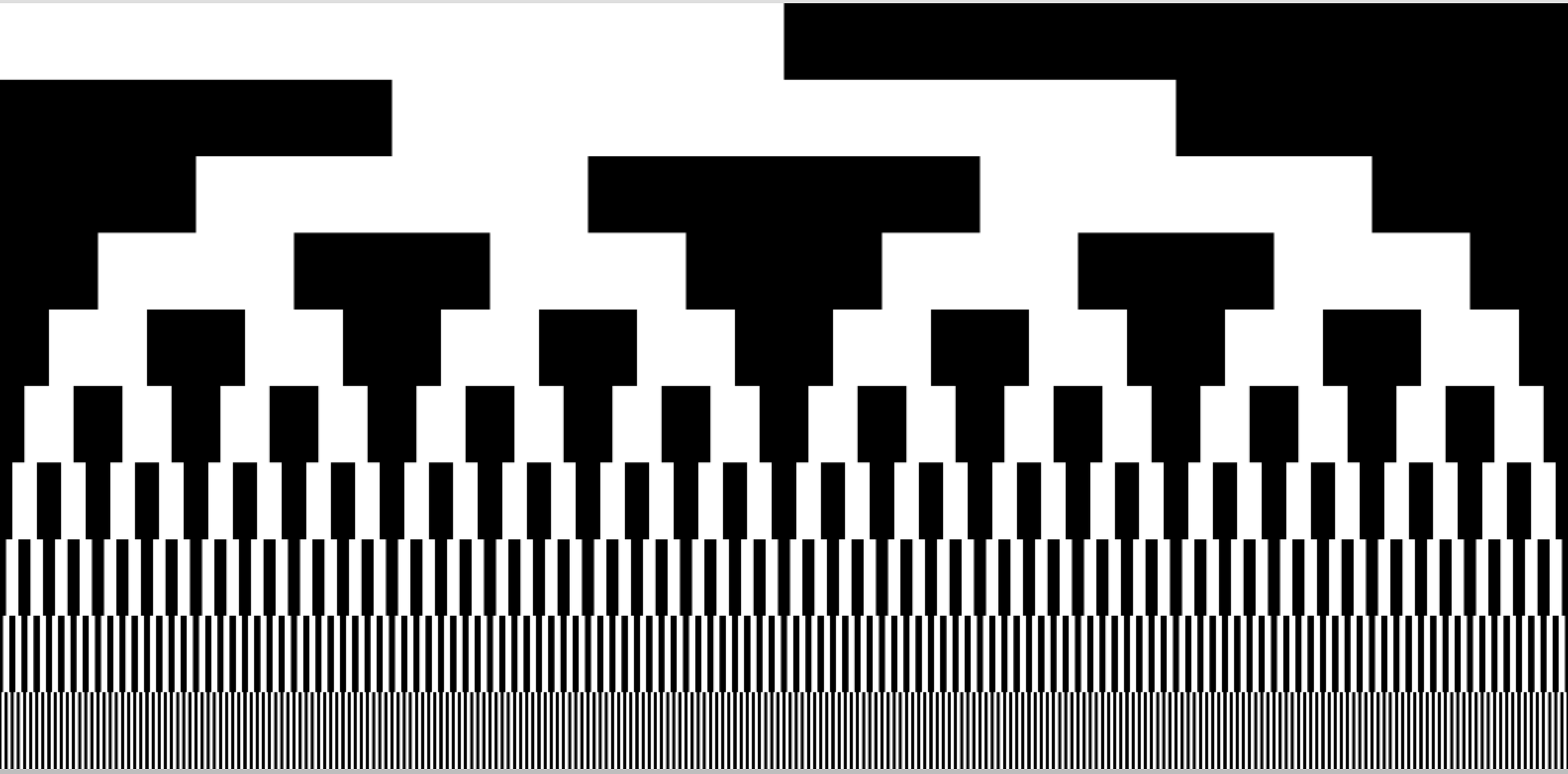
Une autre représentation du code de Gray

Gray encoding using 8 bits

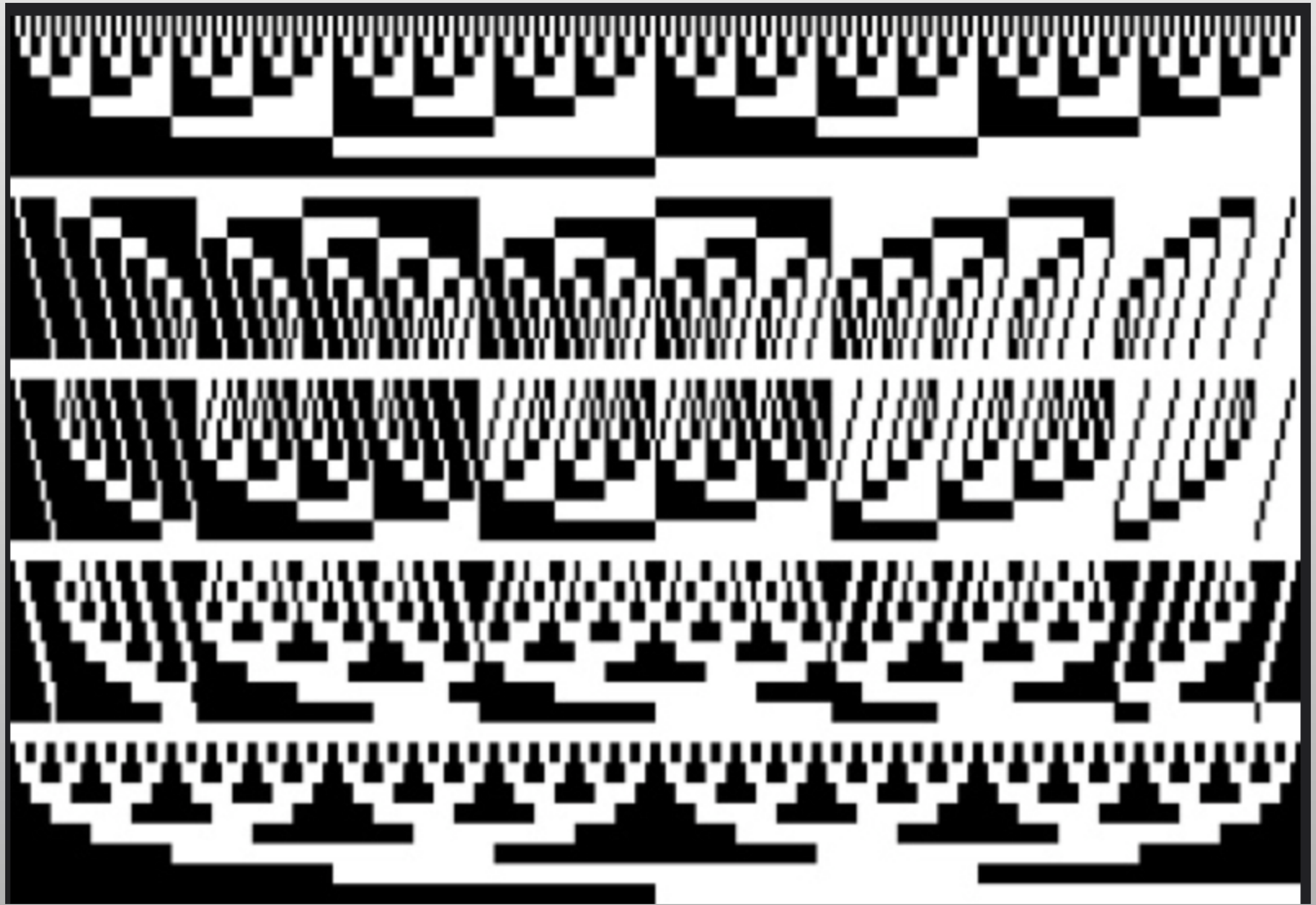
Encore une autre représentation du code de Gray



Encore un autre représentation du code de Gray



Différentes énumérations



Q : Problème, comment passer de la représentation binaire à celle de Gray ?

Q : est l’inverse ?

Binaire vers Gray

Binaire $b_{n-1}b_{n-2}\ldots b_1b_0$

Gray $g_{n-1}=b_{n-1}$, $g_i=b_{i+1}\oplus b_i$

Exemple : 1001 (9) donne 1101 ($10^{\text{ième}}$ code de Gray)

Gray vers Binaire

Gray $g_{n-1}g_{n-2}...g_1g_0$

Binaire $b_{n-1}=g_{n-1}$, $b_i=g_i \oplus b_{i+1}$

Exemple : 1001 donne 1110 donc 1001 est le $15^{ième}$ code de Gray

Un **tableau de Karnaugh** consiste à exprimer la valeur de la fonction sous la forme d'un tableau **torique** dans lequel les variables sont également réparties en entrées des lignes et colonnes et listées selon le code de Gray, comme ici pour une fonction a 5 variables :

abc de	000	001	011	010	110	111	101	100
00								
01								
11								
10								

Dans un tel tableau la **méthode de Karnaugh** consiste à rechercher des rectangles dont les dimensions sont de la forme 2^p (1, 2, 4, 8, 16, etc), dans lesquels la fonction vaut 1 partout et dont p variables sont non-constantes les autres étant fixées par exemple :

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	0	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

Par exemple on ne peut pas regrouper ainsi:

abc de	000	001	011	010	110	111	101	100
00	0	1	1	1	1	0	0	0
01	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0

le paquet est de longueur 2^2, abc changent toutes les trois, c'est donc un mauvais regroupement

On ne peut que regrouper de la façon suivante:

abc de	000	001	011	010	110	111	101	100
00	0	1	1	1	1	0	0	0
01	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	0	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

pour le rectangle **jaune** on constate que la variable b prend les valeurs 0 et 1 alors que les autres variables sont fixes, les mintermes correspondants peuvent être simplifiés en le simple terme **$\neg a \wedge c \wedge \neg d \wedge \neg e$** (2 mintermes à 5 variables remplacés par un terme a 4 variables)

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	0	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

pour le rectangle **vert** on constate que les variables b,c,e prennent les valeurs 0 et 1 alors que les variables a et d sont fixes, les mintermes correspondants peuvent être simplifiés en le simple terme **¬a∧d** (8 mintermes à 5 variables remplacés par un simple terme à 2 variables!)

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	0	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

pour le rectangle **orange** il faut ne pas oublier que le tableau est torique! Ici les variables c et d prennent toutes les valeurs possibles alors que a,b,e sont fixes, le terme obtenu est donc $a \wedge \neg b \wedge \neg e$.

On remplace 4 mintermes à 5 variables par un terme à 3 variables.

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	0	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

les rectangles **violet** et **rouge** ne sont pas simplifiables, on conserve les mintermes initiaux **a∧b∧¬c∧¬d∧e** et **a∧b∧c∧d∧e**

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	0	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

Ainsi la fonction s'écrit en FND :

(¬a∧¬b∧c∧¬d∧¬e) ∨ (¬a∧b∧c∧¬d∧¬e) ∨
(a∧¬b∧c∧¬d∧¬e) ∨ (a∧¬b∧¬c∧¬d∧¬e) ∨
(a∧b∧¬c∧¬d∧e) ∨ (¬a∧¬b∧¬c∧d∧e) ∨
(¬a∧¬b∧c∧d∧e) ∨ (¬a∧b∧c∧d∧e) ∨ (¬a∧b∧¬c∧d∧e) ∨
(a∧b∧c∧d∧e) ∨ (¬a∧¬b∧¬c∧d∧¬e) ∨
(¬a∧¬b∧c∧d∧¬e) ∨ (¬a∧b∧c∧d∧¬e) ∨
(¬a∧b∧¬c∧d∧¬e) ∨ (a∧¬b∧c∧d∧¬e) ∨ (a∧¬b∧¬c∧d∧¬e)

et peut s'écrire en forme simplifiée (disjonction des termes simplifiés) :

(¬a∧c∧¬d∧¬e) ∨ (¬a∧d) ∨ (a∧¬b∧¬e) ∨
(a∧b∧¬c∧¬d∧e) ∨ (a∧b∧c∧d∧e)

on aurait plus simplifier encore un peu plus en remarquant que :

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	0	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

$(\neg a \wedge c \wedge \neg e) \vee (\neg a \wedge d) \vee (a \wedge \neg b \wedge \neg e) \vee$
$(a \wedge b \wedge \neg c \wedge \neg d \wedge e) \vee (a \wedge b \wedge c \wedge d \wedge e)$

attention, il faut toujours essayer de combiner les 1 dans des rectangles aussi grands que possible…

c’est le cas du rectangle **marron** qu’il faut essayer de ne pas laisser seul

¬a∧¬b∧c∧¬d∧e

si on le combine on obtient

¬a∧¬b∧c

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	1	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

abc de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	1	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

c’est le cas du rectangle jaune (on a oublié la propriété de tore du code de Gray)

$\neg a \wedge c \wedge \neg d \wedge \neg e$

si on le combine on obtient

$\neg a \wedge c \wedge e$

abc / de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	1	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

abc / de	000	001	011	010	110	111	101	100
00	0	1	1	0	0	0	1	1
01	0	1	0	0	1	0	0	0
11	1	1	1	1	0	1	0	0
10	1	1	1	1	0	0	1	1

Q: l'afficheur à 7 segments pour les chiffres décimaux codés **en binaire** :

écrire les fonctions associés à chaque segment sous la FND

simplifier les fonctions par la méthode de Karnaugh

Q: même question pour les chiffres hexadécimaux :