DM5 : Code de Gray

A rendre le 25/01 (facultatif). Toutes les fonctions demandées devront utiliser la syntaxe Ocaml et recourir à la récursivité autant que possible. Il est possible d'introduire des fonctions auxilliaires à condition que leur fonctionnement soit clairement décrit. Vous préciserez également la signature de chaque fonction proposée. Il est bien sûr possible d'implémenter ces fonctions sur une machine avant de recopier le code produit sur votre copie.

Introduction

Dans tout le sujet $\oplus$ désigne l'opérateur ou exclusif (XOR) défini sur $\{0,1\}$ par la table de vérité suivante :

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Cet opérateur est étendu aux entiers naturels de la façon suivante : si $a, b \in \mathbb{N}$ sont tels que $a < 2^p$ et $b < 2^p$ pour p un entier naturel et si $a = (a_p...a_0)_2$, $b = (b_p...b_0)_2$, alors $a \oplus b$ est l'entier dont une représentation binaire est $(c_p...c_0)_2$ avec pour tout $i \in [\![0, p]\!]$, $c_i = a_i \oplus b_i$ (autrement dit pour calculer $a \oplus b$, on applique $\oplus$ bit à bit).

Soit $n \in \mathbb{N}$. L'objectif de ce sujet est d'énumérer les n-uplets constitués de 0 et de 1 de sorte à coder les entiers de $[\![0, 2^n - 1]\!]$ de manière astucieuse. Tout n-uplet de 0 et de 1 sera implémenté à l'aide d'une liste.

Partie 1 *Enumération lexicographique*

Dans cette partie on cherche à énumérer les n-uplets de 0 et de 1 dans l'ordre lexicographique. Par exemple, pour $n = 3$, on obtiendra la suite de n-uplets suivante : 000, 001, 010, 011, 100, 101, 110, 111. Remarquons qu'énumérer ces n-uplets dans cet ordre correspond à énumérer les représentations binaires sur 3 bits des entiers de $[\![0, 7]\!]$ dans l'ordre croissant.

1. Ecrire une fonction `suivant` permettant de calculer le n-uplet suivant celui passé en entrée selon l'ordre lexicographique. Par exemple, `suivant [0;1;1;0;1;1]` devra renvoyer `[0;1;1;1;0;0]`.

2. a) Ecrire une fonction `initialisation` associant à un entier n le n-uplet uniquement constitué de zéros.
   b) Ecrire un fonction `affiche_nuplet` permettant d'afficher le n-uplet passé en argument.
   c) En déduire une fonction `affiche_n` permettant d'afficher tous les n-uplets dans l'ordre lexicographique étant donné un entier n.

Partie 2 *Code de Gray*

Coder un entier par sa représentation binaire a un inconvénient : pour passer de la représentation d'un entier à la représentation de l'entier suivant, il est souvent nécessaire de modifier plusieurs bits. Par exemple, si l'on souhaite passer de 5 à 6, c'est-à-dire de $(101)_2$ à $(110)_2$, il y a un risque de passer transitoirement par $4 = (100)_2$ ou $7 = (111)_2$ selon l'ordre dans lequel les modifications de deux derniers bits terminent. Ceci pose problème par exemple si un système lit la valeur passant de 5 à 6 pendant la période transitoire : la mesure risque alors d'être une valeur parasite.

Pour éviter autant que possible ces états transitoires, on souhaite coder les entiers de sorte à ce qu'il n'y ait qu'un seul bit de différence entre la représentation d'un entier et celle de l'entier suivant. Le code de Gray vérifie cette propriété. Pour produire la liste des n-uplets ordonnés selon l'ordre de Gray, on procède de la façon suivante : on calcule les $(n-1)$-uplets selon l'ordre de Gray, on ajoute 0 en tête de chacun de ces $(n-1)$-uplets en les lisant de gauche à droite et on concatène la liste obtenue à celle obtenue en concaténant un 1 en tête des $(n-1)$-uplets lus de droite à gauche. Par exemple : on obtient pour $n = 1$ la liste $[0, 1]$ puis pour $n = 2$ la liste $[00, 01, 11, 10]$ puis pour $n = 3$ la liste $[000, 001, 011, 010, 101, 111, 110, 100]$.

3. Ecrire une fonction `ajout` telle que, si `a` est un entier et `l` est une liste de liste d'entiers, alors `ajout a l` renvoie une liste contenant les éléments de `l` auxquels on a ajouté `a` en tête.

4. Ecrire une fonction `gray_n` prenant en entrée un entier n et renvoyant la liste des n-uplets de 0 et de 1 selon l'ordre de Gray.

5. Evaluer la complexité de votre fonction `gray_n` en fonction de son entrée n et discuter de la possibilité de concevoir un algorithme ayant la même spécification et une meilleure complexité.

Par défintion, un code de Gray d'un entier k tel que $k < 2^n$ est le $(k+1)$-ème élément dans la liste des n-uplets de 0 et de 1 selon l'ordre de Gray. Par exemple, voici la correspondance entre entier et code de Gray pour les entiers entre 0 et 7 :

entier	0	1	2	3	4	5	6	7
code de Gray	000	001	011	010	110	111	101	100

Notons que de la même façon qu'il n'y a pas unicité de la représentation binaire d'un entier, il n'y a pas unicité du code de Gray mais que la seule différence entre deux codes de Gray pour le même entier est la présence d'un nombre différent de zéros en tête du code. Les codes de Gray ont de multiples applications : correction d'erreurs, minimisation de circuits logiques, utilisation dans certains algorithmes génétiques, minimisation d'erreurs lors de la conversion d'un signal, utilisation dans des capteurs de positions...

Partie 3 *Traduction code binaire - code de Gray*

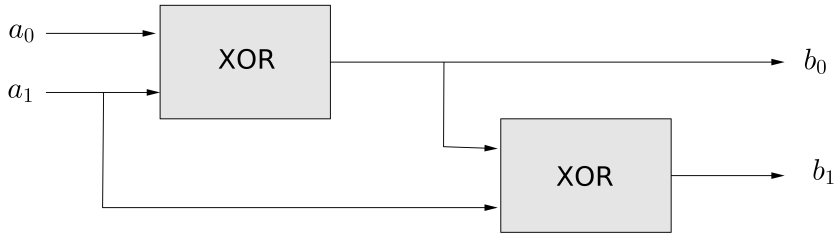
Dans cette partie, on cherche à construire deux fonctions : l'une permettant de déduire le code de Gray d'un entier à partir de sa représentation binaire et l'autre permettant de faire l'opération inverse. On définit une fonction $g : \mathbb{N} \longrightarrow \mathbb{N}$ de la façon suivante. Pour tout $k \in \mathbb{N}$, le calcul de $g(k)$ commence par le calcul de $n \in \mathbb{N}$ tel que $k < 2^n$ puis la construction de la liste des 2^n n-uplets de 0 et de 1 dans l'ordre de Gray : $g(k)$ est alors le nombre dont une représentation binaire est le $(k+1)$-ème de ces n-uplets. Par exemple, en utilisant le tableau ci dessus, $g(0) = 0$, $g(2) = 3$ et $g(7) = 4$.

6. Montrer que la fonction g est bien définie (autrement dit que la valeur de $g(k)$ ne dépend pas de l'entier n tel que $k < 2^n$ choisi dans sa construction).

7. a) Montrer que $g(0) = 0$ et $g(1) = 1$. Montrer que si $k \geq 2$ et si on décompose $k = 2^n + r$ avec $0 \leq r < 2^n$, alors $g(k) = 2^n + g(2^n - 1 - r)$.
   b) En déduire par récurrence que si la représentation binaire de k sur n bits est $(b_n...b_0)_2$ et si on pose $b_{n+1} = 0$, alors la représentation binaire de $g(k)$ est $(a_n...a_0)_2$ avec pour tout $i \in [\![0, n]\!]$, $a_i = b_i \oplus b_{i+1}$.
   c) En déduire une expression simple de $g(k)$ en fonction de k pour tout $k \in \mathbb{N}$.
   d) Ecrire une fonction binaire_vers_Gray permettant de calculer le code de Gray de l'entier dont la représentation binaire est donnée en entrée.

8. a) Montrer que g est une bijection et expliquer comment calculer $g^{-1}(k)$ pour tout $k \in \mathbb{N}$.
   b) Ecrire une fonction Gray_vers_binaire permettant de calculer la représentation binaire de l'entier dont le code de Gray est donné en entrée.

On cherche à réaliser les fonctions g et g^{-1} en utilisant un minimum de portes logiques XOR qu'on représentera par un rectangle sur lequel arrivent deux flèches (ses entrées) et duquel repart une flèche (sa sortie). Par exemple, le circuit logique ci dessous permet de calculer la fonction f telle que $f(0) = 0$, $f(1) = 3$, $f(2) = 1$ et $f(3) = 2$ (typiquement, pour connaître la valeur de $f(1)$, on écrit 1 en binaire sur deux bits, 01, et on calcule les sorties b_0 et b_1 lorsque $a_0 = 1$ et $a_1 = 0$. On obtient $b_0 = 1$ et $b_1 = 1$ et comme 11 est la représentation binaire de 3, $f(1) = 3$).



9. a) Donner un circuit logique à 3 entrées représentant les trois bits d'un entier $k \in [\![0, 7]\!]$, et à trois sorties représentant les trois bits de $g(k)$.
   b) Donner un circuit logique à 3 entrées, représentant les trois bits d'un entier $k \in [\![0, 7]\!]$, et à trois sorties représentant les trois bits de $g^{-1}(k)$.
   c) Expliquer comment généraliser les questions précédentes dans le cas d'un entier k écrit sur n bits. Préciser le nombre de portes logiques XOR qui sera utilisé.

Bonus : Dans le but de minimiser les états transitoires, on souhaite représenter un entier par son code de Gray et ne jamais repasser par sa représentation binaire. Pour ce faire, il faut donc être capable de faire des opérations sur les entiers directement à partir de leurs codes de Gray. Expliquer comment sommer et multiplier des entiers à partir de leurs codes de Gray (on pourra commencer par identifier un algorithme permettant d'ajouter 1 et un algorithme permettant de multiplier par 2). Implémenter ces algorithmes : ils doivent prendre en entrée les codes de Gray de deux entiers p et q et renvoyer le code de Gray représentant la somme (respectivement le produit) de p et q.