Informatique

DM

CENTRALE- Option informatique - 2019

Ipesup

Les candidats devront utiliser le langage Caml pour traiter ce sujet.

Dans tout le problème, n désigne un entier naturel non nul.

XOR

Soit $\oplus$ l'opérateur XOR défini sur $\{0,1\}$ par la table suivante :

$\oplus$	0	1
0	0	1
1	1	0

On prolonge cette définition à $\mathbb{N}$ de la manière suivante.
Soit $(x,y) \in \mathbb{N}^2$ vérifiant $x < 2^p$ et $y < 2^p$ où p est un entier naturel non nul. On décompose x et y en base 2 :

$$x = \sum_{k=0}^{p} a_k 2^k \qquad \text{et} \qquad y = \sum_{k=0}^{p} b_k 2^k,$$

où les coefficients a_k et b_k sont des éléments de $\{0,1\}$. On définit alors $x \oplus y$ par :

$$x \oplus y = \sum_{k=0}^{p} (a_k \oplus b_k) 2^k.$$

On cherche à obtenir tous les n-uplets composés de 0 et de 1 de deux manières différentes.

Ces n-uplets seront implémentés à l'aide de listes.

Exercise 1

Dans cette sous-partie, on obtient ces n-uplets dans l'ordre lexicographique.

1. Écrire une fonction suivant transformant un n-uplet en son suivant dans l'ordre lexicographique.
   Par exemple, si $t = (0;1;0;0;1;1;1)$, après exécution de `suivant(t)`, on a $t = (0;1;0;1;0;0;0)$.
   Cette fonction modifie la liste qu'elle reçoit en argument. De plus, elle renvoie un booléen, valant vrai si elle a pu déterminer un n-uplet suivant, ou bien faux si le n-uplet fourni en argument était le dernier. Dans ce dernier cas, la valeur de ce n-uplet après exécution de la fonction n'est pas spécifiée.
2. Écrire une fonction affichant tous les n-uplets dans l'ordre lexicographique.
   On pourra commencer par écrire une fonction affichant les éléments d'un n-uplet : `affiche_nuplet : int list -> unit`

Exercise 2

Pour certaines applications, par exemple pour éviter des états transitoires intermédiaires dans les circuits logiques ou pour faciliter la correction d'erreur dans les transmissions numériques, on souhaite que le passage d'un n-uplet au suivant ne modifie qu'un seul bit.

Dans un brevet de 1953, Frank Gray définit un ordre des n-uplets possédant cette propriété.

Pour produire la liste des n-uplets dans l'ordre de Gray, un algorithme consiste à partir de la liste des 1-uplets $(0, 1)$ et à construire la liste des $(n+1)$-uplets à partir de celles des n-uplets en ajoutant un 0 en tête de chaque n-uplet puis un 1 en tête de chaque n-uplet en parcourant leur liste à l'envers.

On obtient ainsi
pour $n = 2$ la liste $(00, 01, 11, 10)$,
pour $n = 3$ la liste $(000, 001, 011, 010, 110, 111, 101, 100)$.

Les n-uplets sont représentés par des listes, une liste de n-uplets sera donc une liste de listes.

1. Écrire une fonction `ajout` telle que si a est un entier et l une liste de n-uplets d'entiers, `ajout a l` renvoie une liste contenant les éléments de l auxquels on a ajouté a en tête.
2. Écrire deux fonctions mutuellement récursives `monte` et `descend` prenant en argument un entier n et renvoyant les n-uplets dans l'ordre de Gray. L'une les renverra de $00\ldots0$ à $10\ldots0$, l'autre en sens inverse.
3. Évaluer la complexité de ces fonctions en termes d'appels à `monte` et `descend`.
4. Décrire une façon simple d'améliorer cette complexité.

Exercise 3

Dans tout ce qui suit, l'expression *représentation binaire* désigne la représentation traditionnelle en base 2.

Étant donné $k \in \mathbb{N}^*$, on a de manière unique :

$$k = \sum_{i=0}^{p} a_i 2^i \qquad \text{avec} \qquad p \in \mathbb{N}, \quad a_i \in \{0, 1\}, \quad a_p \neq 0.$$

La représentation binaire canonique de k est $a_p \ldots a_0$ (le bit de poids fort, qui est non nul, en premier).

On dira que tout n-uplet constitué d'un nombre quelconque de 0 suivis de la représentation binaire canonique de k est une représentation binaire de k.

On définit une fonction g sur $\mathbb{N}$ de la manière suivante.

Pour $k \in \mathbb{N}$ et n tel que $k < 2^n$, on considère la liste dans l'ordre de Gray des 2^n n-uplets ; on indice cette liste de 0 à $2^n - 1$. Alors $g(k)$ est le nombre dont une représentation binaire est le n-uplet d'indice k.

Par exemple, si on énumère les 3-uplets selon l'ordre de Gray, on a :

$$(000,\ 001,\ 011,\ 010,\ 110,\ 111,\ 101,\ 100).$$

Dans cette liste, l'élément d'indice 0 est 000, donc $g(0) = 0$, et le 3-uplet d'indice 7 est 100, donc $g(7) = 4$.

Soit n un entier naturel et k un entier compris entre 2^n et $2^{n+1} - 1$:

$$k = 2^n + r \qquad \text{avec} \qquad 0 \leq r < 2^n.$$

1. Démontrer que : $g(k) = 2^n + g(2^n - 1 - r)$.
2. En déduire que, si la représentation binaire de k est $b_n \dots b_0$ et si l'on pose $b_{n+1} = 0$, la représentation binaire de $g(k)$ est $a_n \dots a_0$ où, pour tout j entre 0 et n : $a_j = b_j \oplus b_{j+1}$.
3. Exprimer, pour $k \in \mathbb{N}$, $g(k)$ en fonction de k (à l'aide de $\oplus$).
4. Réaliser les fonctions g et g^{-1} avec des circuits logiques.
5. Donner un circuit logique à 3 entrées représentant les trois bits d'un entier $k \leq 7$ et à trois sorties représentant les trois bits de $g(k)$.
6. Donner un circuit pour l'opération inverse.
7. Si $n \geq 2$, donner un circuit permettant de passer d'un nombre k à n bits à $g(k)$. Faire de même pour l'opération inverse en utilisant le moins possible de portes et préciser le nombre de portes utilisées.

Exercice 4

Dans toute la suite, on note $\mathbb{B} = \{0, 1\}$. Soit $n \in \mathbb{N}^*$ et $\mathbb{B}^n$ l'ensemble des n-uplets binaires.

Pour tout $x = (x_1, \dots, x_n) \in \mathbb{B}^n$, on définit le *poids de Hamming* par

$$\mathrm{wt}(x) = \sum_{i=1}^{n} x_i.$$

Pour $x, y \in \mathbb{B}^n$, on définit la *distance de Hamming* par

$$d(x, y) = \mathrm{wt}(x \oplus y),$$

où $\oplus$ désigne l'opération XOR bit à bit.

1. Montrer que $\mathrm{wt}(x \oplus y) = \sum_{i=1}^{n} (x_i \oplus y_i)$.
2. Calculer le poids de Hamming des entiers 13, 27 et 42.

Exercice 5

On se place dans le cas $n = 4$.

Soit $x = (x_1, x_2, x_3, x_4) \in \mathbb{B}^4$. On définit la fonction booléenne $M : \mathbb{B}^4 \longrightarrow \mathbb{B}$ par

$$M(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{si } \mathrm{wt}(x_1, x_2, x_3, x_4) \geq 3, \\ 0 & \text{sinon.} \end{cases}$$

1. Construire la table de vérité complète de la fonction M.
2. Donner l'expression canonique disjonctive de M.
3. Montrer que $M(x_1, x_2, x_3, x_4) = x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4$.
4. Déterminer le nombre de portes AND, OR et NOT nécessaires pour implémenter directement cette expression et dessiner le circuit logique.

Exercice 6

On souhaite réduire le coût matériel du circuit réalisant la fonction majorité.

1. Construire la carte de Karnaugh à quatre variables associée à la fonction M.
2. En déduire une expression logique simplifiée de la fonction M et redessiner le nouveau circuit.
3. Comparer le nombre de portes logiques nécessaires avant et après simplification.