# Final MT42

## Exercice 1 : Logique

**[On change de copie, svp]**

A) Parmi ces formules, lesquelles sont des tautologies ? Expliquez pourquoi.

a) $(p \to (q \to r)) \to ((p \wedge q) \to r)$
b) $(p \to q) \to (((p \to r) \to q) \to q)$
c) $(p \wedge (q \to r)) \to (q \wedge p)$

B) Un étudiant en MT42 doit passer son examen d'UV. Il fait le raisonnement suivant :

(E1) Si je suis absent à mon examen, je ne réussirai pas à mon examen.
(E2) Si le bus est à l'heure, alors je ne serai pas absent à mon examen.
(E3) Si mon bus n'est pas à l'heure alors, si je me lève tôt alors je pourrai être présent à mon examen (en m'y rendant à pied).
(E4) S'il y a la grève des bus, alors le bus ne sera pas à l'heure.
(E5) Il y a la grève des bus et je me lève tôt.

1. Traduire ces énoncés dans la logique des propositions, en précisant le vocabulaire utilisé. Ne considérez pas ce qui est écrit entre parenthèses lors de la traduction en formules logiques.
2. Transformer les formules en clauses de Horn.
3. En appliquant la méthode de résolution, peut-on démontrer que l'étudiant ne sera pas absent à son examen ?
4. En appliquant la méthode de résolution, peut-on démontrer que l'étudiant réussit son examen ?

## Exercice 2 : Étude de complexité

**[On change de copie, svp]**

On considère la suite numérique suivante appelée suite de Fibonacci:

$$\begin{cases} u_0 &= 0 \\ u_1 &= 1 \\ u_n &= u_{n-1} + u_{n-2} \quad \forall n \geq 2 \end{cases} \tag{1}$$

1. Écrire un algorithme itératif (non-récursif!), appelé Fibonacci1, qui étant donné $n \in \mathbb{N}$ calcule $u_n$ le $n$-ème terme de la suite de Fibonacci.
2. Déterminer la complexité de votre algorithme.
3. On considère une version récursive de l'algorithme:

**Algorithme 1.** Fibonacci2($n$)
**if** $n < 2$ **then**
return($n$)
**else** Fibonacci2($n-1$)+Fibonacci2($n-2$)
**end if**

(a) On note $s(n)$ la fonction qui compte le nombre d'additions dans l'appel de Fibonacci2 pour le paramètre $n$. Donner une expression de $s(n)$ en fonction de $s(n-1)$ et $s(n-2)$.
(b) En déduire par récurrence que $s(n) \geq u_n$ pour $n \geq 2$
(c) On admettra le résultat suivant, dit *formule de Binet*, qui donne

$$u_n = \frac{1}{\sqrt{5}}(\phi^n - \phi'^n), \text{ avec } \phi = \frac{1+\sqrt{5}}{2} \text{ et } \phi' = -\frac{1}{\phi} \tag{2}$$

i. Montrer que $(u_n) \in \Theta(\phi^n)$.

ii. Que pouvez-vous en déduire sur la complexité de l'algorithme Fibonacci2 ?

4. Pour tout $n$ on note $F_n = \begin{pmatrix} u_n \\ u_{n-1} \end{pmatrix}$ et $F_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

   (a) Déterminer une matrice $A$ de taille $2 \times 2$ telle que $F_n = AF_{n-1}$.

   (b) En déduire que $F_n = A^n F_0$.

   (c) En vous inspirant de l'algorithme d'exponentiation rapide (algorithme 1, TD1), écrire un algorithme, Fibonacci3 qui, étant donné $n$, calcule le vecteur $F_n$.

   (d) Étudier la complexité de Fibonacci3.

5. Le tableau suivant donne les temps (en secondes) pour les trois algorithmes décrit dans ce problème pour calculer $u_n$. On a changé les noms des algorithmes et l'ordre. Le symbole $\infty$ signifie que le temps d'exécution est trop important pour que le calcul soit réalisable sur machine.

| $n$ | 100 | 1000 | 10000 | 20000 | 50000 | 80000 | $10^6$ | $10^7$ |
|---|---|---|---|---|---|---|---|---|
| algo_f | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| algo_g | 0 | 0 | 0.187 | 0.656 | 3.750 | 9.563 | 14.531 | $\infty$ |
| algo_h | 0.016 | 0.015 | 0.031 | 0.063 | 0.359 | 0.75 | 1.031 | 103 |

(3)

   Identifier les algorithmes algo_f, algo_g et algo_h avec Fibonacci1, Fibonacci2 et Fibonacci3.

6. On admet que la suite de Fibonacci $(u_n)$ vérifie les relations

$$\begin{cases} u_{2k} &= u_k(2u_{k+1} - u_k) \\ u_{2k+1} &= u_{k+1}^2 + u_k^2 \end{cases} \tag{4}$$

   Justifier, sans détails inutiles, que ces relations permettent d'écrire un algorithme qui calcule $u_n$ avec une complexité en $\mathcal{O}(\ln(n))$.

## Exercice 3 : Fonctions booléennes duales et auto-duales

A) Étude d'une fonction booléenne sur $\mathbb{B}^3$.

On note $\mathbb{B} = \{0, 1\}$ et on considère $f_1 : \mathbb{B}^3 \to \mathbb{B}$ définie par $f_1(x, y, z) = x.\overline{y} + x.y.\overline{z} + \overline{x}.z + y.\overline{z} + \overline{x}.\overline{y}.\overline{z}$

1. Représenter $\mathbb{B}^3$ et repérer les faces couvertes par $f_1$.
2. En déduire les monômes inférieurs à $f_1$ qu'on listera par degré.
3. Déterminer géométriquement les monômes premiers de $f_1$.
4. Proposer une forme réduite de $f_1$ à partir des monômes premiers et vérifier que cette forme réduite est bien équivalente à $f$.

B) Fonctions booléennes duales et autoduales.

Soit $f : \mathbb{B}^n \to \mathbb{B}$, une fonction booléenne $f : (x_1, \ldots, x_n) \mapsto f(x_1, \ldots, x_n)$. On définit la fonction duale de $f$, notée $f^*$, de la manière suivante :

$$f^*(x_1, x_2, \ldots, x_n) = \overline{f(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)} \tag{5}$$

1. Calculer $f_1^*$, la fonction duale de la fonction $f_1$.
2. Démontrer qu'on obtient $f^*$ en échangeant les opérations . et + dans l'expression de $f$.
3. On dit qu'une fonction booléenne est autoduale si et seulement si $f^* = f$.
   (a) Montrer que $f_2$ définie par $f_2(x, y, z) = \overline{x}.\overline{y}.\overline{z} + x.y.\overline{z} + x.\overline{y}.z + \overline{x}.y.z$ est auto-duale.
   (b) Montrer que si $f : \mathbb{B}^n \to \mathbb{B}$ est auto-duale il suffit de connaître la moitié des valeurs de $f$ pour la connaître totalement.
   (c) Déduire de ce qui précède qu'une fonction autoduale sur le cube $\mathbb{B}^3$ est entièrement déterminée par sa connaissance sur une face. Donner alors le nombre de fonction auto-duales sur le cube.
   (d) Déterminer le nombre de fonctions booléennes auto-duales définies sur $\mathbb{B}^n$.

**Correction**

# Exercice 2: Étude de complexité

1.

**Algorithme 2.** Fibonacci1($n$)
$a := 0$
$b := 1$
**for** $i = 2$ to $n$ **do**
  $c := b$
  $b := b + a$
  $a := c$
**end for**
$b$

2. La fonction complexité de l'algorithme est $f(n) = 3 + 4(n-1)$ (on compte le nombre d'opérations élémentaires. Donc cet algorithme a une complexité en $\Theta(n)$ (linéaire).

3. (a) $s(n) = s(n-1) + s(n-2) + 1$ (lorsqu'on appelle Fibonacci2($n$) on fait une addition plus le nombre d'addition pour réaliser Fibonacci2($n-1$) plus le nombre d'additions pour réaliser Fibonacci2($n-2$).

   (b) On a $s(0) = 0$ et $s(1) = 0$ donc $s(2) = 0+0+1 \geq u_2 = 1$. Ensuite on suppose par récurrence (forte) que $s(k) \geq u_k$ pour tout $k \leq n$ donc $s(n+1) = s(n)+s(n-1)+1 \geq u_n+u_{n-1}+1 \geq u_{n+1}+1 \geq u_{n+1}$. La propriété est vraie au rang $n+1$ donc par récurrence toujours vraie.

   (c) $\lim_{n\to\infty} \phi'^n = 0$ car $|\phi'| < 1$ donc $\lim_{n\to+\infty} \dfrac{u_n}{\phi^n} = 1 \Rightarrow (u_n) \in \Theta(\phi^n)$.

   (d) D'après b) $s(n) \geq u_n$ et $u_n \in \Theta(\phi^n)$ on en déduit que $s(n) \geq c\Phi^n$ (ou $s(n) \in \Omega(\phi^n)$)). La comlpexité de l'algorithme récursif Fibonacci2 est donc au moins exponentiel.

4. $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

   $F_n = AF_{n-1} = AAF_{n-2} = \cdots = A^nF_0$.

**(b)**

**Algorithme 3.** Calcul de $F_n$
Fibonacci3($n$)
$A := \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$;
$N := n$;
$R := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$F_0 := \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
**while** $N > 0$ **do**
  **if** N pair **then**
    $A := A \times A$
    $N := N/2$
  **else**
    $R := R \times A$
    $N := N - 1$
  **end if**
**end while**
return $R \times F_0$

Dans cet algorithme $\times$ représente le produit matriciel.

(c) Le coût opératoire du produit de deux matrices est constant (8 multiplications[1]). Le nombre de passage en boucle **While** est borné par $2\dfrac{\ln(n)}{\ln(2))}$ et chaque condition **If** entraîne 11 opérations élémentaires. La fonction complexité vérifie donc $f(n) \leq 22\dfrac{\ln(n)}{\ln(2)}$, soit une complexité en $\mathcal{O}(\ln(n))$

[1] Il existe des algorithmes minimisant le nombre de multiplications pour calculer le produit de deux matrices mais c'est une autre histoire...

5. On a asymptotiquement algo_h dominé par algo_g domniné par algo_f. Donc d'après ce qui précède on a

algo_g=Fibonacci1
algo_f=Fibonacci2
algo_h=Fibonacci3

6. Les relations de récurrence permettent d'exprimer $u_{2k}$ (ou $u_{2k+1}$) en fonction de $u_{k+1}$ et $u_k$. Admettons que l'écriture en base 2 de $n$ nécessite $k$-bits (i.e. $k = \log_2(n)$, alors le calcul de $u_n$ se décompose en $k$ étapes. On a donc une complexité en $\mathcal{O}(\ln(n))$.

# Exercice 3

A) 1.

2. – degré 3: $\overline{x}.\overline{y}.\overline{z}, \overline{x}.y.\overline{z}, \overline{x}.\overline{y}.z, \overline{x}.y.z, x.\overline{y}.\overline{z}, x.y.\overline{z}, x.\overline{y}.z$.
   – degré 2: $\overline{x}.\overline{y}, \overline{x}.\overline{z}, \overline{x}.z, \overline{x}.y, \overline{y}.\overline{z}, y.\overline{z}, x.\overline{z}, x.\overline{y}$.
   – degré 1: $\overline{x}$, $\overline{y}$, $\overline{z}$.
3. Les monômes premiers correspondent aux faces maximales: $\overline{x}$, $\overline{y}$, $\overline{z}$.
4. On a $f(x,y,z) = \overline{x} + \overline{y} + \overline{z}$. On vérifie que $f = f_1$.

B) 1. À partir de la forme simplifiée de $f_1$ on a $f_1^*(x,y,z) = \overline{f_1(\overline{x},\overline{y},\overline{z})} = \overline{\overline{x} + \overline{yz}} = x.y.z$ (lois de Morgan).

2. Pour les fonctions d'une variable c'est clair puisque une telle fonctions est nécessairement réduite à un des 4 cas suivants
   – $g_1(x) = 0$
   – $g_2(x) = x$
   – $g_3(x) = \overline{x}$
   – $g_4(x) = 1$

   Et pour chacune de ces fonctions of a $g_i^*(x) = \overline{g_i(\overline{x}} = g_i(x)$ (la duale s'obtient bien en échangeant + et .. Supposons que le résultat soit vrai pour les fonctions booléennes définies sur $\mathbb{B}^n$. Une fonction booléenne sur $\mathbb{B}^{n+1}$ peut toujours sécrire

$$f(x_1,\ldots,x_n,x_{n+1}) = x_{n+1}g(x_1,\ldots,x_n) + \overline{x_{n+1}}.h(x_1,\ldots,x_n) \tag{6}$$

   Le calcul de $f^*$ donne

$$f^*(x_1,\ldots,x_{n+1}) = \overline{\overline{x_{n+1}}g(\overline{x_1},\ldots,\overline{x_n}) + x_{n+1}.h(\overline{x_1},\ldots,\overline{x_n})} \tag{7}$$

   d'où avec les lois de Morgan

$$f^*(x_1,\ldots,x_{n+1}) = (x_{n+1} + \overline{g(\overline{x_1},\ldots,\overline{x_n})}).(\overline{x_{n+1}} + \overline{h(\overline{x_1},\ldots,\overline{x_n})}) \tag{8}$$

   i.e.

$$f^*(x_1,\ldots,x_n) = (x_1 + g^*(x_1,\ldots,x_n)).(\overline{x_1} + h^*(x_1,\ldots,x_n)) \tag{9}$$

   Par hypothèse de récurrence l'écriture de $h^*$ et $g^*$ s'obtient en échangeant le rôle de + et . ce qui implique le résultat pour toute fonction $f : \mathbb{B}^{n+1} \to \mathbb{B}$.

3. (a) $f_2^*(x) = (\overline{x} + \overline{y} + \overline{z}).(x + y + \overline{z}).(x + \overline{y} + z).(\overline{x} + y + z) = f_2$
   (b) Une fonction autoduale vérifie $f = f^*$ donc $f(x_1,\ldots,x_n) = \overline{f(\overline{x_1},\ldots,\overline{x_n})}$. En particulier $f(\overline{x_1},\ldots,\overline{x_n}) = \overline{f(x_1,\ldots,x_n)}$. Donc on connaît les valeurs de $f$ sur le complémentaire de $(x_1,\ldots,x_n)$ en prendant le complément de la valeur. On peut alors séprarer les $n$-uplets de $\mathbb{B}^n$ en deux ensembles de valeurs par exemple $E_1 = \{(0,x_2,\ldots,x_n), x_i \in \{0,1\}\}$ et $E_2 = \{(1,y_2,\ldots y_n), y_i \in \{0,1\}\}$. Il suffit alors de connaître $f$ sur $E_1$ pour la connaître aussi sur $E_2$.
   (c) $E_1 = \{(0,x_1,x_2), x_i \in \{0,1\}\}$ est une face du cube. Pour caractériser les valeurs de $f$ sur $E_1$ il faut affecter une valeur, 0 ou 1, par sommet soir $2 \times 2 \times 2 \times 2 = 8$ possibilité.
   (d) Le même raisonnement conduit à Card(fonctions boolénnes autoduales de $\mathbb{B}^n$) $= 2^n$.