1 Ordre topologique et applications

1.1.1) Pour le premier graphe le seul ordre possible est défini par $o(1) = 4$, $o(2) = 2$, $o(3) = 1$ et $o(4) = 3$. Pour le deuxième graphe, un ordre possible est celui défini par l'identité.

1.1.2) L'existence d'un cycle dans G est clairement incompatible avec celle d'un ordre topologique. Réciproquement, on montre par récurrence sur le nombre n de sommets que tout graphe à n sommets sans cycle peut être topologiquement ordonné. Pour $n = 1$ c'est évident. Pour $n > 1$, on choisit un sommet sans père (il en existe sans quoi on pourait trouver un cycle dans G), on lui affecte le numéro 1, puis on ordonne topologiquement avec les numéros $2, \ldots, n$ le sous-graphe constitué des $n - 1$ sommets restants et des arêtes joignant ces sommets.

1.1.3)
$o_1 \longleftarrow$ *tableau de* n *entiers*
Pour $i = 1 \ldots n$ *faire* $o_1(o(i)) \longleftarrow i$ *fin pour*
Retourner o_1
La complexité est $O(n)$.

1.2.1)
$u \longleftarrow$ *tableau de* $|S|$ *listes vides*
Pour $i = 1 \ldots |S|$ *faire*
    $\ell \longleftarrow t(i)$
    Tant que $\ell \neq$ *nil faire*
        $j \longleftarrow$ *tête*(ℓ); $u(j) \longleftarrow$ *concat*$(i, u(j))$
        $\ell \longleftarrow$ *queue*(ℓ)
    Fin tant que
Fin pour
Retourner u
La complexité est $O(|S| + |A|)$.

1.2.2) Pour déterminer les degrés entrants il suffit de compter le nombre de parents au lieu de les accumuler dans une liste comme le fait la fonction précédente. Pour déterminer les entrées d'un graphe, on calcule les degrés entrants et on forme la liste des nœuds de degré entrant nul. Ces deux algorithmes ont une complexité $O(|S| + |A|)$.

1.2.3)
$o \longleftarrow$ *tableau de* $|S|$ *zéros*
$u \longleftarrow$ *parents*(t)
$k \longleftarrow 1$

fonction numérote$(i) :=$
    si $o(i) < 0$ *alors erreur(cycle)*
    si $o(i) = 0$ *alors*
        $o(i) \longleftarrow -1$
        $\ell \longleftarrow u(i)$
        Tant que $\ell \neq$ *nil faire*
            $j \longleftarrow$ *tête*(ℓ); *numérote*(j)
            $\ell \longleftarrow$ *queue*(ℓ)
        Fin tant que
        $o(i) \longleftarrow k$; $k \longleftarrow k + 1$
    Fin si
Fin fonction

Pour $i = 1 \ldots |S|$ *faire numérote*(i) *fin pour*
Retourner o

La fonction récursive *numérote* numérote un nœud si cela n'a pas déjà été fait (test « $o(i) = 0$ ») en numérotant au préalable tous les parents du nœud considéré de façon à garantir la condition $(j, k) \in A \Longrightarrow o(j) < o(k)$. Le but de l'affectation provisoire « $o(i) \longleftarrow -1$ » est d'éviter de boucler si le graphe comporte un cycle passant par le nœud i. Ce fait est détecté lorsqu'on repasse en i lors de l'exploration récursive de ses ancêtres et déclenche une exception *erreur(cycle)* supposée interrompre l'algorithme.

La correction de cet algorithme résulte du fait qu'un nœud n'est jamais numéroté avant que tous ses parents ne l'aient été, et que tous les nœuds sont effectivement numérotés grâce à la boucle principale. La terminaison résulte du fait que pour un nœud i donné, on rentre une et une seule fois dans la branche « *si* $o(i) = 0$ ». En ce qui concerne la complexité, pour $i \in \{1, \ldots, |S|\}$ donné, il est effectué exactement $1+ex(i)$ appels *numérote*(i) et un seul de ces appels a une complexité non constante, proportionnelle à $in(i)$, en ne comptant pas le temps passé dans les appels récursifs à *numérote*. Ainsi, chaque nœud contribue pour $1+ex(i)+in(i)$ à la complexité de l'ensemble des appels à *numérote*. En sommant sur tous les nœuds, et en ajoutant la complexité du calcul des parents, on obtient une complexité globale $O(|S| + |A|)$ pour le tri topologique.

1.3)
$o_1 \longleftarrow$ *inverse*(o)
$u \longleftarrow$ *parents*(t)
$t' \longleftarrow$ *tableau de* $|S|$ *listes vides*
Pour $k = |S| \ldots 1$ *faire*
  $j \longleftarrow o_1(k)$; $\ell \longleftarrow u(j)$
  Tant que $\ell \neq$ *nil faire*
    $i \longleftarrow$ *tête*(ℓ); $t'(i) \longleftarrow$ *concat*$(j, t'(i))$
    $\ell \longleftarrow$ *queue*(ℓ)
  Fin tant que
Fin pour
Retourner t'

1.4.1)

i	R(1)	R(2)	R(3)	R(4)	R(5)	Q(1)	Q(2)	Q(3)	Q(4)	Q(5)
5	nil	nil	nil	nil	nil	faux	faux	faux	faux	faux
4	nil	nil	nil	nil	[5]	faux	faux	faux	faux	faux
3	nil	nil	nil	[54]	[5]	faux	faux	faux	faux	faux
2	nil	nil	[453]	[54]	[5]	faux	faux	faux	faux	faux
1	nil	[3542]	[453]	[54]	[5]	faux	faux	faux	faux	faux
	[24531]	[3542]	[453]	[54]	[5]	faux	faux	faux	faux	faux

1.4.2) L'invariant de boucle suivant est clairement satisfait :
à l'entrée de la boucle en ligne 2, pour tout $j \in \{i+1, \ldots, n\}$, *la liste* $R(j)$ *est une liste d'adjacence sans répétition du nœud* j *pour le graphe* G^*.

1.4.3) Si v est un sommet interne, soit u le père de v de plus grand numéro. Alors $(u, v) \in A^-$ et l'application $v \longmapsto (u, v)$ est une injection de l'ensemble des sommets internes de G dans A^-. Ceci prouve que le nombre de sommets internes est majoré par $|A^-|$.

Si u est un sommet quelconque de G, les fils de u dans G^* autres que u sont tous des sommets internes de G. Ceci prouve que le nombre d'arêtes dans G^* est majoré par $|S|(1 + |A^-|)$.

Soit $i \in \{1, \ldots, n\}$ et $j \in t(i)$. Si $Q(j) =$ vrai en ligne 9 alors il existe $k \in t(i)$ tel que $k < j$ et $j \in R(k)$. Il existe donc un chemin dans G menant de i à j en passant par k, et par conséquent $(i, j) \notin A^-$ et la réciproque est immédiate. Ainsi, $Q(j) =$ faux $\Longleftrightarrow (i, j) \in A^-$.

La complexité des instructions effectuées en lignes 10–18 pour un couple (i, j) donné est $O(|R(j)|) = O(|R(i)|)$. Celle des instructions effectuées en lignes 3 – 26 pour un nœud i donné est donc $O(|A^-||R(i)|)$ et la complexité globale de l'algorithme proposé est $O(|A^-||A^*|)$.

1.4.4) Non. Pour $n \in \mathbf{N}^*$ on considère le graphe G de sommets $1, \ldots, 2n$ comportant une arête (i, j) pour chaque couple (i, j) tel que i est impair et j est pair. Chacune de ces arêtes appartient à A^- (car elle définit l'unique chemin reliant i à j), donc $|A^-| = n^2 = \frac{1}{4}|S|^2$.

1.5.1)
1 $r \longleftarrow$ *tableau de* $|S|$ *listes vides*
2 $c \longleftarrow 0$
3 *Pour* $i = |S| \ldots 1$ *faire*
4 $\ell \longleftarrow t(i)$; $j \longleftarrow i$
5 *Tant que* $\ell \neq$ *nil et* $j = i$ *faire*
6 $k \longleftarrow$ *tête*(ℓ); $\ell \longleftarrow$ *queue*(ℓ)
7 *Si* $r(k) \neq$ *nil alors* $j \longleftarrow k$ *fin si*
8 *Fin tant que*

```
 9        Si j = i alors c ⟵ c + 1; r(i) ⟵ [i]
10        sinon r(i) ⟵ concat(i, r(j)); r(j) ⟵ nil
11        Fin si
12    Fin pour
13    s ⟵ tableau de c listes
14    Pour i = |S|...1 faire si r(i) ≠ nil alors s(c) ⟵ r(i); c ⟵ c − 1 fin si fin pour
15    Retourner s
```

L'algoritme ci-dessus est construit de manière à satisfaire l'invariant de boucle suivant, dont la vérification est immédiate :

au début de la boucle 3–12, le tableau r *contient* c *listes non vides qui constituent une partition de* $\{i+1, \ldots, n\}$ *en chaînes non concaténables. Une telle chaîne est rangée, sous forme d'une liste sans répétition classée par ordre topologique croissant, dans la case* r(j) *où* j *est le premier nœud de la chaîne considérée. Les autres cases de* r *contiennent nil.*

La complexité est $O(|S| + |A|)$.

1.5.2.a) Le cas $i \in S_h$ est évident.

Dans le cas $i \notin S_h$, on remarque dans un premier temps que pour tout chemin dans G de longueur au moins égale à 2 commençant par le sommet i, il existe un chemin dans G ayant mêmes extrémités, et dont la deuxième étape est un sommet j tel que $(i,j) \in A^-$ (considérer un chemin de longueur maximale parmi ceux ayant ces extrémités).

Supposons $i \notin S_h$ tel que le degré sortant de i n'est pas nul. L'ensemble des sommets j tels que $(i,j) \in A^-$ est donc non vide, et il existe j parmi ces sommets tel que $R(h,j)$ est minimal. Si $R(h,j) = |S| + 1$, alors aucun chemin issu de i n'atteint un sommet de S_h dans G, donc $R(h,i) = |S| + 1 = R(h,j)$ dans ce cas. Si $R(h,j) = k \in S_h$ alors il existe un chemin dans G allant de i à k en passant par j, et il n'existe aucun chemin partant de i et aboutissant à un nœud de S_h de numéro strictement inférieur à k. On a donc $R(h,i) = k = R(h,j)$ dans ce cas aussi.

1.5.2.b) Il existe un chemin dans G allant de i à j et passant en deuxième étape par un sommet ℓ tel que $(i,\ell) \in A^-$. On a donc $\ell \leqslant j$ et $R(C[j],\ell) \leqslant j$. Si $\ell < j$ alors $(i,j) \notin A^-$, B contient ℓ donc est non vide et $r \leqslant R(C[j],\ell) = j$. Sinon, $\ell = j$ donc $(i,j) \in A^-$ et aucun chemin dans G issu de i ne peut rejoindre un nœud de $S_{C[j]}$ dont le numéro est inférieur ou égal à j, d'où $r > j$.

1.5.2c) L'algorithme suivant, inspiré de celui donné en **1.4**, calcule les listes d'adjacence pour le graphe $G^- = (S, A^-)$ avec une complexité $O(|S| + |A|)$. Ces listes d'adjacences sont de plus classées par ordre croissant :

```
t1 ⟵ tableau de |S| listes
q ⟵ tableau de |S| booléens initialisés à faux
Pour i = 1...n faire
    t1(i) ⟵ nil; ℓ ⟵ t(i)
    Tant que ℓ ≠ nil faire
        j ⟵ tête(ℓ); ℓ ⟵ queue(ℓ)
        Si q(j) = faux alors t1(i) ⟵ concat(j, t1(i)); q(j) ⟵ vrai fin si
    Fin tant que
    ℓ ⟵ t1(i); t1(i) ⟵ nil
    Tant que ℓ ≠ nil faire
        j ⟵ tête(ℓ); ℓ ⟵ queue(ℓ)
        q(j) ⟵ faux; t1(i) ⟵ concat(j, t1(i))
    Fin tant que
Fin pour
Retourner t1
```

L'algorithme suivant calcule alors la table des valeurs de R à l'aide de t_1 et C en utilisant la formule vue en **1.5.2.a**. Sa complexité est $O(k(|S| + |A^-|))$.

```
r ⟵ matrice de taille k × |S| initialisée à |S| + 1
Pour h = 1...k faire
    Pour i = |S|...1 faire
```

Si $C(i) = h$ *alors* $r(h,i) \longleftarrow i$
sinon
$\ell \longleftarrow t_1(i)$
Tant que $\ell \neq$ *nil faire*
$j \longleftarrow$ *tête*(ℓ); $\ell \longleftarrow$ *queue*(ℓ)
Si $r(h,j) < r(h,i)$ *alors* $r(h,i) \longleftarrow r(h,j)$ *fin si*
Fin tant que
Fin si
Fin pour
Fin pour
Retourner r

La complexité du calcul de r, en comptant celle du calcul de t_1 est

$$O(|S| + |A|) + O(k(|S| + |A^-|)) = O(|A| + k(|S| + |A^-|))$$

car $k \geqslant 1$.

1.5.3) On a $(i,j) \in A^*$ si et seulement si $R(C[j], i) \leqslant j$. La connaissance du tableau r calculé à la question précédente permet donc de construire les listes d'incidence pour G^* avec une complexité $O(|S|^2)$, et on peut même les avoir triées pour ce prix. La complexité globale de construction du graphe G^* par cette méthode est

$$O(|S| + |A|) + O(|A| + k(|S| + |A^-|)) + O(|S|^2) = O(k|S|^2).$$

2 Fonctions booléennes et circuits booléens

2.1.1) Fonctions de type ET : il y a huit choix pour a, b, c, donc au plus huit fonctions différentes de type ET. On vérifie que ces huit fonctions sont effectivement différentes comme suit : la table de vérité de $(x_1^a \wedge x_2^b)^c$ comporte trois fois la valeur $\neg c$ et une fois la valeur c, ce qui permet de déterminer c. On détermine ensuite a et b en remarquant que le seul cas où $(x_1^a \wedge x_2^b)^c = c$ est celui où $x_1 = a$ et $x_2 = b$, ce qui permet d'identifier a et b.

Fonctions de type XOR : il y en a deux, par le même type de raisonnement.

Fonctions des deux types : il n'y en n'a aucune car la table de vérité de $(x_1 \oplus x_2)^a$ comporte deux fois les valeurs 0 et 1.

2.1.2) $(x_1 \oplus x_2)^a = (\neg a) \oplus (x_1 \oplus x_2)$ donc une fonction de type XOR est une fonction affine. La réciproque est fausse, $a \oplus x_1$, $a \oplus x_2$ et a avec $a \in \{0,1\}$ sont aussi affines et ne sont pas de type XOR.

2.1.3) $((\neg a)^a \wedge x_2^b)^c = 0^c = c$. $(x_1 \oplus x_1)^a = \neg a$.

2.2.1) Pour $n = 2$, on constate avec **2.1.1** et **2.1.2** que toute fonction booléenne de deux variables soit affine, soit de type ET. Pour $n \geqslant 2$, soit $f \in \Gamma_n$ et g, h les fonctions de Γ_{n-1} obtenues en spécialisant f avec $x_n = 0$ pour g et $x_n = 1$ pour h. Si l'une des deux fonctions g ou h n'est pas affine, l'hypothèse de récurrence s'applique. Sinon, il existe $a, b \in \{0,1\}$ et $S, T \subset \{2, \ldots, n\}$ tels que $g = a \oplus (\bigoplus_{i \in S} x_i)$ et $h = b \oplus (\bigoplus_{i \in T} x_i)$.

Si $a = b$ et $S = T$ alors $f = a \oplus (\bigoplus_{i \in S} x_i)$ est affine.
Si $a \neq b$ et $S = T$ alors $f = a \oplus x_1 \oplus (\bigoplus_{i \in S} x_i)$ est affine.

Si $S \not\subset T$, par exemple $2 \in S \setminus T$. Soit $k(x_1, x_2) = f(x_1, x_2, 0, \ldots, 0)$: la fonction k prend une fois la valeur a, une fois la valeur $\neg a$ et deux fois la valeur b. Elle prend donc trois fois la valeur b et une fois la valeur $\neg b$. D'après **2.1.1**, k est une fonction de type ET. Le cas $T \not\subset S$ se traite de même.

2.2.2) Soit f non monotone et $p, q \in \{0,1\}^n$ tels que $p \leqslant q$ et $f(p) > f(q)$. On a donc $f(p) = 1$ et $f(q) = 0$. L'ensemble des n-uplets $x \in \{0,1\}^n$ tels que $x \leqslant q$ et $f(x) = 1$ est fini non vide ; il admet un élément maximal pour l'ordre partiel $\leqslant$, que l'on note r. On a $r \neq q$, soit $i \in \{1, \ldots, n\}$ tel que $r_i \neq q_i$. Donc $r_i = 0$ et $q_i = 1$ car $r_i \leqslant q_i$. Soit

enfn f' la spécialisation de f obtenue en fixant $x_j = r_j$ pour tout $j \neq i$. Pour $x \in \{0,1\}^n$ on a $f'(x) = f(r) = 1$ si $x_i = 0$ et $f'(x) \neq 1$ si $x_i = 1$ compte tenu du caractère maximal de r. Ainsi, $f'(x) = \neg x_i$.

2.3.1) $(x_1 \wedge x_2) \vee (x_3 \wedge (x_1 \vee x_2))$.

2.3.2) $f_x(x') = (x_1 \iff x'_1) \wedge \ldots \wedge (x_n \iff x'_n)$, et $(x_i \iff x'_i) = (x_i \wedge x'_i) \vee (\neg x_i \wedge \neg x'_i)$.
La conjonction de n expressions booléennes peut être calculée par tout circuit en forme d'arbre binaire dont les nœuds internes sont des portes ET et les entrées sont les expresions booléennes dont on recherche la conjonction. La taille d'un tel circuit, comptée en nombre de nœuds internes est $n-1$, et sa profondeur peut être choisie égale à $\lceil \log_2(n) \rceil$. On en déduit que f_x peut être calculée par un circuit sur $\{0,1,\vee,\wedge,\neg\}$ de taille $(n-1)+5n = O(n)$ et de profondeur $3 + \lceil \log_2(n) \rceil = O(\log n)$.

2.3.3) Si $f \in \Gamma_n$, on a $f = \bigvee_{a \in \{0,1\}^n,\ f(a)=1} f_a$. Donc f peut être calculée par un circuit sur $\{0,1,\vee,\wedge,\neg\}$ de taille $O(n2^n)$ et de profondeur $O(n)$.

2.4.1) $\{0,1,\vee,\wedge,\neg\}$ est complet. Le sous-ensemble $\{\wedge,\neg\}$ l'est aussi et il est minimal car $\neg$ ne permet d'engendrer qu'elle-même et la fonction identité de Γ_1 tandis que $\wedge$ ne permet d'engendrer que des fonctions monotones.

2.4.2.a) Si l'une des conditions **(i)** à **(v)** n'est pas satisfaite alors il n'existe aucun circuit sur Ω calculant respectivement la fonction constante 1, la fonction constante 0, une fonction non monotone, une fonction non auto-adjointe, une fonction non affine. Par contre le caractère nécessaire de **(vi)** est douteux...

2.4.2.b) Calcul de $\neg$: soient f, g, h satisfaisant respectivement **(i)**, **(ii)**, **(iii)**. Si $f(1,\ldots,1) = 0$ alors $f(x_1,\ldots,x_1) = \neg x_1$. De même si $g(0,\ldots,0) = 1$. Sinon, $f(x_1,\ldots,x_1) = 1$ et $g(x_1,\ldots,x_1) = 0$. On peut alors utiliser ces deux fonctions pour construire une spécialisation de h calculant $\neg$ (cf. **2.2.2**).

Calcul des fonctions constantes : si k satisfaisant **(iv)** alors $k(x_1,\ldots,x_1)$ est constant. On peut obtenir l'autre constante en composant avec $\neg$.

Calcul de $\wedge$: soit ℓ satisfaisant **(v)**. Il existe une spécialisation de ℓ qui est du type ET (cf. **2.2.1**). Cette spécialisation peut être réalisée par un circuit sur Ω puisqu'on dispose des constantes. En composant au besoin avec des négations, on obtient un circuit calculant $\wedge$.

On en déduit que si Ω satisfait les propriétés **(i)** à **(v)** alors on peut calculer toutes les fonctions du système complet $\{\wedge,\neg\}$, et donc toutes les fonctions booléennes. Remarquons que la propriété **(vi)** n'a pas servi.

3 Bornes supérieures et bornes inférieures

3.1.1) On montre par récurrence sur m que $((I+X)^m)_{ij} = 1$ si et seulement s'il existe dans G un chemin de i à j de longueur inférieure ou égale à m.

3.1.2) Soit m la plus petite puissance de 2 supérieure ou égale à n : $m = 2^{\lceil \log_2(n) \rceil}$. On calcule $(I+X)^m$ par $\lceil \log_2(n) \rceil$ élévations au carré successives et on extrait le coefficient (i,j) de cette matrice ; c'est $\mathrm{Acc}_{ij}(X)$.

Un coefficient du produit de deux matrices booléennes $n \times n$ peut être calculé avec un circuit sur Ω de taille $O(n)$ et de profondeur $O(\log n)$, donc le produit de deux telles matrices peut être calculé par un circuit sur Ω de taille $O(n^3)$ et de profondeur $O(\log n)$. En concaténant $\lceil \log_2(n) \rceil$ tels circuits derrière un circuit trivial calculant $I+X$, on obtient toutes les fonctions Acc avec un circuit de taille $O(n^3 \log(n))$ et de profondeur $O(\log(n)^2)$.

3.2.1) Notons q_1 et q_2 les nombres de nœuds de degré entrant 1 et 2 et p_1 le nombre de nœuds de degré sortant 1. Le nombre de nœuds internes est q_1+q_2, le nombre total de nœuds est $n+q_1+q_2 = m+p_1+p$ et le nombre total d'arêtes est $q_1+2q_2 \geqslant p_1+2p$. On a donc :

$$2(m+q_1+q_2) \geqslant 2m+p_1+q_1+2p \geqslant 2m+p_1+2p = m+p+n+q_1+q_2$$

soit $q_1+q_2 \geqslant n-m+p$.

3.2.2.a) C'est évident.

3.2.2.b) Il suffit de prouver qu'on peut supprimer les nœuds $s_1, \ldots, s_k$ et modifier les autres nœuds de C pour obtenir un nouveau circuit de taille taille(C) $-$ k calculant f. Détaillons la suppression de s_1.

Si s_1 est une sortie autre que f alors on supprime s_1 et on conserve le reste du circuit.

Si $g(s_1)$ est une fonction constante alors s_1 n'est pas la sortie f car f est non constante. On supprime s_1 et on répercute la constante sur les nœuds fils de s_1 (c'est-à dire qu'on simplifie ces nœuds fils sachant qu'une de leurs entrées est une constante connue).

Si $g(s_1) = \mathrm{id}$ on supprime s_1 et on prolonge les arêtes issues de s_1 vers le nœud père de s_1.

Si $g(s_1) = \neg$ et si le père de s_1 n'est pas une entrée alors on supprime s_1, on remplace le nœud père de s_1 par le nœud calculant la fonction complémentaire et on prolonge les arêtes issues de s_1 comme dans le cas précédent. Remarquer que la modification du nœud père de s_1 ne remet pas en cause son appartenance éventuelle à l'ensemble $\{s_2, \ldots, s_k\}$.

Si $g(s_1) = \neg$ et le père de s_1 est une entrée alors s_1 n'est pas la sortie f par hypothèse sur f. Dans ce cas, on supprime s_1, on prolonge les arêtes issues de s_1 vers son père, et on modifie les nœuds fils de s_1 pour compenser l'abscence de négation sur leur entrée modifiée. Ceci ne remet pas en cause l'appartenance éventuelle de ces fils à $\{s_2, \ldots, s_k\}$.

Si $g(s_1)$ est une fonction de Γ_2 qui n'est ni de type ET ni de type XOR alors c'est l'une des six fonctions $(x_1, x_2) \longrightarrow 0$, $(x_1, x_2) \longrightarrow 1$, $(x_1, x_2) \longrightarrow x_1$, $(x_1, x_2) \longrightarrow \neg x_1$, $(x_1, x_2) \longrightarrow x_2$, $(x_1, x_2) \longrightarrow \neg x_2$. On applique alors celle des règles précédentes qui est appropriée.

3.3) *En français : on dispose d'un circuit optimal calculant* f *dont la restriction à un certain ensemble* Z *pour* α *sélectionne la variable* $x_{\bar{\alpha}}$. *On veut en déduire un circuit contenant deux nœuds de moins et coïncidant avec une fonction de sélection sur un ensemble* Z′ *pour* α *ayant un élément de moins que* Z. *On choisit* $i \in Z$ *et on fixe* $x_i =$ *constante. Prouver qu'on peut alors simplifier le circuit calculant* f′, *la spécialisation de* f, *et économiser au moins deux nœuds.*

Le fait de fixer la valeur de x_i permet de simplifier tous les nœuds fils de l'entrée x_i, et de les transformer en nœuds du type décrit en **3.2.2.b**. On peut alors supprimer ces nœuds car la fonction f′ n'est ni constante ($z \geqslant 2$), ni une négation. Le travail demandé est terminé lorsque $\mathrm{ex}(x_i) \geqslant 2$. Le cas $\mathrm{ex}(x_i) = 0$ est impossible, le circuit calculant f ne peut fournir une réponse correcte pour $\bar{\alpha} = i$ s'il n'utilise pas l'entrée x_i.

Reste le cas $\mathrm{ex}(x_i) = 1$: le nœud s fils de x_i n'est pas du type **3.2.2.b** puisqu'on a un circuit optimal pour f, donc il calcule une fonction de type ET ou XOR. Soit g la fonction booléenne constituant la deuxième entrée de s. Puisque la seule arête issue de x_i aboutit en s, g ne dépend pas de x_i et par conséquent la sortie de s varie avec x_i. Ceci prouve que s n'est pas la sortie f et donc il existe un deuxième nœud t, fils de s. Si s est un nœud de type ET, on choisit x_i constant de sorte que la sortie de s soit constante. Ceci permet après simplification de supprimer s et t. Si s est un nœud de type XOR, on choisit $x_i = g$: ce n'est pas un choix constant mais il est malgré tout valide et permet à nouveau de supprimer s et t.

Conséquence : le nombre minimal de nœuds internes nécessaire pour calculer une fonction de sélection augmente d'au moins deux unités lorsque l'ensemble dans lequel il faut sélectionner grandit d'un élément. Pour sélectionner un élément dans un singleton il faut au moins zéro nœuds internes, d'où la minoration $L_\Omega(f_k) \geqslant 2n - 2$.

3.4.1) Sinon, le couple (x_i, x_j) n'influe sur f que par la fonction de x_i et x_j calculée en s. Ceci est incompatible avec l'hypothèse qu'on peut calculer un ET et un XOR entre x_i et x_j en spécialisant f.

3.4.2) S'il n'existe pas dans Z au moins $|Z| - 1$ indices i ayant la propriété « tout chemin de i à s_f comporte au moins un nœud de degré sortant supérieur ou égal à 2 » alors on peut trouver deux indices n'ayant pas cette propriété et contredire la question précédente.

Caractère distinct : pour $i \in \{i_1, \ldots, i_{|Z|-1}\}$ et ω un chemin de i à s_f, on note $s(\omega)$ le nœud de plus petit numéro dans $S(\omega)$ ayant un degré sortant supérieur ou égal à 2. Si $i, j \in \{i_1, \ldots, i_{|Z|-1}\}$ sont distincts et si ω et ω' sont des chemins de i à s_f et de j à s_f alors $s(\omega) \neq s(\omega')$ d'après la question précédente et le caractère minimal de $s(\omega)$ et $s(\omega')$.

Minoration de $L_\Omega(f)$: on a donc au moins $|Z|-1$ sommets de degré sortant supérieur ou égal à 2, au moins $|Z|$ entrées et une seule sortie. L'inégalité **3.2.1** fournit le résultat.

3.4.3) Si $c \geqslant 2$ alors f_c admet des spécialisations de type ET et de type XOR pour tout couple d'indices dans $\{1,\ldots,n\}$. En effet, $f_c(\underbrace{1,\ldots,1}_{c-2},0,\ldots,0,x,y) = x \wedge y$ et $f_c(\underbrace{1,\ldots,1}_{c-1},0,\ldots,0,x,y) = x \oplus y$. De même, pour $c = 1$ on a $f_1(1,0,\ldots,0,x,y) = (\neg x) \wedge (\neg y)$. et $f_1(0,\ldots,0,x,y) = x \oplus y$. Donc $L_\Omega(f_c) \geqslant 2n-2$ pour tout $c \in \{1,\ldots,n-1\}$.

3.5) $|\Gamma_n| = 2^{2^n}$. On majore le nombre de circuits de taille L par le nombre de graphes à n entrées et L nœuds internes choisis dans Ω, topologiquement ordonnés les entrées ayant les numéros $1,\ldots,n$, tels que tout nœud interne est l'extrémité d'au plus deux arêtes issues de nœuds de numéros inférieurs. Il y a $|\Omega|^L$ façons de choisir les nœuds internes et pour le i-ème nœud interne, $i+n$ façons de choisir l'origine d'une arête (parmi les n entrées, les $i-1$ nœuds internes précédant, plus une origine fictive pour indiquer qu'il n'y a pas d'arête). Ainsi le nombre de circuits de taille L sur Ω est majoré par : $N(L) = |\Omega|^L((n+1)\ldots(n+L))^2$.

Avec $L = \lceil 2^{n-1}/n \rceil$ on a $\log_2(N(L)) = 2L\log_2(L) + O(L) = \log_2(|\Gamma_n|) - 2^n\log_2(n)/n + O(2^n\log_2(n)/n)$. Donc $N(L)$ est infiniment petit devant $|\Gamma_n|$, ce qui prouve qu'il existe pour n suffisament grand des fonctions booléennes non calculables par un circuit à L nœuds internes, ni par conséquent par un circuit de plus petite taille.