1 Classe AC

Définition 1. *Un circuit booléen avec n bits d'entrée est un graphe orienté acyclique (DAG) où les feuilles sont les n nœuds représentant l'entrée, la sortie est l'unique nœud dont le degré d'entrée est nul et chaque nœud est soit un nœud $\wedge$ ou $\vee$ (qui peuvent être d'arité quelconque) soit un nœud $\neg$ (unaire). Étant donné un mot $w \in \{0,1\}^*$ de taille n et un circuit dont la taille d'entrée est n, on peut évaluer le circuit sur w en remplaçant la i-ème feuille d'entrée par $\top$ quand $w_i = 1$ et $\bot$ quand $w_i = 0$.*

Définition 2. *On dit qu'une* famille de circuits $(C_i)_{i\in\mathbb{N}}$ *accepte un langage L quand pour chaque $n \in \mathbb{N}$, C_n a n bits d'entrée et accepte $\{w \in L \mid |w| = n\}$.*

Définition 3. *On définit la classe AC^i comme la classe des problèmes de décision acceptés par un famille de circuits booléens $(C_i)_{i\in\mathbb{N}}$ dont la taille du circuit est polynomiale et dont la profondeur est de taille $O(log(n)^i)$, c'est à dire qu'il existe k tel que pour tout $n \in \mathbb{N}$, la profondeur de C_n est bornée par $log(n^i) \times k$. La taille de C_n est limitée par $P(n)$ pour un certain P.*

Question 1. Justifier pourquoi l'on ne s'intéresse qu'aux circuits de taille polynomiale (et non pas exponentielle ou illimitée).

Solution 1. Étant donné une fonction booléenne $f : \{0,1\}^n \to \{0,1\}$ on peut regarder $f^{-1}(1) = \{(x_1,\dots,x_n) \mid f(x_1,\dots,x_n) = 1\}$.
En posant $\varphi(u_1,\dots,u_n) = \bigwedge_{u_i=1} x_i \wedge \bigwedge_{u_i=0} \neg x_i$ on a alors $\varphi(u_1,\dots,u_n)(x) = 1 \Leftrightarrow (u_1,\dots,u_n) = x$ et donc

$$f(x) = \bigvee_{(u,\dots,u)\in f^{-1}(1)} \varphi(u)$$

On peut représenter n'importe quelle fonction $f(x)$ par une formule (et donc un circuit) de profondeur 4 (à profondeur 1 on a $\bigvee_{u\in f^{-1}(u)}$, à profondeur 2 on a $\bigwedge_{u_i=1}\dots\bigwedge_{u_i=0}$ et à 3 et 4 on a x_i et $\neg x_i$).

Question 2. Montrer que le problème de décider la parité du nombre de 1 dans l'entrée est dans AC^1. Le problème de la parité correspond au langage $\{w \in \{0,1\}^* \mid |w|_1 \text{ est pair}\}$.

Solution 2. On veut réaliser un $\oplus$ de toutes les entrées $w_1,\dots,w_n$. Si l'on calcule $w_1 \oplus (w_2 \dots (w_{n-1} \oplus w_n))$ alors la profondeur va être $n-1$ fois celle d'un $\oplus$ binaire.
Pour simplifier on ne va travailler que sur les n qui sont des puissances de 2 (en remplissant avec des 0 une solution pour n peut être trouvée avec la puissance de 2 supérieure à n).
Si $|w| > 1$ on peut calculer $w'_i = w_{2i} \oplus w_{2i+1}$ on ramène à un w' de taille $|w'| = \lceil\frac{|w|}{2}\rceil$, la profondeur du calcul de w' est constante et la parité du nombre de 1 de w est la même que celle de w'. En procédant ainsi récursivement en $ln(n)$ étapes on arrive à une unique valeur qui est la solution. La profondeur totale de ce circuit est donc $\lceil ln(n)\rceil$ fois la profondeur d'un unique $\oplus$.

Question 3. Montrer que le problème de l'addition est dans AC^0. Pour cela, exhiber une famille de circuit de profondeur bornée qui accepte les mots $w = abc$ avec $|a| = |b| = |c|$ et $a + b = c$ (a, b et c sont en représentation binaire par exemple en petit boutien).

Solution 3. Pour tester la valeur c_i il faut regarder a_i, b_i mais aussi savoir s'il y a une retenue dans le calcul de $a_0 \dots a_{i-1} + b_0 \dots b_{i-1}$. On note r_i la retenue à l'étape i (c'est à dire $a_1 \dots a_i + b_1 \dots b_i = c_1 \dots c_i r_i$)
Si l'on regarde précisément le calcul de $a + b$ les retenues sont crées (i.e. $r_{i-1} = 0$ mais $r_i = 1$) quand $g_i = (a_i \wedge b_i)$ et transmises (i.e. $r_{i-1} = 1$ et $r_i = 1$) quand $p_i = (a_i \vee b_i)$. Donc il y a une retenue après l'étape i s'il existe $j \le i$ tel que $tr^i_j = g_j \wedge p_{j+1} \wedge \dots \wedge p_i$.
Donc pour savoir s'il y a une retenue en $i+1$ on calcule $r_i = \bigvee_j tr^i_j$ et la valeur de c_i doit être égale à $r_i + a_i + b_i$.

Les nœuds p_i et g_i sont de profondeur 1, les nœuds tr^i_j de profondeur 2, le nœud r_i est donc de profondeur 3, $(a_i \oplus b_i) \oplus r_{i-1} = c_i$ est donc de profondeur bornée.

Question 4. Montrer que le problème de la multiplication est dans AC^1. Pour cela, trouver un famille de circuit qui accepte les mots $w = abc$ avec $2|a| = 2|b| = |c|$ et $a \times b = c$ (a, b et c sont en représentation binaire par exemple en petit boutien).

Solution 4. On a vu dans l'addition qu'on sait calculer la somme de 2 nombres. On va proposer une méthode pour calculer la multiplication par une suite d'additions en utilisant les formules suivantes :

$$mult(a_1 \dots a_{2n}, b) = mult(a_1 \dots a_n, b) + mult(a_{n+1} \dots a_{2n}, b) \times 2^n$$
$$mult(0, b) = 0$$
$$mult(1, b) = b$$

Comme la multiplication par 2^n est facile et que l'addition est de profondeur constante, qu'il y a un $ln(n)$ étapes d'additions qui ont chacune une profondeur constante donc la profondeur est $ln(n)$.

Question 5. Montrer que tout langage régulier sur $\{0, 1\}$ est AC^1.

Solution 5. Soit $A = (Q, \{0, 1\}, \delta, q_0, F)$ un automate sur le langage considéré. On note $tr(i, j, q, q')$ le circuit qui calcule si $\delta(q, w_i \dots w_{j-1}) = q'$. Un mot $w_1 \dots w_n$ est accepté quand $\vee_{f\in F} tr(1, n + 1, q_0, f)$
Maintenant on peut calculer $tr(i, j, q, q')$ de la façon suivante :

$$tr(i, j, q, q') = \vee_{q''\in Q} tr(i,(i + j)/2, q, q'') \wedge tr((i + j)/2, j, q'', q')$$

$$tr(i, i + 2, q, q') = (q' = \delta(q, w_i))$$
$$tr(i, i + 1, q, q') = (q' = q)$$

La profondeur de ce circuit est logarithmique car à chaque étape on rappelle récursivement sur des segments de taille moitié avec un circuit de profondeur bornée (c'est un OU d'arité $|Q|$ de ET binaires donc borné par une profondeur $ln(|Q|) + 2$ indépendante de n).

2 Classe NC

Définition 4. *On note NC^i la classe des problèmes de décision acceptés par une famille de circuits booléens $(C_n)_{n\in\mathbb{N}}$ où l'arité des portes $\vee$ et $\wedge$ est limitée à deux et où la famille de circuits est de profondeur $O(ln(n)^i)$ avec une taille polynomiale.*

Question 6. Montrer que pour tout i nous avons l'inégalité $NC^i \subseteq AC^i \subseteq NC^{i+1}$.

Solution 6. Clairement, par définition $NC^i \subseteq AC^i$.
Maintenant étant donné un circuit correspondant à une famille de AC^i on remarque que chaque porte $\wedge$ ou $\vee$ a une arité bornée par la taille du circuit qui est polynomiale en n. En remplaçant chacune de ces portes par un arbre binaire équilibré des mêmes portes on obtient le résultat. Par exemple, si l'on a un nœud $\wedge(n_1, \dots, n_k)$ on créé deux nœuds $m_1 = \wedge(n_1, \dots, n_{k/2})$ et $m_2 = \wedge(n_{k/2+1}, \dots, n_k)$ et on recommence récursivement.

Question 7. Montrer que $NC^0 \neq AC^0$.

Solution 7. Une famille de circuit correspondant à un problème NC^0 ne peut lire qu'un nombre fini de bits et donc ne peut pas, par exemple, gérer l'addition.

2.1 Classe TC

Définition 5. *La classe TC^0 correspond à la classe des problèmes de décision acceptés par une famille de circuits où les portes sont $\neg$ ainsi que des portes de type $T_k(x_1, \ldots, x_n)$ avec $T_k(x_1, \ldots, x_n) = |\{i \mid x_i = \top\}| \geq k$*

Question 8. Montrer que l'on peut faire les portes $\wedge$, $\vee$ et MAJORITÉ avec MAJORITÉ$(x_1, \ldots, x_n) = \top$ quand la moitié de ses entrées sont $\top$ sans changer la profondeur de circuit.

Solution 8. $\wedge(v_1, \ldots, v_n) = T_n(v_1, \ldots, v_n)$ et $\vee(v_1, \ldots, v_n) = T_1(v_1, \ldots, v_n)$ enfin MAJORITÉ$(v_1, \ldots, v_n) = T_{n/2}(v_1, \ldots, v_n)$.

Question 9. Montrer qu'à l'inverse TC^0 correspond aussi à la classe des problèmes de décision acceptés par des circuits de profondeur bornée avec les portes $\wedge$, $\vee$, $\neg$ et MAJORITÉ (d'arités non bornées).

Solution 9. Étant donnée un porte $T_k(x_1, \ldots, x_n)$ on la transforme en la porte MAJORITÉ en rajoutant simplement des entrées $\top$ ou des $\perp$.
Si $2k > n$, $T_k(x_1, \ldots, x_n) =$ MAJORITÉ$(x_1, \ldots, x_n, t_1, \ldots, t_{2k-n})$ avec $t_i = \top$.
Si $2k < n$, $T_k(x_1, \ldots, x_n) =$ MAJORITÉ$(x_1, \ldots, x_n, b_1, \ldots, b_{n-2k})$ avec $b_i = \perp$.
(les portes $\vee$ et $\wedge$ sont donc inutiles)

Question 10. Montrer $TC^0 \subseteq NC^1$

Solution 10. De la même manière que pour $AC^0 \subseteq NC^1$ on va simuler les portes par des cascades de $\vee$ et $\wedge$. Quitte à rajouter des portes $\top$ et $\perp$, on peut facilement supposer que les entrées des portes MAJORITÉ sont d'arité des puissance de 2.
Pour simuler MAJORITÉ$(a_1, \ldots, a_k)$, on va compter le nombre de 1 dans $a_1, \ldots, a_k$. L'idée c'est d'utiliser un half-adder. On va récursivement calculer en profondeur constante des additions de trois nombres x, y, z qui vont produire deux nombres c et s tels que $c + s = x + y + z$. Pour cela c va contenir le reste (*carry*) et s la somme donc $s_i = x_i + y_i + z_i$ et $c_i = (x_i \wedge y_i) \vee (z_i \wedge y_i) \vee (x_i \wedge z_i)$.
Maintenant $half\_popcount(a_1, \ldots, a_{2k})$ retourne deux vecteurs de bits, le nombre de $\top$ et un reste.
Si $(s_1, c_1) = half\_popcount(a_1, \ldots, a_k)$ et $(s_2, c_2) = half\_popcount(a_{k+1}, \ldots, a_{2k})$ on calcule $s', c' = s_1 + s_2 + c_2$ et on renvoie le résultat de $s' + c_1 + c'$.

Question 11. Montrer que le problème de parité est dans TC^0.

Solution 11. Il suffit de tester pour chaque valeur de k si le nombre de 1 se trouve est $2k$ (et donc plus grand que $2k$ mais pas $2k + 1$) :

$$parity(a_1, \ldots, a_n) = \vee_{k \leq n/2} T_{2k}(a_1, \ldots a_n) \wedge \neg T_{2k+1}(a_1, \ldots a_n)$$

Question 12. Montrer que le problème de la multiplication est dans TC^0.

Solution 12. Pour simplifier la présentation de cette solution, nous allons procéder de façon modulaire.

Passage de la multiplication de deux nombres à l'addition de n nombres On ramène facilement le problème de la multiplication de deux nombres de taille n à l'addition de n nombres de taille $2n$ car $(a \times b) = \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n} 2^{k+j}(a_j \wedge b_k)$.

Passage de l'addition de m nombres de tailles n à l'addition en parrallèle de $ln(m)$ nombres de taille $2ln(m)$. Notons $a^1, \ldots, a^m$ les nombres à additionner et $r^i = \sum_j a_i^j$. Comme r^i est la somme de m nombres valant 0 ou 1, $r^i \leq m$ et donc r^i tient sur $ln(m)$ bits. On veut calculer le bit r_j^i en profondeur constante. Notons pour cela $bit(k, j)$ qui vaut 1 si la représentation binaire de k contient un 1 en j-ème position on a :

$$r_j^i = \bigvee_{\substack{1\leq k\leq n\\ bit(k,j)=1}} T_k(a_i^1,\dots,a_i^m)$$

On a par ailleurs que $\sum_k a^k = \sum_i 2^i r^i$ avec :

$$\begin{array}{ccc} a_1^1 & \cdots & a_n^1 \\ \vdots & \ddots & \vdots \\ a_1^m & \cdots & a_n^m \\ \hline r_1 = \sum_k a_1^k & \cdots & r_n = \sum_k a_n^k \end{array}$$

Comme chacun des r^i tient sur $ln(m)$ bits, on va regrouper les r^i en bloc de taille $B = ln(m)$ bits en calculant

$$v^\ell = \sum_{1\leq k\leq ln(m)} 2^k r^{k+\ell}$$

et donc on a toujours :

$$\sum_k a^k = \sum_i 2^i r^i = \sum_\ell v^\ell 2^{B\ell}$$

Nous allons montrer ensuite comment faire le calcul de v^ℓ en profondeur constante et taille polynomiale. Montrons d'abord comment s'en sortir si l'on sait calculer les v^ℓ.
Une fois que les v^ℓ sont calculés on calcule $x = \sum_\ell 2^{3B\ell} v^{3\ell}$ et $y = \sum_\ell 2^{B(3\ell+1)} v^{3\ell+1}$ et $z = \sum_\ell 2^{B(3\ell+2)} v^{2\ell+2}$. Chacun des v^ℓ tient sur $2ln(m)$ bits on a que les sommes plus hauts peuvent être transformées en OU bit à bit car les représentations binaires de ces nombres sont disjointes (entre $2^{B\ell}v^\ell$ et $2^{B(\ell+3)}v^{\ell+3}$ il n'y a pas de bits en commun car $v^\ell \leq 2^{2B}$).
Enfin la dernière étape est de faire la somme $x + y + z$ mais cela peut être fait en profondeur constante puisque la somme est dans $AC^0 \subseteq TC^0$.

Calcul des v^ℓ en profondeur constante Ce que l'on a envie de faire pour calculer v^ℓ c'est d'utiliser sa table de vérité. Le problème c'est que chaque bit de v^ℓ dépend de $ln(m) \times ln(m)$ bits des r^i. Donc la table de vérité peut être exponentielle en $ln(m) \times ln(m)$ ce qui est plus que polynomial en m.
Le calcul d'un v^ℓ correspond donc à faire la somme de $ln(m)$ nombres de taille $2ln(m)$. En itérant *une seule fois* le processus qui nous a permis de passer d'une somme de a^k à une somme de v^ℓ, on se ramène à écrire $v^\ell = \sum 2^{B'j} w^k$. Où les w^k sont de taille logarithmique en la taille v^ℓ et donc $|w^k| \leq 2ln(ln(m)^2) \leq 4ln(ln(m))$
Maintenant les w^k sont de taille très petite par rapport à n et donc le calcul de chaque w^k peut se faire en profondeur constante en utilisant sa table de vérité (comme à la question 2). Passer par la table de vérité produit bien un circuit de profondeur constante mais de taille exponentielle. Cependant cette taille est exponentielle en le nombre de bits dont dépend w^k qui ici est $4ln(ln(m)) \times ln(ln(m))$ et donc sub-linéaire en m. Notez qu'il y a un nombre polynomial de w^k car il y a $ln(m)/ln(ln(m))$ par v^ℓ et qu'il y a $m/ln(m)$ v^ℓ par a^k.
Une fois que les w^k sont calculés on passe des w^k aux v^ℓ comme nous sommes passés des v^ℓ aux a^k en en sommant un tous les trois, ce qui nous donne des nombres x', y' et z' et l'on fait la somme $x' + y' + z'$.