1 Un problème dans BPP

1.1 Définitions

Définition 1. *On rappelle que la classe BPP peut se définir comme la classe des langages L tels qu'il existe un polynôme P et un langage A reconnu en temps polynomial par une machine de Turing tels que :*

$$x \in L \Rightarrow \mathop{\mathbb{P}}_{\varepsilon \in \{0,1\}^{p(|x|)}}\Big((x,\varepsilon) \in A\Big) \geq \frac{2}{3} \qquad x \notin L \Rightarrow \mathop{\mathbb{P}}_{\varepsilon \in \{0,1\}^{p(|x|)}}\Big((x,\varepsilon) \in A\Big) \leq \frac{1}{3}$$

Définition 2. *On rappelle que le* degré d'un monôme $X_1^{d_1} \dots X_n^{d_n}$ *est* $\sum_j d_j$ *et que le* degré d'un polynôme *est le maximum des degrés des monômes le composant. Par exemple $XXXXY + XYZ + Z + ZY$ est de degré 5 (car $XXXXY$ est de degré 5).*

1.2 Polynômes en forme générale

Définition 3. *Un polynôme peut toujours s'écrire sous la forme de sommes de monôme. Un polynôme est en* forme générale *quand il est donné comme variable simple (i.e. X_i), produit, somme ou différence de polynômes en forme générale. Par exemple* $((X_1 + X_2) \times (X_1 - X_2) + X_1) \times X_1$.

On ignore ici volontairement les détails de l'encodage mais on suppose qu'il est non ambigu et que la taille d'un encodage est $O(ln(n))$ pour décrire la n-ième variable, et $O(1)+n_1+n_2$ pour décrire la somme (ou le produit) de deux polynômes de taille n_1 et n_2.

Question 1. Montrer que la fonction qui transforme un polynôme donné en forme générale en un polynôme sous forme de monômes ne peut pas être calculée en temps polynomial.

Solution 1. Considérons le polynôme $(X_1+X_2)\times(X_3+X_4)\times\cdots\times(X_{n-1}+X_n)$, sa taille est $O(n\times ln(n))$ mais sa forme développée a 2^n monômes de taille n.

Question 2. Quelle est la complexité de la fonction qui teste l'égalité de deux polynômes en forme générale en utilisant l'expansion en somme de monômes ?

Solution 2. Soit $s(p)$ la taille du polynôme. Montrons que le nombre $n(p)$ de monômes de p est borné par $2^{s(p)}$ et que le degré est au plus $d(p) \leq s(p)$.

- Pour P_1+P_2 on a $s(P_1+P_2) = 1+s(P_1)+s(P_2)$ mais $n(P_1+P_2) = n(P_1)+n(P_2)$ et $d(P_1+P_2) = max(d(P_1), d(P_2))$.
- Pour P_1-P_2 on a $s(P_1+P_2) = 1+s(P_1)+s(P_2)$ mais $n(P_1+P_2) = n(P_1)+n(P_2)$ et $d(P_1+P_2) = max(d(P_1), d(P_2))$.
- Pour $P_1 \times P + 2$ on a $s(P_1+P_2) = 1+s(P_1)+s(P_2)$ mais $n(P_1+P_2) = n(P_1)\times n(P_2)$ et $d(P_1+P_2) = d(P_1)+d(P_2)$.
- Pour X on a $s(p) = ln(n)$ et $d(p) = n(p) = 1$.

1.3 Lemme de Schwartz-Zippel

Lemme 1. *Soit $P \in \mathbb{Z}[X_1, \dots, X_n]$ non nul de degré d et soit S une partie finie de $\mathbb{Z}$. Si $x_1, \dots, x_n$ sont choisis uniformément dans S alors* $\mathbb{P}\Big(P(x_1, \dots, x_n) = 0\Big) \leq \dfrac{d}{|S|}$

Question 3. Montrer ou admettre ce lemme.

Solution 3. On montre le résultat par récurrence sur le nombre n de variables puis le degré d.

- Pour $n = 0$, le résultat est évident : $d = 0$ et $P = cst$ ou $d = -\infty$ et $P = 0$.
- Pour $n+1$ avec $n \in \mathbb{N}$ alors on a $P_0, \dots, P_d$ tel que $P = \sum_{i=0}^{d} X_{n+1}^{d-i} P_i(X_1, \dots, X_n)$. P est non nul, ssi l'un des P_i est non nul. On choisit i maximal tel que $P_i \neq 0$. Pour que $P(x_1, \dots, x_n, x_{n+1}) = 0$ il faut que le polynôme P_i de degré i s'annule en $(x_1, \dots, x_n)$ ou que le polynôme $P(x_1, \dots, x_n, X_{n+1})$ en la variable X_{n+1} de degré $d - i$, s'annule en $X_{n+1} = x_{n+1}$.
  Par récurrence on a $\mathbb{P}(P_i(x_1, \dots, x_n) = 0) \leq \dfrac{i}{|S|}$ et $\mathbb{P}(P(x_1, \dots, x_n, X_{n+1}) = 0) \leq \dfrac{d-i}{|S|}$ donc $\mathbb{P}(P(x_1, \dots, x_n, x_{n+1}) = 0) \leq \dfrac{d-i}{|S|} + \dfrac{i}{|S|} \leq \dfrac{d}{|S|}$

Question 4. En déduire un algorithme BPP pour tester l'égalité de deux polynômes en forme générale.

Solution 4. Soit P_1 et P_2 des polynômes données avec n variables. On calcule d le degré commun de P_1 et P_2 (sinon ils ne sont pas égaux) et on choisit $x_1, \dots, x_n$ dans $S = 1..3d$ et on renvoie $P = 0$ quand $P(x_1, \dots, x_n) = 0$ et on renvoie $P \neq 0$ sinon.
Notre algorithme ne renvoie jamais $P \neq 0$ en se trompant mais à moins d'une chance sur trois de renvoyer $P = 0$ quand $P \neq 0$.

2 Révision : Théorème de Rice

Définition 4. *Un état* inutile *d'une machine de Turing est un état qui n'est jamais visité pendant un calcul.*

Question 5. Montrer que le problème de décider si une machine de Turing a des états inutiles est indécidable.

Solution 5. Soit M une machine à n états et k symboles de bande. On ajoute 5 états ($init_1$, $init_2$, $init_3$, $init_4$, end) et un symbole de bande de bande η. Pour chaque état $2i$ de la machine, on ajoute une transition $2i, \eta \to 2i+1, >$ et pour chaque $2i+1$ on ajoute $2i+1, \eta \to 2i+2, <$. Les états $init_1$ et $init_2$ vont écrire η sur les deux premières cases puis lancer la machine sur l'état 1. La machine va donc passer sur tous les états et aux deux derniers états on va nettoyer la bande (selon la parité du nombre d'état on va utiliser $init_3$). Alors on lance M normalement et quand M finit on la fait visiter end. De cette façon, M ne termine que si notre nouvelle machine rentre dans l'état end.

Question 6. Expliquer pourquoi on ne peut pas appliquer le théorème de Rice pour résoudre ce problème.

Solution 6. Le théorème de Rice ne s'applique pas car il ne s'agit pas d'une propriété sémantique sur les programmes. Une machine qui utilise tous ses états a la même sémantique qu'une machine à laquelle on ajoute un état inutile.

3 Révision : Castor affairé

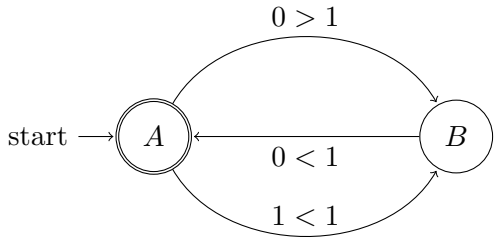
Le problème des « busy beavers » (en français : « castors affairés ») a été introduit par Radó, dans le but de définir « simplement » une fonction non calculable. Le modèle de machines de Turing considéré est le suivant : on suppose les machines déterministes, possédant un ruban bi-infini, un alphabet réduit à $\{0, 1\}$ (on ne distingue pas les 0 du symbole blanc). On suppose de plus que les machines possèdent un unique état final, duquel aucune transition ne sort, et qui n'est pas compté parmi les états de la machine. Enfin, on considérera toujours un ruban initialement complètement blanc et l'on suppose qu'il existe un état d'arrêt duquel ne part aucune transition.
La fonction du castor affairé, notée $\Sigma(n)$, est définie comme le nombre maximum de 1 écrits sur la bande (pas nécessairement consécutifs) après qu'une machine à $n+1$ états (n états d'opération et un état d'arrêt) arrive dans l'état d'arrêt (la machine doit donc s'arrêter).

Question 7. Justifier l'existence de $\Sigma(n)$ pour tout $n \in \mathbb{N}$.

Solution 7. Il y a un nombre fini de machine de Turing avec n état et 2 symboles. Pour chaque machine on peut définir son temps d'arrêt (éventuellement infini) et prendre le max des temps d'arrêt.

Question 8. Que valent $\Sigma(0)$, $\Sigma(1)$, $\Sigma(2)$ (ou plus simplement montrer que $\Sigma(2) \geq 4$) ?

Solution 8. $\Sigma(0) = 0$, $\Sigma(1) = 1$, $\Sigma(2) = 4$
Pour $\Sigma(0)$ et $\Sigma(1)$, c'est évident. Pour $\Sigma(2)$ on constate que la machine ci-dessous écrit bien 4 un.



Pour montrer que c'est le meilleur score il faut énumérer les machines de façon intelligente (par exemple, il n'y a que 3 transitions au plus et la première transition peut par symétrie être $A, 0 \xrightarrow{0>1} B$

Question 9. Montrer que la fonction Σ est strictement croissante.

Solution 9. Étant donné un machine à n états, on construit une machine à $n+1$ états en branchant sur le dernier état une boucle qui écrit un 1 sur le premier 0 qu'elle croise puis s'arrête.

Le modèle de fonction calculable considéré est le suivant : f est calculable si et seulement s'il existe une machine de Turing qui, sur un ruban contenant initialement n symboles 1 consécutifs à droite du symbole blanc de départ, s'arrête après un nombre fini de déplacements en produisant un bloc de $f(n)$ symboles 1 consécutifs.

Question 10. Montrer que la fonction Σ du castor affairé croit strictement plus vite que toute fonction calculable f (i.e. $f(n) = o(\Sigma(n))$).

Solution 10. Soit f une fonction calculable. La fonction $p(n) = 2^n$ est aussi calculable. La composée $p \circ f \circ p$ est aussi calculable. Et la fonction $c_x(n) = x$ peut se construire avec $O(ln(x))$ états (on écrit x en binaire puis on développe). On peut aussi construire une MT $p \circ f \circ p$ en $O(1)$ états donc pour $p \circ f \circ p \circ c_x$ on a une MT qui écrit $2^{f(x)}$ avec $O(ln_2(x))$ états. Comme Σ est strictement croissante on a pour x assez grand que $f(x) = o(\Sigma(O(ln_2(x))))$ mais $\Sigma(O(ln_2(x))) \leq \Sigma(x)$ (par croissance de Σ) et donc Σ grandit plus vite que f.