Correction de l'Examen Final

Exercice 1.
Pour les définitions, voir le cours.
Les inclusions sont :

$$\mathsf{P} \subseteq \mathsf{ZPP} \subseteq \mathsf{RP} \subseteq \mathsf{BPP} \subseteq \mathsf{EXP},$$

et

$$\mathsf{RP} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}.$$

(par contre on ne sait pas comparer BPP et NP)

Pour les **exercices 2 et 3** se référer au cours.

Exercice 4.
On va définir des machines à partir de $\mathcal{M}$ dépendant d'un paramètre entier k, puis choisir une valeur de k garantissant la propriété demandée.
Pour $k \geq 1$ on définit ainsi la machine probabiliste $\mathcal{M}_k$, qui, sur l'entrée x :
– initialise un compteur à la valeur k et une variable b à 0,
– tant que le compteur est différent de 0 :
  – effectue une nouvelle exécution de $\mathcal{M}$ sur l'entrée x ; si cette exécution accepte, alors $b := 1$;
  – décrémente le compteur ;
– si $b = 1$ alors la machine accepte, sinon elle rejette.
Ainsi $\mathcal{M}_k$ effectue k exécutions indépendantes de $\mathcal{M}$ et accepte ssi l'une de ces exécutions accepte.
On a alors :
– si $x \notin L$, alors $Pr[\mathcal{M}_k(x) = 1] = 0$, car $Pr[\mathcal{M}(x) = 1] = 0$.
– si $x \in L$, alors $Pr[\mathcal{M}_k(x) = 0] = Pr[E_1 \wedge E_2 \wedge \cdots \wedge E_k]$, où E_i désigne l'événement "la i-ème exécution de $\mathcal{M}$ rejette", pour $1 \leq i \leq k$.
  Or, comme les exécutions de $\mathcal{M}$ sont indépendantes, on a :

$$Pr[E_1 \wedge E_2 \wedge \cdots \wedge E_k] = \Pi_{i=1}^{k} Pr[E_i],$$

  et $Pr[E_i] \leq (1 - n^{-c})$, pour $1 \leq i \leq k$. Donc :

$$Pr[\mathcal{M}_k(x) = 0] \leq (1 - n^{-c})^k.$$

On veut donc choisir k de telle sorte que : $(1 - n^{-c})^k \leq 2^{-n^d}$.
Il nous suffit pour cela d'avoir : $\log((1 - n^{-c})^k) \leq \log(2^{-n^d})$,
soit $k \geq -n^d / \log(1 - n^{-c})$.
On peut donc prendre $\mathcal{M}' = \mathcal{M}_k$ avec un k satisfaisant cette condition.

Exercice 5.

1. Montrons que FC appartient à NL.
   On considère la machine non déterministe $\mathcal{M}$ suivante :
   sur l'entrée G, (on note m le nombre de sommets de ce graphe orienté)

– pour chaque sommet v_1 de G,
  pour chaque sommet $v_2 \neq v_1$ de G,
  – on initialise un compteur à $\llcorner m \lrcorner$,
  – $v := v_1$,
  – (*) on choisit non déterministiquement un sommet w de G tel que (v, w) est une arête,
  – on décrémente le compteur,
  – si $w \neq v_2$ alors {si le compteur est à 0 on rejette, sinon $v := w$ et on retourne à (*)}
    % à ce point on a trouvé un chemin de v_1 à v_2
  on passe au v_2 suivant
  on passe au v_1 suivant
– on accepte.

On a :
– Si G est fortement connexe, alors : pour tous sommets $v_1 \neq v_2$ de G, il existe un chemin de longueur inférieure ou égale à m de v_1 à v_2, donc une des exécutions de $\mathcal{M}$ accepte G ;
– Si G n'est pas fortement connexe, alors : il existe deux sommets $v_1 \neq v_2$ de G tels qu'il n'y a pas de chemin de longueur inférieure ou égale à m de v_1 à v_2, donc aucune exécution de $\mathcal{M}$ n'accepte G.

Donc la machine non-déterministe $\mathcal{M}$ décide bien le langage FC.

De plus cette machine n'utilise que 5 variables (v_1, v_2, v, w, le compteur) contenant des mots de longueur inférieure ou égale à $\log m$. Donc $\mathcal{M}$ fonctionne en espace logarithmique par rapport à la taille de l'entrée G. Donc FC appartient à NL.

2. Montrons que FC est NL-dur.

On sait que le langage $PATH$ est NL-complet. On va donc essayer de donner une réduction logspace de $PATH$ à FC.

On considère pour cela la fonction f suivante sur les mots de $\{0,1\}^*$:
– si x n'est pas de la forme $< G, s, t >$ avec G est un graphe orienté et s, t deux sommets de G, alors $f(x) = x_0$ constant qui n'est pas de la forme $< G' >$ pour G' un graphe orienté,
– si $x =< G, s, t >$ pour G un graphe orienté et s, t deux sommets de G, alors :
  $f(x) =< G' >$, où G' est le graphe obtenu à partir de G en :
  – ajoutant pour chaque sommet $v \neq s$ une arête de v à s,
  – ajoutant pour chaque sommet $v \neq s, t$ une arête de t à v.

Alors on a :
– Si $< G, s, t > \in PATH$:
  il existe un chemin γ de s à t dans G, donc aussi dans G'. Considérons alors $v_1 \neq v_2$ deux sommets de G'. Si $v_1 \neq s$ et $v_2 \neq t$ on a : une arête de v_1 à s, un chemin γ de s à t, une arête de t à v_2. Donc il existe dans G' un chemin de v_1 à v_2. Les cas où $v_1 = s$ ou $v_2 = t$ sont similaires. Donc $G' \in FC$.
– Si $< G, s, t > \notin PATH$:
  supposons qu'on ait dans G' un chemin γ' de s à t. Quitte à le raccourcir on peut supposer qu'il ne visite qu'une seule fois s et t, donc il n'utilise aucune des "nouvelles" arêtes de G', donc c'est aussi un chemin de G. Ceci contredit le fait que $< G, s, t > \notin PATH$. Donc il n'existe pas de chemin de s à t dans G', donc $G' \notin FC$.

On a donc bien : $f(x) \in FC$ ssi $x \in PATH$.

De plus la fonction f peut être calculée en espace logarithmique, car il suffit pour cela d'effectuer un parcours des sommets de G, en gardant en mémoire le sommet v courant (de taille $\log m$, où m est le nombre de sommets de G).

Donc $PATH \leq_L FC$. On en déduit ainsi que FC est NL-dur, et d'après la question précédente il est NL-complet.

Exercice 6.

1. Supposons que $\mathsf{E} \neq \mathsf{ESPACE}$. Alors $\mathsf{E} \subsetneq \mathsf{ESPACE}$ donc il existe un langage L dans ESPACE qui n'est pas dans E. Soit $c > 0$ tel que $L \in \mathsf{SPACE}(2^{cn})$. Considérons le langage

$$L_{\text{pad}} := \{\langle x, 1^{2^{c|x|}} \rangle : x \in L\}.$$

Alors $L_{\text{pad}} \in \mathsf{PSPACE}$. En effet, pour décider si un mot quelconque y appartient à L_{pad}, on décode d'abord y en $\langle x, 1^{2^{c|x|}} \rangle$ (si y n'est pas de cette forme, on rejette). On applique ensuite à x l'algorithme témoin de l'appartenance de L à $\mathsf{SPACE}(2^{cn})$ et on accepte si et seulement $x \in L$. L'espace utilisé est alors au plus $2^{c|x|}$, qui est bien polynomial en la taille de $y = \langle x, 1^{2^{c|x|}} \rangle$. Montrons maintenant que $L_{\text{pad}} \notin \mathsf{P}$, par contraposée. Supposons qu'il existe un algorithme $\mathcal{A}$ décidant L_{pad} en temps n^d pour un certain $d > 0$. L'algorithme suivant décide L en temps exponentiel : sur l'entrée x, on calcule $\langle x, 1^{2^{c|x|}} \rangle$ (ce qui prend essentiellement le temps d'écrire le couple, donc un temps exponentiel). Puis on applique $\mathcal{A}$ au couple obtenu, et on accepte si et seulement si $\mathcal{A}$ accepte. Le temps de calcul est $\left|\langle x, 1^{2^{c|x|}} \rangle\right|^d = O(2^{c|x|^d}) = O(2^{cd|x|})$. Ainsi, $L \in \mathsf{E}$ ce qui est absurde.

2. On utilise ici aussi la technique du *padding*, mais de manière plus délicate. Le langage de padding utilisé est unaire. Soit L le même langage qu'à la question précédente, et L_{un} le langage défini par

$$L_{\text{un}} := \{1^{1x} : x \in L\}$$

où $1x$ est le mot obtenu en ajoutant un 1 devant le mot x, et est vu comme un entier binaire. Le langage L_{un} est dans PSPACE : sur l'entrée y constituée uniquement de 1 (sinon on rejette), on calcule x tel que $y = 1^{1x}$ (ce qui se fait aisément en espace polynomial), puis on utilise l'algorithme qui témoigne de l'appartenance de L à ESPACE. L'espace utilisé est exponentiel en $|x|$, donc polynomial en $|1^{1x}| = 1x$. Il est également dans P/poly grâce à l'indication. Pour la même raison qu'à la question précédente, L_{un} ne peut pas être dans P.

3. (a) On veut montrer que $L \leq_T^p U$, c'est-à-dire que le langage L peut être décidé en temps polynomial par une machine de Turing avec oracle U. Pour tout n, notons $a(n)$ la taille du conseil α_n ($a(n)$ est par définition un polynôme en n). Pour décider L en temps polynomial grâce à l'oracle U, on utilise l'algorithme suivant : sur une entrée x de longueur n, on teste pour tout $i \leq a(n)$ lequel de $1^{\langle n,i,0 \rangle}$ ou $1^{\langle n,i,1 \rangle}$ appartient à U. Une fois cette boucle effectuée pour tout $i \leq a(n)$ (donc un nombre polynomial de fois), on connaît le conseil α_n. On peut alors exécuter la machine $\mathcal{M}$ en se servant du conseil α_n, et elle permet de décider en temps polynomial si x appartient à L.

(b) Soit $\mathcal{M}$ la machine qui décide L en temps polynomial avec conseil $(\alpha_n)_n$, et $\mathcal{N}$ la machine qui décide L en espace polynomial. Notons toujours $a(n)$ la longueur du conseil α_n. On énumère dans l'ordre lexicographique tous les mots β de longueur $a(n)$. Pour chacun de ces mots, on énumère tous les mots x de longueur n. On lance en parallèle les machines $\mathcal{M}$ avec le conseil β et $\mathcal{N}$ sur l'entrée x. Si les deux sont d'accord, on passe au mot x suivant. Sinon, on arrête l'énumération des x, et on passe

au mot β suivant. Si pour un mot β, les deux machines s'accordent sur tous les mots de longueur n, on pose $\beta_n = \beta$.

Pour les deux énumérations imbriquées, on n'a à chaque fois besoin de retenir que les mots β et x courants. De plus, exécuter $\mathcal{M}$ se faisant en temps poynomial, cela prend un espace au plus polynomial. Enfin, exécuter $\mathcal{N}$ prend par définition un espace au plus polynomial. Ainsi, l'algorithme proposé utilise un espace au plus polynomial en n.

(c) On peut montrer qu'une variante du langage U de la question **3(a)** vérifie les conditions requises. On considère

$$U' = \{1^{\langle n,i,\beta_{n,i}\rangle} : \beta_{n,i} \text{ est le } i\text{-ème bit de } \beta_n\},$$

où $(\beta_n)_n$ est la famille définie à la question **3(b)**. On peut décider L en temps polynomial avec oracle U' ($L \leq_T^p U'$) avec le même algorithme qu'à la question **3(a)** puisque qu'en temps qu'oracle pour L, U' se comporte par définition de la même façon que U. De plus, U' est bien décidable en espace polynomial : sur une entrée 1^t, on décompose t en $\langle n, i, b\rangle$ (où b est un unique bit). On calcule ensuite β_n en espace polynomial en n (et donc en $|1^t|$), puis on teste si le i-ème bit de β_n est b.

(d) En utilisant la clôture de P par réduction Cook-Turing [1], on peut conclure : si U' appartenait à P, alors ce serait le cas de L, ce qui n'est pas.

(e) Les questions précédentes prouvent que si $\mathsf{P} \neq \mathsf{PSPACE} \cap \mathsf{P/poly}$, alors il existe un langage unaire U' dans $\mathsf{PSPACE} \setminus \mathsf{P}$. Considérons le langage

$$U_{\text{bin}} := \{x : 1^{1x} \in U'\}.$$

Ce langage appartient à ESPACE : il suffit sur l'entrée x de calculer 1^{1x} puis de tester l'appartenance de 1^{1x} à U'. De même, il n'appartient pas à E car $U' \notin \mathsf{P}$. Donc $\mathsf{E} \neq \mathsf{ESPACE}$.

4. Par le théorème de Sipser et Gács, on sait que $\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$. Donc par définition $\mathsf{BPP} \subseteq \mathsf{PH}$. Mais comme $\mathsf{PH} \subseteq \mathsf{PSPACE}$, $\mathsf{BPP} \subseteq \mathsf{PSPACE}$.

5. Le théorème d'Adelman nous apprend que $\mathsf{BPP} \subseteq \mathsf{P/poly}$. Ainsi, avec la question précédente, $\mathsf{BPP} \subseteq \mathsf{PSPACE} \cap \mathsf{P/poly}$. Comme $\mathsf{P} \subseteq \mathsf{BPP}$, alors $\mathsf{P} \neq \mathsf{BPP}$ implique $\mathsf{P} \neq \mathsf{PSPACE} \cap \mathsf{P/poly}$. D'où $\mathsf{E} \neq \mathsf{ESPACE}$ grâce à la question **3**.

1. Également appelée *réduction par oracle*.