Corrigé TD 6
Complexité

Luc Lapointe
luc.lapointe@ens-paris-saclay.fr
home.lmf.cnrs.fr/LucLapointe/

Exercice 1 – NL différent de P ?

On admet que la classe $\text{SPACE}(\log^2 n)$ est strictement incluse dans la classe $\text{SPACE}(\log^3 n)$. *Ce résultat est donné par le théorème dit de hiérarchie en espace, qui exploite le fait que* $\log^2 = o(\log^3)$.

Vous voyez passer dans un échange de mail la preuve suivante que $\text{NL} \neq \text{P}$:

```
NL inclus dans L^2 (Savitch)
L^2 strictement dans L^3 (hiérarchie)
L^3 inclus dans P (compter)
```

Qu'en dire ?

Solution

Déjà, toutes les ressources de référence indiquent que l'inclusion stricte de NL dans P est un problème ouvert. Cette preuve est donc probablement fausse !

Le schéma global de la preuve est bon : $A \subseteq B \subsetneq C \subseteq D$ est bien une preuve de $A \neq D$.

Les L avec des puissances désignent $\text{SPACE}(\log^2 n)$ et $\text{SPACE}(\log^3 n)$.

Le théorème de Savitch est reconnu comme étant correct, et montre effectivement que $\text{NL} \subseteq \text{L}^2$.

Le théorème de hiérarchie en espace a bien pour corollaire que $\text{L}^2 \subsetneq \text{L}^3$.

C'est donc la dernière hypothèse qui pose problème. On a bien que $\text{L} \subseteq \text{P}$, mais ça ne vaut plus du tout[1] pour L^3 ! On pourrait penser que c'est vrai avec une intuition qui dirait que « Une exponentielle de logarithmes est bornée par un polynômes », sauf qu'on a en fait l'égalité

$$\exp(\log^3(n)) = n^{\log^2(n)}$$

qui ne permet de conclure à aucune inclusion dans une classe $\text{TIME}\left(n^k\right)$.

Exercice 2 – P-choix

[1] Ou du moins, si c'est vrai, on n'en a pas encore de preuve.

Un langage L appartient à *P-choix*, écrit $L \in \mathrm{P}_c$, s'il existe une fonction $f : \Sigma^* \times \Sigma^* \to \Sigma^*$, calculable en temps polynomial, telle que $\forall x, y \in \Sigma^*$:
- $f(x, y) \in \{x, y\}$,
- si $x \in L$ ou $y \in L$ alors $f(x, y) \in L$.

Dans ce cas, f est appelée la *fonction de choix* pour L.

1. Montrer que $\mathrm{P} \subseteq \mathrm{P}_c$.
2. Montrer que P_c est clos par complémentaire.
3. Montrer que s'il existe un problème NP-dur dans P_c alors $\mathrm{P} = \mathrm{NP}$.

Idée de solution

1. Soit $L \in \mathrm{P}$ et A un algorithme qui le résout en temps déterministe polynomial. f peut consister en : exécuter A sur x, et renvoyer x si et seulement si $x \in L$.
2. Soit $L \in \mathrm{P}_c$ et f associée. La fonction $\bar{\mathrm{f}}(x, y) = f(y, x)$ convient pour montrer que le complémentaire de L est dans P_c.
3. Soit L un problème NP-dur dans P_c. Alors n'importe quel autre problème NP est dans P_c, et notamment SAT. L'objectif est de montrer que SAT est dans P, en utilisant l'existence de fonctions de choix. Soit φ une entrée de SAT. Deux preuves différentes et élégantes sont disponibles à partir de là :
   - Comme P_c est clos par complémentaire, alors notamment TAUTOLOGIE est dans P_c. *Ce n'est pas exactement le complémentaire de SAT, mais TAUTOLOGIE est interréductible avec le complémentaire de SAT.* On utilise f la fonction de choix de TAUTOLOGIE sur l'entrée $(\neg\varphi, \bot)$. Si la réponse est $\neg\varphi$, alors φ n'est pas satisfiable. Si la réponse est $\bot$, alors φ est satisfiable.
   - P_c peut être vue comme une classe où on dispose d'une forme faible de non-déterminisme. Nous pouvons l'utiliser pour résoudre SAT avec l'algorithme suivant :

```
Soit n le nombre de variables dans phi.
Soit f la fonction de choix de SAT.
phi[0] <- phi
Pour i allant de 1 à n :
  phi[i] <- f(phi[i-1][x_i <- true], phi[i-1][x_i <- false])
Évaluer phi[n], renvoyer le résultat obtenu.
```

Exercice 3 – Caractériser P = NP

Étant donné un alphabet Σ, une fonction $f : \Sigma^* \to \Sigma^*$ est un *morphisme* si $f(\Sigma) \subseteq \Sigma$ et pour tout $m = m_1 \ldots m_n \in \Sigma^*$, $f(m) = f(m_1) \ldots f(m_n)$. Autrement dit, f est entièrement déterminée par les valeurs prises sur Σ.

Montrer que $\mathrm{P} = \mathrm{NP}$ si et seulement si P est clos par morphisme.

Idée de solution

Supposons $\mathrm{P} = \mathrm{NP}$. Soit L un langage dans P, et soit f un morphisme. On propose un algorithme non-déterministe en temps polynomial pour décider $f(L)$ qui consiste à deviner l'image réciproque de chaque lettre de Σ par f, puis à résoudre en temps polynomial le problème sur l'image réciproque de l'entrée avec l'algorithme en temps polynomial qui résout L. On conclut avec $\mathrm{P} = \mathrm{NP}$.

Supposons maintenant que P est clos par morphisme. Soit L dans NP. On considère M la machine déterministe qui le résout avec un certificat. En particulier, le problème de décision L' qui prend en entrée un certificat de taille exactement polynomiale en l'entrée et une entrée de L est dans P. On modifie M pour séparer artificiellement les lettres utilisées pour le certificat et pour l'entrée, et on considère le morphisme qui est l'identité sur les caractères de l'entrée, et un caractère $\perp$ sur les caractères du certificat. L'image de L' par ce morphisme, qui est toujours dans P, est *quasiment* L, sauf qu'il y a un nombre de caractères inutiles en entrée polynomial en la taille de l'entrée. Étant donné que ce nombre est fonction de la taille de l'entrée, il ne donne aucune info, et on peut en déduire que L est dans P.

Exercice 4 – Des problèmes à classer

Pour les problèmes suivants, donner une borne inférieure et supérieure sur leur appartenance aux classes de complexité parmi L, NL, P, NP (ou co-NP) et PSpace. Donner une borne inférieure correspond à établir la dureté du problème pour des réductions en espace logarithmique. Idéalement, ces deux bornes sont les mêmes, mais ce n'est parfois ce n'est pas le cas. On rappelle différents problèmes complets.

Le problème Reach est NL-complet.

Entrée: Un graphe G et deux sommets s et t.
Question: Existe-t-il un chemin de s à t ?

Le problème BinOpGen est P-complet.

Entrée: Un ensemble fini S, une opération binaire $* : S \times S \to S$, un sous-ensemble initial $I \subseteq S$ et une cible $t \in S$.
Question: t est-il dans la clôture transitive de l'ensemble I par $*$?

---

1. Le problème $\text{BinOpGen}_{\text{assoc}}$ qui correspond au problème BinOpGen mais avec une opération binaire $*$ associative.
2. Le problème MonotoneCir de la satisfaction d'un circuit.

   Entrée: Un circuit booléen C et l'un de ses sommets n

Question: $v(n) = \top$?

où un circuit booléen est un graphe acyclique $C = (V, E)$ avec deux types de sommets : $\wedge$ et $\vee$. La valeur booléenne d'un sommet v est alors la conjonction ou la disjonction des sommets vers lesquels il pointe, initialisée avec la convention que $\wedge\, \emptyset = \top$ et $\vee\, \emptyset = \bot$.

3. Le problème 1-Reach de l'atteignabilité dans un graphe de degré 1.
4. Le problème MAJSAT de la satisfiabilité par la moitié des valuations.

   Entrée: Une formule *quantifiée* φ de la logique propositionnelle.

   Question: Est-ce que au moins la moitié des valuations satisfont φ ?
5. Le problème 2-SAT de la satisfiabilité d'une 2-CNF.

Idée de solution

1. Dans P en utilisant le même algorithme que BinOpGen. En fait, le problème est même dans NL : deviner le successeur petit à petit, en retenant juste le sommet courant. Notez que c'est une technique qui ne fonctionne pas si la loi de composition n'est pas associative, car un chemin peut nécessiter de faire des compositions « lointaines » avant de faire celles proches de la source.

   Réduction depuis Reach pour montrer qu'il est NL-dur : pour $G = (S, A), s, t$ une entrée de Reach, on construit l'entrée de $\text{BinOpGen}_{\text{assoc}}$ $X, *, I, t'$ suivante :
   - $X = S^2$
   - $t' = (s, t)$
   - $I = A \cup \{(x, x), x \in S\}$
   - $(s_1, s_2) * (s_3, s_4) = \begin{cases} (s_1, s_4) \text{ si } s_3 = s_4 \\ (s_1, s_2) \text{ sinon.} \end{cases}$
2. Dans P en étiquetant petit à petit les sommets du circuit. Également P-dur, cf Wikipedia.
3. Dans L, en retenant le sommet en cours et le nombre de sommets visités. Donc notamment, L-complet pour les réductions logspace (comme tout problème non-trivial dans L).
4. Dans PSpace, en testant toutes les valuations et en les comptant. NP-dur, en rajoutant autant de variables y_i que de variables dans l'entrée de SAT, et en rajoutant au début de la formul $(y_0 \vee \neg y_n) \wedge \ldots \wedge (y_n \vee \neg y_n)$.

   MAJSAT est un problème complet pour la classe probabiliste PP, et prouver que MAJSAT est dans NP ou est PSpace-dur résoudrait des problèmes ouverts concernant PP. cf le cours de Complexité avancée de M1.
5. NL-complet. Pour montrer que c'est NL, on utilise que NL = coNL. Deviner au fur et à mesure un « chemin » de variables (une chaîne

d'implications) qui doivent nécessairement être vraies, qui commence à une variable x et finit par atteindre une variable $\neg x$.

Pour NL-dur : depuis REACH. Pour chaque arête (x, y) rajouter une clause $\neg x \vee y$, choisie ainsi pour modéliser $x \Rightarrow y$.