Examen du cours Complexité (L3)

Les documents (notes, polycopiés, ..) et calculatrices (téléphone, tablette, ..) ne sont pas autorisés.

Date : 19 janv. 2023 à 14h00 / Durée : 2 heures

Exercice 1 : Propriétés des classes de complexité

1. Donnez toutes les inclusions existant entre les 4 familles de langages suivantes : $\mathsf{TIME}(O(1))$, $\mathsf{NTIME}(O(1))$, $\mathsf{SPACE}(0)$, ainsi que la classe DEF des langages de la forme $F \cdot A^*$ où A est un alphabet fini et où $F \subseteq A^*$ est un langage fini sur A. Justifiez vos réponses.

   Solution:
   Un langage $L \subseteq A^*$ est dans $\mathsf{TIME}(O(1))$ s'il existe un n_0 tel que tout mot $x \in A^*$ est décidé (accepté ou rejeté) en temps $\leq n_0$, c.-à-d. sans pouvoir lire plus que les n_0 premières lettres du mot. Donc pour un mot w de longueur n_0, L contient soit tous les mots commençant par w soit aucun. In fine, $L = (L \cap A^{n_0}).A^* \cup L \cap A^{<n_0}$. Ainsi L est constitué d'un langage de DEF combiné à un langage fini. Et clairement tous les langages de cette forme sont dans $\mathsf{TIME}(O(1))$ (et sont réguliers). Le même raisonnement s'applique à $\mathsf{NTIME}(O(1))$ qui coïncide donc avec $\mathsf{TIME}(O(1))$.
   Notons que $\mathsf{TIME}(O(1))$ contient des langages qui ne sont pas dans DEF, p.ex. tous les langages finis non vides. Par ailleurs, tous les langages réguliers sont dans $\mathsf{SPACE}(0)$ mais $\mathsf{SPACE}(0)$ contient des langages réguliers qui ne sont pas dans $\mathsf{TIME}(O(1))$, p.ex. le langage $(ab)^*$. On a donc

   $$\mathsf{DEF} \subsetneq \mathsf{TIME}(O(1)) = \mathsf{NTIME}(O(1)) \subsetneq \mathsf{SPACE}(0).$$

2. Montrez que, pour tous langages L_1, L_2 dans NP, les langages $L_1 \cap L_2$ et $L_1 \cup L_2$ sont aussi dans NP.

   Solution:
   Pour $L_1 \cap L_2$ on construit, à partir de machines non déterministes $\mathcal{M}_1$ et $\mathcal{M}_2$ reconnaissant respectivement L_1 et L_2, une machine $\mathcal{M}$ qui simule les deux machines et n'accepte un mot x que si toutes deux acceptent x. Pour reconnaître $L_1 \cup L_2$, la machine devine si elle doit simuler $\mathcal{M}_1$ ou $\mathcal{M}_2$.

3. Est-ce que, pour tous langages NP-complets L_1, L_2, le langage $L_1 \cap L_2$ est NP-complet ?

   Solution:
   $L_1 \cap L_2$ n'est pas forcément NP-complet, p.ex., il peut être vide.

4. Donnez deux langages NP-complets L_1 et L_2 dont l'union $L_1 \cup L_2$ est dans P (aussi appelé PTIME).
   Déduisez-en que si $\mathsf{P} \neq \mathsf{NP}$ alors la classe des langages NP-complets n'est pas fermée par union.

Solution:
Soit $L \subseteq \{a,b\}^*$ un langage NP-complet (il en existe). On note L' la version de L écrite avec deux nouvelles lettres a', b'. Alors $L_1 \stackrel{\text{def}}{=} L \cup \{a',b'\}^*$ et $L_2 \stackrel{\text{def}}{=} L' \cup \{a,b\}^*$ sont NP-complets. Et $L_1 \cup L_2 = \{a,b\}^* + \{a',b'\}^*$ est dans P.
Pour la 2ème question, on reprend le langage $L_1 \cup L_2$. S'il est NP-complet alors tout langage de NP se réduit à un langage de P et donc P = NP. Par contraposée, on déduit que si P $\neq$ NP alors $L_1 \cup L_2$ n'est pas NP-complet, donc la classe des langages NP-complets n'est pas fermée par union.

Exercice 2 : Autour de SubsetSum.

On rappelle que SubsetSum demande, étant donnés une liste d'entiers naturels $a_1, \ldots, a_m$ et un objectif $g \in \mathbb{N}$, si on peut obtenir g comme somme de certains des a_i, c.-à-d.

$$\text{SubsetSum} = \{(a_1, a_2, \ldots, a_m), g \mid \exists I \subseteq \{1, \ldots, m\} : g = \sum_{i \in I} a_i\},$$

où on suppose une représentation telle que la taille d'une instance est en $O(\lceil \log g \rceil + \sum_i \lceil \log(a_i) \rceil)$ (p.ex. les entiers sont écrits en base 2 et séparés par un symbole non numérique). On a vu en cours que ce problème est NP-complet.
Dans cet exercice, on va considérer certaines variantes de SubsetSum :
— $\mathbb{Z}$-SubsetSum : ici g ainsi que les a_i ne sont plus des entiers naturels mais des entiers relatifs.
— 2dim-SubsetSum : ici g et les a_i ne sont plus des entiers naturels mais des vecteurs de $\mathbb{N}^2$.
— Half-SubsetSum : ici on ne donne pas g, seulement la liste $a_1, \ldots, a_m$, et on demande s'il existe $I \subseteq \{1, \ldots, m\}$ tel que $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$ (c.-à-d. si on peut partitionner la liste en deux morceaux de même poids).
— 2SubsetSum : ici on se donne deux objectifs g_1 et g_2 et on demande s'il existe $I_1, I_2 \subseteq \{1, \ldots, m\}$ tels que $I_1 \cap I_2 = \emptyset$ et $\sum_{i \in I_j} a_i = g_j$ pour $j = 1, 2$ (c.-à-d. si on peut réaliser à la fois g_1 et g_2 sans utiliser un même a_i dans les deux sommes).
— Multi-SubsetSum : ici on demande s'il existe une suite $i_1, \ldots, i_\ell$ d'indices tels que $g = \sum_{k=1}^{\ell} a_{i_k}$ (on a donc le droit de réutiliser un même a_i plusieurs fois). Notons que $\ell > m$ est possible.
On liste maintenant ces variantes par ordre (subjectif) de difficulté.

5. Montrez que $\mathbb{Z}$-SubsetSum est NP-complet.

Solution:
$\mathbb{Z}$-SubsetSum est NP-difficile car il *généralise* SubsetSum, et il existe une réduction triviale de SubsetSum à $\mathbb{Z}$-SubsetSum.
Il suffit donc de montrer que $\mathbb{Z}$-SubsetSum est dans NP : l'algorithme qui devine I puis vérifie en temps polynomial que $g = \sum_{i \in I} a_i$ fait évidemment l'affaire.

6. Montrez que 2dim-SubsetSum est NP-complet.

Solution:
Ici aussi le problème est clairement NP-difficile car il généralise SubsetSum (p.ex. une réduction montrant SubsetSum $\leq$ 2dim-SubsetSum va transformer chaque entier a_i ainsi que g en un vecteur en lui ajoutant une 2ème coordonnée valant 0). On voit que 2dim-SubsetSum est dans NP car deviner I et le valider se fait toujours en temps polynomial même pour des vecteurs.

7. Montrez que Half-SubsetSum est NP-complet.

Solution:

Ici aussi le problème est dans NP (on devine I et on le valide).

Pour montrer qu'il est NP-difficile on exhibe une réduction SubsetSum $\leq$ Half-SubsetSum. P.ex. une réduction possible prend une instance $(a_1, \ldots, a_m), g$ de SubsetSum et lui associe $(a_1, \ldots, a_m, S+g, 2S-g)$ avec $S = \sum_{i=1}^{m} a_i$ (on note que la somme des nombres générés est $4S$, la réduction est clairement correcte).

Il reste à vérifier que la réduction est bien logspace. Il faut ici calculer $S+g$ et $2S-g$. Avec un espace logarithmique, on peut parcourir les a_i, additionner les p-ièmes chiffres (partant de la droite) pour $p = 1, 2, ..$, mémoriser une retenue, mais on ne peut pas mémoriser l'intégralité de $S+g$ ou $2S-g$. On les construit donc un chiffre à la fois, en commençant par la droite (c.-à-d. par les chiffres les moins significatifs). Le plus simple est alors de les écrire à l'envers et de composer avec une autre réduction logspace qui remet à l'endroit les deux derniers nombres.

8. Montrez que 2SubsetSum est NP-complet.

Solution:

Ici aussi le problème est dans NP (on devine I_1, I_2 et on les valide). Il est NP-difficile car on montre Half-SubsetSum $\leq$ 2SubsetSum avec une réduction qui vérifie que $S = \sum_{i=1}^{m} a_i$ est pair et qui définit alors $g_1, g_2 \stackrel{\text{def}}{=} \frac{S}{2}$.

9. Montrez que Multi-SubsetSum est NP-complet.

Solution:

On note tout d'abord que ce problème est dans NP car il s'agit d'un cas particulier du problème CheminPondere vu en cours.

NP-difficulté : on donne une réduction montrant SubsetSum $\leq$ Multi-SubsetSum. Soit $(a_1, \ldots, a_m), g$ une instance de SubsetSum. On définit les entiers suivants : $S \stackrel{\text{def}}{=} \sum_{i=1}^{m} a_i$ comme plus haut, L est un entier tel que $2^L > \max(g, S, 2^m)$ et $M = 2L + m$. La réduction produit alors l'instance de Multi-SubsetSum

$$r\big((a_1, \ldots, a_n), g\big) = (A_{1,0}, A_{1,1}, A_{2,0}, A_{2,1}, \ldots, A_{m,0}, A_{m,1}), G$$

où $G \stackrel{\text{def}}{=} m2^M + 2^{L+m+1} - 2^{L+1} + g = m2^M + 2^{L+1} + 2^{L+2} + \cdots + 2^{L+m} + g$ et où, pour chaque i, $A_{i,j} \stackrel{\text{def}}{=} j \times a_i + 2^{L+i} + 2^M$. Notons que $G < (m+1)2^M$ car $L + m + 1 \leq M$ et car $g < 2^L$.

Correction de la réduction : — Si $g = \sum_{i \in I} a_i$ alors $G = \sum_{i \in I} A_{i,1} + \sum_{i \notin I} A_{i,0}$. Donc $x \in$ SubsetSum $\implies r(x) \in$ Multi-SubsetSum.

— Supposons $G = \sum_{k=1}^{\ell} A_{i_k, j_k}$. Puisque $m2^M \leq G < (m+1)2^M$ et puisque $A_{i,j} > 2^M$ pour tous i, j, on a $\ell \leq m$ (c.-à-d. : la somme utilise au plus m nombres). Par ailleurs $A_{i,0} \leq A_{i,1} < 2^M + 2^{L+m+1}$ donc il faut aussi $m \leq \ell$ (car $m2^{L+m+1} < 2^M$).

Maintenant qu'on a prouvé $\ell = m$, on voit qu'il faut prendre pour chaque $i = 1, \ldots, m$, un et un seul parmi $A_{i,0}$ et $A_{i,1}$, seule façon de réaliser la composante $2^{L+1} + 2^{L+2} + \cdots + 2^{L+m}$ dans G. On peut donc écrire $G = \sum_{k=1}^{m} A_{k,b_k}$. On a alors $g = \sum_{i \in I} a_i$ en prenant $I = \{k \mid b_k = 1\}$. Donc $r(x) \in$ Multi-SubsetSum $\implies x \in$ SubsetSum et la réduction est correcte.

Complexité de la réduction : Elle est logspace car elle n'a besoin que de faire quelques sommes simples comme dans la question 7, ainsi que quelques comparaisons.

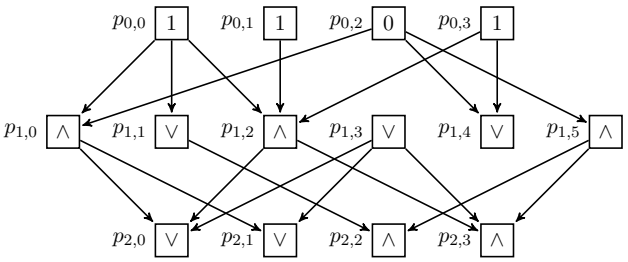
Exercice 3 : Autour de CircuitValue

On rappelle que CircuitValue est le problème de décider, étant donnés un circuit logique $\mathcal{C}$ et une de ses portes p_{goal}, si p_{goal} s'évalue à vrai (à 1) dans $\mathcal{C}$.

Dans cet exercice on s'intéresse à des circuits logiques d'une certaine forme.

Dans un circuit *stratifié*, les portes logiques sont données niveau par niveau et les entrées d'une porte au niveau ℓ sont nécessairement au niveau $\ell - 1$.

La figure ci dessous donne un exemple de circuit stratifié où chaque porte est soit une disjonction ($\vee$) soit une conjonction ($\wedge$). Les portes $p_{0,i}$ du premier niveau n'ont pas d'entrée et sont donc des disjonctions vides (représentées par 0) ou des conjonctions vides (1). On aurait ainsi pu représenter la porte $p_{1,3}$ au moyen d'une étiquette 0 à la place d'un $\vee$.



Le problème $\mathsf{Alt}_2\mathsf{CircuitValue}[\vee, \wedge]$ est la version de CircuitValue restreinte à des circuits stratifiés *d'alternation au plus 2*, c.-à-d. constitués d'un premier niveau avec des constantes 0 ou 1, puis d'un nombre arbitraire $k \geq 0$ de niveaux ne contenant que des $\vee$, puis enfin de $k' \geq 0$ niveaux ne contenant que des $\wedge$. On n'a donc pas de porte $\vee$ sous une porte $\wedge$ (car on ne compte pas les 1 du premier niveau comme des $\wedge$).

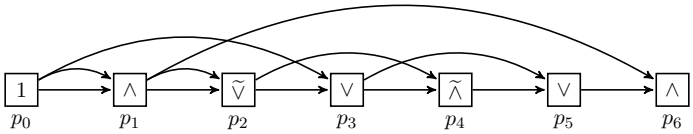
10. Montrez que $\mathsf{Alt}_2\mathsf{CircuitValue}[\vee, \wedge]$ est NL-complet.

Solution:

Le problème est NL-difficile car on montre facilement $\mathsf{GAP} \leq \mathsf{Alt}_2\mathsf{CircuitValue}[\vee, \wedge]$. P.ex. à une instance G, s, t de GAP où le graphe orienté $G = (V, E)$ a $n \stackrel{\text{def}}{=} |V|$ sommets, on associe un circuit $\mathcal{C}$ qui contient n niveaux de n portes. La porte $p_{i,u}$ au niveau $i > 0$ est une porte $\vee$ avec comme entrées tous les $p_{i-1,v}$ tels que (v, u) est une arête de E, ainsi que $p_{i-1,u}$. On a donc un "chemin" de $p_{j,v}$ à $p_{i,u}$ dans $\mathcal{C}$ ssi $j \leq i$ et G a un chemin $v \stackrel{*}{\to} u$ de longueur au plus $i - j$. On fixe alors $p_{0,u} = 1$ si $u = s$, $= 0$ sinon, et on demande d'évaluer la porte $p_{n,t}$. On note que ce circuit n'utilise aucune porte $\wedge$ et est donc d'alternation 1.

Il reste à montrer que le problème est dans NL. On va utiliser le fait que GAP et coGAP sont dans NL. Sans perte de généralité, on considère une instance $(\mathcal{C}, p_{goal})$ où p_{goal} se trouve au dernier niveau (on peut supprimer les niveaux inutiles par une transformation logspace) ; où il y a au plus une entrée valant 1 (on peut fusionner les entrées identiques par une transformation logspace), et où k et k' sont non nuls (si nécessaire on peut insérer un niveau neutre composé de $\wedge$ ou de $\vee$ par une transformation logspace). Avec ces hypothèses p_{goal} s'évalue à 0 ssi il existe une porte p au dernier niveau des portes $\vee$ telle qu'on ne puisse pas atteindre p depuis l'entrée 1 et telle qu'on puisse atteindre p_{goal} depuis p (preuve omise). Un algorithme non déterministe pour décider $\mathsf{coAlt}_2\mathsf{CircuitValue}$ devine p, puis enchaîne la vérification que $p \stackrel{*}{\to} p_{goal}$ (avec l'algorithme de GAP) puis la vérification que $1 \not\stackrel{*}{\to} p$ (avec l'algorithme de coGAP). Finalement, puisque $\mathsf{coAlt}_2\mathsf{CircuitValue}$ est dans NL, et puisque $\mathsf{NL} = \mathsf{coNL}$, on obtient bien $\mathsf{Alt}_2\mathsf{CircuitValue} \in \mathsf{NL}$.

Seq-CircuitValue est une version « séquentielle » de CircuitValue où les portes logiques sont alignés séquentiellement et où chaque porte (sauf la première) a exactement deux entrées dont nécessairement la porte immédiatement précédente. Voici un exemple où on utilise les opérateurs $\widetilde{\wedge}$ et $\widetilde{\vee}$ (NAND et NOR) en sus des classiques $\wedge$ et $\vee$. La porte d'entrée est une constante, 0 ou 1.



11. Montrez que Seq-CircuitValue$[\wedge, \vee, \widetilde{\wedge}, \widetilde{\vee}]$ est PTIME-complet.

Solution:
Tout comme CircuitValue, le problème est évidemment dans PTIME. On montre qu'il est PTIME-difficile dans la question suivante.

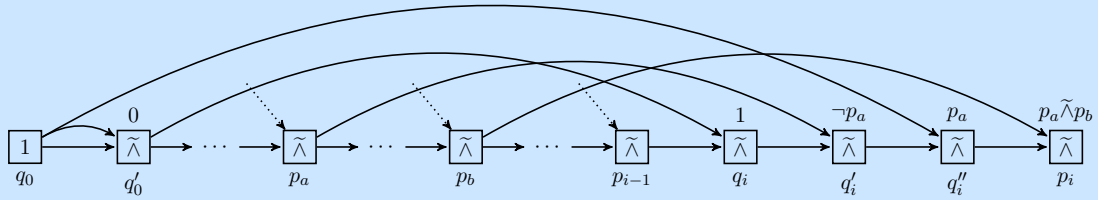
12. Montrez que la version Seq-CircuitValue$[\widetilde{\wedge}]$ où toutes les portes sauf la première portent un $\widetilde{\wedge}$ est encore PTIME-complet.

Solution:
Seq-CircuitValue$[\widetilde{\wedge}]$ est une restriction du problème précédent, donc il est dans PTIME.

Pour montrer qu'il est PTIME-difficile, on réduit CircuitValue$[\widetilde{\wedge}]$ à Seq-CircuitValue$[\widetilde{\wedge}]$. Sans perte de généralité, on suppose qu'une porte d'un circuit de CircuitValue$[\widetilde{\wedge}]$ a toujours *exactement deux arcs entrants*, sauf pour les portes d'entrée qui sont des constantes 0 ou 1.

La réduction $\mathcal{C} \mapsto \mathcal{C}'$ produit un circuit séquentiel commençant par $\mathcal{C}'_0$ qui ne contient qu'une entrée 1 suivie d'une porte $\widetilde{\wedge}$ (les portes q_0 et q'_0). On énumère alors les portes de $\mathcal{C}$ dans un ordre topologique $p_1, p_2, \ldots, p_k$ (c.-à-d., les entrées d'une porte avant la porte elle-même) et, pour chaque porte p_i de $\mathcal{C}$, on prolonge $\mathcal{C}'_{i-1}$ avec de nouvelles portes dont la dernière simule p_i, obtenant ainsi $\mathcal{C}_i$.

De façon générale, si p_i est définie dans $\mathcal{C}$ comme le NAND de p_a et p_b avec $a \leq b < i$, on construit jusqu'à p_{i-1} puis on ajoute quatre portes q_i, q'_i, q''_i et p_i comme indiqué sur la figure.



La contrainte de séquentialité est bien respectée. Par ailleurs, quelle que soit la valeur de p_{i-1} dans $\mathcal{C}'$, q_i s'évaluera à 1, q'_i à la valeur de $1 \widetilde{\wedge} p_a$, soit $\neg p_a$, q''_i à celle de p_a, puis enfin p_i à celle de $p_a \widetilde{\wedge} p_b$ comme voulu. (Le cas où p_i est une porte constante est plus simple à traiter. P.ex. utiliser directement q_i pour simuler p_i convient si, dans $\mathcal{C}$, p_i est la constante 1.)