Complexité - TD 05

Benjamin Bordais

17 Décembre 2020

Exercise 1 Espace Poly-Logarithmique

On rappelle le théorème de hiérarchie en espace :

Théorème 1 *Pour deux fonctions constructibles $f, g \geq \log$ telles que $f(n) = o(g(n))$, on a $\mathsf{DSPACE}(f) \subsetneq \mathsf{DSPACE}(g)$.*

On définit à présent la classe de complexité :

$$\mathsf{PolyLog} = \bigcup_{k>0} \mathsf{SPACE}(\log^k(n))$$

.

1. Montrer que PolyLog n'a pas de problème complet pour des réductions en espace logarithmique. Que peut-on déduire quant à la comparaison entre les classes P et PolyLog ?
2. On rappelle que $\mathsf{PSPACE} = \bigcup_{k>0} \mathsf{SPACE}(n^k)$. Est-ce que PSPACE a un problème complet pour des réductions en espace logarithmique ? Pourquoi est-ce que la preuve de la question précédente ne s'applique pas à PSPACE ?

Solution :

1. Supposons par l'absurde qu'il existe un problème PolyLog-complet L pour des réductions en espace logarithmique. Alors, comme $L \in \mathsf{PolyLog}$, il existe $k \in \mathbb{N}$ tel que $L \in \mathsf{SPACE}(\log^k)$. Montrons alors que $\mathsf{SPACE}(\log^k) = \mathsf{SPACE}(\log^{k+1})$, ce qui est une contradiction avec le théorème de hiérarchie en espace. Soit $L' \in \mathsf{SPACE}(\log^{k+1}) \subseteq \mathsf{PolyLog}$. Il existe une réduction f de L' vers L qui peut être calculée en espace logarithmique puisque L est PolyLog-complet. Considérons à présent une machine de Turing qui, sur une entrée w, calcule $f(w)$ en espace logarithmique et qui simule ensuite une machine de Turing décidant L en espace $\log^k$ sur $f(w)$. Il faut remarquer ici qu'il est important de ne pas stocker $f(w)$ sur un ruban de travail (sinon, l'espace utilisé pourrait excéder la borne $\log^k$). À la place, on peut utiliser une bande virtuelle où l'on calcule les bits de $f(w)$ uniquement lorsque l'on en a besoin sans avoir à se souvenir du calcul dans son intégralité (d'une manière analogue à ce que l'on peut faire pour montrer que la relation "peut être réduit en espace logarithmique" est transitive). On a $|f(w)| = O(|w|^c)$ pour un $c \geq 0$ (qui ne dépend pas de w). Ainsi, l'espace utilisé pour vérifier si $f(w)$ est dans L est plus petit que $\log^k(|f(w)|)$, et donc plus petit que $c^k \cdot \log^k(O(|w|)) = O(\log^k(|w|))$. On conclut avec le théorème d'accélération en espace (théorème 2 du polycopié) pour avoir $L' \in \mathsf{SPACE}(\log^k)$. On obtient $\mathsf{SPACE}(\log^k) = \mathsf{SPACE}(\log^{k+1})$, d'où la contradiction. Ainsi, un tel L ne peut pas exister.

2. PSPACE possède des problème complets pour des réductions en espace logarithmique (tel que TQBF). Cependant, si l'on veut appliquer la preuve précédente pour montrer que $\mathsf{SPACE}(n^k) = \mathsf{SPACE}(n^{k+1})$, on a un problème : comme $|f(w)|$ est en $O(|w|^c)$, on a $|f(w)|^k$ en $O(|w|^{c \cdot k}) \neq O(|w|^k)$ si $c > 1$.

Exercise 2 Fonction de choix

Un langage L appartient à P-*choice*, écrit $L \in \mathsf{P}_c$, s'il existe une fonction $f : \Sigma^* \times \Sigma^* \to \Sigma^*$, calculable en temps polynomial telle que pour tout $x, y \in \Sigma^*$:

— $f(x, y) \in \{x, y\}$,

— si $x \in L$ ou $y \in L$, alors $f(x, y) \in L$.

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

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

Solution :

1. Considérons $L \in \mathsf{P}$. On construit f telle que, sur une entrée $w = (x, y)$, f vérifie en temps polynomial que $x \in L$. Dans ce cas, f renvoie x, sinon elle renvoie y. On a alors f fonction de choix pour L et $L \in \mathsf{P}_c$.
2. Considérons $L \in \mathsf{P}_c$ et sa fonction de choix f. Considérons f' telle que $f'(x, y) = x$ si $f(x, y) = y$ et y sinon. On peut alors vérifier que f' est une fonction de choix pour le complémentaire $\bar{L}$ de L.
3. Soit une réduction en temps polynomial tr de SAT vers $L \in \mathsf{P}_c$ avec la fonction de choix f. On met en place l'algorithme suivant qui fonctionne en temps polynomial et décide SAT :

```
algo_SAT(φ):
        if φ = True:
        then accept;
        elif φ = False:
        then reject;
        else
        let x ∈ Var(φ);
        if f(tr(φ[x ← True]), tr(φ[x ← False])) = tr(φ[x ← True]):
        then algo_SAT(φ[x ← True]);
        else algo_SAT(φ[x ← False])
```

   Calculer f et tr peut se faire en temps polynomial et il y a $|Var(\varphi)|$ appels récursifs où $Var(\varphi)$ est l'ensemble des variables apparaissant dans la formule φ. Ainsi, l'algorithme fonctionne en temps polynomial. La correction de l'algorithme vient de la définition d'une fonction de choix.

Exercise 3 Clôture par morphisme

Étant donné un alphabet fini Σ, une fonction $f : \Sigma^* \to \Sigma^*$ est un morphisme si $f(\Sigma) \subseteq \Sigma$ et pour tout $a = a_1 \cdots a_n \in \Sigma^*$, $f(a) = f(a_1) \cdots f(a_n)$ (f est entièrement déterminée par les valeurs prises sur Σ).

Montrer que $\mathsf{P} = \mathsf{NP}$ si et seulement si P est clos par morphisme (i.e. si $L \in \mathsf{P}$, alors $f(L) \in \mathsf{P}$ pour tout morphisme f).

Solution :

— Supposons que $\mathsf{P} = \mathsf{NP}$. Considérons f un morphisme et $L \in \mathsf{P} = \mathsf{NP}$. Montrons que $f(L) \in \mathsf{NP}$. On considère une machine de Turing non-déterministe M qui, sur une entrée $w \in \Sigma^*$, devine un mot $a \in \Sigma^*$ tel que $|a| = |w|$ et vérifie ensuite que $f(a) = w$ et que $a \in L$ en temps polynomial. Il vient alors que $f(L) \in \mathsf{NP} = \mathsf{P}$ et P est clos par morphisme.

— Supposons à présent que P est clos par morphisme. On montre que $\mathsf{SAT} \in \mathsf{P}$, ce qui prouve que $\mathsf{NP} \subseteq \mathsf{P}$ puisque SAT est NP-complet. Considérons le langage suivant :

$$L = \{(\phi, v) \mid v \text{ est une valuation satsfaisant } \phi\}$$

On a que $L \in \mathsf{P}$ car on peut vérifier en temps polynomial qu'une valuation satisfait une formule. De plus, on peut supposer que l'alphabet Σ est égal à l'union disjointe $\Sigma_\phi \uplus \Sigma_v$ et que symboles utilisés pour encoder ϕ (resp. v) sont dans Σ_ϕ (resp. Σ_v). Ainsi, si l'on considère le morphisme f tel que $f(a) = a$ pour tout $a \in \Sigma_\phi$ et $f(a) = 0$ pour tout $a \in \Sigma_v$, on obtient :

$$f(L) = \{(\phi, 0^n) \mid \phi \text{ a n variables et est satisfiable}\}$$

Par clôture par morphisme, il vient que $f(L) \in \mathsf{P}$. Puisque qu'une instance de SAT peut se réduire en temps polynomial (et même en espace logarithmique) à une instance de $f(L)$, il vient que $\mathsf{SAT} \in \mathsf{P}$. Ainsi, $\mathsf{P} = \mathsf{NP}$.

Exercise 4 Théorème de Ladner

On souhaite montrer que si $\mathsf{P} \neq \mathsf{NP}$ alors il existe $L \in \mathsf{NP} \backslash \mathsf{P}$ tel que L ne soit pas NP-complet.

On considère un codage des machines de Turing dans les entiers tel que à toute machine de Turing correspondent un nombre infini d'entiers, on notera M_i la machine de Turing codé par l'entier i. On définie récursivement la fonction $H : \mathbb{N} \mapsto \mathbb{N}$ et l'ensemble SAT_H comme suit :

$$\mathsf{SAT}_H = \{\psi.0.1^{n^{H(n)}} \mid \psi \in \mathsf{SAT} \wedge n = |\psi|\}$$

$$H(n) = \min \begin{cases} \log(\log(n)) \\ \min\limits_{i<\log(\log(n))} \{i \mid M_i \text{ decide } \mathsf{SAT}_H \text{ en temps } i \cdot |x|^i \text{ pour tout } |x| < \log(\log(n))\} \end{cases}$$

La définition ci-dessus n'est pas valide en 0 et en 1, on prendra donc $H(0) = H(1) = 0$. On va maintenant montrer les résultats suivants :

1. Montrez que H et SAT_H sont bien définies, que H est croissante et qu'elle se calcule en temps polynomial.
2. Montrez que si SAT_H est dans P alors H est bornée.
3. Réciproquement montrez que si H est bornée alors SAT_H est dans P.

Établisser à présent les propriétés suivantes :

(i) $\mathsf{SAT}_H \in \mathsf{NP}$;

(ii) $\mathsf{SAT}_H \notin \mathsf{P}$;

(iii) SAT_H n'est pas NP-complet.

Solution :

1. Soit $m, n \in \mathbb{N}$. Pour une formule $\psi \in \mathsf{SAT}$ telle que $|\psi| = n$, on a $|\psi.0.1^{n^{H(n)}}| < \log(\log(m))$ si $n+1+n^{H(n)} < \log(\log(m))$, il faut donc $n < \log(\log(m)) \leq m$. Ainsi, la définition de $H(m)$ utilise les définitions de $H(n)$ pour $n < m$. Par induction, $H(m)$ est donc bien définie.

   Soit $i \in \mathbb{N}$ et $i < H(n) \leq \log(\log(n)) \leq \log(\log(n+1))$. Par définition, M_i ne décide pas SAT_H en temps $i.x^i$ pour les mots de tailles $< \log(\log(n))$. Donc M_i ne décide pas SAT_H en temps $i.x^i$ pour les mots de tailles $< \log(\log(n+1))$, ce qui implique que $H(n+1) \neq i$, et donc que $H(n) \leq H(n+1)$.

   Montrons maintenant que $H(n)$ se calcule en temps polynomial. On donne deux algorithme mutuellement récursifs pour calculer H et décider SAT_H.

```
M_H(n) :=
  forall i < log(log(n)) do
    forall x s.t. |x| < log(log(n)) do
    b ← true;
    Simule M_i sur x pendant i·|x|^i etapes:
      If (Calcul non fini):
        b ← false;
      Elif (Res ≠ M_SAT_H(x)):
        b ← false;
    done;
    if b then (return i);
  done;
  return log(log(n))
```

   L'algorithme pour décider SAT_H.

```
M_SAT_H(x) :=
  ψ.0.1^k ← x;
  if k = |ψ|^{M_H(|ψ|)} then
  return SAT(ψ);
  else return false;
```

   Posons $T(n)$ le temps de calcul de $M_H(n)$ et $T'(n)$ le temps de calcul de $M_{\mathsf{SAT}_H}(n)$. On a $\log(\log(n))$ tours dans la boucle extérieure, et $2^{\log(\log(n))} = \log(n)$ dans la boucle intérieure de $M_H(n)$. La simulation de M_i sur x se fait en temps $i \cdot |x|^i \leq \log(\log(n)) \cdot (\log(\log(n)))^{\log(\log(n))}$. Pour ce qui est de l'algorithme pour SAT_H, on doit utiliser un temps exponentiel (disons 2^{n^c} pour un c donné) pour décider si $\psi \in \mathsf{SAT}$. On obtient les bornes suivantes :

$$T(n) \leq \log(\log(n)) \cdot \log(n) \cdot \Big(\log(\log(n)) \cdot (\log(\log(n)))^{\log(\log(n))} + T'(\log(\log(n)))\Big)$$
$$T'(n) \leq n + T(n) + 2^{n^c}$$

   Asymptotiquement on a donc :

$$T(n) \leq n + \log(n) \cdot \log(\log(n)) \cdot T'(\log(\log(n)))$$
$$T'(n) \leq n + 2^{n^c} + T(n)$$

Donc pour $n > N$ pour un certain $N \in \mathbb{N}$, on a :

$$\begin{aligned} T(n) &\leq n + \log(n) \cdot \log(\log(n)) \cdot \Big(\log(\log(n)) + 2^{\log(\log(n))^c} + T(\log(\log(n)))\Big) \\ &\leq n + \log(n) \cdot \log(\log(n)) \cdot (n + T(\log(\log(n)))) \\ &\leq n^2 + \log^2(n) \cdot T(\log(\log(n))) \end{aligned}$$

On en conclut que $T(n)$ est polynomial.

2. Supposons que SAT_H soit dans P. Soit M une machine de Turing décidant SAT_H en temps $C \cdot n^k$, et soit $j > \max(k, C)$ tel que $M_j = M$ (on rappelle qu'un nombre infini d'entiers correspondent à la machine M, par hypothèse).

   Soit alors n tel que $j < \log(\log(n))$. On sait que M_j décide SAT_H en temps au plus $j \cdot |x|^j$ (car $C \cdot |x|^k \leq j \cdot |x|^j$) pour tout x de taille $< \log(\log(n))$. Il vient que $H(n) \leq j$, et donc H est bornée.

3. Supposons que H soit bornée. Comme H est croissante on sait que H est stationnaire, soit j sa valeur limite. Ainsi, pour tout n tel que $j < \log(\log(n))$, on a que M_j décide SAT_H en temps $j \cdot |x|^j$ pour tout x de taille $< \log(\log(n))$. Par conséquent M_j décide SAT_H en temps $j \cdot |x|^j$ pour tout x. C'est-à-dire : $\mathsf{SAT}_H \in \mathsf{P}$.

(i) Vérifier qu'un mot x est dans SAT_H peut se faire en temps non-déterministe polynomial : on considère l'expression $\phi = \psi.0.1^k$ telle que $\phi = x$ (si cela n'est pas possible, on rejette immédiatement), on vérifie ensuite en temps polynomial que $k = |\psi|^{H(|\psi|)}$ (ce qui bien possible étant donné que l'on peut calculer $H(|\psi|)$ en temps polynomial et $H(|\psi|) \leq \log(\log(|\psi|))$). Il suffit finalement de vérifier la satisfiabilité de ψ en temps polynomial non déterministe..

(ii) Supposons par l'absurde que SAT_H soit dans P, notons M_S une machine de Turing décidant SAT_H en temps polynomial. Par la question 2, on a que H est bornée par un certain $j \in \mathbb{N}$. On peut alors décider SAT en temps polynomial : sur une entrée ψ telle que $n = |\psi|$, on calcule $H(\psi)$ (en temps polynomial en n) et $\psi.0.1^{n^{H(\psi)}}$ (en temps $O(n^j)$, donc polynomial), puis on simule M_S sur $\psi.0.1^{n^{H(\psi)}}$.

   Cet algorithme décide SAT en temps polynomiale, ce qui contredit l'hypothèse $\mathsf{P} \neq \mathsf{NP}$.

(iii) Supposons à présent que SAT_H soit NP-complet. Nous allons alors montrer que SAT peut se résoudre en temps polynomial (ce qui contredit encore l'hypothèse $\mathsf{P} \neq \mathsf{NP}$).

   Notons f une réduction de SAT vers SAT_H en temps asymptotiquement borné par n^c (à partir d'un $n = N$). Comme on suppose $\mathsf{P} \neq \mathsf{NP}$, on sait que $\mathsf{SAT}_H \notin \mathsf{P}$, et donc H n'est pas bornée. Soit donc n_0 tel que $\forall n \geq n_0, H(n) \geq 2 \cdot c$. Notons M^{exp}_{SAT} un algorithme déterministe arbitraire décidant SAT.

```
1  M'_SAT(φ) :=
2          ψ.0.1^k ← f(φ);
3          if k ≠ |ψ|^H(|ψ|)
4          then return false;
5          elif |ψ| ≤ n_0
6          then return M^exp_SAT(ψ);
7          else return M'_SAT(ψ);
```

   La fonction M^{exp}_{SAT} n'est appelé que sur des instances de tailles $\leq n_0$, et donc se réalise en temps constant. Si la fonction M'_{SAT} est appelé récursivement alors on

sait que $|\psi| > n_0$, et donc que $H(|\psi|) \geq 2 \cdot c$. De plus on sait que $k = |\psi|^{H(|\psi|)}$. Si $n \geq N$ on sait que $f(x)$ est calculé en temps au plus n^c, donc $|\psi.0.1^k| \leq n^c$ et ainsi $|\psi| + 1 + |\psi|^{H(|\psi|)} \leq n^c$. On en déduit que pour tout $n \geq N$:

$$|\psi| \leq n^{c/H(|\psi|)} \leq n^{c/(2.c)} = n^{1/2} = \sqrt{n}$$

Notons $T(n)$ la complexité temporelle de M'_{SAT}. On a alors que pour tout $n > N$ et pour deux constante r, t donnée :

$$T(n) \leq n^t + T(\lfloor \sqrt{n} \rfloor) + r$$

On en déduit que $T(n)$ est polynomial. En effet, pour $m \in \mathbb{N}$ tel que $m^t \geq 2\sqrt{m}^t + r$ et $u = \max_{n \leq \max(N,m)}\{T(n)\}$, on peut montrer par récurrence que, pour tout $n \geq 0$, on a $T(n) \leq 2 \cdot n^t + u + r$.