Hiérarchie polynomiale

Brault Vincent

19 décembre 2008

1 Rappel

Nous supposons connu le cours de langage formel de Monsieur Carton ainsi que tout ce qui a été vu dans les TD de Monsieur Fouque.
On rappelle que l'ensemble des machines alternantes décidable en temps polynomial est APSPACE et qu'il est égal à l'ensemble des machines décidables en espace polynomial c'est à dire PSPACE.
Ce que nous allons essayer de voir, c'est ce que l'on peut trouver entre les ensembles NP$\bigcup$co-NP et PSAPCE

2 Prérequis sur les oracles

Définition 1. Soit A un langage
On appelle machine de Turing avec oracle A une machine de Turing avec une bande spéciale appelé bande oracle
Cette bande est munie de trois états particuliers $q_?, q_o$ et q_n. La machine peut écrire un mot sur cette bande puis entrer dans l'état $q_?$, cette bande agit comme une machine de Turing qui dit ou non si le mot appartient au langage A sauf qu'elle le dit instantanément et renvoie suivant la réponse l'état q_o pour oui ou q_n pour non (en anglais, on a les états q_y et q_n)

Définition 2. Soit A une classe de complexité et B un langage, on note A^B la classe des langages reconnus par une machine de Turing de classe A avec un oracle de langage B
Soit A et B deux classes de complexité, on note A^B la classe des langages reconnus par une machine de Turing de classe A avec un oracle appartenant à la classe B, c'est à dire que $A^B = \bigcup_{L \in B} A^L$

Proposition 3. *Soit A et C une classe de langage et B un langage complet dans la classe C alors* $A^B = A^C$

Démonstration. Comme le langage B est dans C alors on a trivialement $A^B \subseteq A^C$
L'autre inclusion vient du fait que B est complet dans C et donc que tout langage de C peut se réduire à B. □

Proposition 4. *On a NP⊆NP^{NP}*

Démonstration. On prend une machine non déterministe qui décide en temps polynomial
Alors on peut rajouter n'importe quel oracle, puisque on n'est pas obligé de l'utiliser
D'où NP⊆NP^{NP}
□

Proposition 5. *Il existe un langage A tel que $P^A = NP^A$*

Démonstration. Pour démontrer l'égalité, nous prenons A un langage PSPACE-complet ; nous allons démontrer les inclusions suivantes :
$PSPACE \subseteq P^A \subseteq NP^A \subseteq NPSPACE \subseteq PSPACE$
Pour la première inclusion, on prend L un langage de PSPACE. Comme A est PSPACE-complet, on choisit la machine de Turing qui réduit le langage L à celui de A et on lui ajoute l'oracle A, ce qui nous donne ce que l'on souhaite
La deuxième inclusion vient juste du fait que $P \subseteq NP$.
Pour la troisième inclusion, soit M une machine non déterministe en espace polynomial qui résout le problème de décidabilité de A. Alors une machine non déterministe en temps polynomial avec l'oracle A peut être simulée en espace polynomial par M.
Enfin, la dernière inclusion est obtenue par le théorème de Savitch qui dit que $PSPACE = NPSPACE$
D'où $P^A = NP^A$

□

Proposition 6. *Il existe un langage B tel que $P^B \neq NP^B$*

Démonstration. Nous recherchons donc un oracle B qui puisse montrer la puissance des machines non déterministes sur les machines déterministes
L'idée est en fait de commencer par trouver le langage L tel que L∈ $NP^B - P^B$
Soit L :=$\{O^n | x \in B$tel que$|x| = n\}$
Il apparaît évident que L∈ NP^B et cela pour tout oracle B
Le reste de la preuve consiste à construire l'oracle B tel que L $\notin P^B$
Pour cela, on commence par énumérer toutes les machines qui ont une bande oracle (symbolisé par le ?) en faisant en sorte qu'elles apparaisent une infinité de fois, on obtient ainsi la liste suivante : $M_1^?, M_2^?, M_3^?$...
Nous allons construire B par diagonalisation, ce sera la réunion d'une suite croissante $(B_n)_{n\in\mathbb{N}}$
On introduit également un ensemble X qui répresente les mots interdits
A la phase initiale, B_0 et X sont tous les deux vides

Pour faire la récurrence à la i^{eme} étape, on suppose que l'on a B_{i-1} et X
On lance le programme $M_i^B(O^i)$ pour $i^{log(i)}$ pas
A chaque fois, on questionne la machine : x $\in B$?
Cette question se répond de la façon suivante :
Si $|x| <$ i on regarde si x $\in B_{i-1}$
Si oui, on accepte
Si non, on continue
Si $|x| \geq$ i, c'est que M_{i-1}^B a tout le temps refusé donc on rejette et on ajoute x dans l'ensemble X
Si après $i^{log(i)}$ pas ou moins on rejette, il faut empêcher M_i^B d'accepter (O^i) et dans cette perspective, on définit $B_i = B_{i-1} \bigcup \{x \in A^* || x| = i \text{ et } x \notin X\}$
Donc $L(M_i^B) \neq L$ sous condition que l'ensemble que l'on vient de définir est non vide
C'est pour cela que nous avons demander de ne faire que $i^{log(i)}$ pas car dans le pire cas (celui qui consisterait à ce qu'à chaque étape, on rajoute un mot dans X), on a $|X| \leq \sum_{j=1}^{i} j^{log(j)} = \sum_{j=1}^{i} e^{(log(j))^2} < 2^i \leq |A^*|^i$ en supposant que l'alphabet A n'est pas unaire (ce qui est une supposition correcte)
Si M_i^B accepte O^i, on pose $B_i = B_{i-1}$ et ainsi, $L(M_i^B) \neq L$
Si M_i^B échoue pour s'arrêter après $i^{log(i)}$ pas, il est possible que le polynôme P est alors trop grand pour la valeur i. Comme on a construit la suite de telle sorte que les machines reviennent une infinité de fois, on pose $B_i = B_{i-1}$ pour que ce cas soit évalué plus tard
Au final, on prend B=$\cup_{i \in \mathbb{N}} B_i$ et notre construction permet le résultat

□

3 Définition par les machines alternantes

Dans cette partie, nous allons commencé par définir la hiérarchie polynomiale à l'aide des quanteurs et des machines alternantes. Pour cela, nous allons introduire un certain nombre de notation en espérant que cela rendra au lecteur la notion plus compréhensible.

Définition 7. Soit p un polynôme et L un langage. Nous appelons témoin un symbole # hors de l'alphabet de départ A et notons $\exists^p L$ et $\forall^p L$ les ensembles suivants :
$\exists^p L := \{x \in A^* | (\exists y \in A^* avec |y| \leqslant p(|x|)) \text{ et on a } x\#y \in L\}$
$\forall^p L := \{x \in A^* | (\forall y \in A^* avec |y| \leqslant p(|x|)) \text{ et on a } x\#y \in L\}$
De là, nous construisons les classes d'ensemble correspondant avec C une classe de langage :
$\exists^P C := \{\exists^p L | \text{p est un polynôme et } L \in C\}$
$\forall^P C := \{\forall^p L | \text{p est un polynôme et } L \in C\}$

A l'aide de ces ensembles, nous allons pouvoir donner une première définition des ensembles qui fondent la hiérarchie polynomiale.

Définition 8. Les classes de la hiérarchie polynomiale sont définies par récurrence :
$\sum_0^P := \prod_0^P :=$P
$\sum_{k+1}^P := \exists^P \prod_k^P$
$\prod_{k+1}^P := \forall^P \sum_k^P$
Ainsi, on définit la hiérarchie polynomiale par :
PH $:= \bigcup_{n\in\mathbb{N}} \sum_n^P$

Proposition 9. *Pour tout $k \in \mathbb{N}$ on a :*
$\sum_k^P =$*co-*$\prod_k^P$

Démonstration. Nous allons procéder par récurrence sur k $\in \mathbb{N}$
Etape 1 : Initialisation
On sait que P est stable par complémentaire
Donc $\sum_0^P$ = P = co-P = co-$\prod_0^P$
Etape 2 : Récurrence
On suppose qu'il existe k $\in \mathbb{N}$ tel que $\sum_k^P =$co-$\prod_k^P$, qu'en est-il de k+1 ?
$\sum_{k+1}^P = \exists^P \prod_k^P = \{\exists^p L |$p est un polynôme et $L \in \prod_k^P\}$
D'après l'hypothèse de récurrence, on a $\sum_k^P =$co-$\prod_k^P$
Soit L $\in \sum_{k+1}^P$, donc il existe un polynôme p et un langage L'$\in \prod_k^P$ tel que L=$\exists^p L'$
Le complémentaire de L est $\forall^p L''$ avec L'' $\in$co-$\prod_k^P = \sum_k^P$
Donc le complémentaire de L est dans $\prod_{k+1}^P$
D'où on obtient co-$\sum_{k+1}^P \subseteq \prod_{k+1}^P$
Ce qui donne, par symétrie, le résultat

□

Le lien entre cette définition de la hiérarchie polynômiale et les machines alternantes vient du fait que si on a $n \in \mathbb{N}$ alors $\sum_n^P = \exists^P \forall^P \exists^P ... Q^P P$ avec Q qui vaut $\exists$ si n est impair et $\forall$ si n est pair

4 Définition par les oracles

En feuilletant différents documents parlant de la hiérarchie polynomiale, je me suis aperçu que certains auteurs mettent le P en exposant ou à la suite (c'est à dire $\sum_k^P$ ou $\sum_k P$) suivant s'ils ont introduit les ensembles par les machines alternantes ou les oracles.
Pour ma part, je préfère garder la notation précédente mais je souhaitais avertir le lecteur pour qu'il ne soit pas surpris en lisant d'autres sources. Et si en écrivant, je prends l'autre notation, j'espère qu'il ne m'en tiendra pas rigueur

Définition 10. On définit les ensembles suivants :
$NP_1 := NP$
$\Delta_0^P := P$
Et pour tout $k \geq 0$ on a :
$NP_{k+1} := NP^{NP_k}$
$\Delta_{k+1}^P := P^{NP_k}$

Proposition 11. *La hiérarchie polynomiale peut être définie de la façon suivante :*
$\Delta_0^P = \sum_0^P = \prod_0^P = P$
Et pour tout $k \geq 0$ *on a :*
$\sum_{k+1}^P = NP^{\sum_k^P} = NP_{k+1}$
$\prod_{k+1}^P = co\text{-}NP^{\sum_k^P}$

Démonstration. Grâce à la proposition précédente, il est nécessaire de prouver uniquement que pour tout k ≥ 1, on a $\sum_k^P = NP_k$
Nous allons procéder par récurrence sur k. Le cas n=1 étant trivial on passe au cas k ≥ 1
Dans un premier temps, nous allons montrer que $\sum_k^P \subseteq NP_k$
Soit L un langage de $\sum_k^P$ alors il existe un polynôme p tel que l'on obtient par définition $L = \{x \in A^* | y \text{ avec } |y| \leq p(|x|) x \in \prod_{k-1}^P\}$
On choisit la machine de Turing M qui devine y et qui utilise un oracle de NP_{k-1} pour vérifier que y convient
Grâce à l'hypothèse de récurrence, on a $\sum_k^P \subseteq NP_k$
Réciproquement, on suppose que L $\in NP_k = NP^{NP_{k-1}}$
Donc on a une machine de Turing M^K qui reconnait L avec K un oracle de NP_{k-1}
Soit L', le langage reconnu par K
Par réccurence, on sait que L' $\in \sum_{k-1}^P$ donc il existe un polynôme q tel que $L' = \{y \in A^* | z \text{ avec } |z| \leq q(|y|) \text{ et } y\#z \in \prod_{k-2}^P\}$
On sait que x $\in$ L si et seulement s'il est accepté par M^K
Le certificat de x sera par conséquent la chaîne y enregistrée comme un calcul de la machine
A chaque fois que l'on utilise la machine, elle questionne l'oracle qui lui répond des fois oui et des fois non
Pour chaque 'oui', notre certificat y a son propre témoin z_i et polynôme q_i tel que $|z_i| \leq q_i(|y|)$
Comme on a une liste finie car notre machine décide en temps polynomial, on prend le polynôme p parmis les q_i tels que $p(|x\#y|) = max\{q_i(|x\#y|)\}$
Ainsi on a l'existence de p tel que pour tout z $\in A^*$ avec $|z| \leq p(|x\#y|)$ on ait $x\#y\#z \in \sum_{k-2}^P$
C'est à dire que y est tel que $x\#y \in \prod_{k-1}^P$
Ceci étant vrai pour tout x et que la longueur de y dépend de la machine, on

a l'existence d'un polynôme r tel que pour tout x ∈ L, il existe y ∈ A^* avec $|y| \leq r(|x|)$ et $x\#y \in \prod_{k-1}^P$
Autrement dit, L ∈ $\sum_k^P$
D'où, on conclut que $NP_k \subseteq \sum_k^P$

□

Le lien avec les oracles est assez apparent et on peut reformuler l'ensemble de la façon suivante : $\sum_k^P = NP^{NP^{\cdots NP}}$

Proposition 12. *Pour tout k ≥ 0, on a* Δ_k^P =*co-*Δ_k^P *c'est à dire que* Δ_k^P *est autodual*

Démonstration. Soit M une machine de Turing de $\Delta_k^P = P^{\sum_k^P}$, comme la machine est déterministe, il suffit juste d'inverser les états d'acceptation et de rejet pour obtenir une machine qui accepte exactement son complémentaire

□

5 Les inclusions dans la hiérarchie polynomiale

Nous savons déjà que $\sum_k^P$ =co-$\prod_k^P$. Nous allons montrer les inclusions au sein de la hiérarchie polynomiale.

Proposition 13. *Pour tout k ≥ 0, on a :* $\sum_k^P \bigcup \prod_k^P \subseteq \Delta_{k+1}^P$

Démonstration. Comme on sait que Δ_{k+1}^P est autodual, il suffit juste de montrer que $\sum_k^P \subseteq \Delta_{k+1}^P$ et on aura l'union par complémentarité.
Autrement dit il ne reste qu'à prouver que $NP_k \subseteq P^{NP_k}$
Si on prend une machine M dans NP_k, on doit juste prendre la machine qui possède M comme oracle et pour chaque entrée, on demande juste à l'oracle de répondre
D'où $\sum_k^P \subseteq \Delta_{k+1}^P$ ce qui donne le résultat

□

Proposition 14. *Pour tout k ≥ 0, on a :*$\Delta_k^P \subseteq \sum_k^P \bigcap \prod_k^P$

Démonstration. Commençons par démontrer que $\Delta_k^P \subseteq \sum_k^P$
Pour cela, on rappelle que $\Delta_k^P = P^{NP_k}$ et $\sum_k^P = NP^{NP_k}$
Donc, comme P ⊆ NP, on a la première inclusion
Avec le résultat précédent on a :
Δ_k^P =co-Δ_k^P ⊆co-$\sum_k^P = \prod_k^P$
D'où la deuxième inclusion
Ce qui donne $\Delta_k^P \subseteq \sum_k^P \bigcap \prod_k^P$

□

Proposition 15. *Pour tout $k \geq 0$, on a $\sum_k^P \subseteq PSPACE$*

Démonstration. Soit k $\geq$ 0. On a vu que, par définition, les machines qui reconnaissent les langages de $\sum_k^P$ sont les machines alternantes et comme nous l'avons rappelé dans la première partie, APSPACE=PSPACE
D'où $\sum_k^P \subseteq$ PSPACE

□

Corollaire 16. *$PH \subseteq PSPACE$*

Démonstration. On a pour tout k $\geq$ 0, $\sum_k^P \subseteq$ PSPACE
Comme la suite des ensembles $\sum_k^P$ est croissante, on a le résultat

□

Proposition 17. *S'il existe un $i \in \mathbb{N}^*$ tel que $\sum_i^P = \prod_i^P$*
Alors pour tout $j > i$, on a $\sum_j^P = \prod_j^P = \Delta_j^P = \sum_i^P$

Démonstration. Il suffit de montrer que $\sum_{i+1}^P = \sum_i^P$ car par l'une des propositions précédentes, on aura $\Delta_{i+1}^P = \sum_i^P$ et, par complémentarité, $\prod_{i+1}^P = \prod_i^P = \sum_i^P$
Le reste s'obtiendra alors par récurrence
On sait déjà que $\sum_i^P \subseteq \sum_{i+1}^P$
Soit L $\in \sum_{i+1}^P$ donc il existe un polynôme p tel que
L =$\{x \in A^* | y \in A^*$tel que$|y| \leq p(|x|)$et x#y $\in \prod_i^P\}$
Or $\sum_i^P = \prod_i^P$ donc L =$\{x \in A^* | y \in A^*$tel que$|y| \leq p(|x|)$et x#y $\in \sum_i^P\}$
Soit x $\in$ L alors il existe y $\in A^*$ tel que $|y| \leq p(|x|)$ et x#y $\in \sum_i^P$
Donc il existe un polynôme q et z $\in A^*$ tel que $|z| \leq q(|x\#y|)$ et x#y#z $\in \sum_{i-1}^P$
On a $|y\#z| \leq q(|x\#y|) + p(|x|) + 1$
Or $q(|x\#y|) = q(|x|) + q(1) + q(|y|) \leq q(|x|) + q(1) + M$
où $M = sup\{q(t) | t \in [|0; p(|x|)|]\}$
Donc on a montré qu'il existe un polynôme r=p+1+q+q(1)+M et un mot y#z tel que $|y\#z| \leq r(|x|)$ avec $x\#y\#z \in \sum_{i-1}^P$
Comme c'est vrai pour tout x $\in$ L, on a que L $\in \sum_i^P$
D'où $\sum_{i+1}^P \subseteq \sum_i^P$
Ce qui nous donne le résultat final

□

Corollaire 18. *Si P=NP ou NP=co-NP, la hiérarchie polynomiale est restreinte à NP*

Nous venons de le voir, la hiérarchie polynomiale n'a plus aucun sens si on découvre que P=NP
Toutefois, comme l'idée admise actuellement est que P$\neq$NP, on peut constater

que la hiérarchie polynomiale est un "filtre" qui permet de distinguer les différentes classes des machines de PSPACE

6 Problèmes complets et hiérarchie polynomiale

Si on suppose que la hiérarchie polynomiale ne s'écroule pas, il est intéressant de se poser la question de complétude

Définition 19. Soit i$\geq$ 1 et ϕ une expression booléenne avec des variables partitionnées en i sous variables $X_1, ..., X_i$
Le problème $QSAT_i$ consiste à répondre si "l'équation booléenne" $\exists X_1 \forall X_2 \exists X_3 ... Q X_i \phi$ où Q symbolise $\exists$ si i est impair et $\forall$ sinon est satisfiable

Proposition 20. *Pour tout i$\geq$1, le problème $QSAT_i$ est $\sum_i^P$-complet*

Démonstration. On a vu dans la section 3 que si $i \in \mathbb{N}$ alors $\sum_i^P = \exists^P \forall^P \exists^P ... Q^P P$ avec Q qui vaut $\exists$ si i est impair et $\forall$ si i est pair
Une autre façon de le formuler est que si L$\in \sum_i^P$ alors il existe une relation polynomiale R telle $L = \{x \in A^* | \exists y_1 \forall y_2 \exists y_3 ... Q y_i$ tels que $(x, y_1, y_2, ..., y_i) \in R\}$
Avec cette remarque, on voit que les problèmes $QSAT_i$ sont dans $\sum_i^P$
Regardons maintenant la complétude :
Soit L $\in \sum_i^P$, donc il existe une relation polynomiale R telle que l'on ait $L = \{x \in A^* | \exists y_1 \forall y_2 \exists y_3 ... Q y_i$ tels que $(x, y_1, y_2, ..., y_i) \in R\}$. Pour simplifier, supposons que i est impair, c'est à dire Q=$\exists$ (par dualité, on a l'autre résultat)
On a donc une machine de Turing M qui accepte précisément lorsque on lui donne en entrée $(x, y_1, y_2, ..., y_i)$
Avec une démonstration semblable à celle pour montrer que SAT est NP-complet, on peut écrire une formule booléenne ϕ qui répresente les calculs de la machine M. Ces variables peuvent être divisées en i+2 classes : la variable X représente le premier symbole que l'on met en entrée (celui de la liste $(x, y_1, y_2, ..., y_i)$). De même, les Y_i représentent les variables suivantes. Enfin, il nous faut une variable Z qui incorpore tous les autres aspects du calcul de la machine
Si on se donne des valeurs X,$Y_1, Y_2, ..., Y_i$ fixées, l'expression résultante est satisfiable si et seulement si les variables d'entrées représentent un mot accepté par la machine, c'est à dire que $(X, Y_1, Y_2, ..., Y_i) \in R$
Prenons une entrée x, on introduit sa valeur booléenne $\hat{\text{X}}$ dans la formule ϕ
x$\in L \Leftrightarrow \exists y_1 \forall y_2 \exists y_3 ... \exists y_i$ avec $(x, y_1, y_2, ..., y_i) \in R$
Ce qui se traduit en terme booléen par : pour toute valeur $\hat{\text{X}}$, il existe une valeur pour Y_1 telle que pour toute valeur de Y_2,..., il existe une valeur pour Y_i et il existe une valeur de Z qui rend la formule ϕ vrai
Conclusion, $x \in L \Leftrightarrow \exists Y_1 \forall Y_2 \exists Y_3 ... \exists Y_i$ Z$\phi(\hat{\text{X}})$, ce qui est exactement un problème de $QSAT_i$

□

Proposition 21. *S'il existe un problème HP-complet, alors la hiérarchie polynomiale se stationne à partir d'un certain rang*

Démonstration. Supposons qu'il existe un langage L qui soit HP-complet
Comme HP=$\bigcup_{n\in\mathbb{N}} \sum_n^P$, il existe i $\in \mathbb{N}$ tel que L$\in \sum_i^P$
Or comme L est HP-complet, tout langage L' $\in \sum_{i+1}^P$ peut se ramener à L
Mais comme tous les niveaux de la hiérarchie sont clos par réduction, on a que L' $\in \sum_{i+1}^P$
Donc, par la proposition 17, la hiérarchie se stationne

□

7 Conclusion

Nous avons pu constater que, en admettant P≠NP, la hiérarchie polynomiale est une façon d'augmenter nos moyens de comparer la complexité des programmes PSPACE
En revanche, si P=NP, toute la théorie s'écroule (comme beaucoup d'autres d'ailleurs)

Il reste une question très intéressante qui n'est toujours pas résolue à ce jours : on sait déjà que PH $\subseteq$ PSPACE mais on ne sait pas si l'inclusion est stricte
L'intérêt de cette question réside dans le fait que PSPACE admet un problème complet et donc, si on a l'égalité, on a vu que la hiérarchie stationnerait. Ce qui signifie que l'un des $QSAT_i$ serait complet. Mais rien ne justifie à l'heure actuelle qu'un de ces problèmes aient plus de raisons d'être PSPACE-complet que les autres et une des hypothèses est qu'ils seraient tous complets. Or parmis eux se trouvent $QSAT_1$ et ceci impliquerait que NP=co-NP

8 Bibliographie

Nous recommandons le livre de Papadimitriou "Computation Complexity" pour tout ce qui parle des oracles
Nous recommandons le cours de Jean Goubault-Larrecq pour une définition de la hiérarchie polynomiale par les machines alterantes
Références complémentaires qui m'ont servi pour l'exposé :
"Langages formels, calculabilité et complexité" de Olivier Carton
Les groupe de lecture sur "Theory of computation", Lecture 9 et 10. CS682,spring 2004
Wikipédia