Quelques propriétés de la complexité en espace

S. Dasse-Hartaut

16 février 2007

0.1 Définitions, lemmes

On va ici s'intéresser à trois théorèmes, qui ne sont pas vraiment intuitifs :

le théorème du Gap de Borodin donne l'égalité entre deux espaces, pour lesquels on ne connait à la base qu'une inclusion, et semble contredire le théorème de hiérarchie. Il n'y a cependant aucune contradiction, parce que le théorème de hiérarchie demande des hypothèses plus strictes.

Le théorème d'accélération de Blum affirme l'existence de fonctions ne possédant aucun meilleur programme (c'est-à-dire aucune machine de Turing optimale en espace).

Le théorème de l'union donne l'existence sous certaines conditions, d'une fonction S(n) satisfaisant une relation mettant en cause un ensemble de fonctions totales récursives.

Nous allons utiliser une numérotation pour les machine de Turing. Il y a bijection entre les machines de Turing bien construites et les entiers, et le codage ainsi que le décodage, est calculable efficacement.

Définition 1. On dira d'une propriété des entiers qu'elle est vraie presque toujours si elle est fausse seulement pour un nombre fini d'entiers.

Si une propriété est vraie presque toujours, alors sa négation n'est pas vraie presque toujours.

Une propriété des entiers peut être vraie infiniment souvent en même temps que la négation de cette propriete.

Lemme 1. L'ensemble des triplets < M, n, m > où M est une MT et n,m sont des entiers, tel que l'espace maximal utilisé par M sur des entrées de taille n est exactement m est récursif.

Preuve. Dans un espace maximal m le temps de calcul est borné (par c^m, avec c la taille de l'alphabet de bande) sinon on sait que la machine boucle. On peut donc simuler la machine M sur toutes les entrées de taille n et vérifier que l'espace maximal est bien m. □

Lemme 2. Si L = L(M) est le langage accepté par une MT qui est bornée en espace S(n) presque toujours alors il existe M tel que L = L(M) et M est borné en espace par S(n).

Preuve. La MT M ne respecte pas l'espace S(n) seulement sur un nombre fini d'entrées. Il existe donc une machine M qui fait la même chose que M sauf sur un nombre fini d'entrées. (Cet argument n'est pas constructif.) □

0.2 Gap de Borodin

0.2.1 Théorème de Gap de Borodin

Théorème 3 (**gap de Borodin**). *Etant donnée une fonction totale récursive g, telle que $g(n) \geqslant n$, il existe une fonction totale récursive S, telle que $DSPACE(S(n)) = DSPACE(g(S(n)))$.*

Preuve. Pour g donnée, on cherche à construire S telle que, si L un langage est dans $DSPACE(g(S(n)))$, L est nécessairement dans $DSPACE(S(n))$.

Pour cela, on établit une bijection entre les machines de Türing (MT) et les entiers. On note M_i la i^{eme} machine de Türing. Soit $S_i(n)$ l'espace maximal utilisé par M_i sur une entrée quelconque de taille n. Si, sur des entrées de taille n, M_i s'arrête toujours alors M_i est bornée en espace par $S_i(n)$, sinon $S_i(n)$ est défini égal à $\perp$, où $\perp$ est considéré plus grand que tous les entiers.

On cherche à construire S(n) totale récursive, telle que, pour tout k :
$S_k(n) \leqslant S(n)$ presque toujours, ou
$S_k(n) \geqslant g(S(n))$ infiniment souvent.

On crée donc un algorithme qui calcule S récursivement :

Pour tout j donné plus petit que n, on peut décider si $S_i(n) \leqslant j$ ou si $S_i(n) > g(j)$. On definit alors $S(n)$ par le plus petit entier j tel que pour tout i entre 1 et n on aie $S_i(n) \leqslant j$, si aucune de ces valeurs ne vaut $\perp$.

Sinon on impose 1 pour valeur initale de j, et on regarde s'il existe un i tel que M_i est entre $j+1$ et $g(j)$. Si c'est le cas, on réitère, avec j valant $S_i(n)$, sinon, on donne à S(n) la dernière valeur de j. ou $S_i(n) > g(j)$.

Cet algorithme, pour n donné, s'arrête nécessairement, car on considère un nombre fini de machines sur un nombre fini d'entrées (de taille n), donc il existe une taille d'espace donnée, pour n fixé, au delà de laquelle toute machine boucle.

Montrons que S ainsi définie satisfait bien les conditions demandées :
- La fonction S est récursive
- Soit k fixé, montrons que $S_k(n) \leqslant S(n)$ presque toujours, ou $S_k(n) \geqslant g(S(n))$ infiniment souvent :
s'il existe un entier n tel que $S_k(n) > S(n)$, par construction, on a $S_k(n) \geqslant g(S(n))$.
S'il existe une infinité d'entiers n tels que $S_k(n) > S(n)$, il existe donc une infinité d'entiers n tels que $S_k(n) \geqslant g(S(n))$.

On suppose alors qu'il existe L dans DSPACE(g(S(n)) mais pas dans DSPACE(S(n)).
Soit alors M_k une machine qui décide L avec $S_k(n) \leqslant g(S(n))$. Par construction, on est assuré que pour tout $n \geqslant k$, $S_k(n) \leqslant S(n)$, donc $S_k(n) \leqslant S(n)$ presque toujours.
D'après le deuxième lemme , cela implique que L est dans DSPACE(S(n)), ce qui contredit les hypothèses faites sur L.
On a donc construit S telle que $DSPACE(S(n)) = DSPACE(g(S(n)))$.

□

0.2.2 Exemples

Pour k entier,il existe S, tel que $DSPACE(S(n)) = DSPACE(S(n)^k)$.
Il existe S, tel que $DSPACE(S(n)) = DSPACE(2^{S(n)})$.
Il existe S, tel que $DSPACE(S(n)) = DSPACE(S(n)!)$.

0.2.3 Conséquence

Il existe f telle que $DTIME(f(n)) = NTIME(f(n)) = DSPACE(f(n)) = NSPACE(f(n))$

Preuve. On possède déjà les inclusions

$$DTIME(f(n)) \subseteq NTIME(f(n)) \subseteq NSPACE(f(n))$$

et

$$DTIME(f(n)) \subseteq DSPACE(f(n)) \subseteq NSPACE(f(n))$$

Il suffit alors de trouver f pour lequel $NSPACE(f(n)) \subseteq DTIME(f(n))$. Pour tout langage L,

$$L \in NSPACE(f(n)) \Rightarrow L \in DSPACE(f(n)^2) \Rightarrow L \in DTIME(c^{f(n)^2}) \Rightarrow L \in DTIME(f(n)^{f(n)^2})$$

Le théorème du gap de Borodin assure l'existence de f telle que $DTIME(f(n)) = DTIME(f(n)^{f(n)^2})$, d'où le résultat.

□

0.3 Speed up de Blum

Nous allons maintenant voir qu'il existe un langage pour lequel aucun algorithme n'est optimal en terme d'espace.

Théorème 4 (Speed up de Blum). *:*
Soit $r(n) \geqslant n^2$ une fonction totale récursive non décroissante. Il existe un langage récursif L tel que pour toute MT, M_i qui décide L en espace $S_i(n)$ il existe une MT M_j qui décide L en espace $S_j(n)$ tel que $r(Sj(n)) \leqslant Si(n)$ presque toujours.

Preuve. L'idée générale de la preuve est de construire un langage L, tel que L peut être reconnu, plus "rapidement"(en espace) qu'avec n'importe quelle machine le reconnaissant, à l'aide d'une nouvelle machine de Türing indicée par un entier, et que, i augmentant, l'espace nécessaire à M_i pour reconnaître L est de plus en plus petit. On utilise un raisonnement diagonal.

Définissons h de par h(1) = 2 et h(n) = r(h(n - 1)).

On note M_i la i-ème machine de Turing suivant la bijection utlisée précédemment entre les machines de Türing et les entiers. La longueur du code de M_i est log(i)(par construction) et M_i est bornée en espace par $S_i(n)$.

Nous allons construire L tel que
1) Si L est reconnu par M_i alors $S_i(n) \geqslant h(n-i)$ presque toujours.
2) Pour tout k, il existe une MT M_j tel que L = L(M_j) et $S_j(n) \leqslant h(n-k)$.

Montrons déjà que pour L ainsi construit,pour toute M_i acceptant L, il existe M_j qui accepte L tel que $S_i(n) \geqslant r(S_j(n))$ presque toujours.
Soit M_i qui accepte L en espace S_i. On choisit M_j tel que $S_j(n) \leqslant h(n-i-1)$. Ceci est possible par la condition (2), avec $k = i-1$.
Par la condition (1), on a alors $S_i(n) \geqslant h(n-i) = r(h(n-i-1)) \geqslant r(S_j(n))$ presque toujours.

Nous devons maintenant construire L qui satisfait (1) et (2). Le langage L que l'on va définir ne contiendra qu'au plus un mot de taille n, pour tout n, de la forme 0^n, s'il existe.
On calcule L, en décidant pour tout n, en commençant à $n = 1$, et en incrémentant à chaque étape, si 0^n appartient à L. Pour cela, on proscrit certaines MT M_i, à chaque étape. Une machine proscrite n'accepte pas L.

Soit $\sigma(n)$ le plus petit $j \leqslant n$ tel que $S_j(n) < h(n-j)$ et M_j n'a pas été proscrite aux étapes j, pour $j < n$.
A l'étape n, si $\sigma(n)$ existe, alors on proscrit M(n). 0^n sera placé dans L si et suelement si$\sigma(n)$ existe et 0^n n'est pas accepté par M(n).

Montrons maintenant que L satisfait la condition (1).
Si L = L(M_i) alors $S_i(n) \geqslant h(n-i)$ presque toujours.
Supposons qu'il existe une infinité d'entiers n, tels que $S_i(n) < h(n-i)$. S'il existe k tel que $\sigma(k) = i$, alors M_i est proscrite, et 0^k est dans L, si et seulement si 0^n n'est pas accepté par M_i : c'est absurde.
Cependant, il existe une infinité d'entiers n, tels que $S_i(n) < h(n-i)$. Le nombre d'entiers plus petits que i étant fini, il existe n_0, tel que pour tout j, avec $j < i$ et $S_j(n_0) < h(n_0 - j)$, M_j est proscrite.
En effet, soit l'ensemble E des entiers j, tels que $j < i$, $S_j(n) < h(n-j)$,et M_j est non proscrite. Si on est à l'étape n, avec $S_i(n) < h(n-i)$, soit on proscrit M_i, soit on enlève un élément de E. Si on suppose qu'on ne peut proscrire M_i,il existe donc n, tel qu'à l'étape n, E est vide. Or si à l'étape n, E est vide, il existe $m > n$ tel que $S_i(m) < h(m-i)$, avec $\sigma(m) = i$. On proscrit alors M_i, ce qui est absurde : L satisfait la condition (1).

Montrons maintenant que L satisfait la condition (2).
Pour tout k, il existe une machine de Turing M_j telle que $L = L(M_j)$ et $S_j(n) \leqslant h(n-k)$.
On va construire M, une machine satisfaisant, pour un k donné, L = L(M). Il existe alors j tel que $M = M_j$ et on a $S_j(n) \leqslant h(n-k)$.
Pour déterminer si $0^n \in L$, la MT M doit simuler $M_{\sigma(n)}$ sur 0^n.
Or, pour calculer $\sigma(n)$, M doit déterminer quelles machines M_i ont été proscrites au cours des étapes N = 1, . . . , n - 1. Cependant, pour construire la liste des M_i proscrits, il faut tester si M_i utilise, sur (0^l), plus de $h(l-i)$ cases mémoire pour $0 \leqslant l \leqslant n$ et $1 \leqslant i \leqslant n$. Pour $i < k + l - n$, ceci nécessite plus de h(l - i) = h(l - (k + l - n)) = h(n - k) cases mémoire. On aurait donc besoin de trop d'espace.
Pour contourner ce problème, on amène la machine M à traiter à part les cas qui demandent trop d'espace. En effet, pour k fixé, il existe un n_1 tel que tout M_i avec $i \leqslant k$ qui est proscrit, est proscrit à l'étape $n_1 > n$.

On inclut donc dans le code du programme la liste de tous les éléments de L de taille inférieure ou égale à n_1 : il y en a un nombre fini. On inclut de même la liste de toutes les M_i proscrites lors des premières n_1 étapes : il y en a aussi un nombre fini.

Pour $n \leqslant n_1$, la machine M n'utilise aucun espace parce que toutes les réponses à ses calculs sont incluses dans son code.

Dans le cas où $n > n_1$, pour calculer $\sigma(n)$ et simuler $M_{\sigma(n)}(0^n)$, on a besoin de simuler les machines $M_i(0^l)$ pour $n1 < l \leqslant n$ et $k < i \leqslant n$ pour voir si M_i est proscrit à l'étape l.

Premièrement, la simulation de $M_i(0^l)$ se fait en espace $h(l-i) \leqslant h(n-k-1)$ cases mémoire de M_i puisque $l \leqslant n$ et $i > k$.

Les cases mémoire de M_i peuvent être plus grosses que celles de M. Comme $i_{\leqslant} n$, la taille du code de la machine M_i est d'au plus log(n). Chaque symbole de M_i peut donc être codé en utilisant log(n) symbole. Or $\log(n)h(n-k-1) \leqslant h(n-k-1)^2 \leqslant h(n-k)$.

Deuxièmement, la simulation de $M_{\sigma(n)}(0^n)$ nécessite $h(n-\sigma(n)) \leqslant h(n-k)$ puisque comme $n > n_1$ si $\sigma(n)$ existe il doit être plus grand que k (par choix de n_1.

Troisièmement, la machine M doit garder une liste des M_i proscrits. Il y en a au plus n et donc l'espace nécessaire est au plus $n\log(n)$. Clairement pour n suffisamment grand $n\log n \leqslant h(n-k)$. Les cas nécessitant trop d'espace sont codés dans la machine elle-même.

Le langage L satisfait donc les conditions (1) et (2), d'après la première étape de la preuve, on en déduit qu'on connait L, tel que pour toute machine de Turing M qui décide L en espace $S(n)$, il existe une machine M' qui décide L en espace $S'(n)$, avec $r(S'(n)) \leqslant S(n)$ presque toujours.

□

0.4 Union

Théorème 5. *(Union)*
Soit $\{fi|i = 1, 2, ...\}$ une liste énumérable de fonctions totales récursives. C'est-à-dire qu'il existe une machine M_f telle que $M_{M_f(x)}$ calcule f_x. Supposons encore que $f_i(n) < f_{i+1}(n)$, alors il existe S tel que $DSPACE(S(n)) = \bigcup_{i\geqslant 1} DSPACE(f_i(n))$.

Preuve. Nous allons construire une fonction totale calculable $S(n)$ satisfaisant les deux conditions suivantes :

1) Pour tout i, $S(n) > fi(n)$ presque toujours.

2) Si S_j est la complexité d'espace de M_j et pour tout i, $S_j(n) > f_i(n)$ infiniment souvent alors $S_j(n) > S(n)$ infiniment souvent.

Montrons que les conditions (1) et (2) impliquent $DSPACE(S(n)) = \bigcup_{i\geqslant 1} DSPACE(fi(n))$:

La première condition assure que $\bigcup_{i\geqslant 1} DSPACE(f_i(n)) \subseteq DSPACE(S(n))$, la seconde que DSPACE(S(n)) ne contient que les ensembles qui se trouvent dans $DSPACE(f_i(n))$ pour un i.

En effet, si $L \in \bigcup_{i\geqslant 1} DSPACE(f_i(n))$ alors pour un certain m, $L \in DSPACE(f_m(n))$.

Par (1) $f_m(n) < S(n)$ presque toujours, et donc par un lemme précédent, $L \in DSPACE(S(n))$.

Si $L \in DSPACE(S(n))$ alors il existe j tel que $L = L(M_j)$ et $\forall n, S_j(n) \leqslant S(n)$.

Supposons maintenant que $L \notin \bigcup_{i\geqslant 1} DSPACE(f_i(n))$.

Pour aucun i, $L \in DSPACE(f_i(n))$.

Pour chaque i et k, si M_k accepte L, $S_k(n) > f_i(n)$ infiniment souvent.

La condition (2) s'applique.

Il existe n pour lequel $S_k(n) > S(n)$.

En choisissant $k = j$ on obtient une contradiction.

On peut alors donner une façon de générer $S(n)$:

1) Initialiser LISTE=(liste vide)

2) pour $i = 1, 2, ...$ faire :

3) Si pour tout $i_j = k$ dans LIST $f_k(n) \geqslant S_j(n)$. alors

4) ajouter $i_n = n$ à LISTE et définir $S(n) = f_n(n)$. sinon

5) Parmi les $i_j = k$ tel que $f_k(n) < S_j(n)$ on prend le minimum sur k.puis pour ce k le minimum sur i.

6) On définit $S(n) = f_k(n)$

7) On remplace $i_j = k$ par $i_j = n$ dans LISTE

8) On ajoute $i_n = n$ dans LISTE

Clairement, $S(x)$ est calculable, on effectue la procédure de génération jusqu'à ce que $n = x$ et alors on retourne la valeur appropriée : S est alors totale récursive. Il faut maintenant montrer que S satisfait les deux conditions.

Montrons que $S(n)$ satisfait la condition (1).

Pour tout m, $S(n) > f_m(n)$ presque toujours.

On assigne une valeur à $S(n)$ soit à la ligne 4 ou à la ligne 6 de la procédure décrite précédemment.

Pour $n > m$ la valeur assignée à la ligne 4 est $f_n(n) \geqslant f_m(n)$.

Si la valeur est assignée à la ligne 6, alors il se peut que la valeur de k soit inférieure à m et donc que $S(n) = f_k(n) > f_m(n)$.

Si cela arrive, on remplace à la ligne 7 l'élément de la liste par un plus gros. Eventuellement, pour n suffisamment grand, il ne restera plus de $k < m$ et à partir de ce moment $S(n) > f_m(n)$.

Montrons que $S(n)$ satisfait la condition (2).

Si S_j est la complexité d'espace de M_j et pour tout i, $S_j(n) > f_i(n)$ infiniment souvent alors $S_j(n) > S(n)$ infiniment souvent.

Comme pour tout i, $S_j(n) > f_i(n)$, la condition 3 ne sera pas respectée infiniment souvent et à chaque fois la valeur retournée sera inférieure à $S_j(n)$.

□