Sujet de rédaction

Simulation d'une machine de Turing à k bandes par une machine à 2 bandes

Michaël MATHIEU

20 décembre 2008

1 Présentation

On sait que l'on peut simuler une machine de Turing à k bandes sur une machine à une seule bande, mais cela peut aller jusqu'à élever la complexité au carré. Plus formellement, si $\mathcal{M}$ est une machine à k bandes qui calcule une entrée de taille n en temps $O(t(n))$, alors on peut construire une machine $\mathcal{M}'$ à une bande calculant la même fonction que $\mathcal{M}$ en temps $O(t(n)^2)$. On peut montrer que dans certains cas, on ne peut s'affranchir de cette augmentation de complexité.

Nous allons par conséquent nous intéresser, non pas à une machine $\mathcal{M}'$ à une bande, mais à une machine à 2 bandes, et nous allons montrer que dans ce cas, on peut construire une machine $\mathcal{M}'$ (à 2 bandes) qui calcule une entrée de taille n en temps $O(t(n)\log(t(n)))$. Dans une seconde partie, on présentera une application de ce résultat à la détermination de la hiérarchie des classes de complexité.

2 Énoncé et preuve du théorème

Théorème :

Soit $\mathcal{M}$ une machine de Turing déterministe à $k > 2$ bandes, calculant en temps $O(t(n))$. Alors on peut construire une machine de Turing $\mathcal{M}'$ déterministe à 2 bandes qui calcule le même langage que $\mathcal{M}$, en temps $O(t(n)\log(t(n)))$.

Remarque : Avec le théorème d'accélération linéaire, on peut même assurer que $\mathcal{M}'$ soit en temps exactement $t(n)\log(t(n))$.

Preuve : On note Σ l'alphabet de $\mathcal{M}$. Il s'agit de définir la machine $\mathcal{M}'$. Cette machine possède deux bandes, la première, dite bande principale, sera la bande qui servira à simuler les bandes de $\mathcal{M}$, tandis que le seconde, dite bande de travail, ne servira en fait qu'à stocker de façon temporaire des données à déplacer sur la bande principale.

On utilise une machine $\mathcal{M}'$ à bandes bi-infinies, mais étant on sait qu'une machine à bandes bi-infines peut être simulée sur une machine "classique" sans perte de complexité asymptotique.

La machine $\mathcal{M}'$ travaille avec des k-uplets de couples de lettres de Σ. En plus de simuler k bandes sur la bande principale, on subdivise encore une fois chacune de ces bandes. Autrement dit, chacune des bandes de $\mathcal{M}$ sera simulée sur une "sous-bande" de couples. On attribue donc deux piste à chaque bande, que l'on appelle piste haute et piste basse. L'alphabet de $\mathcal{M}'$ est donc $\left(\Sigma^2\right)^k$

On introduit également un découpage "horizontal" de la bande principale en blocs de travail : la bande principale est virtuellement divisée en blocs B_k de la façon suivante :

- B_0 est composé de la case d'indice 0 de piste basse de la bande (la piste haute n'est pas utilisée dans ce bloc) ;
- pour tout $n \geq 1$, B_n est composé des cases d'indice 2^{n-1} à $2^n - 1$;
- pour tout $n \geq 1$, B_{-n} est composé des cases d'indice $-2^n + 1$ à -2^{n-1}.

Un bloc possède trois états possibles (en fait, il en possède d'autres, mais ils ne sont jamais utilisés lors de la simulation) :

- *état vide* : les deux pistes sont vides sur B_k ;
- *état moitié plein* : la piste haute est vide mais pas la piste basse ;
- *état plein* : les deux pistes sont pleines.

#	#	#	#	#	#	a_{-2}	$\cdot$	a_2	#	#	#	#
#	#	#	#	a_{-4}	a_{-3}	a_{-1}	a_0	a_1	a_3	a_4	a_5	#
#	#	#	#	#	#	#	$\cdot$	b_2	b_7	b_8	#	#
#	#	#	#	#	#	b_{-1}	b_0	b_1	b_3	b_4	b_5	b_6
#	#	#	#	c_{-6}	c_{-5}	c_{-2}	$\cdot$	c_1	#	#	#	#
c_{-10}	c_{-9}	c_{-8}	c_{-7}	c_{-4}	c_{-3}	c_{-1}	c_0	c_2	c_3	#	#	#

Figure 1 : La bande principale de $\mathcal{M}'$. Les doubles barres verticales représentent les blocs, tandis que les doubles barres-horizontales symbolisent les "sous-bandes".

L'idée intuitive de l'algorithme est de travailler sur les blocs, de sorte que les opérations coûteuses de déplacement de la tête de lecture, que l'on rencontre si l'on simule avec une seule bande, ne se produisent que rarement, en fait une fois tous les 2^n déplacements.

Au départ, le mot d'entrée est placé sur la piste basse de la bande de $\mathcal{M}'$ qui correspond à la bande d'entrée de $\mathcal{M}$. Lors de la simulation de $\mathcal{M}$ par $\mathcal{M}'$, au lieu de déplacer la tête de $\mathcal{M}'$ sur la bande principale, on va déplacer le contenu de la bande. On utilise pour cela deux fonctions, `tasse-droite` et `tasse-gauche`, qui vont respectivement déplacer la bande vers la droite et vers la gauche. On ne décrira que la fonction `tasse-droite`, l'autre étant similaire.

`tasse-droite :`
faire
$i \leftarrow$ le plus petit indice $i > 0$ tel que B_i n'est pas plein
si B_i est à moitié plein, alors
déplacer la totalité de B_{i-1} dans la piste haute de B_i
sinon
déplacer la totalité de B_{i-1} dans la piste basse de B_i
finsi
jusqu'à $i = 1$
i $\leftarrow$ le plus grand indice $i < 0$ tel que B_i n'est pas vide
si B_i est à moitié plein, alors
déplacer le contenu de la piste basse de B_i dans les pistes
basses des B_{i+1}, B_{i+2}, ... , B_0
sinon
déplacer le contenu de la piste haute de B_i dans les pistes
basses des B_{i+1}, B_{i+2}, ... , B_0
finsi

#	#	#	#	#	·	#	#	#	#	#	#	
a_{-5}	a_{-4}	a_{-3}	a_{-2}	a_{-1}	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

après un `tasse-droite` :

#	#	#	#	#	·	a_0	#	#	#	#	#	
a_{-5}	a_{-4}	a_{-3}	a_{-2}	#	a_{-1}	a_1	a_2	a_3	a_4	a_5	a_6	a_7

après un autre `tasse-droite` :

#	#	#	#	#	·	#	a_0	a_1	#	#	#	
a_{-5}	a_{-4}	#	#	a_{-3}	a_{-2}	a_{-1}	a_2	a_3	a_4	a_5	a_6	a_7

après un `tasse-gauche` :

#	#	#	#	a_{-3}	·	#	a_0	a_1	#	#	#	
a_{-5}	a_{-4}	#	#	a_{-2}	a_{-1}	#	a_2	a_3	a_4	a_5	a_6	a_7

Figure 2 : Deux `tasse-droite` suivis d'un `tasse-gauche`

On s'aperçoit que ce que la propriété énoncée ci-dessus, à savoir l'état soit *plein*, soit *moitié plein*, soit *vide* des blocs, est préservé par cet algorithme. En effet, on ne remplit les parties hautes que lorsque les parties basses sont pleines. On remarque également que le caractère exponentiel des tailles des blocs assure qu'il y aura toujours la place nécessaire, lors des déplacements de blocs (après la boucle de l'algorithme).

Ainsi, lors de la simulation du déplacement de tête de la bande p de $\mathcal{M}$ vers la gauche, on applique la fonction `tasse-droite` , en considérant les blocs correspondants à la bande p, et en utilisant la bande de travail comme une pile pour ef-

fectuer les déplacements. On se convainc facilement que cette méthode de simulation est correcte, il reste à démontrer que sa complexité est en $O(t(n)\log(t(n)))$.

Intuitivement, le fait de découper en blocs de taille exponentielle va réduire la complexité amortie. En effet, on n'effectuera un gros déplacement que rarement ; et le plus souvent, le travail se fera au voisinage de 0, ainsi, les déplacement seront peu coûteux.
Plus formellement, à un appel de `tasse-droite`, on note i l'indice du bloc le plus éloigné de B_0 qu'il faudra modifier (c'est-à-dire le plus petit i tel que B_i n'est pas plein. Alors l'appel coûtera $O(2^i)$ transitions. Il faut savoir à quelle fréquence i sera atteint lors de la mise à jour de la bande. Une fois i modifié, tous les blocs $B_1, B_2, \ldots, B_{i-1}$ seront remplis à moité. Donc même s'il n'y a pas d'appel de `tasse-gauche` entre temps, il faut $2^{i-1}-1$ transitions pour remplir à nouveau entièrement ces blocs, et donc être contraint de modifier une nouvelle fois B_i. Par conséquent, le bloc B_i est modifié au plus une fois toutes les 2^i-1 transitions.

Comme $\mathcal{M}$ ne peut atteindre une case d'indice supérieur à $t(n)$ en temps $t(n)$, on sait que $\mathcal{M}'$ n'aura pas à atteindre un bloc d'indice strictement supérieur à $\lfloor \log_2(t(n)) \rfloor + 1$.
Soit $|i| \leq \lfloor \log_2(t(n)) \rfloor + 1$. On ne peut atteindre le bloc B_i qu'une fois toutes les $2^{|i|-1}$ transitions, et son déplacement nous coûte moins de $C2^{|i|-1}$ transitions (puisque B_i est de taille $2^{|i|-1}$), avec C une constante indépendante de n et i. Ainsi, la somme des accès au bloc B_i nous coûte au plus $C2^{|i|-1}\frac{t(n)}{2^{|i|-1}} = Ct(n)$ transitions.
La complexité totale est égale à la somme de ces termes, autant de fois qu'il y a de bandes dans $\mathcal{M}$. Ainsi, la complexité totale est majorée par

$$k \sum_{i=-\lfloor \log_2(t(n)) \rfloor - 1}^{\lfloor \log_2(t(n)) \rfloor + 1} C \cdot t(n) = O(t(n)\log(t(n)))$$

3 Application aux théorèmes de hiérarchie

Grâce au théorème précédent, on peut établir un corollaire, qui est un théorème de hiérarchie temporelle, valable même pour des machines à $k \geq 2$ bandes.

On note $TIME_k(t(n))$ la classe des langages qui sont calculés par une machine de Turing à k bandes en temps $t(n)$. On note

$$TIME(t(n)) = \bigcup_{k \geq 1} TIME_k(t(n))$$

Théorème (de hiérarchie temporelle) : :

Soient $t_1 : \mathbb{N} \to \mathbb{N}$ et $t_2 : \mathbb{N} \to \mathbb{N}$ deux fonctions telles que t_1 soit constructible en temps, que $\lim_{n \to +\infty} \frac{t_1(n)\log(t_1(n))}{t_2(n)} = 0$, et pour n assez grand, et que $t_2(n) \geq t_1(n) \geq n$.
Alors il existe un langage qui soit dans $TIME(t_2(n))$ mais pas dans $TIME(t_1(n))$.

Preuve : Remarquons d'abord que quitte à ajouter des bandes, on peut compter, lors de la simulation d'une machine de Turing, le nombre de transitions réalisé pendant la simulation, et s'arreter si ce nombre dépasse une certaine borne.

On utilise un argument diagonal. Le soucis est que pour construire une machine qui va executer $\mathcal{M}$ sur $< \mathcal{M} >$, il faut prendre garde au fait que $\mathcal{M}$ peut avoir un nombre arbitraire de bandes, ce qui ne sera pas le cas de la machine qui simule, et donc pourrait causer des problèmes pour effectuer une simulation efficace. On va par conséquent imposer que $\mathcal{M}$ ait seulement 2 bandes, et utiliser le théorème précédent pour s'y ramener, en ayant une borne précise du temps de simulation. On construit donc la machine $\mathcal{N}$ qui prend en entrée le codage $< \mathcal{M} >$ d'une machine $\mathcal{M}$ et s'exécute comme suit :

calculer $t_2(n)$
exécuter $\mathcal{M}$ sur $< \mathcal{M} >$
    si le temps d'exécution dépasse $t_2(n)$ alors
        accepter (cas 1)
    sinon
        si l'exécution a retourné $VRAI$ alors
            refuser (cas 2)
        sinon
            accepter (cas 3)

La machine $\mathcal{N}$ est sans problèmes dans $TIME(t_2(n))$ (on rappelle que $t_2(n)$ est calculable en temps $O(t_2(n))$). Il reste donc à montrer qu'elle n'est pas dans $TIME(t_1(n))$.

Supposons que $\mathcal{L}(\mathcal{N}) \in TIME(t_1(n))$. Alors par le théorème précédent, il existe une machine de Turing $\mathcal{T}$ à deux bandes qui détermine l'appartenance à $\mathcal{L}(\mathcal{N})$ en temps $O(t_1(n)\log(t_1(n)))$. Lors du calcul d'une entrée de taille n, la machine $\mathcal{T}$ effectue donc au plus $Ct_1(n)\log(t_1(n))$ transitions, où C est une constante qui ne dépend pas de l'entrée.

Comme $\lim_{n\to+\infty} \frac{t_1(n)\log(t_1(n))}{t_2(n)} = 0$, il existe n_0 assez grand pour avoir, pour tout $n \geq n_0$, $Ct_1(n)\log(t_1(n)) \leq t_2(n)$.

Quitte à ajouter des transitions inaccessibles à $\mathcal{T}$, on peut supposer que $\mathcal{T}$ est de taille supérieure à n_0. De la sorte, $\mathcal{T}$ appliquée à l'entrée $< \mathcal{T} >$ n'est pas dans le cas 1 de l'algorithme ci-dessus. Mais alors $< \mathcal{T} > \in \mathcal{L}(\mathcal{N})$ si et seulement si $< \mathcal{T} > \in \mathcal{L}(\mathcal{T})$ si et seulement si $< \mathcal{T} >$ est acceptée par $\mathcal{T}$ en temps inférieur à $Ct_1(n)\log(t_1(n)) \leq t_2(n)$ (par définition de $\mathcal{T}$) si et seulement si $\mathcal{N}$ rejette $< \mathcal{T} >$ (puisqu'on est alors dans le cas 1) si et seulement si $< \mathcal{T} > \notin \mathcal{L}(\mathcal{N})$, ce qui est absurde.

Application : Ce théorème permet, par exemple, de démontrer que $PTIME \subsetneq EXPTIME$. En effet, si $k, h > 0$, on a $\lim_{x\to+\infty} \frac{n^k \log(n^k)}{2^{hn}} = 0$, et on peut appliquer le théorème de hiérarchie.