Le *hardest* langage

Pierre Bertin

11 février 2007

Le *hardest* langage est l'équivalent pour les langages algébriques des problèmes NP-complets. Un problème A est NP-complet si tout problème B de la classe NP se réduit polynomialement à A. Le *hardest* langage sera donc un langage L_0 qui vérifie que tout langage L algébrique se réduit à L_0. Il faut encore choisir la réduction appropriée : ce sera la réduction par morphisme.
Il faut donc démontrer le théorème suivant :

Théorème 1 *Il existe un langage algébrique L_0 sur un alphabet A tel que pour tout langage algébrique L sur un alphabet B il existe un morphisme $\phi : B^* \to A^*$ tel que :*

$$L\backslash\{\epsilon\} = \phi^{-1}(L_0\backslash\{\epsilon\})$$

1 Définition du langage L_0

On aura besoin du langage de Dyck D_2^* engendré par la grammaire suivante :

$$\begin{array}{l} A = \{a_1,\bar{a}_1,a_2,\bar{a}_2\} \\ S \to SS + a_1 S \bar{a}_1 + a_2 S \bar{a}_2 + \epsilon \end{array}$$

Définition 1 *Soit $T = \{a_1,\bar{a}_1,a_2,\bar{a}_2,c,\$\}$*
Soit L_0 le langage défini sur $T \cup \{d\}$par :

$$\begin{array}{ll} L_0 = \{\epsilon\} \cup \{x_1 c y_1 c z_1 d \dots d x_n c y_n c z_n | & n \geq 1 \\ & x_i,z_i \in T^* \ \forall i \\ & y_1 y_2 \dots y_n \in \$D_2^* \ , \ y_i \neq \epsilon\} \end{array}$$

Le langage L_0 choisit un sous-mot dans chaque groupe encadré par des d de telle sorte que la concaténation de ces sous-mots appartienne à $\$D_2^*$.
Voici un exemple de grammaire qui engendre L_0 :

$$\begin{array}{l} S \to \epsilon \ + \ US'a_1VS'\bar{a}_1W \ + \ US'a_2VS'\bar{a}_2W \\ S' \to S'S' \ + \ a_1VS'\bar{a}_1V \ + \ a_2VS'\bar{a}_2V \ + \ \epsilon \\ U \to^* \ T^*c\$ \\ V \to^* \ cT^*dT^*c \ + \ \epsilon \\ W \to^* \ cT^*d \end{array}$$

2 Construction du morphisme

Soit L un langage algébrique. On suppose que L ne contient pas le mot vide. Soit G=(B,V,P) une grammaire qui reconnaît le langage L. On suppose que G est en forme normale de Greibach, c'est-à-dire que toutes les règles sont de la forme $Y_i \to b_k Y_{j_1} \dots Y_{j_m}$. On suppose de plus que l'axiome ne se trouve jamais dans le membre de droite.

L'idée est de construire une application $\xi : P \to T^*$ de telle sorte que : S se dérive en w par la suite de règles $p_1 \dots p_n$ ssi $\xi(p_1) \dots \xi(p_n) \in \$D_2^*$.

Par exemple, si V= $\{S, Y_2, \dots, Y_n\}$ on construit ξ :

$$\begin{cases} \text{si p} = S \to bY_{j_1} \dots Y_{j_m}, & \xi(p) = \ \$\ \bar{a}_1 \bar{a}_2^1 \bar{a}_1\ a_1 a_2^{j_m} a_1 \dots a_1 a_2^{j_1} a_1 \\ \text{si p} = Y_i \to bY_{j_1} \dots Y_{j_m}, & \xi(p) = \quad \bar{a}_1 \bar{a}_2^i \bar{a}_1\ a_1 a_2^{j_m} a_1 \dots a_1 a_2^{j_1} a_1 \end{cases}$$

Il faut encore construire le morphisme proprement dit. Soit b une lettre de B et $P_b = \{p_1, \dots, p_m\}$ l'ensemble des règles dont le membre de droite commence par b. Alors :

$$\phi(b) = c\xi(p_1)c \dots c\xi(p_m)cd$$

Il faut encore montrer qu'un mot w appartient à L si et seulemeny si ϕ(w) appartient à L_0. La démonstration se construit à peu près ainsi :

Le mot w appartient à L si et seulement si S se dérive en w.

Si w=$b_1 \dots b_m$, S se dérive en w si et seulement si il existe des règles $p_1, \dots, p_m$ telles que le membre de droite de p_i commence par b_i et que cette suite constitue une dérivation de S.

Enfin, si on a bien construit l'application ξ il suffit pour ça que $\xi(p_1) \dots \xi(p_m)$ soit un mot de $\$D_2^*$.

On rajoute plein de rigueur dans cette démonstration et le tour est joué.

3 Applications

Tout d'abord, on peut remarquer que la dernière démonstration ressemble beaucoup aux démonstrations de NP-complétude. Pour montrer la NP-complétude de SAT on avait codé l'existence d'un calcul réussi par une formule à satisfaire. Ici on code l'existence d'une 'érivation par l'éxistence de sous-lots qui vérifient une certaine caractéristique.

Les probmlèmes de NP-complétude et de *hardest* langage sont donc très semblables. Ils ont les mêmes applications. Pour démontrer qu'une propriété est vraie pour tous les langages algébriques, il suffit de montrer qu'elle est vraie pour L_0 et qu'elle passe par les morphismes.

Par exemple, si on trouve une machine de Turing qui reconnaît L_0 avec une complexité en temps $O(n^3)$, alors tout langage algébrique est reconnaissable par une machine de Turing avec une complexité en temps $O(n^3)$. En effet, pour tout morphisme ϕ et tout mot w, $|\phi(w)| \leq C|w|$. Les mêmes conclusions s'appliquent pour la complexité en espace.

On peut encore faire d'autres parallèles : P=NP ssi SAT $\in$ NP, de même tous les langages algébriques sont reconnaissables par un automate à pile déterministe ssi L_0 l'est. C'est beaucoup moins intéressant puisqu'on sait que la première affirmation est fausse. On peut cependant prendre le problème à l'envers. Si P $\neq$ NP, alors SAT n'appartient pas à P, puisque tous les langages algébriques ne sont pas reconnaissables par un automate à pile déterministe, L_0 ne l'est pas.

Références

[1] Sheila A. Greibach , *The hardest context-fre language*, Siam Journal of Computing No.4 Dec. 1973