NP-complétude de la séparation par automate

Antoine Dequay

21 septembre 2022

Notes

— Prof : Nathalie Bertrand.

— Leçon : 909, 928.

— Références :

  — Floyd, p. 664

Théorème 1 On définit le problème de séparation par automate SEP :

— Entrée : S et T deux langages finis disjoints et k un entier,

— Sortie : True si et seulement si il existe un automate à k états qui accepte les mots de S et rejette ceux de T.

Ce problème est NP-complet.

Preuve. On commence par montrer que ce problème est dans NP. En effet, en temps polynomial, étant donné un automate $\mathcal{A}$ déterminisé construit de manière non-déterministe, on vérifie en temps linéaire que $S \subset \mathcal{L}(\mathcal{A})$ et $T \cap \mathcal{L}(\mathcal{A}) = \emptyset$.

Soit $F = \bigwedge_{i=0}^{n-1} C_i$ une forme normale conjonctive sur les littéraux $x_0, \dots, x_{r-1}$.

Pour montrer que le problème est NP-dur, on va réduire polynomialement CNF-SAT, qui est NP-dur (car équivalent à SAT), à SEP. Pour cela, on va chercher à définir S et T pour traduire sur un automate les conditions exprimées par F.

On commence par fixer les nombre d'états de l'automate : On choisit $k = n+2r$ et l'alphabet $\Sigma = \{a, b\}$. On note $l_0, \dots, l_{2r-1} = x_0, \dots, x_{r-1}, \overline{x_0}, \dots, \overline{x_{r-1}}$, et on pose enfin $S_0 = \{\epsilon, a^k\}$, $T_0 = \{a^i, i \in [\![1, k-1]\!]\}$.

a va servir à donner la forme de l'automate et imposer sa résolution, alors que b va permettre d'y imposer des contraintes pour traduire F. De même, les $(S_i)_i$ seront à reconnaître par l'automate, au contraire des $(T_i)_i$.

Si $\mathcal{A}$ est un automate acceptant S_0 et refusant T_0 et que $\mathcal{A}$ a k états, alors $\mathcal{A}$ est une boucle de transitions étiquetées par a, avec $F = \{i\}$. On note $C_0 = i$ et $Q = \{C_0, \dots, C_{n-1}, l_0, \dots, l_{2r-1}\}$ dans l'ordre des transitions :

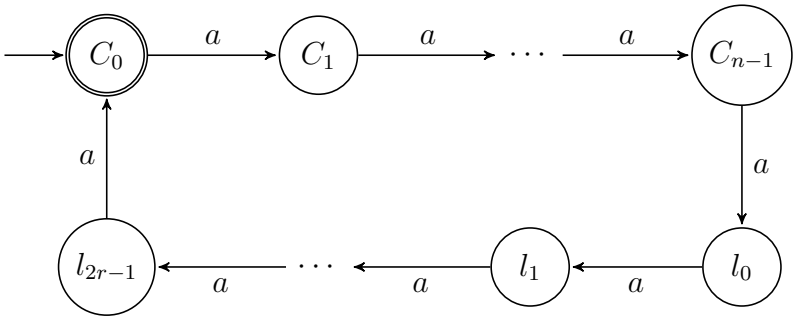

FIGURE 1: Automate à compléter au fil des explications.

On utilise le codage suivant pour les transitions étiquetées par b, pour traduire les relations entre les $(C_i)_i$ et les $(l_i)_i$: $C_i \xrightarrow{b} l_j \iff$ l_j est un littéral de C_i.

On définit donc :

$$T_1 = \left\{a^i b a^j, i \in [\![0, n-1]\!], j \in [\![0, k-1]\!]\right\} \setminus \left\{a^i b a^{2r-j}, l_j \text{ littéral de } C_i\right\}$$

Pour plus de commodité, sans perdre de généralité, on fixe C_0 à "vrai" et $\overline{x_0}$ à faux. Pour imposer des valeurs de vérité à chaque clause et littéraux, on veut : $l_j \xrightarrow{b} C_0 \iff l_j$ est un littéral vrai et $l_j \xrightarrow{b} \overline{x_0} \iff l_j$ est un littéral faux. On construit pour cela :

$$T_2 = \left\{a^{n+i}ba^j, i \in [\![0, 2r-1]\!], j \in [\![1, k-1]\!], j \neq r\right\}$$

On veut également ne pas avoir x_i et $\overline{x_i}$ codés à vrai, pour $i \in [\![0, r-1]\!]$:

$$T_3 = \left\{a^{n+j}ba^{n+j+r}, j \in [\![0, r-1]\!]\right\}$$

Enfin, pour résoudre le problème SAT, il faut que chaque clause C_i ait un littéral vrai :

$$S_1 = \left\{a^ibb, i \in [\![0, n-1]\!]\right\}$$

En définissant $S = S_0 \cup S_1$ et $T = T_0 \cup T_1 \cup T_2 \cup T_3$, on a bien des langages finis disjoints.

La réduction proposée est bien polynomiale ; en effet, S_0 est linéaire, T_0, T_3 et S_1 quadratiques et T_1 et T_2 cubiques en la taille de l'entrée.

Un algorithme qui résout le problème de séparation de S et T avec k états donne une solution de SAT. Par ce qui précède, il est donc NP-dur, donc NP-complet.

□