TD1 - Conception et vérification de programmes

Bastien Maubert

1 Démineur est NP-complet

Nous allons prouver que le jeu de démineur est essentiellement NP-complet. Pour cela nous considérons le *Problème du Démineur* : étant donnée une grille de $m*n$ cases, chacune pouvant être blanche, être marquée comme minée $(*)$ ou contenir un chiffre entre 0 et 8, décider s'il existe une disposition des mines dans les cases blanches qui corresponde aux informations disponibles. En d'autres termes, les données fournies sont-elles consistantes ?

$Q1$) : Montrer que ce problème est dans NP.

Pour montrer que le problème du Démineur est NP-dur, nous le réduisons au problème SAT. L'idée est, étant donnée une instance Φ du problème SAT vue comme un circuit booléen, de construire une instance du problème de démineur qui simule ce circuit.

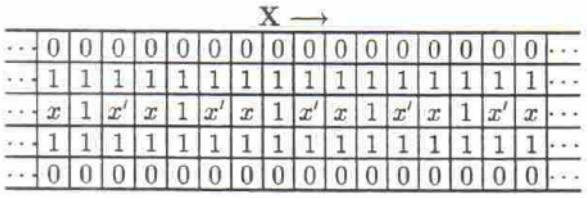

FIGURE 1 – Un fil

La figure 1 montre comment simuler un fil. Il y a exactement deux solutions possibles (en supposant que ça se passe bien aux extrémités) : soit tous les x soit tous les x' sont des mines. On prendra la convention que, étant donnée une orientation du fil, si les cases blanches *avant* les lignes de 1 sont des mines, le fil porte la valeur *vrai*, si ce sont les cases blanches *après* les lignes de 1, il porte la valeur *faux*.

$Q2$) : Donner un gadget permettant de commencer/terminer un fil sans contraindre la valeur. Puis donner un gadget permettant de commencer/terminer un fil en forçant la valeur.

$Q3$) : Donner un gadget permettant de faire tourner un fil à angle droit, et un gadget permettant de diviser un fil en trois.

$Q3$) : Donner un gadget permettant de simuler une porte NON.

On suppose maintenant qu'on a un moyen de simuler la porte ET, et de faire se croiser des fils. On peut donc réaliser n'importe quel circuit logique.

$Q4$) : Soit l'instance du problème SAT Φ donnée en forme normale conjonctive. Décrire comment construire une instance du problème du Démineur qui soit consistante si et seulement si la formule est satisfiable.

Plus de details : voir [1]

2 L'universalité des expressions régulières est PSPACE-complet

On dit qu'une expression régulière e sur un alphabet A est *universelle* si le langage qu'elle dénote est universel, *i.e* $\mathcal{L}(e) = A^*$.

Dans cet exercice on prouve que le problème de décider l'universalité d'une expression rationelle est PSPACE-complet.

$Q1$) : Montrer que ce problème est dans PSPACE (rappel : PSPACE = NPSPACE d'après le théorème de Savitch).

Pour montrer la borne inférieure, on montre directement (méthode force brute) que tout problème PSPACE peut être codé en temps polynomial vers ce problème.

Soit $\mathcal{M} = (\Sigma, Q, q_0, q_{acc}, q_{rej}, \delta)$ une machine de Turing déterministe fonctionnant en espace $f(n)$, où f est un polynôme. Soit x le mot d'entrée, $n = |x|$.

On va construire une expression régulière e qui représente l'ensemble des mots qui *ne codent pas* une exécution acceptante de la machine $\mathcal{M}$ sur x. Ainsi, on aura $\mathcal{L}(e) = A^*$ ssi aucun mot ne code une exécution acceptante, ce qui équivaut à dire que x est rejeté par $\mathcal{M}$.

$Q2$) : Comment encoder une exécution de la machine par un mot sur un alphabet A ?

$Q3$) : Donner une expréssion régulière qui dénote l'ensemble des mots qui ne codent pas une exécution acceptante.

Plus de détails : voir [2]

Références

[1] R. Kaye. Minesweeper is np-complete. *The Mathematical Intelligencer*, 22(2) :9–15, 2000.

[2] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE, 1972.