Université Bordeaux 1. Master Sciences & Technologies, Informatique.
Examen UE INF 461, Modèles de calcul.
Session 1 2008–2009. 17 décembre 2008. Durée : 3h00.
Manuscrits et polycopiés autorisés, livres interdits.
La notation tiendra compte du soin apporté à la rédaction.

Exercice 4 On considère le problème MOT-CONCORDANT suivant.

Donnée Des mots $u_1, \ldots, u_p$ sur l'alphabet $\{0, 1, *\}$, tous de même taille n.

Question Trouver s'il existe un mot y sur l'alphabet $\{0, 1\}$, également de taille n, qui concorde avec chaque mot u_i sur au moins une position. C'est-à-dire, si on note $u_i = a_{i,1} \cdots a_{i,n}$, qu'on demande l'existence d'un mot $y = y_1 \cdots y_n$ tel que :

(1) Pour toute position k entre 1 et n : $y_k = 0$ ou $y_k = 1$, et
(2) Pour tout i entre 1 et p : il existe une position ℓ telle que $y_\ell = a_{i,\ell}$.

1. Le problème a-t-il une solution sur l'instance $u_1 = 01*$, $u_2 = **0$, $u_3 = 1*0$, $u_4 = *1*$?

   Solution. Oui : 010 ou 110.

2. Le problème a-t-il une solution sur l'instance $u_1 = 01*$, $u_2 = **0$, $u_3 = 1*1$, $u_4 = *1*$?

   Solution. Oui : 110.

3. Décrire un algorithme qui vérifie en temps polynomial (par rapport à $p.n$) si un mot y donné est une solution du problème MOT-CONCORDANT. Peut-on conclure que le problème appartient à la classe P ? à la classe NP ?

   Solution. Algorithme en pseudo-langage (le mot u_i étant stocké dans le tableau `u[i]`) :

```
Pour i ← 1 à p faire
  # test que y et u[i] coïncident sur une position.
  ok ← Faux
  Pour j ← 1 à n faire
    Si y[j] == u[i][j] alors ok ← true
  FinPour
  Si ok == Faux alors Retourner Faux
FinPour
Retourner Vrai
```

   Il y a deux boucles imbriquées, la complexité est $O(np)$. On déduit que l'on peut **vérifier** en temps polynomial qu'un mot y est bien solution du problème MOT-CONCORDANT, donc le problème est dans la classe NP : on devine le mot y en temps $O(n)$ et on vérifie en temps $O(np)$ que c'est bien une solution. Par contre, cela ne prouve pas que le problème soit dans la classe P : pour obtenir un algorithme à partir de celui ci-dessus, on pourrait générer tous les mots y potentiels, et pour chacun d'eux, tester en temps $O(np)$ s'il est une solution. Mais comme il y a 2^n tels mots, cela donne un algorithme de complexité exponentielle.

On veut construire une réduction polynomiale de 3-SAT à MOT-CONCORDANT. Soit $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_p$ une formule 3-CNF sur n variables $x_1, \ldots, x_n$, avec la convention qu'aucune clause ne contient à la fois une variable et sa négation. Pour chaque clause c_i, on construit un mot $u_i = a_{i,1} \cdots a_{i,n}$. La $k^{\text{ème}}$ lettre $a_{i,k}$ de u_i est définie ainsi :

$$a_{i,k} = \begin{cases} 0 & \text{si } \neg x_k \text{ apparaît dans } c_i, \\ 1 & \text{si } x_k \text{ apparaît dans } c_i, \\ * & \text{si ni } x_k, \text{ ni } \neg x_k \text{ n'apparaissent dans } c_i. \end{cases}$$

4. Montrer comment à partir d'une valuation des variables $x_1, \ldots, x_n$ qui rend φ vraie, on peut construire une solution de MOT-CONCORDANT sur l'instance $u_1, \ldots, u_p$ ainsi définie.

   Solution. Soit $f : \{x_1, \ldots, x_n\} \to \{0, 1\}$ une valuation des variables qui rend φ vraie. Alors le mot $y = f(x_1) \cdot f(x_2) \cdots f(x_n)$ est solution de MOT-CONCORDANT sur la donnée $u_1, \ldots, u_p$. En effet, montrons que chacun des mots u_i coïncide avec y sur au moins une position. On sait que la valuation f rend c_i vraie. C'est donc que :

   (*a*) soit c_i contient un littéral x_k et $f(x_k) = 1$,

   (*b*) soit c_i contient un littéral $\neg x_k$ et $f(x_k) = 0$.

   Mais par définition de u_i, dans le cas (*a*), la k-ème lettre de u_i est 1, c'est donc la même que la k-ème lettre de y qui est $f(x_k)$. De même, dans le cas (*b*), la k-ème lettre de u_i est 0 par définition de u_i, et c'est à nouveau la même que la k-ème lettre de y, qui est $f(x_k)$.

5. Inversement, montrer que si MOT-CONCORDANT a une solution y sur l'instance $u_1, \ldots, u_p$, alors φ est satisfaisable (en interprétant $y = y_1 \cdots y_n$ comme une valuation des variables $x_1, \ldots, x_n$).

   Solution. On doit maintenant définir f à partir de y. On définit $f(x_k) = y_k$, la k-ème lettre de y. On doit montrer que f rend la formule initiale $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_p$ vraie, c'est-à-dire qu'elle rend chaque clause c_i vraie. On sait que y et u_i coïncident sur une position, disons k. On a à nouveau deux cas :

   – Si à la position k on trouve un 1 dans y et u_i, alors $f(x_k) = 1$ par définition de f à partir de y, et c_i contient le littéral x_k par définition de u_i à partir de c_i. Donc la valuation f rend vraie la clause c_i.

   – Si à la position k on trouve un 0 dans y et u_i, alors $f(x_k) = 0$ par définition de f à partir de y, et c_i contient le littéral $\neg x_k$ par définition de u_i à partir de c_i. Donc à nouveau, la valuation f rend vraie la clause c_i.

6. Que déduit-on des questions précédentes concernant le problème MOT-CONCORDANT ?

   Solution. D'après la question 3, on sait que MOT-CONCORDANT est dans la classe NP.

   Les questions 4 et 5 donnent une réduction polynomiale de 3-SAT à MOT-CONCORDANT : à partir d'une donnée φ de 3-SAT, on a bien construit, en temps polynomial, une donnée $u_1, \ldots, u_p$ de MOT-CONCORDANT telle que :

   3-SAT a une réponse positive sur φ
   $\iff$
   MOT-CONCORDANT a une réponse positive sur $u_1, \ldots, u_p$.

Donc 3-SAT $\leqslant_p$ MOT-CONCORDANT, et comme 3-SAT est NP-complet, il en est de même de MOT-CONCORDANT.

Exercice 5 1. Montrer que la fonction $f(x, y) = y^x$ est primitive récursive (avec par convention : $0^0 = 1$). On pourra utiliser le fait que la multiplication est primitive récursive.

Solution. On a $f(0, y) = 1$ et $f(x + 1, y) = y \times f(x, y)$. Comme la fonction constante 1 (qui intervient dans la définition de $f(0, y)$ et la multiplication (qui intervient dans la définition de $f(x + 1, y)$) sont primitives récursives, on en déduit que f l'est également : on a simplement utilisé le schéma de récursion.

2. Montrer que l'ensemble des fonctions primitives récursives de $\mathbb{N}$ dans $\mathbb{N}$ est dénombrable.

Solution. Toute fonction primitive récursive peut s'écrire à partir de l'ensemble dénombrable des fonctions nulles, projections et successeur en utilisant un nombre fini de fois la composition et la récursion. Cela montre que l'ensemble est dénombrable.

3. En utilisant un argument de diagonalisation, montrer qu'il existe des fonctions totales de $\mathbb{N}$ dans $\mathbb{N}$, calculables, mais non primitives récursives.

Solution. On part d'une numérotation effective $f_0, f_1, f_2, \ldots$ des fonctions primitives récursives de $\mathbb{N}$ dans $\mathbb{N}$. La fonction f définie par $f(n) = 1 + f_n(n)$ est calculable : partant d'un entier n, on détermine f_n (la numérotation est effective), on calcule $f_n(n)$ et on ajoute 1. Mais elle n'est pas primitive récursive. En effet, si elle l'était, elle serait l'une des fonctions de la suite $f_0, f_1, \ldots$, et on aurait donc $f = f_k$ pour un certain k. En particulier $f(k) = f_k(k)$. Mais par définition de f, on a $f(k) = f_k(k) + 1 \neq f_k(k)$. Il est donc impossible que f soit une fonction primitive récursive.