

Parcours : MCSB71–74      Date : 17 Décembre 2007  
 Code UE : INF461      Durée : 3h00  
 Épreuve de : Modèles de calcul - M. Zeitoun

**Exercice 1** Répondre par vrai ou faux aux questions suivantes, sans donner une justification.  
 Barème : +0,5 pour une réponse juste, −0,5 pour une réponse fausse, 0 pour une absence de réponse.

1. L'ensemble des graphes orientés finis est dénombrable.

Vrai : l'ensemble  $\mathcal{G}_{k,n}$  des graphes orientés ayant  $k$  sommets et  $n$  arêtes est fini, et l'ensemble des graphes finis est  $\bigcup_{k=0}^{\infty} \bigcup_{n=0}^{\infty} \mathcal{G}_{k,n}$ . C'est donc une union dénombrable d'ensemble dénombrables.

2. Tout langage fini est décidable.

Vrai : c'est même le cas pour tout langage rationnel, puisqu'un automate fini est, en particulier, une machine de Turing qui s'arrête sur toute entrée.

3. Pour tous  $K, L \subseteq \Sigma^*$  avec  $\Sigma = \{0,1\}$ , si  $K$  se réduit à  $L$ , alors  $\Sigma^* \setminus K$  se réduit à  $\Sigma^* \setminus L$ .

Vrai : si  $K$  se réduit à  $L$ , il existe une fonction calculable  $f : \Sigma^* \rightarrow \Sigma^*$  telle que

$$\forall x \in \Sigma^* \quad x \in K \iff f(x) \in L,$$

ce qui s'écrit aussi

$$\forall x \in \Sigma^* \quad x \in \Sigma^* \setminus K \iff f(x) \in \Sigma^* \setminus L.$$

4. Tout langage sur un alphabet à une lettre est décidable.

Faux. Un langage sur un alphabet à une lettre peut être vu comme une partie de  $\mathbb{N}$  (par exemple,  $\{a, a^3\}$  peut être vu comme la partie  $\{1,3\}$ ). De même,  $(a^2)^*$  correspond à l'ensemble des entiers pairs). Or, l'ensemble de ces parties n'est pas dénombrable, contrairement à l'ensemble des langages décidables sur une lettre.

5. Étant donné une machine de Turing  $M$ , un mot  $w$  et un entier  $k$ , on peut décider si  $M$  accepte  $w$  en au plus  $k$  pas de calcul.

Vrai. Il suffit de tester : lancer  $M$  et de l'arrêter au bout de  $k$  pas de calcul (de toutes les façons possibles si  $M$  est non déterministe).

6. Étant données deux machines de Turing  $M_1$  et  $M_2$ , on peut décider si  $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$ .

Faux. Même lorsque  $M_2$  est une machine fixée qui rejette toute entrée, puisque sinon on saurait décider si  $\mathcal{L}(M_1) = \emptyset$ , ce qui est indécidable (d'après le théorème de Rice, par exemple).

7. Il existe une infinité de fonctions récursives totales qui ne sont pas primitives récursives.

Vrai. L'ensemble des fonctions récursives primitives est dénombrable : soit  $f_0, f_1, f_2, \dots$  une énumération effective de cet ensemble. Pour  $m > 0$ , la fonction  $g_m$  définie par  $g_m(k) = f_k(k) + m$  est clairement calculable. Si elle était récursive primitive, on aurait  $g_m = f_\ell$  pour un certain  $\ell$ , donc  $g_m(\ell) = f_\ell(\ell) = f_\ell(\ell) + m$ , ce qui est impossible. Chacune des fonctions  $g_m$  est donc une fonction récursive totale mais n'est pas primitive récursive. Enfin, comme  $g_m = g_{m-1} + 1$ , ces fonctions sont deux à deux différentes.

8. Le langage des mots sur l'alphabet ASCII représentant un programme C syntaxiquement correct est décidable.

Vrai. Ce que fait un compilateur C est justement de décider si un mot (un programme) appartient à ce langage, c'est-à-dire représente un programme syntaxiquement correct.

9. Le complément de tout ensemble récursivement énumérable est aussi récursivement énumérable.  
Faux. Sinon, tout ensemble récursivement énumérable serait décidable, ce qui n'est pas le cas.
10. Pour tout entier  $k \geq 1$ , il existe une bijection primitive récursive de  $\mathbb{N}^k$  dans  $\mathbb{N}$ .  
Vrai. La fonction  $c_2 : \mathbb{N}^2 \rightarrow \mathbb{N}$  définie par  $c_2(i, j) = \frac{(i+j)(i+j+1)}{2} + i$  est une bijection primitive récursive, et on peut généraliser à tout  $k$ . Par exemple, on peut définir  $c_3(x, y) = c_2(x, c_2(y, z))$ , etc.

**Exercice 2** Dans cet exercice les entiers seront codés en binaire. On pourra utiliser les machines usuelles de manipulation d'entiers en binaire (comparaisons, opérateurs arithmétiques,...) sans en refaire les constructions. On note  $L \subseteq \{0,1\}^*$  l'ensemble des représentations binaires des entiers non premiers (c'est-à-dire, des entiers  $n \in \mathbb{N}$  qui ont un diviseur différent de  $n$  et de 1).

1. Décrire de façon informelle, mais précisément, une machine de Turing déterministe qui décide le langage  $L$ . Donner une borne supérieure (raisonnable) pour le temps de calcul de cette machine de Turing sur une entrée de taille  $k$ .

Il s'agit juste de retranscrire un algorithme de test de non primalité en termes de machines de Turing. On utilise plusieurs bandes. Sur la 1ère bande, on conserve l'entrée. Sur la 2ème bande, on initialise un entier en binaire à la valeur 2. Tant que cet entier est strictement inférieur à l'entrée (ce qu'on vérifie par une machine de comparaison), on teste s'il divise l'entrée. Si c'est le cas, on sort sur OK, sinon, on incrémente l'entier de la bande 2 et on reprend la boucle *tant que*. Si l'entier de la 2ème bande devient égal à l'entrée, on sort sur KO.

Dans le pire des cas, chaque test de divisibilité échoue. Il y a donc  $2(n-1)$  tests (comparaison et divisibilité), chacun prenant un temps polynomial en la taille  $k$  de la donnée  $n$ . Or, le nombre de bits de l'entier  $n > 0$  est  $k = \lceil \log_2(n+1) \rceil$ . Donc  $n = 2^{O(k)}$ , la complexité est exponentielle.

2. Décrire de façon informelle, mais précisément, une machine de Turing non-déterministe qui décide le langage  $L$  avec une complexité (en temps) polynomiale. Justifier cette complexité.

La machine, de façon non-déterministe, génère un entier avec moins de bits que  $n$  (on dit qu'elle *devine* cet entier). Si cet entier est  $\leq 1$  ou  $\geq n$ , elle répond KO. Sinon, elle répond OK si cet entier divise  $n$  et KO sinon.

Ainsi, la machine répond toujours KO si  $n$  est premier, et répond OK pour au moins un chemin de calcul si  $n$  n'est pas premier. Par définition, elle décide donc la non primalité de façon non déterministe. Cette fois, la machine travaille en temps polynomial, car générer un entier de  $k$  bits consomme  $O(k)$  unités de temps, et le test de divisibilité est polynomial en  $k$ .

**Exercice 3** Montrer que la fonction  $f : \mathbb{N} \rightarrow \mathbb{N}$  définie par  $f(n) = \lfloor \log_2(\sqrt{n^3+1}) \rfloor$  est primitive récursive. Ici,  $\lfloor x \rfloor$  désigne le plus grand entier inférieur ou égal à  $x$ . On pourra utiliser les fonctions primitives récursives et les schémas de construction de fonctions primitives récursives vus en cours.

Par définition,  $k = \lfloor \log_2(\sqrt{n^3+1}) \rfloor$  est l'unique entier tel que  $k \leq \log_2(\sqrt{n^3+1}) < k+1$ , donc le plus petit entier tel que  $2^{k+1} > \sqrt{n^3+1}$ , ce qui est équivalent à  $(2^{k+1}-1)^2 > n^3$ . Par ailleurs, on a clairement  $f(n) \leq n^3+1$ . Finalement,  $f(n) = \min\{k \leq n^3+1 \mid (2^{k+1}-1)^2 > n^3\}$ . Comme les fonctions  $n \mapsto n^3+1$  et  $(k,n) \mapsto (2^{k+1}-1)^2 > n^3$  sont primitives récursives, on en déduit que  $f$  l'est.

Remarque : il est faux de dire que la fonction  $f$  est la composition des fonctions primitives récursives  $n \mapsto \sqrt{n^3+1}$  et  $n \mapsto \log_2(n)$ , car ces fonctions prennent des valeurs non entières.

**Exercice 4** Soit  $\Sigma = \{0,1\}$ . L'ordre *hiérarchique*  $\leq$  est un ordre (total) sur  $\Sigma^*$  défini par :

$$u < v \text{ si } |u| < |v|, \text{ ou } |u| = |v| \text{ et } u <_{\text{lex}} v.$$

où  $<_{\text{lex}}$  est l'ordre lexicographique, avec sur  $\Sigma$  l'ordre  $0 <_{\text{lex}} 1$ .

1. Donner les 5 premiers mots de  $\Sigma^*$ , dans l'ordre hiérarchique.

Les premiers mots dans l'ordre hiérarchique sont  $\varepsilon < 0 < 1 < 00 < 01 < 10 < 11 < 000 < 001 < 010 < 011 < 100 < 101 < 110 < 111$ .

- Donner le principe d'une machine de Turing qui sur l'entrée  $k > 0$  codée en unaire, calcule le  $k$ -ième mot de  $\Sigma^*$  dans l'ordre hiérarchique.

On constate que le successeur d'un mot  $x$  dans l'ordre hiérarchique est  $0^{k+1}$  si  $x$  est de la forme  $1^k$ , et le successeur de  $x$  sinon (en base 2). Le principe d'une machine de Turing calculant le  $k$ -ième mot de  $\Sigma^*$  est donc le suivant. On utilise 2 bandes. La première contient l'entrée  $k$  en unaire, la seconde est initialement vide.

On commence par effacer le premier symbole de  $k$  (ainsi, si  $k = 1$ , on aura calculé le mot vide sur la seconde bande). Ensuite, pour chaque symbole qui apparaît dans  $k$ , on efface ce symbole (on le remplace par  $\square$ ) et on incrémente le mot sur la seconde bande, en binaire, sauf s'il est de la forme  $1^k$ . Dans ce dernier cas on remplace chaque 1 sur la seconde bande par 0, et on ajoute un 0 supplémentaire.

On dit qu'un langage infini  $L \subseteq \Sigma^*$  peut être énuméré en ordre croissant s'il existe une bijection calculable  $f : \mathbb{N} \setminus \{0\} \rightarrow L$  qui à chaque  $n > 0$  associe le  $n$ -ième mot de  $L$  dans l'ordre hiérarchique.

- Montrer que tout langage décidable infini peut être énuméré en ordre croissant.

Pour calculer  $f(n)$ , on génère successivement chaque mot de  $\Sigma^*$  dans l'ordre hiérarchique grâce à la machine de la question 2. Pour chacun de ces mots, on teste s'il est dans  $L$  grâce à une machine de Turing qui décide  $L$ . La valeur de  $f(n)$  sera le  $n$ -ème mot ayant passé ce test avec succès (on utilise un compteur pour savoir combien de mots ont passé le test). Le fait que le langage soit infini garantit qu'on ne peut pas effectuer une suite infinie de tests négatifs, et donc que le calcul termine.

- Montrer que tout langage qui peut être énuméré en ordre croissant est décidable.

Pour décider si un mot donné  $x$  est dans  $L$ , on calcule successivement  $f(0), f(1), f(2), \dots$  jusqu'à ce que

- soit l'une de ces valeurs soit égale à  $x$ . Dans ce cas,  $x$  est dans  $L$  et on sort sur OK.
- soit l'une des valeurs, disons  $f(k)$ , est supérieure à  $x$  dans l'ordre lexicographique. Dans ce cas, on sort sur KO. Ceci arrive nécessairement si  $x \notin L$  car la suite  $(f(n))_n$  est croissante. Réciproquement, si on sort sur KO, alors  $x \notin L$ . En effet, comme  $f$  est croissante, si  $\ell \geq k$ , alors  $f(\ell) \geq f(k) > x$ , donc  $f(\ell) \neq x$ . Et pour  $\ell < k$ , on a aussi  $f(\ell) \neq x$  car sinon on serait sorti sur OK (cas précédent).

**Exercice 5** On dit qu'une machine de Turing  $M$  sur l'alphabet d'entrée  $\Sigma = \{0,1,\dots,9\}$  calcule la constante 2008 si quelle que soit sa donnée, elle s'arrête en ayant sur sa bande les 4 caractères 2008, entourés de  $\square$ .

- Décrire une machine de Turing qui calcule la constante 2008.

Il suffit de faire une machine qui efface sa bande et écrit 2, 0, 0, 8.

- Décrire un algorithme qui, à partir d'une machine de Turing  $M_1$ , construit une machine de Turing  $M$  telle que  $M_1$  accepte le mot vide si et seulement si  $M$  calcule la constante 2008.

La machine  $M$  commence par effacer sa bande, puis lance la machine  $M_1$ . Si  $M_1$  s'arrête sur KO, alors  $M$  continue en bouclant. Si  $M_1$  s'arrête sur OK, alors  $M$  efface sa bande et écrit 2, 0, 0, 8.

Le problème 2008 est le suivant :

**Donnée** Une machine de Turing  $M$  (codée par un mot de  $\{0,1\}^*$ ).

**Question** La machine de Turing  $M$  calcule-t-elle la constante 2008?

- Exprimer précisément le résultat de la question précédente en terme de réduction, en justifiant.

Le problème « étant donnée une machine  $M$ , accepte-t-elle le mot vide ? » se réduit au problème 2008.

- Montrer que le problème 2008 est indécidable, en justifiant.

Le problème « étant donnée une machine  $M$ , accepte-t-elle le mot vide ? » est indécidable d'après le théorème de Rice, car il s'écrit : étant donnée une machine de Turing  $M$ , est-ce que  $\varepsilon \in \mathcal{L}(M)$ , et

la propriété « contenir  $\varepsilon$  » n'est pas une propriété triviale des langages récursivement énumérables. Comme ce problème se réduit au problème 2008 (qu.3), il est lui-même indécidable.

**Remarque** On ne peut pas appliquer directement le théorème de Rice, car *écrire quelque chose sur sa bande* n'est pas une propriété du langage accepté par une machine de Turing.

**Exercice 6** On rappelle que le problème de correspondance de Post (PCP en abrégé) est le suivant.

**Donnée**  $n$  paires de mots  $(x_1, y_1), \dots, (x_n, y_n)$ .

**Question** Existe-t-il une suite d'indices  $i_1, \dots, i_k$  telle que  $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$  ?

On rappelle également que ce problème est indécidable sur un alphabet ayant au moins deux lettres. On se place sur l'alphabet  $\Sigma = \{1, 2, \dots, 9\}$ . On note  $v : \Sigma^* \rightarrow \mathbb{N}$  la fonction qui à un mot associe l'entier ayant ce mot comme représentation en base 10. Par exemple,  $v$  associe au mot 1234 de  $\Sigma^*$  l'entier 1234. On note  $|w|$  la longueur d'un mot  $w$ .

1. Exprimer  $v(xx')$  en fonction de  $v(x)$ ,  $v(x')$  et  $|x'|$ .

On calcule facilement :  $v(xx') = 10^{|x'|}v(x) + v(x')$ .

On considère la fonction  $f$  qui à un couple de mots  $(x, y)$  associe la matrice  $\begin{pmatrix} 10^{|x|} & 10^{|x|} - 10^{|y|} & 0 \\ 0 & 10^{|y|} & 0 \\ v(x) & v(x) - v(y) & 1 \end{pmatrix}$ .

2. Montrer que  $f(x, y)f(x', y') = f(xx', yy')$  pour tous mots  $x, x', y, y'$  sur l'alphabet  $\Sigma$ .

Il suffit de faire le calcul en utilisant que  $|xx'| = |x| + |x'|$  et donc  $10^{|x|} \cdot 10^{|x'|} = 10^{|x|+|x'|} = 10^{|xx'|}$ .

On rappelle que si  $A = (a_{i,j})_{1 \leq i, j \leq 3}$  et  $B = (b_{i,j})_{1 \leq i, j \leq 3}$  sont des matrices  $3 \times 3$ , leur produit  $C = (c_{i,j})_{1 \leq i, j \leq 3}$  est donné par  $c_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + a_{i,3}b_{3,j}$ .

3. Soit  $(x_1, y_1), \dots, (x_n, y_n)$  une instance du PCP, et soit  $M_i = f(x_i, y_i)$  pour  $i = 1, \dots, n$ . Déduire de la question précédente que  $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$  si et seulement si la composante (3,2) de la matrice  $M_{i_1} \cdots M_{i_k}$  est nulle.

D'après la question précédente, la composante (3,2) de  $M_{i_1} \cdots M_{i_k}$  est  $v(x_{i_1} \cdots x_{i_k}) - v(y_{i_1} \cdots y_{i_k})$ . Compte tenu du fait qu'on n'a pas 0 dans l'alphabet,  $v$  est injective, et donc cette composante est nulle si et seulement si  $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$ .

4. En déduire que le problème suivant est indécidable, en justifiant soigneusement.

**Donnée**  $n$  matrices  $M_1, \dots, M_n$  carrées  $3 \times 3$ , à coefficients dans  $\mathbb{Z}$ .

**Question** Existe-t-il un produit de la forme  $M_{i_1} \cdots M_{i_k}$  dont la composante (3,2) est nulle ?

Notons P le problème ci-dessus. On a montré que le PCP a une solution sur l'instance  $(x_1, y_1), \dots, (x_n, y_n)$  si et seulement si le problème P a une solution sur l'entrée  $M_1, \dots, M_n$ . On a donc  $\text{PCP} \leq P$  : le PCP se réduit à P. Comme le PCP est indécidable, ce problème l'est aussi.

**Exercice 7** On représente l'entier  $n$  par le  $\lambda$ -terme  $[n] = \lambda s z . s^n z$ , où  $s^0 z = z$  et  $s^{n+1} z = s(s^n z)$ . On rappelle que les Booléens sont conventionnellement représentés par  $[T] = \lambda x y . x$  (true) et  $[F] = \lambda x y . y$  (false). Soit  $Z = \lambda a (a (\lambda b [F]) [T])$ . Vérifier que  $Z[0] \xrightarrow{\beta}^* [T]$ , et  $Z[n] \xrightarrow{\beta}^* [F]$  si  $n \neq 0$ .

Fait en TD.