Master informatique, semestre 1
Modèles de calcul, devoir 2003

Exercice 1

Un corrigé est disponible ci-dessous.

La méthode de la *réduction* (polynomiale) d'un problème à un autre est essentielle en théorie de la *complexité*. Elle peut servir aussi en théorie de la *calculabilité* (bien entendu sans rien supposer sur la complexité de la réduction). On sait (théorème de Rice) que l'ensemble des procédures p qui calculent l'identité (autrement dit telles que, pour tout x : $p(x) = x$), est indécidable ; cet exercice montre, en utilisant des réductions, qu'il n'est pas récursivement énumérable, et son complémentaire non plus.

Par définition, on dira, dans cet exercice, qu'un problème de décision A se réduit au problème de décision B s'il existe une fonction f *calculable totale* telle que :

$$A\,(x) = 1 \text{ si et seulement si } B\,(f\,(x)\,) = 1$$

(A et B sont des prédicats, et les booléens sont notés 0 et 1) ; pour les trois premières questions, on suppose que A se réduit à B, et on note E, F les ensembles définis par :

$$E = \{\ x \mid A\,(x) = 1\ \},\ F = \{\ x \mid B\,(x) = 1\ \}$$

(autrement dit, A et B sont les fonctions caractéristiques des ensembles E et F).

1. Montrer que si F est décidable, E l'est aussi.
2. Montrer que si F est récursivement énumérable, E l'est aussi.
3. Expliquer comment formuler les résultats précédents lorsqu'on veut prouver qu'un ensemble est indécidable, ou non récursivement énumérable. *Note* : cette question est volontairement formulée de façon vague, la réponse est très courte, mais indispensable pour résoudre les questions 5 et 7.

On considère la procédure suivante :

```
int gq (int p (int), int x, int n) {
  int y = p(x);
  return n;
}
```

On note $q = f\,(\,p, x)$ la fonction définie par : $q(n) = gq\,(\,p, x, n)$.

4. Montrer que f réduit le problème de la *terminaison* du calcul de $p(x)$ à celui de savoir si q calcule l'identité.
5. En déduire que l'ensemble des procédures qui *ne* calculent *pas* l'identité n'est pas récursivement énumérable. *Attention* : la réponse correcte est courte, mais délicate ; seul un raisonnement rigoureux rapportera des points. *Indication* : vérifier d'abord (c'est facile) que si un problème A se réduit à B, le complémentaire de A se réduit au complémentaire de B.

On remplace gq par une procédure gr, où la variable t représente le temps ; on rappelle que h désigne la fonction calculable *totale* telle que $h\,(\,p, x, t\,) = 1$ si et seulement si le calcul $p(x)$ est terminé au temps t :

```
int gr (int p (int), int x, int t) {
  if (h (p, x, t)) return p(x);
  else return t;
}
```

On note $r = g\,(\,p, x)$ la fonction définie par : $r\,(\,t\,) = gr\,(\,p, x, t\,)$.

6. Montrer que g réduit le problème de la terminaison du calcul de $p(x)$ à celui de savoir si r *ne* calcule *pas* l'identité.
7. En déduire que l'ensemble des procédures qui calculent l'identité n'est pas récursivement énumérable.

Exercice 2

Un corrigé est disponible ci-dessous.

Un problème A appartient à la classe ***NP*** s'il existe un *vérifieur* polynomial V pour A, c'est-à-dire une procédure polynomiale V telle que :

$$A\,(u) = 1 \text{ si et seulement si : il existe } x \text{ tel que } V\,(u, x) = 1.$$

Le théorème de Cook et Levin affirme qu'on peut alors construire en temps polynomial, à partir de l'argument u, une formule booléenne Φ telle que :

$$A\,(u) = 1 \text{ si et seulement si } \Phi \text{ est satisfiable.}$$

La construction de Φ s'effectue en traduisant les calculs du vérifieur par des opérations booléennes (ce qui a lieu pendant l'exécution matérielle du vérifieur par les circuits du processeur) ; le but de cet exercice est d'effectuer cette construction dans le cas particulier du problème SUBSET-SUM, qui consiste à savoir si un entier t est somme d'un sous-ensemble d'entiers choisis dans un ensemble donné $E = \{ u_1, u_2, \ldots u_n \}$.

Pour que les formules puissent être écrites explicitement, on considère dans la suite l'exemple :

$$E = \{7, 12, 17, 20, 26\}$$

si bien que tous les calculs pourront être réalisés en codant chaque entier sur 7 bits (expliquer brièvement pourquoi).

1. Ecrire la formule "$x = y$" (qui est vraie si et seulement si les deux variables booléennes x et y ont même valeur) sous forme normale conjonctive (conjonction de clauses).
2. Un additionneur utilise comme composante élémentaire un circuit qui, à partir de trois bits x, y, z, calcule un bit s (somme) et un bit r (retenue). Par exemple si $x = 1$, $y = 0$, et $z = 1$, on a $s = 0$ et $r = 1$. Ecrire les formules booléennes qui définissent s et r en fonction de x, y et z.
3. Ecrire une formule booléenne $\sigma\,(x, y, z, s, r)$ qui soit vraie si et seulement si s et r sont la somme et la retenue décrites à la question précédente. *Note* : la nouveauté de cette question est que σ est construite à partir des opérateurs booléens ordinaires, sans utiliser l'égalité. La réponse doit être donnée, si possible, en forme normale conjonctive ; il existe une réponse avec 14 clauses. Si cette question ne vous amuse pas, n'hésitez pas à passer à la suivante…
4. On considère trois nombres a, b et c, codés chacun sur 7 bits ; on les représente donc par 21 variables booléennes : $a_0 \ldots a_6$, $b_0 \ldots b_6$, $c_0 \ldots c_6$. A l'aide de σ (cf. question précédente), construire une formule S, comportant les 21 variables booléennes précédentes, et 6 variables $r_1 \ldots r_6$ pour les retenues intermédiaires, de telle sorte que S soit *satisfiable* si et seulement si $a + b = c$ (sans débordement).
5. On considère six nombres $u_1 \ldots u_5$ et t, codés chacun sur 7 bits ; on les représente donc par 42 variables booléennes. A l'aide de S (cf. question précédente), construire une formule T, comportant les 42 variables booléennes précédentes, et des variables supplémentaires (combien ?) pour les sommes partielles, de telle sorte que T soit *satisfiable* si et seulement si $u_1 + \ldots + u_5 = t$ (sans débordement).
6. Reste à coder le choix d'un sous-ensemble de E par cinq variables booléennes $x_1 \ldots x_5$. Dans la représentation binaire de u_i , on remplace alors chaque bit égal à 1 par la variable x_i ; ainsi u_i est annulé s'il ne fait pas partie du sous-ensemble choisi. A l'aide de T (cf. question précédente), construire une formule Φ comportant les 5 variables booléennes $x_1 \ldots x_5$, les 7 variables booléennes $t_0 \ldots t_6$ qui codent t, et des variables supplémentaires (combien ?), de telle sorte que Φ soit *satisfiable* si et seulement si t est somme d'un sous-ensemble de $\{7, 12, 17, 20, 26\}$.
7. Donner une estimation précise du nombre de variables booléennes et du nombre de clauses

composant Φ lorsque l'ensemble E comporte n nombres de taille k.

Corrigé

Exercice 1

Voir l'énoncé ci-dessus.

1. Par hypothèse on a, pour tout x : $A\,(x) = B\,(f\,(x))$; dire que F est décidable équivaut à dire que B est *calculable* ; donc A, fonction *composée* de deux fonctions calculables, est calculable, ce qui signifie que E est décidable.
2. On sait que récursivement énumérable équivaut à semi-décidable ; donc par hypothèse la fonction *semi-caractéristique* de F est calculable ; appelons-la B', et montrons comme dans la question précédente que, pour tout x : $A'\,(x) = B'\,(f\,(x))$, où A' désigne la fonction semi-caractéristique de E, ce qui prouvera que E est semi-décidable, autrement dit récursivement énumérable. Pour cela distinguons deux cas :
   - si $x \in E$, alors $A'\,(x) = A\,(x) = 1$, de même que $B'\,(f\,(x)) = B\,(f\,(x)) = 1$, car $f\,(x) \in F$;
   - sinon $A'\,(x)$ et $B'\,(f\,(x))$ sont tous deux indéfinis.
3. En prenant les *contraposés* des résultats précédents, on obtient :
   - E non décidable $\Rightarrow$ F non décidable ;
   - E non récursivement énumérable $\Rightarrow$ F non récursivement énumérable.
4. Appelons H le problème de terminaison : $H\,(p, x) = 1$ si et seulement si le calcul de $p\,(x)$ termine ; et appelons I le problème défini par : $I\,(q) = 1 \Leftrightarrow \forall n\ q\,(n) = n$. Si $H\,(p, x) = 1$, la procédure q exécute l'instruction `return n`, donc $I\,(q) = 1$. Si au contraire $H\,(p, x) = 0$, la procédure q ne termine pas, à cause de l'instruction `y = p(x)` ; donc q calcule la fonction *jamais définie*, et $I\,(q) = 0$. On a donc bien montré l'*équivalence* :
   $$H\,(p, x) = 1 \Leftrightarrow I\,(q) = 1\ .$$
5. L'équivalence $A\,(x) = 1 \Leftrightarrow B\,(f\,(x)) = 1$ peut aussi s'écrire $A\,(x) = 0 \Leftrightarrow B\,(f\,(x)) = 0$, puisque A et B sont des prédicats ; donc si f réduit A à B, f réduit aussi $\neg A$ à $\neg B$. La question précédente réduit donc $\neg H$ à $\neg I$; or on sait que l'ensemble défini par $\neg H$, à savoir :
   $$\{\ (p, x)\ |\ p\,(x) \text{ ne termine pas}\ \}$$
   n'est pas récursivement énumérable ; les questions 2 et 3 montrent donc que l'ensemble défini par $\neg I$, à savoir :
   $$\{\ q\ |\ q \text{ ne calcule pas l'identité}\ \}$$
   n'est pas récursivement énumérable. *Note* : si la question 3 n'a pas été traitée, il suffit de conclure ici par un raisonnement par l'absurde, ce qui revient au même que d'utiliser le contraposée d'une implication.
6. Si $H\,(p, x) = 1$, alors il existe t tel que `h(p, x, t)` est vrai, et donc pour t suffisamment grand, la fonction calculée par r est constante (sa valeur est $p\,(x)$, indépendamment de t ; on aurait pu d'ailleurs faire suivre `return` de n'importe quelle constante : 0, 1, 2…) ; on en déduit que r ne calcule pas l'identité, soit $I\,(r) = 0$. Si au contraire $H\,(p, x) = 0$, alors pour tout t `h(p, x, t)` est faux, la procédure q exécute l'instruction `return t`, et donc $I\,(q) = 1$.
7. On a réduit H à $\neg I$, ce qui revient au même que de réduire $\neg H$ à I. Par le même raisonnement que dans la question 5, on en déduit que l'ensemble des procédures qui calculent l'identité n'est pas récursivement énumérable.

Exercice 2

Voir l'énoncé ci-dessus.

1. $(\neg x \vee y) \wedge (x \vee \neg y)$ est la conjonction de deux clauses, vraie si seulement si x et y ont même

valeur ; on peut remarquer que la première clause représente l'implication $x \Rightarrow y$, tandis que la seconde représente l'implication inverse $y \Rightarrow x$.

2. Additionneur de 3 bits (s désigne la somme, r la retenue).

x	y	z	s	r
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

La table ci-contre fournit les formules suivantes, en forme normale disjonctive :

$$s = (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge y \wedge z)$$

$$r = (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge \neg z) \vee (x \wedge y \wedge z)$$

Pour la retenue r, la dernière conjonction peut être regroupée, et simplifiée, avec n'importe laquelle des précédentes, d'où :

$$r = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$$

On a des formules symétriques, en forme normale conjonctive :

$$s = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$

$$r = (x \vee y) \wedge (x \vee z) \wedge (y \vee z)$$

Ces formules expriment que la somme vaut 1 si un ou trois opérandes valent 1, tandis que la retenue vaut 1 si au moins deux des opérandes valent 1 ; elles illustrent que des conditions simples, comme celles-ci, ont une traduction simple (quoique non évidente), par des expressions booléennes.

3. Chaque ligne de la table ci-dessus peut être interprétée comme une implication ; par exemple la ligne 6 indique que, pour $xyz = 101$, alors $s = 0$ et $r = 1$; autrement dit :

$$(x \wedge \neg y \wedge z \Rightarrow \neg s) \wedge (x \wedge \neg y \wedge z \Rightarrow r)$$

En remplaçant l'implication $p \Rightarrow q$ par sa définition $\neg p \vee q$, et en appliquant une loi de Morgan, la première formule devient une *clause* :

$$\neg x \vee y \vee \neg z \vee \neg s$$

On obtient ainsi les huit premières clauses de σ :

$$(x \vee y \vee z \vee \neg s) \wedge (x \vee y \vee \neg z \vee s) \wedge (x \vee \neg y \vee z \vee s) \wedge (\neg x \vee y \vee z \vee s) \wedge$$
$$\wedge (x \vee \neg y \vee \neg z \vee \neg s) \wedge (\neg x \vee y \vee \neg z \vee \neg s) \wedge (\neg x \vee \neg y \vee z \vee \neg s) \wedge (\neg x \vee \neg y \vee \neg z \vee s)$$

La première clause indique que si aucune des trois variables x, y, z n'a la valeur 1, alors $s = 0$; les trois suivantes indiquent que si l'une des variables x, y, z vaut 1, et une seulement, alors $s = 1$; les trois suivantes indiquent que si deux des variables x, y, z valent 1, et deux seulement, alors $s = 0$; on laisse le lecteur deviner le sens de la dernière clause…

On peut obtenir de la même façon huit clauses pour la retenue r, mais dans ce cas les huit implications peuvent se réduire à six, grâce aux mêmes simplifications que pour la question précédente ; trois implications du type $x \wedge y \Rightarrow r$, qui expriment que la retenue vaut 1 dès que deux des variables x, y, z valent 1 ; et trois implications du type $\neg x \wedge \neg y \Rightarrow \neg r$, qui expriment que la retenue vaut 0 dès que deux des variables x, y, z valent 0 ; d'où les six dernières clauses qui composent la formule σ :

$$(\neg x \vee \neg y \vee r) \wedge (\neg x \vee \neg z \vee r) \wedge (\neg y \vee \neg z \vee r) \wedge (x \vee y \vee \neg r) \wedge (x \vee z \vee \neg r) \wedge (y \vee z \vee \neg r)$$

On a bien obtenu 14 clauses. La formule σ comporte 5 variables, donc sa table de vérité comporte 32 lignes, dont 8 pour lesquelles $\sigma = 1$, et 24 pour lesquelles $\sigma = 0$; en examinant ces dernières, certains étudiants ont réussi à en regrouper 22 deux par deux, d'où 11 clauses avec 4 variables, deux clauses restantes avec 5 variables, et 13 clauses au total ; mais la formule obtenue est légèrement plus longue que celle ci-dessus (dont certaines clauses sont plus

courtes). Cet exemple illustre que minimiser le nombre de clauses, ou minimiser la taille de la formule, est un problème complexe.

4. Dans la question précédente r désigne un seul bit ; à partir de maintenant r désigne une suite de six variables booléennes r_i ; on a l'équivalence suivante :

$$a + b = c \Leftrightarrow \exists r\, S(a, b, c, r) = 1 ,$$
$$\text{avec } S(a, b, c, r) =$$
$$\sigma(a_0, b_0, 0, c_0, r_1) \wedge \sigma(a_1, b_1, r_1, c_1, r_2) \wedge \ldots \wedge \sigma(a_6, b_6, r_6, c_6, 0).$$

Le premier zéro exprime que la retenue initiale est nulle, et le dernier zéro exprime l'absence de débordement. S'il existe un vecteur r (composé de six retenues) qui satisfait S, celui-ci est certainement unique, pour a, b et c donnés.

Exemple : avec $a = 7$ (0000111 en binaire), et $b = 12$ (0001100 en binaire), S devient :

$$\sigma(1, 0, 0, c_0, r_1) \wedge \sigma(1, 0, r_1, c_1, r_2) \wedge \sigma(1, 1, r_2, c_2, r_3) \wedge \ldots \wedge \sigma(0, 0, r_6, c_6, 0)$$

- Les huit premières clauses qui composent $\sigma(1, 0, 0, c_0, r_1)$ sont toutes satisfaites, sauf $\neg x \vee y \vee z \vee s$, qui se réduit à s, d'où $c_0 = 1$; les six dernières clauses sont toutes satisfaites, sauf $\neg r \vee y \vee z$, qui se réduit à $\neg r$, d'où $r_1 = 0$;
- $\sigma(1, 0, r_1, c_1, r_2)$ devient donc $\sigma(1, 0, 0, c_1, r_2)$, qui est satisfaite, comme ci-dessus, pour les seules valeurs $c_1 = 1$ et $r_2 = 0$;
- $\sigma(1, 1, r_2, c_2, r_3)$ devient donc $\sigma(1, 1, 0, c_2, r_3)$; les huit premières clauses qui composent cette formule sont toutes satisfaites, sauf $\neg x \vee \neg y \vee z \vee \neg s$ qui se réduit à $\neg s$, d'où $c_2 = 0$; les six dernières clauses sont toutes satisfaites, sauf $r \vee \neg x \vee \neg y$ qui se réduit à r, d'où $r_3 = 1$;
- etc.

Cet exemple illustre comment l'exécution d'un algorithme déterministe (ici c'est l'algorithme élémentaire d'addition, chiffre par chiffre), peut être exprimé par une formule booléenne en forme normale conjonctive ; une fois les arguments donnés (ici les deux entiers a et b à additionner), certaines clauses sont vérifiées ipso facto (ce sont celles qui sont inutiles pour *ce* calcul), tandis que les autres *déterminent progressivement* (car l'algorithme est déterministe) les valeurs des variables booléennes (autres que celles qui représentent les arguments) qui interviennent dans la formule — ici ce sont les variables qui représentent le résultat c de l'addition, et les retenues (variables intermédiaires).

Notons aussi que dans un programme, une seule variable permet de stocker les valeurs *successives* de la retenue ; par contre la formule booléenne contient nécessairement une variable pour chaque valeur de la retenue : r_j , pour $1 \leqslant j \leqslant 6$, désigne la valeur de la retenue lors de l'étape numéro j de l'algorithme, autrement dit à l'*instant j* (avec les notations de la section section 5.3 du cours, r désigne l'un des bits b_i de la mémoire, et r_j sa valeur $b_{i\,j}$ à l'instant j).

5. On a l'équivalence suivante :

$$u_1 + \ldots + u_5 = t$$
$$\Leftrightarrow \exists\, v_1 \cdots v_3\, \rho_1 \cdots \rho_4\, T(u_1, \cdots, u_5, t, v_1, \cdots, v_3, \rho_1, \ldots, \rho_4) = 1 ,$$
$$\text{avec } T(u_1, \ldots, u_5, t, v_1, \ldots, v_3, \rho_1, \ldots, \rho_4) =$$
$$S(u_1, u_2, v_1, \rho_1) \wedge S(v_1, u_3, v_2, \rho_2) \wedge S(v_2, u_4, v_3, \rho_3) \wedge S(v_3, u_5, t, \rho_4)$$

Les v_i désignent trois nombres de 7 bits, utilisés pour les sommes partielles, et les ρ_i désignent quatre vecteurs de retenues (chacun de longueur 6), utilisés par S ; la formule T comporte donc 3 * 7 + 4 * 6 = 45 variables booléennes auxiliaires. S'il existe des valeurs de ces 45 variables

qui satisfont T, celles-ci sont uniques, pour $u_1 \ldots u_5$ et t donnés.

6. $u_1 = 7$, soit 0000111 en binaire, donc on passe de T à Φ en remplaçant les sept variables booléennes qui représentent u_1 par x_1 , x_1 , x_1 , 0, 0, 0, 0 (on commence par le bit d'indice 0, qui est le bit de poids le plus faible, ou encore le bit le plus à droite dans la représentation binaire) ; de même $u_2 = 12$, soit 0001100 en binaire, et les sept variables booléennes suivantes de T sont remplacées par 0, 0, x_2 , x_2 , 0, 0, 0 ; etc. Pour le reste Φ possède les mêmes variables booléennes que T, soit sept variables pour représenter t, et 45 variables auxiliaires ; on aura donc :

$$t \text{ est somme d'un sous-ensemble de } \{7, 12, 17, 20, 26\} \Leftrightarrow \exists x\, v\, \rho\, \Phi\, (x, t, v, \rho) = 1$$

où x désigne une suite de sept variables booléennes, v une suite de 21 variables booléennes, et ρ une suite de 24 variables booléennes (t est une suite de sept paramètres booléens, qui n'apparaissent donc pas derrière le quantificateur existentiel).

7. Variables booléennes composant Φ (à l'exception de t, qui est un paramètre) : n variables pour représenter x, k (n - 2) variables pour représenter v (n - 2 sommes intermédiaires), (k - 1)(n - 1) variables pour représenter ρ (n - 1 vecteurs de retenues), soit :

$$n + k\,(n - 2) + (k - 1)(n - 1) = 2kn - 3k + 1.$$

Pour $k = 7$ et $n = 5$, on retrouve bien 50 variables booléennes (45 présentes dans T, et $x_1 \ldots x_5$). La formule Φ est la conjonction, comme T, de n - 1 occurrences de S, et chaque formule S est la conjonction de k occurrences de σ, formées chacune de 14 clauses ; dont le nombre de clauses de Φ vaut :

$$14\,k\,(n - 1) \text{ — soit 392 pour } k = 7 \text{ et } n = 5.$$

La taille des données du problème SUBSET-SUM, $u_1 \ldots u_n$ et t, vaut $k\,(n + 1)$, donc la taille de Φ est essentiellement *proportionnelle* à la taille des données.