Université BORDEAUX I le 11 décembre 1997

Maîtrise d'Informatique
Calculabilité et Complexité – Corrigé du devoir à la maison

Exercice 1 *Répondre par "oui" ou par "non" aux questions suivantes. Justifier vos réponses (mais brièvement !).*

a) La fonction universelle $U(x; m)$ est primitive récursive.

NON : toute primitive récursive est une fonction totale, ce qui n'est pas le cas de la fonction universelle.

> **Remarque.** La fonction universelle étant récursive (d'après le théorème d'équivalence, les fonctions calculables sont les fonctions récursives), l'important était la distinction entre "récursive" et "primitive récursive": invoquer le problème d'arrêt (ou n'importe quel argument d'indécidabilité) était hors de propos.

b) Le complément d'un ensemble fini est toujours décidable.

OUI : tout ensemble fini est décidable, et le complémentaire d'un ensemble décidable est décidable.

c) Il existe une machine de Turing qui calcule le n-ième nombre premier (en fonction de n).

OUI : l'ensemble des nombres premiers est décidable, il existe donc un algorithme pour l'énumérer de façon croissante; d'après la thèse de Church, on peut trouver une machine de Turing réalisant cet algorithme. (Note : l'invocation de la thèse de Church n'est pas vraiment nécessaire; il n'est pas très difficile de montrer que la fonction à calculer est primitive récursive.)

> **Remarque.** Certains ont utilisé l'argument suivant: l'ensemble des nombres premiers est décidable, donc énumérable, donc il peut s'écrire $\{p_0, p_1, \ldots, p_n, \ldots\}$ de façon qu'on puisse calculer p_n en fonction de n. L'argument n'est pas recevable: l'énoncé précisait "**le** n-ième nombre premier", ce qui suppose un ordre implicite (l'ordre croissant est assez classique).

d) Le domaine des valeurs de la fonction universelle $U(x; m)$ est décidable.

OUI : c'est $\mathbb{N}$ tout entier (en effet, la fonction identité est calculable, il existe donc un entier x tel que, pour tout entier n, $n = U(x; n)$).

Remarque. Beaucoup utilisent l'argument suivant: $U(N, N)$ est l'ensemble des valeurs d'une fonction non totale, or tout ensemble énumérable est l'ensemble des valeurs d'une fonction totale, donc $U(N, N)$ est non énumérable, a fortiori non décidable.

Raisonnement faux: pour tout ensemble énumérable E **il existe** une fonction totale f telle que $f(N) = E$, mais ça ne prouve pas que **toute** fonction f telle que $f(N) = E$ soit totale. Par exemple, $\{0\}$ est l'ensemble des valeurs de la fonction constante nulle (totale), mais c'est aussi l'ensemble des valeurs de la fonction nulle en 0 et non définie partout ailleurs (qui est strictement partielle).

e) Toute fonction dont le domaine de définition est fini, est algorithmiquement calculable.

OUI : il suffit de programmer le calcul par une instruction "switch".

Remarque. Certains répondent "non", en utilisant l'argument suivant: la fonction f définie sur $[0, 100]$ par $f(n) = 1$ si $U(n, n)$ termine, et $f(n) = 0$ sinon, n'est pas calculable (ceux qui ont des scrupules ajoutent "enfin, ça dépend de la numérotation des fonctions").

C'est l'erreur classique sur la notion de calculabilité: une fonction calculable n'est pas une fonction pour laquelle **on connaît** un algorithme de calcul, mais une fonction pour laquelle **il existe** un algorithme. Même si une fonction f n'est pas calculable, pour tout n fixé, il existe un programme de calcul de $f(n)$ (si f n'est pas définie en n: `while(1);`, et si $f(n) = y$: `return y;`).

f) Tout ce qu'on peut programmer en C, on peut aussi programmer en machine à registres.

OUI : cela découle de la thèse de Church (et de l'équivalence des modèles de calcul). On peut noter que les programmes en langage assembleur produits par les compilateurs C sont à peu près des programmes de machine à registres.

Remarques. Personne n'a ergoté sur les ambiguïtés de C.

J'ai beaucoup apprécié une copie qui disait en substance "si le langage C était plus puissant que les machines à registres, on pourrait jeter à la poubelle toute la théorie de la calculabilité."

Exercice 2 *Donner une méthode de numérotation des listes* $(a_1, a_2, \ldots a_n)$, $n \geq 0$, $a_i \in \mathsf{N}$ *qui vérifient les conditions suivantes:*

a) $a_1 \leq a_2 \leq \ldots \leq a_n$;

b) il est interdit d'avoir plusieurs entiers naturels égaux à la fin de la liste (c'est-à-dire, si $n \geq 2$, alors $a_{n-1} < a_n$).

Donner la formule explicite correspondante.

A toute liste $(a_1, a_2, \ldots, a_n)$ vérifiant les conditions (a) et (b), on peut associer la liste

$$f(a_1, a_2, \ldots, a_n) = (a_1, a_2 - a_1, \ldots, a_n - a_{n-1})$$

sans zéro à la fin (sauf si $n = 1$ et $a_1 = 0$).

Réciproquement, à toute liste $(b_1, b_2, \ldots, b_n)$ ($n \geq 0$) sans zéros à la fin (sauf si c'est la liste (0)), on peut associer la liste $(b_1, b_1 + b_2, \ldots, \sum_{1 \leq i \leq n} b_i)$ dont l'image par f est justement $(b_1, b_2, \ldots, b_n)$.

f est ainsi une bijection de l'ensemble des listes vérifiant (a) et (b) sur l'ensemble des listes sur lesquelles est définie la numérotation g de Gödel : $g \circ f$ est donc une numérotation des listes considérées, avec $g \circ f() = 0$, et pour $n \geq 1$

$$g \circ f(a_1, a_2, \ldots, a_n) = 2^{a_1} \times 3^{a_2 - a_1} \times \ldots \times p_n^{a_n - a_{n-1}}$$

(p_n désignant le n-ième nombre premier).

> **Remarque.** Une justification (même brève) de la bijectivité s'imposait.

Exercice 3 *Ecrire un schéma de récursion pour la fonction*

$f(n) = n$-ième chiffre après la virgule du nombre $\log_7 17 = 1.4559836410\ldots$

(ici le "0-ième chiffre après la virgule" est le chiffre 1 avant la virgule). L'utilisation des fonctions déjà définies comme récursives en TD est autorisée.

Pour tout n, $f(n) = g(n) \bmod 10$ où $g(n)$ est la partie entière de $10^n \log_7 17$, c'est-à-dire le plus grand entier x tel que $x \leq 10^n \log_7 17$, soit encore tel que $7^x \leq 17^{(10^n)}$. Comme on a toujours $7^{(2 \times 10^n)} = 49^{(10^n)} > 17^{(10^n)}$, on peut écrire

$$g(n) = \max\{x \leq 2 \times 10^n \;/\; 7^x \leq 17^{(10^n)}\}$$

ce qui donne un schéma de maximisation bornée pour g (la multiplication, l'exponentiation, $\leq$ sont primitives récursives); donc g est primitive récursive, et par conséquent f aussi.

> **Remarques.** L'écueil principal consistait à introduire des fonctions non entières (logarithmes, puissances de 10 négatives...) dans le schéma de récursion proposé.
>
> Ceux qui ont invoqué la maximisation bornée ont généralement omis d'indiquer une borne, erreur rédhibitoire.

Exercice 4 *Soit $A \subseteq \mathsf{N}$ un ensemble décidable. Pour tout $x \in A$ on met dans B le nombre $x - (t^2 + 2t + 3)$ où t est le plus grand entier tel que $t^2 + 2t + 3 \leq x$. L'ensemble B peut-il ne pas être décidable? Justifier votre réponse.*

Si on définit f par $f(x) = x-(t^2+2t+3)$ où t est le plus grand entier tel que $t^2 + 2t + 3 \leq x$, la construction donnée est $B = f(A)$. Par conséquent B est énumérable (il suffit de prendre $b_n = f(a_n)$, où les a_n sont une énumération de A).

f est une fonction affine par morceaux: on a $f(x) = x - 3$ sur $[3, 5]$, $f(x) = x - 6$ sur $[6, 10]$, etc. Intuitivement, pour montrer qu'un nombre y n'est pas dans B, il faudrait montrer que $y + 3$ n'est pas dans $A \cap [3, 5]$, puis que $y + 6$ n'est pas dans $A \cap [6, 10]$, etc. Mais ce procédé ne termine pas, ce n'est donc pas un algorithme.

On est ainsi conduit à conjecturer que B n'est pas nécessairement décidable.

La démonstration de cette conjecture se fait par une technique classique: soit $B = \{b_0, b_1, \ldots, b_n, \ldots\}$ un ensemble énumérable, mais non décidable; on construit une suite croissante a_n telle que $B = f(A)$, où A est l'ensemble des valeurs de la suite a_n.

Il suffit pour cela de poser $a_n = b_n + t_n^2 + 2t_n + 3$ où

$$t_0 = \min\{k \ / \ 2k + 3 > b_0\}$$

et pour $n \geq 1$

$$t_n = \min\{k \geq a_{n-1} \ / \ 2k + 3 > b_n\}$$

De $t_n \geq a_{n-1}$ pour $n \geq 1$, on déduit que la suite a_n est croissante, donc $A = \{a_0, a_1, \ldots, a_n, \ldots\}$, que l'on peut énumérer de façon croissante, est décidable.

D'autre part, pour tout n, on a $t_n^2 + 2t_n + 3 \leq a_n$ et

$$(t_n + 1)^2 + 2(t_n + 1) + 3 = 2t_n + 3 + t_n^2 + 2t_n + 3 > b_n + t_n^2 + 2t_n + 3 = a_n$$

donc t_n est bien le plus grand entier t tel que $t^2+2t+3 \leq a_n$, et par conséquent $b_n = f(a_n)$.

Donc $B = f(A)$.

> **Remarque.** Beaucoup se sont arrêtés à l'analyse du problème, qui permet de formuler une conjecture, mais ne constitue pas une preuve (on n'a pas trouvé d'algorithme de décision, mais on ne peut pas affirmer qu'il n'en existe aucun).

Exercice 5 *Construire une machine de Turing à une bande qui, pour un entier $x \in \mathbf{N}$ représenté en système* ternaire *(c'est-à-dire, sur la base 3), calcule $2x$.*

> **Remarque.** Correctement traité par tous ceux qui ont essayé. A noter qu'un automate sous forme de graphe ou de tableau est plus lisible qu'un système de règles de réécriture.

Exercice 6 *Montrer qu'il existe une fonction* non calculable *$f(n)$ telle que pour tout $m \geq 1$ la fonction*

$$f_m(n) = f(n) \ mod \ m$$

soit calculable et totale.

Pour tout entier n, on pose $f(n) = k!$ si le calcul de $U(n;n)$ termine en k pas de programme, et $f(n) = 0$ si ce calcul ne se termine pas.

La fonction f est totale, et n'est pas calculable (le problème d'arrêt de $U(n;n)$, que l'on sait indécidable, se réduit en effet à $f(n) \neq 0$).

Mais, pour tout entier $m \geq 1$, la fonction f_m, qui est évidemment totale, est calculable : en effet, pour tout $k \geq m$, $k! \bmod m = 0$. Pour calculer $f(n) \bmod m$, il suffit donc de lancer une partie limitée du calcul de $U(n;n)$; si le calcul n'est pas terminé après m étapes, le résultat est 0.

> **Remarque.** La plupart des solutions proposées par les étudiants comportaient un vice de forme: un paramètre m non défini apparaissait dans la définition de f.
>
> Ceux qui s'en sont rendu compte ont écrit: soit m un paramètre fixé, et donné la définition de f, par exemple: $f(n) = n + m$ si $U(n;n)$ termine, et $f(n) = n$ sinon; f n'est pas calculable, mais $f_m(n) = n \bmod m$ est calculable. C'est juste, mais ça ne répond pas aux conditions demandées (f_m doit être calculable pour **tout** m, et non pour **un** m particulier).