Université BORDEAUX 1 — U.F.R. Mathématiques et Informatique
Epreuve de J.Bétréma — 23 janvier 2004

Master d'Informatique
INF 300 - Modèles de calcul

Durée: 3 heures
Tous documents interdits.

Exercice 1

Un corrigé est disponible ci-dessous.

1. Un ensemble d'entiers E comportant un seul élément est *décidable* : expliquer brièvement pourquoi, en écrivant une procédure qui décide si un entier x appartient à E.
2. Montrer que la réunion de deux ensembles décidables est décidable.
3. En déduire que tout ensemble *fini* est décidable.

Dans la suite de l'exercice, on suppose que E est un ensemble d'entiers naturels *infini*, et on définit la fonction f de la façon suivante :

a. E possède un plus petit élément u_0 ;
b. on définit $f(0) = u_0$;
c. l'ensemble $E - \{ u_0 \}$ (E privé de u_0) possède un plus petit élément u_1 , et on définit $f(1) = u_1$;
d. l'ensemble $E - \{ u_0 , u_1 \}$ possède un plus petit élément u_2 , et on définit $f(2) = u_2$;
e. et ainsi de suite…

Questions :

4. La fonction f est-elle totale ? A-t-on $E =$ **Val** (f) ?
5. Il existe des ensembles infinis non récursivement énumérables : expliquer brièvement pourquoi.
6. La construction de f semble pourtant montrer que tout ensemble infini est récursivement énumérable ; expliquer *précisément* où est le bug (au cas où il se trouverait dans l'une des étapes a-e de la définition de f, expliquer laquelle). *Note* : la réponse demandée est courte et précise ; si vous ne la voyez pas, inutile de perdre du temps à rédiger un discours brumeux qui ne vous rapportera aucun point.

Exercice 2

Un corrigé est disponible ci-dessous.

A et B désignent deux ensembles *infinis* d'entiers, auxquels on associe l'ensemble C défini par :

$$C = \{ 2n \mid n \in A \} \cup \{ 2n + 1 \mid n \in B \}$$

1. On suppose A et B récursivement énumérables, et on appelle f et g deux fonctions calculables totales qui les énumèrent. Ecrire une procédure h qui énumère C. *Note* : la réponse est facile, mais les lapsus le sont aussi ; vérifier soigneusement qu'on a bien **Val** (h) = C.
2. Inversement, si C est récursivement énumérable, peut-on en déduire que A et B le sont aussi ? La réponse, qu'elle soit positive ou négative, doit être correctement justifiée.
3. Soit $H = \{ (p, x) \mid$ le calcul de p (x) termine $\}$, où p désigne une procédure (à un argument), et x un entier. Expliquer pourquoi H est récursivement énumérable, et pourquoi son complémentaire H' (noté d'habitude "H barre", diffcile à coder en HTML, désolé) ne l'est pas. *Note* : il ne suffit pas de répondre "ce sont des théorèmes du cours" ; on demande d'exposer *brièvement* l'essentiel des démonstrations de ces deux résultats.
4. Expliquer *brièvement* comment numéroter les couples (p, x), où p désigne une procédure (à un argument), et x un entier.
5. On appelle φ la fonction de numérotation de la question précédente, et on considère l'ensemble

E défini par :

$$E = \{ 2\,\varphi(p, x) \mid (p, x) \in H \} \cup \{ 2\,\varphi(p, x) + 1 \mid (p, x) \in H' \}$$

(rappel : H' désigne le complémentaire de H). E est-il récursivement énumérable ? Son complémentaire E' est-il récursivement énumérable ? *Note* : justifier soigneusement les réponses.

Exercice 3

Un corrigé est disponible ci-dessous.

On code un *couple* par le terme

$$C = \lambda\, x\, y\, f\, .\, f\, x\, y$$

1. Soit $P = \lambda\, t\, .\, t\, (\lambda\, x\, y\, .\, x)$; réduire $P\, (C\, M\, N)$, où M et N désignent des λ-expressions quelconques. *Note* : écrire et disposer de façon très lisible les différentes étapes de la réduction ; des gribouillis ne rapporteront aucun point.
2. Coder un *triplet* par une λ-expression T construite à l'aide de C. *Note* : une réponse directe, construite sans utiliser C, ne rapportera aucun point.
3. Coder l'expression **second** telle que **second** $(T\, M_1\, M_2\, M_3) \rightarrow M_2$, où les M_i sont des λ-expressions quelconques.

Exercice 4

Un corrigé est disponible ci-dessous.

On suppose donné un algorithme A résolvant le problème de décision **SAT**. Soit F une formule logique, dont les variables booléennes sont $x_1 \ldots x_n$; on suppose que F est *satisfiable*. Ecrire un algorithme efficace B, qui utilise A pour calculer une suite de n valeurs booléennes $b_1 \ldots b_n$ qui satisfont F ; dans cet exercice, on ne se préoccupe pas de la complexité de A pour évaluer la complexité de B (on considère A comme un *oracle*, dont l'exécution a un coût constant, indépendant de la taille de son argument) — autrement dit le but de l'exercice est de montrer que *trouver* une suite de valeurs qui satisfont F, n'est pas plus difficile que de résoudre le problème de décision **SAT**.

Corrigé.

Note sur le barème : les copies ont été notées sur 30 (8 points pour l'exercice 1, 10 points pour l'exercice 2, 7 points pour l'exercice 3, et 5 points pour l'exercice 4). La meilleure note a été 23, et les notes comprises entre 12 et 23 ont été ajustées pour obtenir des notes comprises entre 12 et 19.

Exercice 1

Voir l'énoncé ci-dessus.

Barème : 4 points pour la série des trois premières questions.

1. Soit $E = \{ a \}$; la procédure qui calcule la fonction caractéristique de E s'écrit :

```
int p (int x) {
  return x == a;
}
```

Note : il est étonnant, au niveau bac + 4, que la plupart des étudiants écrivent encore (même si ce n'est pas faux) :

```
int p (int x) {
  if (x == a) return 1;
  else return 0;
}
```

2. Si `p` et `q` désignent des procédures qui calculent les fonctions caractéristiques de E et F, voici une procédure `u` qui décide si x appartient à l'union des deux ensembles :

```
int u (int x) {
  return p(x) || q(x);
}
```

ou bien :

```
int u (int x) {
  if (p(x)) return 1;
  return q(x);
}
```

Note : il y a beaucoup de façons de rédiger cette réponse, par exemple on peut écrire que si les prédicats $x \in E$ et $x \in F$ sont calculables, alors il en va de même pour :

$$x \in E \cup F = (x \in E) \vee (x \in F).$$

3. Démonstration par récurrence sur le cardinal n de l'ensemble :
   - Si $n = 0$, l'ensemble est vide, sa fonction caractéristique est :

```
int vide (int x) { return 0; }
```

   - Un ensemble E de cardinal $n + 1$ est la réunion d'un ensemble de cardinal n, décidable par hypothèse de récurrence, et d'un ensemble de cardinal 1. On utilise alors les questions 1 et 2 pour conclure que E est décidable.

Barème : 4 points pour la série des trois dernières questions.

4. Oui, f est totale, car E est infini. De même, pour tout élément x dans E, il existe un entier i tel que $x = f(i)$: i est le rang de x dans E, une fois celui-ci ordonné.
5. L'ensemble des couples (p, x) tels que le calcul $p(x)$ ne termine pas, n'est pas récursivement énumérable. Via la numérotation des procédures et des couples, un tel ensemble peut être vu comme un ensemble d'entiers.

   Note : c'est le seul exemple d'ensemble non récursivement énumérable vu en cours, et c'est l'exemple fondamental. Mais on peut en citer d'autres, par exemple l'ensemble des procédures qui calculent l'identité (ou n'importe quelle fonction calculable donnée) : voir le devoir 2003 ou le devoir 2004. On peut aussi donner un argument de dénombrement : les ensembles récursivement énumérables forment un ensemble dénombrable (comme les procédures qui les énumèrent), ce qui n'est pas le cas des ensembles infinis quelconques d'entiers.
6. La fonction *min*, qui fournit le plus petit élément d'un ensemble E, n'est en général pas *calculable*, et donc f non plus. D'ailleurs f énumère E en ordre croissant, et donc si f est calculable, E est décidable (voir la feuille d'exercices du chapitre 4). Inversement, si E est décidable, *min* et f sont calculables.

Exercice 2

Voir l'énoncé ci-dessus.

1. (2 points). Dans la procédure suivante, `else` est optionnel, et *n/2* désigne le quotient entier de n par 2 :

```
int h (int n) {
  if (n % 2 == 0)
    return 2 * f (n/2);
  else
    return 1 + 2 * g (n/2);
}
```

2. (2 points). Oui, le plus simple est d'utiliser l'équivalence entre *récursivement énumérable* et *semi-décidable* :

```
int scA (int x) {
  return scC (2 * x);
}
```

La procédure `scA` calcule la fonction *semi-caractéristique* de l'ensemble A, à partir d'une procédure similaire pour C. Le cas de B est symétrique.

On peut aussi écrire une procédure d'énumération des éléments de A, construite à partir d'une procédure h qui énumère les éléments de C :

```
int f (int n) {
  int x = h (n);
  if (x % 2 == 0)
    return x / 2;
}
```

Cette procédure calcule une fonction *partielle*, mais ce n'est pas gênant pour conclure que A est récursivement énumérable (voir la feuille d'exercices du chapitre 4). Cependant, les règles du langage C entraînent que $f(n)$ n'est pas indéfini, mais aléatoire, lorsque $h(n)$ est impair ; on peut y remédier en ajoutant une boucle infinie `while (1)` en fin de procédure. Une solution moins artificielle est de transformer f en une procédure *totale*, par exemple :

```
int f (int n) {
  int x, c = -1;
  for (i == 0; c < n; i++) {
    x = h (i);
    if (x % 2 == 0) c++;
  }
  return x / 2;
}
```

3. (2 points). Question de cours, voir chapitre 4, sections 2 et 3.
4. (2 points). Question de cours, voir chapitre 1, sections 2 et 5.
5. (2 points). Non, E n'est pas récursivement énumérable, sinon, d'après la question 2, H' serait récursivement énumérable, ce qui est faux (question 3). Le complémentaire E' de E est obtenu en échangeant les rôles de H et H', donc il n'est pas non plus récursivement énumérable.

   Cet exercice montre donc comment fabriquer simplement un ensemble E tel que ni lui ni son complémentaire ne soient récursivement énumérables, à partir d'un ensemble (ici H', mais tout autre exemple conviendrait) dont on sait seulement qu'il n'est pas récursivement énumérable.

   Note : beaucoup d'étudiants se sont trompés en calculant E'. Le complémentaire de l'union de deux ensembles est bien l'intersection des complémentaires, mais le complémentaire de :

$$F = \{\ 2n \mid n \in A\ \}$$

   n'est pas $G = \{\ 2n \mid n \in A'\ \}$, car les entiers impairs ne sont ni dans F ni dans G ! Le complémentaire F' est constitué de l'union de G et de tous les entiers impairs ; l'oubli est pourtant facile à déceler, car il entraîne qu'on trouve E' … vide, ce qui est curieux (mais ne semble pas déconcerter certains étudiants).

Exercice 3

Voir l'énoncé ci-dessus.

1. (3 points). Cette question fait partie de la feuille d'exercices du chapitre 6 : C est le terme **pair**, P le terme **first**, et $\lambda\, x\, y\, .\, x$ est le terme **true**. Revoici une réduction :

$$P\ (C\ M\ N) \to (C\ M\ N)\ (\lambda\, x\, y\, .\, x) = C\ M\ N\ (\lambda\, x\, y\, .\, x)$$

Les parenthèses à gauche sont en effet inutiles, et comme C est une fonction de trois variables, on obtient :

$$C\ M\ N\ (\ \lambda\ x\ y\ .\ x\ )\ \rightarrow\ (\ \lambda\ x\ y\ .\ x\ )\ M\ N\ \rightarrow\ M$$

2. (2 points). Il faut considérer un triplet comme un couple, dont le premier élément est lui-même un couple :

$$T = \lambda\ x\ y\ z\ .\ C\ (\ C\ x\ y\ )\ z$$

3. (2 points). Pour obtenir le second élément d'un triplet défini comme ci-dessus, il faut sélectionner le premier élément du triplet, qui est lui-même un couple, dont on sélectionne alors le second élément, soit :

$$\textbf{second} = \lambda\ t\ .\ S\ (\ P\ t\ )$$

en notant S le jumeau de P, soit $S = \lambda\ c\ .\ c\ (\ \lambda\ x\ y\ .\ y\ )$; vérifions :

$$\textbf{second}\ (\ T\ M_1\ M_2\ M_3\ )\ \rightarrow\ S\ (\ P\ (\ T\ M_1\ M_2\ M_3\ )\ )\ \rightarrow$$
$$\rightarrow\ S\ (\ P\ (\ C\ (\ C\ M_1\ M_2\ )\ M_3\ )\ )\ \rightarrow\ S\ (\ C\ M_1\ M_2\ )\ \rightarrow\ M_2$$

car, par symétrie avec P (voir première question), $S\ (\ C\ M\ N\ )\ \rightarrow\ N$.

Exercice 4

(5 points). Voir l'énoncé ci-dessus. Voici l'algorithme B :

- On remplace x_1 par 1 dans F, soit $F_1 = F\ (\ 1,\ x_2\ \ldots\ x_n\ )$; on interroge A pour savoir si F_1 est satisfiable : si oui, on choisit $b_1 = 1$; sinon on change d'avis, on remplace x_1 par 0 dans F, soit $F_1 = F\ (\ 0,\ x_2\ \ldots\ x_n\ )$; inutile d'interroger à nouveau A, puisque F est satisfiable, et on pose $b_1 = 0$.
- On est sûr par construction que $F_1 = F\ (\ b_1,\ x_2\ \ldots\ x_n\ )$ est satisfiable ; on remplace x_2 par 1 dans F_1, ce qui fournit F_2, on calcule $A\ (\ F_2\ )$, on fixe b_2 en fonction de la réponse — en fait on pose $b_2 = A\ (\ F_2\ )$ — et on ajuste $F_2 = F\ (\ b_1\ ,\ b_2\ ,\ x_3\ \ldots\ x_n\ )$.
- Et ainsi de suite : en n étapes, on détermine ainsi des valeurs de $b_1\ \ldots\ b_n$ qui satisfont F.