Correction du CC

1 Calculabilité

Exercice 1

Soit c la fonction de codage pour les couples d'entiers vue en cours : $c : \mathbb{N} \to \mathbb{N}$ telle que $c(x, y) = \frac{(x+y)(x+y+1)}{2} + x$.

1. Soit h la fonction de codage pour les triplets définie par $h(x, y, z) = c(c(x, y), z)$. Quel est le doublet codé par 50 ? Quel est le triplet codé par 50 ?
2. Soit h' la fonction de codage pour les triplets définie par $h'(x, y, z) = c(x, c(y, z))$. Quel est le doublet codé par 50 ? Quel est le triplet codé par 50 ?

1. On cherche x, y tels que $c(x, y) = 50$. Pour ce faire, on pose $t = x + y$ et on cherche $t_{max} = max\{t \mid \frac{t(t+1)}{2} \leq 50\}$.
$$t_{max} = 9$$
En effet, on a $\frac{9 \times 10}{2} = 45 \leq 50$ et $\frac{10 \times 11}{2} = 55 > 50$, 9 est donc bien le t maximum. Il ne reste qu'à trouver x et y :
$$x = 50 - \frac{t_{max}(t_{max} + 1)}{2} = 50 - 45 = 5$$
$$y = t - x = 9 - 5 = 4$$
On a donc :
$$c(5, 4) = 50$$
On a trouvé le doublet qui se code en 50. On veut trouver le triplet qui se code en 50. On sait que $h(x, y, z) = c(c(x, y), z) = 50$. Donc :
$$c(x, y) = 5$$
$$z = 4$$
Il ne reste qu'à trouver le doublet codé par 5. On procède de la même manière :
$$t_{max} = 2$$
En effet, on a $\frac{2 \times 3}{2} = 3 \leq 5$ et $\frac{3 \times 4}{2} = 6 > 5$, 2 est donc bien le t maximum. Il ne reste qu'à trouver x et y :
$$x = 5 - \frac{t_{max}(t_{max} + 1)}{2} = 5 - 3 = 2$$
$$y = t - x = 2 - 2 = 0$$
Le triplet codé par 50 est $(2, 0, 4)$.
2. Le doublet codé par 50 n'a pas changé, c'est toujours $(5, 4)$. On a maintenant :
$$x = 5$$
$$c(y, z) = 4$$

Il ne reste donc qu'à trouver le doublet codé par 4. On procède de manière similaire :

$$t_{max} = 2$$

En effet, on a $\frac{2\times 3}{2} = 3 \leq 4$ et $\frac{3\times 4}{2} = 6 > 4$, 2 est donc bien le t maximum. Il ne reste qu'à trouver y et z :

$$y = 4 - \frac{t_{max}(t_{max}+1)}{2} = 4 - 3 = 1$$
$$z = t - x = 2 - 1 = 1$$

Le triplet codé par 50 est $(5, 1, 1)$.

Exercice 2

1. Soient f et g, deux fonctions calculables. Soit $E = \{x \mid f(x)$ est défini, $g(x)$ est défini et $f(x) < g(x)\}$. Montrez que E est récursivement énumérable.
2. Le complémentaire de E est-il toujours récursivement énumérable ? Justifiez complètement votre réponse.
3. Soit p une procédure définie pour tout x. Montrez que savoir si $\forall x, p(x)$ est premier est un problème indécidable.
4. Soit une suite f_i de fonctions totales de $\mathbb{N}$ dans $\mathbb{N}$. Donner une fonction croissante h qui n'appartienne pas à cette suite.

1. Il y a deux manières de faire pour montrer que E est récursivement énumérable :
   — construire une procédure automatique qui énumère E,
   — construire une procédure automatique qui calcule la fonction semi-caractéristique de E.

   Commençons par construire une procédure automatique qui énumère E. On pose p_f la procédure automatique qui calcule f et p_g la procédure automatique g. La procédure automatique p_E suivante énumère E :

```
void p_E() {
    int n = 1;
    while (1) {
        for (int i = 0; i <= n; ++i) {
            if (p_f(i) termine en n etapes et p_g(i) termine en n etapes et p_f(i) < p_g(i))
                afficher(i);
        }
        n += 1;
    }
}
```

   On est obligés d'introduire un facteur temps dans cette procédure d'énumération, car on ne sait pas si i est dans le domaine de définition de f et de g. S'il l'est, les procédures automatiques finiront par calculer le résultat et afficheront i si $p_f(i) < p_g(i)$. Cette procédure automatique énumère bien E.

   Procédons en construisant la fonction semi-caractéristique $\mathcal{X}_E$ (avec toujours p_f et p_g les procédures automatiques qui calculent f et g) :

```
int X_E(int n) {
    if (p_f(n) < p_g(n)) return 1;
    while(1) ; // Pas sûr que cette boucle soit nécessaire
               // -> si l'élément n'est pas dans l'ensemble, a-t-on le droit de return 0 ?
}
```

Cette procédure calcule bien la fonction semi-caractéristique de E : f et g sont calculables, donc si n est sur le domaine de définition de ces deux fonctions, et que $f(n) < g(n)$, alors la procédure renvoie 1. Si n est sur le domaine de définition de f et g mais que $f(n) \geq g(n)$, alors la procédure automatique ne s'arrête pas. De même, si n n'est pas sur le domaine de définition de f ou sur le domaine de définition de g, elle ne s'arrête pas. C'est bien une procédure automatique qui calcule la fonction semi-caractéristique de E.

2. Non, le complémentaire de E n'est pas toujours récursivement énumérable. En effet, si les fonctions f et g ne sont pas totales, on ne peut pas dire si un élément n'appartient pas à E, et ainsi, on ne peut pas savoir si cet élément appartient à $\bar{E}$.

   Si les fonctions f et g sont totales, alors $\bar{E}$ est récursivement énumérable, et E et $\bar{E}$ sont décidables.

3. Il y a deux techniques pour prouver que le problème est indécidable :

   — construire une fonction contradictoire,

   — construire un prédicat non trivial et invoquer le théorème de Rice.

   Commençons par construire une fonction contradictoire :

```
int gamma(int x) {
    if (premier(gamma(x))) return 42;
    else return 17;
}
```

   Cette fonction prouve que le prédicat premier ne peut pas exister. En effet, si $\gamma(x)$ est premier, alors $\gamma(x) = 42$, qui n'est pas premier. Au contraire, si $\gamma(x)$ n'est pas premier, alors $\gamma(x) = 17$, qui lui, est premier. Donc la fonction γ ne peut pas exister, et la fonction premier ne peut pas exister non plus.

   Procédons maintenant par l'invocation du théorème de Rice. Soit le prédicat $P(p) = 1$ si la procédure p renvoie tout le temps un nombre premier, 0 sinon. On peut contruire la fonction quarante_deux suivante :

```
int quarante_deux() { return 42; }
```

   On a $P(\text{quarante\_deux}) = 0$. Ensuite, on peut construire la fonction dix_sept suivante :

```
int dix_sept() { return 17; }
```

   On a $P(\text{dix\_sept}) = 1$. Le prédicat P est non trivial, donc, par le théorème de Rice, ce problème est indécidable.

4. Soit la fonction $h(n) = \sum_{i=0}^{n} f_i(i) + 1$. Cette fonction est croissante car $h(0) = f_0(0) + 1$, $\forall n \in \mathbb{N}. h(n) = h(n-1) + f_n(n) + 1$ et que $f_i : \mathbb{N} \to \mathbb{N}$, on a $h(n) \geq h(n-1) + 1$. De plus, cette fonction est totale, car f_i est une famille de fonctions totales. On suppose que h appartienne à la suite de fonctions totales de $\mathbb{N} \to \mathbb{N}$. Ainsi, il existe k tel que $h = f_k$. Or, $h(k) = \sum_{i=0}^{k}(f_i(i) + 1) = h(k-1) + f_k(k) + 1 \neq f_k(k)$. On a trouvé une contradiction, h ne peut donc pas appartenir à la famille f_i.

2 Partie complexité

Exercice 3

1. Q est NP-complet et R se réduit polynomialement à Q. Que sait-on sur R ?
2. Q est NP-complet et Q se réduit polynomialement à R. Que sait-on de R ?
3. Vous trouvez un algorithme polynomial pour résoudre le problème du stable. Quelles sont parmi les affirmations suivantes celles qui peuvent être vraies ?
   (a) Vous vous êtes trompés
   (b) P = NP
   (c) Le problème du stable n'est pas NP-complet

1. On ne sait pas grand chose sur R. La seule information que la réduction polynomiale nous donne est que Q est plus difficile que R, donc, dans le pire des cas, R est NP-complet, mais on n'en sait pas plus.
2. Si Q est NP-complet et qu'il se réduit polynomialement à R, alors R est plus difficile que Q, donc R est NP-dur (et R est NP-complet si $R \in \mathcal{NP}$).
3. (a) C'est une possibilité en effet.
   (b) Oui, mais du coup, je ne publierai pas mes résultats de recherches, sous peine de représailles.
   (c) Non, même si ce problème est faisable en temps polynomial, il reste NP-complet, car un problème NP-complet se réduit à celui-ci, d'où le P = NP de la question précédente.