Oraux blancs MPI
type CCMT

Florian Bourse

Exercice 1

Nous noterons `unit list` le type utilisé pour représenté $\mathbb{N}$.

On considère la variante Halt suivante du problème de l'arrêt, qui est indécidable :

> **Instance :** Une fonction OCaml `f` de type `unit -> unit`.
> **Solution :** V si et seulement si l'appel `f ()` termine.

On rappelle qu'un problème est semi-décidable s'il existe une fonction qui termine et renvoie V sur toute instance positive du problème et ne renvoie jamais V sur une instance négative du problème.

On suppose l'existence d'une fonction `runs` qui prend en entrée une fonction `f`, un argument `x` et un entier `n` et qui renvoie F si `f x` termine en moins de n étapes de calcul et V si `f x` n'a pas encore terminé au bout de n étapes de calcul.

1. Montrer que Halt est semi-décidable.
2. On considère aussi le complémentaire de Halt, co-Halt ci-dessous :

   Instance : Une fonction OCaml `f` de type `unit -> unit`.

   Solution : F si et seulement si l'appel `f ()` termine.

   Montrer que co-Halt n'est pas semi-décidable.
3. Considérons le problème $\text{Halt}_\forall$ ci-dessous :

   > **Instance :** Une fonction OCaml `f` de type `unit list -> unit`.
   > **Solution :** V si et seulement si pour toute valeur `n`, l'appel `f n` termine.

   Que peut-on dire de la décidabilité, de la semi-décidabilité de $\text{Halt}_\forall$?
4. Considérons le problème $\text{Halt}_\exists$ ci-dessous :

   > **Instance :** Une fonction OCaml `f` de type `unit list -> unit`.
   > **Solution :** V si et seulement si il existe une valeur `n` telle que l'appel `f n` termine.

   Que peut-on dire de la décidabilité et de la semi-décidabilité de $\text{Halt}_\exists$?

Exercice 2

Soit une fonction f pour laquelle il existe une valeur i telle que $f(i) = 0$. Nous cherchons une telle valeur i en divisant l'espace de recherche en deux parties : les nombres positifs et les nombres négatifs. Notre programme va lancer deux fils d'exécution concurrents pour chercher dans chacun de ces ensembles.

```
bool found; int res;

void p() {
  int i = 0;                // p1
  while (!found) {          // p2
    i = i + 1;              // p3
    found = (f(i) == 0);    // p4
  }
  res = i;                  // p5
}

void q() {
  int j = 1;                // q1
  while (!found) {          // q2
    j = j - 1;              // q3
    found = (f(j) == 0);    // q4
  }
  res = j;                  // q5
}

int cherche_zero() {
  pthread_t id[2];
  pthread_create(&id[0], NULL, p, NULL);
  pthread_create(&id[1], NULL, q, NULL);

  pthread_join(id[0], NULL);
  pthread_join(id[1], NULL);

  return res;
}
```

Étudier la correction de la fonction ci-dessus en corrigeant les erreurs si nécessaire.