TD 3 : terminaison et correction des algorithmes

Exercice 1 – Premier exemple

Considérons la fonction `expo : int -> int -> int` répondant à la spécification suivante :

Précondition : $n \geq 0$
Postcondition : `expo(a, n)` $= a^n$

Python

```python
def expo(a, n):
    p = n
    x = 1
    b = a
    while p != 0:
        if p % 2 == 1:
            x = x * b
        b = b * b
        p = p // 2
    return x
```

▶ **Question 1** Montrer que "$xb^p = a^n$ et $p > 0$" est un invariant de boucle.

▶ **Question 2** Montrer la correction partielle, puis la correction totale, de la fonction.

Exercice 2 – Algorithme de recherche dichotomique

Considérons maintenant la fonction de recherche dichotomique, répondant à la spécification suivante :

Entrée : un tableau $t = [t_0, \dots, t_{n-1}]$, un élément x
Sortie : s'il existe, un indice $i \in [\![0, n-1]\!]$ tel que $t_i = x$, n sinon

▶ **Question 1** Écrire une première fonction `recherche_naive : 'a array -> 'a -> int` en pseudo-code répondant à cette spécification.
Modifions un peu la spécification attendue :

Entrée : un tableau $t = [t_0, \dots, t_{n-1}]$, un élément x
Précondition : t est trié
Sortie : s'il existe, un indice $i \in [\![0, n-1]\!]$ tel que $t_i = x$, sinon n

Pseudo-code

```
1 : fonction RECHERCHE(x, t) :
2 :     deb, fin ← 0, n
3 :     Tant que fin − deb > 0, faire :
4 :         milieu ← (deb + fin)/2                ▷ division entière
5 :         Si t_milieu = x alors
6 :             Renvoyer milieu
7 :         Sinon, si t_milieu < x
8 :             deb ← milieu + 1
9 :         Sinon
10 :            fin ← milieu
11 :        Fin si
12 :    Fin tant que
13 :    Renvoyer n
14 : Fin fonction
```

▶ **Question 2** Montrer la terminaison de `Recherche`.

▶ **Question 3** Montrer que si l'algorithme renvoie un entier $i < n$, alors ce résultat est correct. A-t-on déjà la correction partielle ?

▶ **Question 4** Pour $a \leq b$, notons $t\,[a : b]$ la portion du tableau comprise entre les indices a inclus et b exclu. Montrer que la boucle respecte l'invariant suivant :

$$x \notin t\,[0 : deb] \text{ et } x \notin t\,[fin : n]$$

Cette opération est appelée *slicing*, et dispose d'une syntaxe dédiée en Python, mais pas en OCaml ni en C.

▶ **Question 5** En déduire la correction (totale) de l'algorithme.

Exercice 3 – Correction du tri par insertion et du tri fusion

Pour cet exercice, vous avez besoin de reprendre le code de votre fonction de tri par insertion et de tri fusion écrite en DM, corrigée si besoin à l'aide de l'indication de corrigé donnée sur `cahier-de-prepa.fr`.

▶ **Question 1** Formaliser les spécifications des fonctions `insere` et `tri_insertion`, avec la présentation et le vocabulaire vu en cours.

▶ **Question 2** Montrer la correction de ces fonctions.

▶ **Question 3** Faire de même pour le `tri_fusion`.

Exercice 4 – Problème de l'arrêt (spoiler de la MPI)

On définit le problème de l'arrêt comme suis :

Problème de l'arrêt

Entrée : un programme P exprimé comme un mot sur un alphabet Σ, qui prend en entrée un mot sur Σ, et qui renvoie `vrai` ou `faux`, un mot u sur l'alphabet Σ

Sortie : `vrai` si P termine sur l'entrée u, `faux` sinon

▶ **Question 1** Montrer qu'il existe un algorithme partiellement correct pour cette spécification (on suppose que l'on peut exécuter P à partir de sa représentation comme un mot sur Σ).

▶ **Question 2** Montrer qu'il n'existe pas d'algorithme correct décidant le problème de l'arrêt. *On supposera qu'un algorithme en pseudo-code est exprimable sur l'alphabet Σ.*

Exercice 5 – Le principe d'induction bien fondée

La relation binaire $\leq$ sur un ensemble X est une relation d'ordre si elle est réflexive, antisymétrique et transitive.
On dit que la relation d'ordre $\leq$ sur X est *bien fondée* si tout sous-ensemble non vide Y de X possède un élément minimal, c.-à-d. un élément de Y tel qu'il n'existe pas d'élément de Y strictement plus petit pour $\leq$.
Un ordre $\leq$ total et bien fondé est appelé un *bon ordre*, et X un ensemble *bien ordonné* par $\leq$.

▶ **Question 1** Montrer le principe d'induction bien fondée :

Théorème 1 – Principe d'induction bien fondée

Soit $\mathcal{P}$ un prédicat sur X, et $\leq$ une relation d'ordre bien fondée sur X. Supposons que pour tout $x \in X$, si tout élément $y < x$ vérifie $\mathcal{P}$, alors x vérifie $\mathcal{P}$. Alors, $\mathcal{P}(x)$ est `vrai` pour tout $x \in X$.

▶ **Question 2** Comment s'appelle le principe d'induction bien fondée pour $X = \mathbb{N}$ et $\leq$ l'ordre canonique sur les entiers naturels ?

▶ **Question 3** Montrer que $\leq$ sur X est bien fondé si et seulement s'il n'existe pas de suite infinie strictement décroissante d'éléments de X, si et seulement si le principe de récurrence forte est vrai pour $\leq$ sur X pour tout prédicat $\mathcal{P}$ sur X.

▶ **Question 4** Montrer que la relation d'ordre naturelle sur $\mathbb{N}$ est un ordre bien fondé, mais pas sur $\mathbb{Z}$.

▶ **Question 5** Montrer que si $\leq_X$, resp. $\leq_Y$ sont deux ordres bien fondés sur l'ensemble X, resp. Y, alors l'ordre lexicographique $\leq_{X\times Y}$ sur l'ensemble $X \times Y$ est bien fondé. *Cet exemple est généralisable à tout produit d'ensembles bien ordonnés.*

▶ **Question 6** Montrer que l'ordre lexicographique sur les suites (infinies) d'entiers naturels n'est pas bien fondée.

Exercice 6 – La fonction d'Ackermann est bien définie

▶ **Question 1** Rappeler la définition de l'ordre lexicographique sur l'ensemble $\mathbb{N}^2$. *Dans la suite, on le note $\leq_{lex}$.*

▶ **Question 2** Montrer par induction bien fondée que l'algorithme suivant *termine*, c'est-à-dire que la séquence d'instructions exécutées dans l'algorithme sur toute entrée est finie (pas de "boucles infinies") :

Pseudo-code

```
1 : fonction ACKERMANN(m,n) :
2 :     Si m = 0 alors
3 :         retourne n + 1
4 :     Sinon, si n = 0
5 :         retourne ACKERMANN(m-1,1)
6 :     Sinon
7 :         k ← ACKERMANN(m,n-1)
8 :         retourne ACKERMANN(m-1, k)
9 :     Fin si
10 : Fin fonction
```

Indication : on peut utiliser dans cette question que $\leq_{lex}$ est bien fondé.

▶ **Question 3** Écrire explicitement la sortie de ACKERMANN$(1,k)$, ACKERMANN$(2,k)$ et ACKERMANN$(3,k)$ pour tout $k \in \mathbb{N}$.
La fonction d'Ackermann calculée par cet algorithme est l'exemple classique de fonction *non primitive récursive* : il n'existe pas d'algorithme (n'utilisant pas d'appels récursifs) calculant cette fonction sans utiliser de boucles non bornées, c'est-à-dire sans utiliser de **while**. Le montrer est cependant un exercice difficile.