Chapitre 1 : correction et terminaison des algorithmes-Exercices

9 février 2023

Exercice 1
On donne ci-dessous un algorithme de multiplication qui prend en entrée deux entiers a et b.

```
1: r ⟵ 0
2: while b>0 do
3:     b ⟵ b − 1
4:     r ⟵ r + a
5: end while
6: return r
```

1. Proposer des préconditions et de postconditions pour cet algorithme.
2. Montrer que, sous les préconditions choisies, le programme termine quels que soient les arguments a et b.
3. Proposer un invariant pour la boucle `while` et montrer que c'est bien un invariant.
4. Montrer que le programme est fortement correct vis-à-vis de la spécification choisie.

Exercice 2
On rappelle ci-dessous l'algorithme de la division euclidienne, qui prend en entrée deux entiers a et b pour calculer leur pgcd.

```
1: while b > 0 do
2:     a, b ⟵ b, a%b
3: end while
4: return a
```

1. L'algorithme est-il partiellement correct ?
2. Proposer des préconditions et postconditions.
3. Montrer que l'algorithme termine en utilisant un variant de boucle.
4. Montrer qu'il est fortement correct.

Exercice 3
On propose ci-dessous un algorithme qui prend en entrée un entier naturel n et doit retourner la somme $\sum_{k=0}^{n} k$.

```
1: s ⟵ 0
2: for i allant de 0 à n do
3:     s ⟵ s + i
4: end for
5: return s
```

1. Proposer des préconditions et postconditions pour l'algorithme.
2. Justifier rapidement que l'algorithme termine et prouver sa correction au moyen d'un invariant de boucle adapté.

Exercice 4
On donne ci-dessous un algorithme de recherche du plus petit élément d'une liste. L'algorithme prend en entrée une liste de nombres entiers ou flottants L.

```
1: m ⟵ L[0]
2: for i allant de 1 à len(L) − 1 do
3:     if L[i] < m then
4:         m = L[i]
5:     end if
6: end for
7: return m
```

1. Proposer des préconditions et postconditions.
2. Justifier rapidement que l'algorithme termine et montrer qu'il est correct en utilisant un invariant de boucle adapté.

Exercice 5
On dit qu'une liste de nombre entier est un palindrome si elle est identique à la liste obtenue en « retournant »la liste initiale, c'est-à-dire en lisant les éléments de la fin au début.

Par exemple, les liste $[1,2,3,3,2,1]$ et $[0,1,2,3,2,1,0]$ sont des palindromes.

On donne ci-dessous un algorithme prenant en entrée une liste L et déterminant si L est un palindrome.

On rappelle que si x est un nombre réel, $\llcorner x \lrcorner$ désigne le plus grand entier p tel que $p \leq x$.

```
1: n ⟵ len(L)
2: p ⟵ ⌞n/2⌟
3: i ⟵ 0
4: while i < p and L[i] = L[n − 1 − i] do
5:     i ⟵ i + 1
6: end while
7: return i = p
```

1. Proposer des précondition et postcondition pour cet algorithme.
2. Montrer que l'algorithme termine.
3. Montrer que l'algorithme est correct en utilisant un invariant de boucle adapté.

Exercice 6
On revient dans cet exercice sur la recherche par dichotomie d'un élément dans un tableau trié

1. On propose dans un premier temps un algorithme prenant en entrée un tableau de nombres entiers T et déterminant si T est trié.

```
1: i ⟵ 0
2: while i < len(T) − 1 and T[i] ≤ T[i + 1] do
3:     i ⟵ i + 1
4: end while
5: return i = len(T) − 1
```

   Proposer des préconditions et postconditions pour cet algorithme. Montrer qu'il termine et qu'il est correct relativement à cette spécification.

2. On propose maintenant une version de l'algorithme de recherche dichotomique d'un entier elt dans un tableau de nombres entiers **triés** T.

   a. Appliquer l'algorithme « à a la main » pour $T = [0,1,3,4]$ et $elt = 2$. Quelles sont les valeurs finales de a et b?

   b. Montrer que l'algorithme termine.

   c. Proposer une spécification et un invariant de boucle et montrer la correction de l'algorithme.

```
1: a, b ⟵ 0, len(T) − 1
2: c ⟵ (a + b)//2
3: while b − a ≥ 0 and T[c] ≠ elt do
4:    if T[c] < elt then
5:       a ⟵ c + 1
6:    else
7:       b ⟵ c − 1
8:    end if
9:    c ⟵ (a + b)//2
10: end while
11: return b ≥ a
```

Exercice 7
On donne ci-dessous une version de l'algorithme de tri bulle, qui prend en entrée une liste T de nombres entiers ou flottants.

```
1: n ⟵ len(T)
2: for i allant de 0 à n − 2 do
3:    for j allant de 0 à n − 2 − i do
4:       if T[i] < T[i + 1] then
5:          Echanger T[i] et T[i + 1]
6:       end if
7:    end for
8: end for
9: return T
```

1. Proposer une postcondition pour l'algorithme.

2. Proposer un invariant pour chacune des deux boucles et vérifier la correction de l'algorithme.

Corrigé 1 **1.** On peut remarquer que si $b < 0$, l'algorithme retourne a qui n'est pas la bonne réponse en général. Il est prudent de choisir comme précondition $P(a,b)$: a entier et b entier vérifiant $b \geq 0$.

La postcondition exprime le fait que le résultat est le produit de a et b. $Q(a,b,r)$: $r = a \times b$.

2. Si $b > 0$, b est un variant de boucle et l'algorithme termine.

3. Pour exprimer l'invariant, il est utile de noter b_0 la valeur initiale de b, b désignant la valeur de la variable a un moment quelconque de l'exécution. On propose comme invariant de boucle $r = a \times (b_0 - b)$.

Initialisation Avant la première entrée dans la boucle, on a $r = 0$ et $b_0 - b = b_0 - b_0 = 0$, donc l'égalité est vérifiée.

Conservation Supposons que l'invariant soit vérifié en début de boucle. Le tableau ci-dessous donne l'évolution des varaibles b et r lors du passage dans la boucle (a reste inchangé).

Initial	final
b	$b-1$
r	$r+a$

Vérifions que l'invariant est conservé en fin de boucle : on a $a\times(b_0-(b-1))=a\times(b_0-b)+a$, or $a\times(b_0-b)=r$ d'après l'invariant en début de boucle, donc $a\times(b_0-(b-1))=r+a$ et l'invariant reste conservé.

4. La terminaison du programme a déjà été vérifiée.

Lorsque le programme, on a $b=0$, donc, d'après l'invariant, $a=a\times(b_0-0)=a_0\times b_0$ car la valeur de a est restée inchangée. La postcondition est vérifiée.

Corrigé 2 **1.** Dans la cas où $b<0$, la boucle n'est pas exécutée et la programme retourne a, qui n'est pas le bon résultat en général : le programme n'est pas faiblement correct sans spécifications adaptées.

2. **Préconditions** $P(a,b)$: a entier, b entier tel que $b>0$
Postcondition $Q(a,b,r)$: $r=pgcd(a,b)$.

3. On peut noter que b est un variant de boucle : b est positif et à chaque exécution de la boucle, b se voit affecter la valeur $a\%b$ donc il décroît strictement.

4. **Invariant de boucle** Notons a_0 et b_0 les valeurs initiales de a et b. Montrons que $pgcd(a,b)=pgcd(a_0,b_0)$ est un invariant de boucle.

La propriété est évidemment vraie avant le premier passage dans la boucle.

Supposons que l'invariant soit vrai en début de boucle. A la sortie de la boucle, a et b ont pris respectivement les valeurs de b et r. L'entier r est le reste de la division euclidienne de a par b, donc il existe un entier naturel q tel que $a=qb+r$.
Si un entier d divise a et b, il divise b et $r=a-qb$.
Réciproquement, si d divise b et r, il divise b et $a=qb+r$.
La liste des diviseurs communs à (a,b) et à (b,r) est identique, donc $pgcd(b,r)=pgcd(a,b)$, et l'invariant est conservé.

Correction L'algorithme termine lorsque $b=0$. On a dans ce cas $pgcd(a,b)=a$, d'où d'après l'invariant de boucle, $a=pgcd(a_0,b_0)$. Le résultat retourné vérifie bien la post-condition.

Corrigé 3 **1.** La spécification d'entrée est ici simple $P(n):n\geq 0$.

On peut écrire la spécification de sortie : $Q(n,s):s=\sum_{k=0}^{n}k$.

2. La terminaison ne pose pas de problème, le nombre d'exécution de la boucle étant prédéfini.

Il reste à montrer la correction de l'algorithme par un invariant adapté. Dans le cas d'une boucle `for`, il faut être attentif à l'endroit dans le code ou on vérifie cet invariant. Ici on va vérifier la correction de $s=\sum_{k=0}^{i}k$ à la fin de la ligne 3.

Initialisation Il est à noter que i n'étant pas défini avant l'entrée dans la boucle. On peut contourner le problème de deux manières.

- Vérifier que l'invariant n'est vérifié qu'après une première exécution de la boucle. Après celle-ci, on a $s = 0+0 = 0$ et $\sum_{k=0}^{0} k = 0$ donc l'invariant est vérifié.
- Plus élégant, considéré que l'indice de boucle étant incrémenté de 1 en 1, il vaut -1 avant l'entrée de la boucle, et que la somme vaut par convention 0 si $i < 0$.

Conservation Supposons maintenant que l'invariant est vérifié en début de boucle : $s = \sum_{k=0}^{i} k$. En fin de boucle, la nouvelle valeur de s est $s+(i+1) = \sum_{k=0}^{i} k + i + 1 = \sum_{k=0}^{i+1} k$ et l'invariant est vérifié.

Terminaison Lors de la dernière exécution de la boucle, on a $i = n$, et l'invariant permet de vérifier la spécification de sortie $s = \sum_{k=0}^{n} k$.

Corrigé 4 **1.** La précondition ne pose pas de problème particulier. $P(L)$: L est un tableau d'entiers.

Pour la postcondition, il faut utiliser la quantificateurs pour exprimer le fait que le résultat m est un élément du tableau et qu'il est le plus petit.

$$Q(L,m) : \exists j \in [\![0; len(L)-1]\!] \text{ tel que } L[j] = m \text{ et } \forall k \in [\![0; len(L)-1]\!], m \leq L[k]$$

2. La terminaison de l'algorithme ne pose pas de problème, le nombre de passage dans la boucle étant prédéfini.

On montra sa correction en utilisant un invariant exprimant le fait qu'en fin de boucle, m est le plus petit des éléments trouvés jusque là.

$$\exists j \in [\![0; i]\!] \text{ tel que } L[j] = m \text{ et } \forall k \in [\![0; i]\!], m \leq L[j]$$

Initialisation L'invariant est vrai en début de boucle si on suppose par convention que $i = 0$ avant le premier passage dans la boucle.

Conservation Supposons que l'invariant est vrai en début de boucle. On a alors deux cas possibles.

- Si $L[i+1] \geq m$, la valeur de m reste inchangée et m restant le plus petit élément des éléments du tableau d'indice compris entre 0 et $i+1$, l'invariant reste vérifié.
- $Si L[i+1] < m$, on a alors d'après l'invariant de boucle $\forall k \in [\![0; i]\!], L[i+1] \leq L[k]$ et comme $L[i+1]$ égal à lui même, $\forall k \in [\![0; i+1]\!], L[i+1] \leq L[k]$. La variable m prend la valeur $L[i+1]$ et les deux parties de l'invariant restent vérifiées.

Dans les deux cas, l'invariant reste vérifié.

Terminaison Lorsque l'algorithme termine, on a $i = n$: l'invariant nous donne directement la postcondition.

Corrigé 5 **1.** La précondition ne pose pas de problème. $P(L)$; L est une liste d'entiers.

Pour la postcondition, il suffit d'exprimer le fait que le résultat est vrai si est seulement si les p premiers éléments de la liste sont égaux aux p derniers en comptant de la fin, où $p = \llcorner \frac{n}{2} \lrcorner$ avec $n = len(L)$. (Notons que si n est impair, l'élément central, d'indice p, est forcément égal à lui-même.)

$Q(L,r) : r = True \Leftrightarrow \forall\, j \in [\![0; p-1]\!],\ L[j] = L[n-1-j]$

2. On peut remarque que $p - i$ est un variant de boucle, donc que l'algorithme termine.

3. On utilise un invariant de boucle exprimant le fait qu'à la fin de la boucle, les i premiers éléments sont égaux au i derniers lus dans l'ordre inverse : $\forall\, j \in [\![0; i-1]\!],\ L[j] = L[n-1-j]$.

Initialisation Si $i = 0$, l'intervalle $[\![0; i-1]\!]$ est vide et l'invariant est évidemment vérifié, donc l'invariant est vrai avant le premier passage dans la boucle.

Conservation Supposons l'invariant vrai en début de boucle. On a donc $\forall\, j \in [\![0; i-1]\!],\ L[j] = L[n-1-j]$.
Pour que la boucle s'exécute, on doit avoir $L[i] = L[n-1-i]$. En fin de boucle, l'indice a pris la valeur $i+1$ et l'invariant $\forall\, j \in [\![0; (i+1)-1]\!],\ L[j] = L[n-1-j]$ est vérifié.

Terminaison La boucle s'arrête si $i = p$ ou $L[i] \neq L[n-i]$.
Dans le premier cas, l'algorithme retourne $True$ et, d'après l'invariant de boucle, on a $\forall\, j \in [\![0; p-1]\!],\ L[j] = L[n-1-j]$. L'équivalence de la postcondition est donc vérifiée.
Dans le deuxième cas, l'algorithme retourne $False$. De plus, on a $L[i] \neq L[n-1-i]$, donc l'affirmation $\forall\, j \in [\![0; p-1]\!],\ L[j] = L[n-1-j]$ est fausse et l'équivalence de la postcondition est également vérifiée. L'algorithme donne donc bien le résultat attendu.

Corrigé 6 **1. Précondition** $P(T)$: T est un tableau de nombres de type entiers ou flottants.

Postcondition On doit exprimer le fait que le résultat r retourné par l'algorithme est égal à $True$ si et seulement si le tableau est trié, c'est à dire si et seulement si pour tout entier j compris entre 0 et $len(T)-2$, $T[i] \leq T[i+1]$. On peut pour cela utiliser une équivalence.

$Q(T,r)\ : r = True \Leftrightarrow \forall\, j \in [\![0; len(T)-2]\!], T[j] \leq T[j+1]$

Terminaison de l'algorithme On remarque que $len(T) - i$ est un variant de boucle. En effet, $len(T) - i$ reste strictement supérieur à 1 d'après la condition d'arrêt, et, à chaque passage dans la boucle, i augmente de 1 donc le variant diminue de 1.

Définition d'un invariant de boucle On exprime par un invariant de boucle le fait qu'à la fin de la $i^{\text{ème}}$ boucle, le tableau des élément d'indice 0 à i est trié. Cet invariant peut s'exprimer plus formellement par

$$R(i)\ : \forall\, j \in [\![0; i-1]\!], T[j] \leq T[j+1]$$

Pour $i = 0$, le tableau $[\![0; i-1]\!]$ est vide et l'invariant est vérifié.

Soit i un entier tel que $R(i)$ soit vérifié. Pour que la boucle s'exécute, on doit avoir $T[i] \leq T[i+1]$. Le tableau des élément allant d'indice O à $i+1$ est donc trié, et $R(i+1)$ est vérifiée. La propriété $R(i)$ est donc bien un invariant de boucle.

Vérification de la postcondition D'après la condition d'arrêt, l'algorithme s'arrête lorsque $i = len(T)-1$ ou $T[i] > T[i+1]$.

Dans le premier cas, la valeur r retournée par l'algorithme est $True$, et par ailleurs, d'après l'invariant de boucle, on a $\forall j \in [\![0; len(T)-2]\!], T[j] \leq T[j+1]$. L'équivalence entre les deux termes de la postcondition est donc vérifiée.

Dans le second cas, la valeur r retournée est $False$ et $T[i] > T[i+1]$, donc l'affirmation $\forall j \in [\![0; len(T)-2]\!], T[j] \leq T[j+1]$ est fausse : l'équivalence entre les deux termes de la postcondition est donc également vérifiée.

La postcondition est donc vérifiée, et l'algorithme est correct.

2. On propose maintenant une version de l'algorithme de recherche dichotomique d'un entier elt dans un tableau de nombres entiers **triés** T.

```
1: a, b ⟵ 0, len(T) − 1
2: c ⟵ (a + b)//2
3: while b − a ≥ 0 and T[c] ≠ elt do
4:    if T[c] < elt then
5:       a ⟵ c + 1
6:    else
7:       b ⟵ c − 1
8:    end if
9:    c ⟵ (a + b)//2
10: end while
11: return b ≥ a
```

a. Les variables a et b prennent initialement les valeurs 0 et 3.

Premier passage dans la boucle c prend la valeur 1. On a $T[1] = 1$ dont $T[1] < elt$ et a prend la valeur 2.

Second passage c prend la valeur 2. On a $T[2] = 3$ dont $T[2] > elt$ et b prend la valeur 1.

On a alors $b - a < 0$: les valeurs finales de a et b sont $a = 2$ et $b = 1$ et l'algorithme retourne $False$.

b. On peut remarque que $b - a$ est un invariant de boucle. En effet, d'une part, $b - a \geq 0$ d'après la condition d'arrêt.

D'autre part, lors du passage dans la boucle, on a $a \leq c \leq c$. Soit b prend la valeur $c-1$, et dans cas b diminue strictement et $b - a$ diminue strictement, soit a prend la valeur $c+1$, et dans ce cas a augmente strictement et $b - a$ diminue strictement.

c. **Précondition** T est un tableau de nombres entiers ou réels et elt un élément du même type que les éléments de T. De plus, T doit être trié, ce qui s'exprime par

$$P(T) : \forall i \in [\![0, len(T)-2]\!], T[i] \leq T[i+1]$$

Postcondition La postcondition exprime le fait que le résultat r est égal à $True$ si et seulement si elt appartient au tableau initial.

$$Q(r,T,elt) : r = True \Leftrightarrow \exists j \in [\![0; len(T)-1]\!] \, T[j] = elt$$

Invariant de boucle Attention, cet invariant peut être délicat à écrire. L'idée est qu'à une étape donnée, l'élément elt se trouvait dans le tableau initial si et seulement si il se trouve dans la sous-tableau dont les indices vont de a à b.

$$R(i) : \exists j \in [\![0; len(T)-1]\!] \setminus T[j] = elt \Leftrightarrow \exists j \in [\![a; b]\!] \setminus T[j] = elt$$

Avant la premier passage dans la boucle, $a = 0$ et $b = n-1$ donc l'invariant est trivialement vérifié.

Supposons que l'invariant soit vérifié à l'étape i.

La boucle ne s'exécute que si $T[c] \neq elt$. Le tableau étant initialement trié, si $T[c] > elt$, elt ne peut appartenir au tableau que si il est compris entre $T[a]$ et $T[c-1]$, et si $T[c] < elt$, elt ne peut appartenir au tableau que si il est compris entre $T[c+1]$ et $T[b]$. Dans ces deux cas, elt appartient au tableau initial si et seulement si l'indice de l'élément de T correspondant est compris entre les nouvelles valeurs de a et b en fin de boucle, et l'invariant est vérifié à l'étape $i+1$.

Correction D'après la condition d'arrêt, la boucle termine si et seulement si $b < a$ ou $T[c] = elt$.

Dans le premier cas, le résultat r vaut $False$. De plus, $[\![a; b]\!]$ est vide, donc il n'existe pas d'indice j dans cet ensemble tel que $T[j] = elt$ et donc, d'après l'invariant de boucle, il n'existe pas d'indice j dans $[\![0; len(T)-1]\!]$ tel que $T[j] = elt$. L'équivalence de la postcondition est vérifiée.

Dans le second cas, r vaut $True$ et $T[c] = elt$, donc l'équivalence de la postcondition est également vérifiée.

La correction de l'algorithme est donc vérifiée.