Terminaison et correction

INFORMATIQUE COMMUNE - TP nº 2.1 - Olivier Reynet

À la fin de ce chapitre, je sais :

☞ programmer les algorithmes donnés en exemples.
☞ prouver la terminaison d'un algorithme simple.
☞ prouver la correction d'un algorithme simple.

A Terminaison

A1. Prouver la terminaison de l'algorithme 1.

Algorithme 1 Palindrome

```
1 :  Fonction PALINDROME(w)
2 :      n ← la taille de la chaîne de caractères w
3 :      i ← 0
4 :      j ← n − 1
5 :      tant que i < j répéter
6 :          si w[i] = w[j] alors
7 :              i ← i + 1
8 :              j ← j − 1
9 :          sinon
10 :             renvoyer False
11 :     renvoyer Vrai
```

Solution : Si la condition en ligne 6 est invalidée, l'algorithme se termine. Si ce n'est pas le cas, on utilise le variant de boucle $v = j - i$. On vérifie qu'il est bien initialement positif (n-1), à valeurs entières et qu'il décroît strictement de deux unités à chaque tour de boucle. Nécessairement, v va donc atteindre la valeur 0. La condition $i < j$ est donc invalidée et la boucle se termine. L'algorithme palindrome se termine.

A2. Prouver la terminaison de l'algorithme 2.

Solution : Si la condition en ligne 2 est validée, l'algorithme se termine. Si le nombre n est impair, l'algorithme se termine également trivialement. Si ce n'est pas le cas, on utilise le variant de boucle $v = n$. On vérifie qu'il est bien initialement positif, à valeurs entières et qu'il décroît

Algorithme 2 Est une puissance de deux

```
Fonction EST_PUISSANCE_DE_DEUX(n)
    si n < 2 alors
        renvoyer Faux
    sinon
        m ← n mod 2
        tant que m = 0 répéter
            n ← n//2
            m ← n mod 2
        renvoyer n = 1
```

strictement (car divisé par deux en division entière) à chaque tour de boucle. Nécessairement, v va donc atteindre la valeur 1 (car n est pair et 2//2 vaut 1). La condition $m = 0$ est donc invalidée car $1 mod 2 = 1$ et la boucle se termine. L'algorithme est_puissance_de_deux se termine.

A3. Prouver la terminaison de l'algorithme récursif 3.

Algorithme 3 Somme des n premiers entiers

```
Fonction INT_SUM(n)
    si n=0 alors
        renvoyer 0
    sinon
        renvoyer n + INT_SUM(n-1)
```

Solution : On procède par récurrence sur n.

Initialisation : pour $n = 0$, l'algorithme se termine en renvoyant 0.

Hérédité : On suppose que l'algorithme se termine pour le paramètre $n-1$. L'opération $n + int\_sum(n-1)$ n'est qu'une addition est se termine donc.

Conclusion : l'algorithme se termine pour toute valeur de n.

B Correction

B1. Prouver la correction partielle de l'algorithme 4.

Solution : On choisit l'invariant $\mathcal{I}$: *à la fin de l'itération i, m est le plus grand élément de $t[0:i]$*.

Initialisation : à l'entrée de la boucle, $m = t[0]$. À la fin de l'itération pour $i = 1$, m est le plus grand élément de $t[0,1]$ à cause du test en ligne 8 et de l'affectation afférente en ligne 9.

Hérédité : On suppose que l'invariant est vérifié pour l'itération $k-1$, c'est à dire que m est le plus grand élément de $t[0:k-1]$. À la fin de l'itération k, si $t[k]$ est plus grand que m, alors celui-ci est affecté à m. Donc, m est le plus grand élément de $t[0:k]$ à la fin de l'itération k.

Algorithme 4 Élément maximum d'un tableau

Fonction MAX(t)
  si t est vide **alors**
    renvoyer $\emptyset$
  sinon
    $n \leftarrow$ la taille du tableau
    $m = t[0]$
    pour $i = 1$ à $n-1$ **répéter**
      si m < t[i] **alors**
        $m \leftarrow t[i]$
    renvoyer m

Conclusion : $\mathcal{I}$ est vérifié à chaque itération. C'est bien un invariant de boucle. À la sortie de la boucle, on a parcouru tout le tableau et m est le plus grand élément du tableau. L'algorithme est correct

B2. Prouver la correction partielle de l'algorithme de tri par sélection 5

Algorithme 5 Tri par sélection

Fonction TRIER_SELECTION(t)
  $n \leftarrow$ taille(t)
  pour i de 0 à $n-1$ **répéter**
    min_index ← i ▷ indice du prochain plus petit
    pour j de $i+1$ à $n-1$ **répéter** ▷ pour tous les éléments non triés
      si t[j] < t[min_index] **alors**
        min_index ← j ▷ c'est l'indice du plus petit non trié!
    échanger(t[i], t[min_index]) ▷ c'est le plus grand des triés!

Solution : On choisit les invariants suivants :

1. $\mathcal{I}_1$: $t[0:i]$ *est trié et ses éléments sont plus petits que les autres éléments de* t. pour la boucle sur i
2. $\mathcal{I}_2$: $t[min\_index]$ *est le plus petit élément de* $t[i:j]$. pour la boucle sur j.

On commence par prouver la correction de la boucle sur j, avec l'invariant $\mathcal{I}_2$.
Initialisation : à l'entrée de la boucle, min_index vaut i et $j = i+1$. À la fin de la première itération, t[min_index] est nécessairement le plus petit élément des deux $t[i]$ ou $t[i+1]$.
Hérédité : À la fin de l'itération k, si l'invariant est vérifié pour $k-1$, t[min_index] est nécessairement le plus petit élément de $t[k-1:k]$ et donc de $t[i:k]$.
Conclusion : $\mathcal{I}_2$ est vérifié à chaque itération. C'est bien un invariant de boucle.
Pour l'invariant $\mathcal{I}_1$:
Initialisation : pour $i = 0$, à la fin de l'itération, comme la boucle sur j est correcte, on a placé le plus petit élément du restant du tableau dans la case d'indice 0. Par ailleurs, le tableau $t[0]$ possède une seule case : il est donc trivialement trié. $\mathcal{I}_1$ est donc vérifié.

Hérédité : supposons que $\mathcal{I}_1$ soit vérifié à l'entrée de la kième itération. Alors $t[0:k-1]$ est correctement trié et tous les éléments du restant du tableau (à droite de $k-1$) sont plus grands que $t[k-1]$. Le minimum du restant du tableau de droite restant est alors placé à l'indice k. Comme il est plus grand que $t[k-1]$, le tableau $t[0:k]$ est correctement trié.

Conclusion : Comme le tableau est correctement trié pour $i=0$ et que l'invariant est héréditaire, l'invariant $\mathcal{I}_1$ est vérifié pour toutes les itérations de la boucle.

B3. Prouver la correction de l'algorithme du tri par insertion 6.

Algorithme 6 Tri par insertion

```
Fonction INSERTION(t, i)
    à_insérer ← t[i]
    j ← i
    tant que t[j-1] > à_insérer et j>0 répéter
        t[j] ← t[j-1]                          ▷ faire monter les éléments
        j ← j-1
    t[j] ← à_insérer                           ▷ insertion de l'élément
Fonction TRIER_INSERTION(t)
    n ← taille(t)
    pour i de 1 à n-1 répéter
        INSERTION(t,i)
```

Solution : On choisit d'abord de prouver la correction de l'algorithme d'insertion.

La terminaison de la fonction est garantie par j qui est un variant de la boucle tant que.

On utilise l'invariant suivant pour la correction de la boucle tant que $\mathcal{I}$: *à chaque itération, le tableau t[0:i] est correctement trié et t[j] = t[j+1].*

Initialisation : à l'entrée de la boucle, j vaut i. À la fin de la première itération, on a fait monter l'élément t[j-1] en i. Le tableau t[0:i] contient t[j-1] à l'indice i-1 et à l'indice i. Il est donc correctement trié et les deux derniers éléments sont égaux.

Hérédité : À la fin de l'itération, si l'invariant est vérifié, t[0,i] est trié et on a $t[j]=t[j+1]$. On fait monter (recopie) l'élément j-1 en j. Le tableau t[0,i] est toujours trié et $t[j-1]=t[j]$.

Conclusion : $\mathcal{I}$ est donc vérifié à chaque itération. C'est bien un invariant de boucle. À la fin de la boucle, on a t[j-1] < à_insérer et $t[j]=t[j+1]$. L'élément à_insérer se voit attribuer la place j : il n'écrase aucune valeur du tableau puisqu'on les a décalé. Le tableau t[0:i] est correctement trié.

Pour la correction de la fonction *trier_insertion*, on choisit l'invariant de boucle suivant : $\mathcal{I}$: *à chaque itération, le tableau t[0:i] est trié.*

Initialisation : à l'entrée de la boucle, i vaut 1 et t[0] est un tableau trivialement trié. L'insertion de l'élément t[1] dans le tableau étant correcte, le tableau t[0:1] est donc correctement trié à la fin de la première itération.

Hérédité : À la fin de l'itération k, si l'invariant est vérifié pour $k-1$, t[0,k-1]] est trié. Comme la fonction d'insertion est correcte, $t[0:k]$ est correctement trié à la fin de la kième itération.

Conclusion : $\mathcal{I}$ est vérifié à chaque itération. C'est bien un invariant de boucle. À la fin de la boucle, on a parcouru tous les éléments du tableau. t est donc complètement trié. L'algorithme est donc correct.

C Algorithme d'Euclide du PGCD

Algorithme 7 Algorithme d'Euclide (optimisé)

```
Fonction PGCD(a, b)                         ▷ On suppose que (a, b) ∈ ℤ, b ⩽ a.
    a ←| a |
    b ←| b |
    r ← a mod b
    tant que r > 0 répéter                  ▷ On connaît la réponse si r est nul.
        a ← b
        b ← r
        r ← a mod b
    renvoyer b                              ▷ Le pgcd est b
```

On cherche à prouver la terminaison et la correction de l'algorithme d'Euclide 7. Dans ce but, on rappelle quelques éléments mathématiques importants.

Théorème 1 — Division euclidienne . Soient $a \in \mathbb{Z}$ et $b \in \mathbb{N}^\star$. Alors il existe un unique couple $(q, r) \in \mathbb{Z} \times \mathbb{N}$ tel que les deux critères suivants sont vérifiés :

$$\begin{cases} a = bq + r \\ 0 \leqslant r < b \end{cases}$$

Démonstration.

1. Existence : a et b étant donné, on pose $q = \lfloor \frac{a}{b} \rfloor$. Par définition de partie entière, on a : $0 \leqslant \frac{a}{b} - \lfloor \frac{a}{b} \rfloor < 1$. En multipliant par b, on obtient : $0 \leqslant a - b \times \lfloor \frac{a}{b} \rfloor < b$. En choisissant donc $q = \lfloor \frac{a}{b} \rfloor$ et $r = a - b \times \lfloor \frac{a}{b} \rfloor$, on a bien :
$$\begin{cases} a = bq + r \\ 0 \leqslant r < b \end{cases}$$

2. Unicité : supposons que l'on ait deux couples (q, r) et (q', r') appartenant à $\mathbb{Z} \times \mathbb{N}$: $a = bq + r = bq' + r'$ avec $0 \leqslant r < b$ et $0 \leqslant r' < b$. Cela peut également s'écrire : $b(q' - q) = r - r'$. Or, on a l'encadrement $-b < r - r' < b$. On en conclut que $-b < b(q' - q) < b$ et donc que $-1 < q' - q < 1$. Mais q et q' sont des entiers d'après nos hypothèses de départ. Donc, on en déduit de $q' - q = 0$. Il s'en suit que $q = q'$ et que $r = r'$. Il s'agit donc bien du même couple.

■

Théorème 2 — Existence du PGCD. Parmi tous les diviseurs communs de deux entiers a et b non nuls, il y en a **un** qui est le plus grand. Ce dernier est nommé plus grand commun diviseur de a et de b. On le note $\mathrm{PGCD}(a, b)$.

Démonstration. Soit $a \in \mathbb{N}^\star$. Tous les diviseurs de a sont bornés par $|a|$. On peut tenir le même raisonnement pour ceux de b. Donc, parmi les diviseurs de a et de b, il y en a donc un plus grand. ■

Théorème 3 — Propriété du PGCD. Soit a et b deux entiers.

1. Si $b = 0$, alors $\mathrm{PGCD}(a, b) = a$.
2. Si $b \neq 0$, alors $\mathrm{PGCD}(a, b) = \mathrm{PGCD}(b, a \mod b)$.

Démonstration. Démonstration de l'égalité de l'ensemble $\mathcal{D}_{ab}$ des diviseurs de a et de b et de l'ensemble $\mathcal{D}_{br}$ des diviseurs de b et de r par double inclusion.

$\mathcal{D}_{ab} \subset \mathcal{D}_{br}$ **:** La division euclidienne étant unique comme nous l'avons montré au théorème 1, il existe un entier q tel que $a = qb + r$. Ce qui peut s'écrire : $a - qb = r$. Si γ est un diviseur de a et de b, alors on peut écrire : $a - bq = \gamma a' + \gamma b' q = \gamma(a' - b'q) = r$. On a donc montrer qu'un diviseur de a et de b est un diviseur de r.

$\mathcal{D}_{br} \subset \mathcal{D}_{ab}$ **:** De même, si η est un diviseur de b et de r, alors on a : $a = bq + r = \eta(b'q + r')$, ce qui signifie que η est un diviseur de a.

Donc, $\mathcal{D}_{ab} = \mathcal{D}_{br}$. Ceci est vrai, y compris pour le plus grand des diviseurs de a et de b. ■

■ **Définition 1 — Suite des restes de la division euclidienne.** Soient a et b des entiers. On définit la suite des restes de la division euclidienne comme suit :

$$r_0 = |a| \tag{1}$$
$$r_1 = |b| \tag{2}$$
$$q_k = \lfloor r_{k-1}/r_k \rfloor, 1 \leqslant k \leqslant n \tag{3}$$

Alors on a :

$$r_{k-1} = q_k r_k + r_{k+1} \tag{4}$$
$$r_{k+1} = r_{k-1} \mod r_k \tag{5}$$

Théorème 4 — Stricte décroissance de $(r_n)_{n\in\mathbb{N}}$. La suite des restes de la division euclidienne est positive, strictement décroissante et minorée par zéro.

C1. Donner une preuve du théorème 4.

> **Solution :** D'après le théorème 1, le reste r de la division euclidienne de a et de b est tel que : $0 \leqslant r < b$. Donc, la suite est minorée par zéro. Cette borne est atteinte lorsque r_k est un multiple de r_{k-1}. C'est une suite positive car elle est initialisée à des valeurs positives. Elle est strictement décroissante car $r_{k-1} < r_k$ d'après la définition de la division euclidienne 1.

C2. Montrer que r est un variant de boucle pour l'algorithme d'Euclide.

Solution : On observe qu'un élément de la suite des restes est calculé à chaque tour de boucle (cf. algorithme 7 ligne 8). D'après la question précédente, r est positif, **strictement** décroissant et minoré par zéro. r est donc un variant de boucle. La condition d'arrêt, r est donc atteinte. Le programme se termine.

C3. Prouver la correction de l'algorithme d'Euclide.

Solution : On choisit l'invariant $\mathcal{I}$: *à chaque tour de boucle, le PGCD de b et r est le PGCD des deux nombres passés en paramètres.*.

Initialisation : L'invariant est vérifié à l'entrée de la boucle car $r = a \mod b$ et d'après le point 2 du théorème 3 on a $\text{PGCD}(a, b) = \text{PGCD}(b, a \mod b)$.

Hérédité : Si l'invariant est vérifié à l'entrée de la boucle, la propriété du PGCD fait qu'il est vérifié à la fin de la boucle.

Conclusion : À la sortie de la boucle, le reste est nul (d'après la démonstration de la terminaison) et la propriété du PGCD nous indique que le PGCD de b et r est le PGCD recherché. Or $\text{PGCD}(b, 0) = b$. Le PGCD vaut donc b, ce que renvoie la fonction. L'algorithme est correct.