Correction et terminaison

Algorithmique

Questions fondamentales

— Donne-t-il un résultat ou ne s'arrête-t-il jamais ? (Notion de **terminaison**)

— Donne-t-il le résultat attendu ou calcule-t-il autre chose ? (Notion de **correction**)

— Donne-t-il le résultat en un temps raisonnable ou faut-il attendre très longtemps ? (Notion de **complexité**)

Ce chapitre traite des deux premières questions. La complexité fait l'objet d'un chapitre dédié.

Exemple fil rouge : la division euclidienne

Basons-nous sur un exemple concret : celui de la division euclidienne.

```
def division(a : int, b : int):
    """
    Entrée : a et b sont deux entiers positifs non nuls
    Sortie : quotient q et reste r de la division euclidienne de a par b
    """
    q = 0
    r = a
    while r >= b:
        q = q + 1
        r = r - b
    return (q, r)

print(division(22, 3))
```

La trace d'un algorithme consiste à noter la valeur de chaque variable à chaque étape. C'est un outil essentiel pour comprendre le fonctionnement d'un programme.

Écrivons la trace pour $a = 22$ et $b = 3$:

Étape	q	r	$r \geq b$
Initialisation	0	22	Vrai
Tour 1	1	19	Vrai
Tour 2	2	16	Vrai
Tour 3	3	13	Vrai
Tour 4	4	10	Vrai
Tour 5	5	7	Vrai
Tour 6	6	4	Vrai
Tour 7	7	1	Faux

À la sortie de la boucle, $q = 7$ et $r = 1$. On a bien $22 = 3 \times 7 + 1$ avec $0 \leq 1 < 3$.

Preuve de la terminaison : le variant de boucle

Pour montrer que cet algorithme se termine, on doit prouver que la boucle `while` ne s'exécute pas indéfiniment. Pour cela nous allons utiliser ce que l'on appelle un *variant de boucle*.

Un **variant de boucle** est une quantité qui doit :

— être un **entier positif ou nul** à l'entrée de chaque tour de boucle ;

— **décroître strictement** à chaque itération.

Puisqu'un entier positif qui décroît strictement finit nécessairement par atteindre 0 ou devenir négatif, la boucle se terminera forcément.

Application à la division euclidienne

Dans notre exemple, un bon variant de boucle est r lui-même (ou plus précisément, la valeur de `r` au début de chaque tour).

— **Positif à l'entrée de chaque tour ?** Oui, car la boucle ne s'exécute que si $r \geq b > 0$, donc $r > 0$.

— **Décroissance stricte ?** Oui, car à chaque tour, r est remplacé par $r - b$, et $b > 0$, donc la nouvelle valeur de r est strictement inférieure à l'ancienne.

La boucle se termine donc nécessairement.

Preuve de la correction : l'invariant de boucle

Un **invariant de boucle** est une proposition qui :

— est vérifiée **avant** la première itération (initialisation) ;

— **reste vraie** après chaque itération (conservation).

Un invariant bien choisi, combiné à la condition d'arrêt de la boucle, permet de montrer que l'algorithme produit le résultat attendu.

Application à la division euclidienne

L'invariant de boucle est

$$a = b \times q + r \quad \text{avec } r \geq 0.$$

Initialisation. Avant la boucle, $q = 0$ et $r = a$, donc $b \times q + r = b \times 0 + a = a$. De plus, $r = a \geq 0$ car a est un entier positif. L'invariant est vérifié.

Conservation. Supposons que $a = b \times q + r$ et $r \geq 0$ au début d'un tour. Après les affectations $q \leftarrow q + 1$ et $r \leftarrow r - b$, on obtient :

$$b \times (q + 1) + (r - b) = bq + b + r - b = bq + r = a.$$

De plus, le tour n'a lieu que si $r \geq b$, donc la nouvelle valeur $r - b$ est positive ou nulle. L'invariant est conservé.

Conclusion. À la sortie de la boucle, l'invariant $a = bq + r$ est toujours vrai, et la condition d'arrêt nous donne $r < b$. On a donc bien $a = bq + r$ avec $0 \leq r < b$: c'est exactement la définition de la division euclidienne. L'algorithme est correct.

Méthode générale

1. Choisir et exprimer un invariant judicieux.
2. Démontrer qu'il est vérifié avant d'entrer dans la boucle en utilisant les préconditions, c'est-à-dire les conditions que doivent vérifier les entrées (ici, $a \geq 0$ et $b > 0$).
3. Démontrer que s'il est vérifié au début d'une itération quelconque, il l'est aussi au début de l'itération suivante (utiliser le corps de la boucle).
4. Instancier l'invariant en sortie de boucle et, grâce à la négation de la condition du `while`, en déduire une postcondition, c'est-à-dire une propriété vraie à la sortie de l'algorithme (ici, $a = bq + r$ avec $0 \leq r < b$).

Un deuxième exemple : la multiplication par additions successives

```
def multiplication(a, b):
    """
    Entrée : a et b sont deux entiers naturels
    Sortie : le produit a * b
    """
    resultat = 0
    compteur = 0
    while compteur < b:
        resultat = resultat + a
        compteur = compteur + 1
    return resultat

print(multiplication(7, 5))
```

Terminaison. Le variant de boucle est b − compteur. Cette quantité est un entier positif (car la boucle s'exécute tant que compteur $< b$) et décroît strictement de 1 à chaque tour. La boucle se termine.

Correction. L'invariant de boucle est : resultat = $a \times$ compteur.

— *Initialisation.* resultat = 0 et compteur = 0, donc resultat = $a \times 0$. Vérifié.

— *Conservation.* Si resultat = $a \times$compteur au début d'un tour, après les affectations on obtient resultat+a = $a \times$ compteur + a = $a \times$ (compteur + 1). Vérifié.

— *Conclusion.* En sortie de boucle, compteur = b (car la boucle s'arrête quand compteur $\geq b$), donc resultat = $a \times b$. L'algorithme est correct.

Exercices

Exercice 1 Écrire la trace de l'algorithme de division euclidienne pour $a = 15$ et $b = 4$.

Exercice 2 Pour chacune des boucles suivantes, proposer un variant de boucle et prouver que la boucle se termine.

a)

```
def mystere_a(n):
    i = n
    while i > 0:
        i = i - 1
    return i
```

b)

```
def mystere_b(n):
    i = 1
    while i < n:
        i = i * 2
    return i
```

Exercice 3 Considérons l'algorithme suivant qui calcule a^n :

```
def puissance(a, n):
    """Entrée : a un nombre, n un entier naturel
       Sortie : a^n"""
    resultat = 1
    compteur = 0
    while compteur < n:
        resultat = resultat * a
        compteur = compteur + 1
    return resultat
```

1. Écrire la trace pour $a = 2$ et $n = 4$.
2. Proposer un variant de boucle et prouver la terminaison.
3. Proposer un invariant de boucle et prouver la correction.

Exercice 4 Considérons l'algorithme suivant qui recherche le maximum d'une liste :

```
def maximum(L):
    """Entrée : L une liste non vide d'entiers
       Sortie : le plus grand élément de L"""
```

```
    m = L[0]
    for i in range(1, len(L)):
        if L[i] > m:
            m = L[i]
    return m
```

1. Proposer un invariant de boucle pour cet algorithme. (Indication : que peut-on dire de `m` par rapport à `L[0], L[1], …, L[i]` ?)
2. En déduire que l'algorithme est correct.

Exercice 5 L'algorithme d'Euclide calcule le PGCD (plus grand commun diviseur) de deux entiers naturels :

```
def pgcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

1. Écrire la trace pour $a = 48$ et $b = 18$.
2. Proposer un variant de boucle et prouver la terminaison.
3. Quel est l'invariant de boucle ? (Indication : on admet que, pour $b \neq 0$, $\text{pgcd}(a, b) = \text{pgcd}(b, r)$, où r est le reste de la division euclidienne de a par b, calculé en Python par `a % b` ; on pourra en déduire que $\text{pgcd}(a, b)$ ne change pas d'une itération à l'autre.)

Exercice 6 L'algorithme suivant prétend calculer la somme des entiers de 1 à n. Montrez, en utilisant un invariant de boucle, que cet algorithme est **incorrect**. Corrigez-le ensuite.

```
def somme_fausse(n):
    s = 0
    i = 0
    while i < n:
        s = s + i
        i = i + 1
    return s
```