Chapitre 7

Terminaison et correction de boucles

7.1 Introduction

• La programmation est une activité complexe. Un programme manipule un nombre de données pouvant être important, dont les valeurs fluctuent durant l'exécution du programme.

• Il est important de pouvoir justifier rigoureusement qu'un programme s'arrête et retourne bien le résultat recherché.

• C'est ce que l'on appelle respectivement la *terminaison de programme* et *correction de programme*.

•Nous allons faire un peu de terminaison et de correction de programmes simples consistant en de simples itérations d'une boucle `for` ou `while`.

7.2 Terminaison de boucle

7.2.1 Boucle `while`

• Lors de l'utilisation d'une boucle `while`, il est important de s'assurer que la boucle ne tournera pas indéfiniment.

• L'exemple suivant boucle infiniment et nécessite de forcer l'arrêt.

```
k = 10
while (k >= 0):
    print("bonjour")
    k = k+1
```

• Lors de l'utilisation d'une boucle `while`, il est important de s'assurer que la boucle ne tournera pas indéfiniment.

• L'exemple suivant boucle infiniment et nécessite de forcer l'arrêt.

```
k = 9
while (k >= 0):
    print("bonjour")
    k = k+1
```

• Il faut justifier que la condition de la boucle `while`, ici `(k >= 0)` finira nécessairement par ne plus être satisfaite.

C'est mieux ainsi :

```
k = 9
while (k >= 0):
    print("bonjour")
    k = k-1
```

• La boucle s'arrêtera : en effet k prend des valeurs entières ET décroît strictement à chaque passage dans la boucle. Il finira donc par ne plus être positif : la condition finira par ne plus être satisfaite.
(Au bout de 10 itérations, `"bonjour"` sera écrit 10 fois).

• Attention les deux conditions `k` entier et strictement monotone sont en généralement nécessaires pour que `k` finisse par dépasser une valeur seuil ("plancher" dans le cas décroissant ou "plafond" dans le cas croissant).

```
k = 2
n = 0
while (k >= 0):
    k = k - 1/(2**n)
    n = n + 1
```

• Avec des valeurs non entières la condition que `k` est strictement décroissante ne suffit pas en général pour que `k` prenne des valeurs strictement négatives. En effet ici $\forall\, n \in \mathbb{N}$, $\sum_{k=0}^{n} \frac{1}{2^k} < 2$. Avec des valeurs réelles et des capacités infinies la boucle tournerait indéfiniment car k ne prendrait que des valeurs positives.

• L'éxécution produit ici aussi une boucle infini, k finissant par prendre une valeur constante très petite mais > 0.

Avec des valeurs de k non entières.

• Exemple :

```
k = 2
n = 1
while (k >= 0):
    k = k - 1/n
    n = n + 1
```

• La boucle s'arrête :

En effet la suite $\sum_{i=1}^{n} \frac{1}{i}$ est strictement croissante et n'est pas majorée par 2 (on démontre que $\lim_{n\mapsto\infty} \sum_{i=1}^{n} \frac{1}{i} = +\infty$).

$\Longrightarrow$ La suite $k_n = 2 - \sum_{i=1}^{n} \frac{1}{i}$ est strictement décroissante et n'est pas minorée par 0. Donc `k` finit par prendre une valeur strictement négative et la condition `k` positif de la boucle `while` finit par ne plus être vérifiée.

7.2.2 Variant de boucle

Définition : On appelle **variant de boucle** une quantité entière et positive qui décroît strictement à chaque passage dans la boucle.

• Ainsi si l'on exhibe un variant de boucle, nécessairement la boucle finit par s'arrêter.

• Exemple : Algorithme d'Euclide pour le calcul du pgcd de deux entiers naturels.

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

• On a toujours $0 \leqslant$ `a % b` $<$ `b` puisque `a % b` est le reste dans la division euclidienne de `a` par `b`.

Donc `b` est un entier positif dont la valeur décroît strictement à chaque passage dans la boucle (instruction `b = a % b`).

Donc `b` est un variant de boucle. La boucle s'arrête.

7.2.3 Boucle `for`

• Pour une boucle `for` : En général une boucle `for` finit toujours par s'arrêter : sauf si l'on fait n'importe quoi :

Il faut s'interdire de rajouter des éléments à une liste parcourue :

```
L = [1,2,3]
for i in L:
    print i
    L.append(0)    # Ajout de l'élément 0 en fin de liste
```

Produit :

```
1
2
3
0
⋮
```

boucle sans fin. Il faut forcer l'arrêt du programme.

• **Pour une boucle `for` bien écrite le problème de la terminaison ne se pose pas : c'est une raison pour préférer une boucle for à une boucle while lorsque c'est possible (lorsque nombre d'itérations connues à l'avance).**

7.3 Invariants de boucle

7.3.1 Présentation

• Prouver qu'un programme finit par s'arrêter est une première étape. Prouver qu'il retourne le résultat escompté est une deuxième étape.

• Concentrons nous sur un programme simple, constitué d'une boucle.

• Prouver que le résultat calculé dans une boucle est bien le résultat attendu peut souvent se faire grâce à un *Invariant de boucle*.

Définition Pour une boucle, un invariant de boucle est

- une propriété qui est vraie avant l'entrée dans la boucle et qui reste vraie après chaque passage dans la boucle.
- ou une quantité numérique qui demeure inchangée après chaque passage dans la boucle.

• Ainsi la propriété reste vraie à la sortie de la boucle.

```
⋮
# Invariant I vrai avant la boucle
for (ou while) ...
...    ⋮
...    # Invariant I vrai après chaque passage
# => Invariant I vrai après la boucle
```

7.3.2 Exemple : l'algorithme d'Euclide

• Algorithme d'Euclide pour le pgcd.

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

Ici on peut regarder la quantité numérique $pgcd(a,b)$ qui reste inchangée à chaque passage dans la boucle. En effet si r désigne le reste dans la division euclidienne de a par b, et q le quotient, alors :

– $a = q.b + r$ donc tout diviseur de b et r est aussi diviseur de a et b.
– $r = a - q.b$ donc tout diviseur de a et b est aussi diviseur de b et r.
– Ainsi a, b et b, r ont mêmes diviseurs communs :

• Soit d diviseur de b et r : $r = dr'$ et $b = db'$.
Alors $a = qdb' + dr' = d(qb' + r')$ donc d divise aussi a (et b).

• Soit d diviseur de a et b : $a = da'$ et $b = db'$.
Alors $r = a'd - qb'd = (a' - qb')d$ donc d divise aussi r (et b).
$\Longrightarrow$ En particulier a, b et b, r ont même PLUS GRAND diviseur commun :

$$\boxed{pgcd(a,b) = pgcd(b,r)}$$

• Soit r le reste de la division euclidienne de a par b : `r = a % b`. Alors $pgcd(a,b) = pgcd(b,r)$.

• Soient a_k, b_k et r_k les valeurs de `a,b,r` au k-ième passage dans la boucle :

$a_{k+1} = b_k$, $b_{k+1} = r_k$.

Donc $pgcd(a_{k+1}, b_{k+1}) = pgcd(b_k, r_k) = pgcd(a_k, b_k)$.
Ainsi la valeur de $pgcd(a,b)$ est invariante. C'est l'**invariant de boucle**

• La boucle s'arrête lorsque $b = 0$. Le programme retourne a qui égale $pgcd(a,0)$ mais aussi $pgcd(a_k, b_k)$ pour un certain entier k.

• Puisque $pgcd(a,b) = pgcd(a_k, b_k)$ est un invariant, le résultat retourné est bien le pgcd des paramètres `a` et `b`. □

On a démontré :

Théorème 1 *Avec pour paramètres deux entiers naturels a et b, l'algorithme d'Euclide retourne* $pgcd(a,b)$.