Plan du chapitre

1 Preuve de correction d'une boucle énumérative

Dans ce paragraphe on présente la méthode permettant de prouver qu'un algorithme utilisant une boucle énumérative répond au problème posé. On parle de correction de l'algorithme au sens où l'on démontre que l'algorithme est donne une réponse correcte au prblème posé.

Exercice 1

Que renvoie la fonction suivante ?

```
1 def f(n):
2       p=1
3       for i in range(1,n+1):
4             p*=i
5       return p
```

SOLUTION. Pour démontrer la conjecture effectuée, on va raisonner par récurrence, suivant le même principe qu'en mathématiques. On définit, pour $k \in [\![0, n]\!]$, la propriété :

HR(k) : "à l'issue de la k-ième itération de la boule énumérative, la variable p contient $k!$".

On adopte la convention que la 0-ième itération 0 correspond à l'initialisation.

Notons tout de suite que lors de la k-ième itération, l'indice `i` vaut k et que la boucle énumérative contient n itérations.

— Si $k = 0$, la propriété HR(0) est vérifiée car $p = 1 = 0!$.

— Soit $k \in [\![0, n-1]\!]$. Si HR(k) est vérifiée cela signifie qu'à l'issue de la k-ième itération, la variable p contient $k!$.

Ainsi, à l'issue de l'itération suivante, la variable p contient donc $p \times (k+1) = (k+1)!$ car la variable $i = k+1$ lors de la $k+1$-ième itération.

Par récurrence, on en déduit que $HR(k)$ est vraie pour tout $k \in [\![0, n]\!]$ et par conséquent $HR(n)$ est vraie : p contient $n!$ à l'issue de la n-ième (et dernière) itération.

□

Définition 2

On appelle la propriété HR(k) un **invariant de boucle** : c'est-à-dire une propriété valide à chaque itération de la boucle énumérative `for`.

Exercice 3

Que renvoie la fonction suivante ?

```
1 def s(n):
2       s=0
3       for i in range(n):
4             s+=i
5       return s
```

Le démontrer à l'aide d'un invariant de boucle.

SOLUTION. On définit pour $k \in [\![0, n]\!]$:

$$\text{HR}(k) : \text{« à l'issue de la } k\text{-ième itération, } s = \sum_{i \in [\![0,k-1]\!]} i \text{ ».}$$

On adopte la convention que la 0-ième itération correspond à l'initialisation et qu'une somme sur un ensemble vide est nulle.

Notons tout de suite que la boucle énumérative contient n itérations et que lors de la k-ième itération, l'indice `i` est égal à $k-1$.

— HR(0) est vraie car à l'issue de l'initialisation, la variable s est nulle et la somme $\sum_{i \in [\![0,k-1]\!]} i$ est nulle car portant sur l'ensemble d'indice $[\![0, -1]\!] = \emptyset$.

— Soit $k \in [\![0, n-1]\!]$. Si HR(k) est vérifiée, cela signifie qu'à l'issue de la k-ième itération, $s = \sum_{i=0}^{k-1} i$.

A l'issue de la $k+1$-ième itération, on a donc

$$s = \left(\sum_{i=0}^{k-1} i\right) + k = \sum_{i=0}^{k} i$$

car lors de la $k+1$-ième itération, l'indice `i` est égal à k.

On conclut par récurrence et on en déduit en particulier que HR(n) est vraie.

À l'issue de la n-ième (et dernière itération) $s = \sum_{k=0}^{n-1} i$ c'est-à-dire $\frac{n(n-1)}{2}$.

□

Exercice 4

Que renvoie la fonction suivante ?

```
1 def maxi(L):
2     m=L[0]
3     for i in range(1,len(L)):
4         if L[i]>m:
5             m=L[i]
6     return m
```

Le démontrer à l'aide d'un invariant de boucle.

SOLUTION. Soit `L` une liste et $n =$ `len(L)`.

On définit pour $k \in [\![0, n-1]\!]$:

$$\text{HR}(k) : \text{« à l'issue de la } k\text{-ième itération, } m = \max \texttt{L[0:k+1]} \text{ ».}$$

On adopte la convention que la 0-ième itération correspond à l'initialisation et qu'une somme sur un ensemble vide est nulle.

Notons tout de suite que la boucle énumérative contient $n-1$ itérations et que lors de la k-ième itération, l'indice `i` est égal à k.

— HR(0) est vraie car à l'issue de l'initialisation, la variable `m` est égale à `L[0]` qui est bien le maximum de la liste partielle `L[0:1]` (sous-liste réduite à `L[0]`).

— Soit $k \in [\![0, n-2]\!]$. Si HR(k) est vérifiée, cela signifie qu'à l'issue de la k-ième itération,

$$\texttt{m} = \max \texttt{L[0:k+1]}.$$

Lors de la $k+1$-ième itération `i` $= k+1$.
Dans cette itération de la boucle `for`, on compare `m` à L[k+1] :
— Si `L[k+1]` $> m$ alors on a trouvé un élément de `L` plus grand que le maximum partielle de la sous-liste `L[0:k+1]`. La valeur `m` est modifiée et vaut alors `m=L[k+1]`. Par conséquent `m` est le maximum de la liste partielle `L[0:k+2]`.
— Sinon, c'est-à-dire si `L[k+1]` $\leqslant m$ alors le maximum de la liste partielle `L[0:k+1]` est encore le maximum de la liste partielle `L[0:k+2]`. La valeur de `m` dans ce cas n'est pas modifiée.
Dans les deux cas, à l'issue de la $k+1$-ième itération `m` est bien le maximum de la liste partielle `[0:k+2]`.
On conclut par récurrence et on en déduit en particulier que HR($n-1$) est vraie : à l'issue de la $n-1$-ième (et dernière) itération `m` contient le maximum de la liste `L[0:n-1+1]` c'est-à-dire le maximum de la liste `L[0:n]=L`. □

Exercice 5

Écrire une fonction `puissance2(n)` calculant la puissance `n` de 2 et démontrer que la fonction réalise le travail demandé à l'aide d'un invariant de boucle.

SOLUTION.

```
1 def puissance2(n):
2     p=1
3     for i in range(1,n+1):
4         p=2*p
5     return p
```

On utilise l'invariant de boucle, pour $k \in [\![0, n]\!]$: $HR(k)$: «$p = 2^k$».

□

Exercice 6

Une méthode naturelle de tri d'une liste d'objets consiste à insérer un par un les objets, à leur place, dans une liste partiellement triée. On parle de tri par insertion.
C'est par exemple ce que l'on fait lorsqu'on classe des copies d'élèves par ordre alphabétique. La première copie étant posée sur la table, la deuxième copie se place avant ou après celle-ci en suivant l'ordre alphabétique. La troisième copie est ensuite classée parmi les deux précédentes et ainsi de suite. On parle de tri par insertion.

1. Écrire une fonction `tri_insertion` d'argument `L` une liste d'entiers naturels et renvoyant la liste composée des mêmes éléments que `L`, triée par ordre croissant.
2. Prouver la correction de cette fonction.

SOLUTION.

```
def tri_insertion(L):
    t=[L[0]]
    for i in range(1,len(L)):
        k=0
        while k<len(t) and L[i]>t[k]:
            k+=1
        t.insert(k,L[i])
    return t
```

On définit pour $k \in [\![0, n-1]\!]$, la propriété :

$HR(k)$: «à l'issue de la k-ième itération, la liste `t[0:k+1]` est triée».

Il s'agit d'un invariant de boucle. □

2 Preuve de terminaison et de correction d'une boucle conditionnelle

2.A Preuve de terminaison

Commençons par étudier un exemple :

Exercice 7

```
def f(u):
    while u>0:
        if u%2==0:
            u=u//2
        else:
            u=3*u+1
        print(u)
```

Que renvoient les 10 premiers affichages `print(u)` produits par l'appel $f(10)$?
Que peut-on prévoir ?

L'exemple précédent est un exemple de boucle `while` infinie : la boucle conditionnelle ne se termine pas. On en trouve des exemples moins élaborés parfois dans les copies !

```
i=1
while i<10:
    i=1
```

```
n=0
while n*(n+1)%2==0:
    n+=1
```

On comprend donc l'intérêt de prévoir, et savoir démontrer, qu'une boucle conditionnelle `while` se termine. On parle de preuve de **terminaison** d'une boucle conditionnelle.

Exercice 8

Prouver la terminaison de la boucle while dans la fonction suivante :

```
def f(n):
    """renvoie la plus petite puissance de 2
    supérieure ou égale à n
    avec n un entier naturel non nul"""
    p=1
    while p<n:
        p=2*p
    return p
```

SOLUTION. L'entier `p` vaut initialement 1 et il est multiplié par 2 à chaque itération de la boucle `while`.

La suite $(2^p)_{p\in\mathbb{N}}$ diverge vers $+\infty$ donc pour tout $n \in \mathbb{N}$ fixé, il existe $p_0 \in \mathbb{N}$ tel que $2^{p_0} > n$.
La condition $p < n$ va donc cesser d'être réalisée au bout d'un certain nombre p_0 d'itérations.
La boucle `while` se termine donc bien après un certain nombre p_0 d'itérations. □

Remarques

On peut même prévoir le nombre d'itérations : $\left\lfloor \frac{\ln(n)}{\ln(2)} \right\rfloor + 1$.

Définition 9

Un variant de boucle est une variable (souvent un nombre) qui varie lors de l'exécution d'une boucle conditionnelle.

..

Exercice 10

Prouver la terminaison de la boucle `while` lors de l'appel `somme(L)` où `L` une liste de nombres.

```
def somme(L):
    """renvoie la somme des éléments de L"""
    i=0
    s=0
    while i<len(L):
        s+=L[i]
        i+=1
    return s
```

SOLUTION. La variable `i` est un variant de boucle dont la valeur augmente d'une unité à chaque itération. La suite $(i)_{i\in\mathbb{N}}$ diverge vers $+\infty$ donc pour tout $n \in \mathbb{N}$ fixé avec `n=len(L)` il existe $i_0 \in \mathbb{N}$ tel que $i_0 \geqslant n$. La boucle `while` va donc se terminer. □

Exercice 11

Prouver la terminaison de la boucle `while` lors de l'appel `palindrome(m)` avec `m` une chaîne de caractères.

```
1  def palindrome(m):
2       i=0
3       j=len(m)-1
4       test=True
5       while i<=j and test:
6            if m[i]==m[j]:
7                 i+=1
8                 j-=1
9            else:
10                test=False
11      return test
```

SOLUTION. La variable `j-i` est un variant de boucle. A chaque itération de la boucle `while` :
— Si la condition `m[i]==m[j]` n'est pas vérifiée alors on sort de la boucle `while` et de la fonction avec la commande `Return False`. Dans ce cas la terminaison de la boucle est assurée.
— Si la condition `m[i]==m[j]` est vérifiée alors la variable `j-i` est diminuée de deux unités.

La suite $(j - i - 2k)_{k\in\mathbb{N}}$ est une suite d'entiers strictement décroissante donc nécessairement négative ou nulle à partir d'un certain rang. La boucle conditionnelle se termine donc. □

Exercice 12

Prouver la terminaison de que la boucle `while` de la fonction suivante :

```
1  def division(a,b):
2       """renvoie le quotient et le reste
3       de la division euclidienne de a par b>0"""
4       q=0
5       r=a
6       while r>=b:
7            q+=1
8            r-=b
9       return q,r
```

SOLUTION. La variable `r` est un variant de boucle.

Cette variable diminue de `b>0` à chaque itération donc la condition `r>=b` ne sera plus vérifiée à partir d'une certaine itération. □

2.B Preuve de correction d'une boucle `while`

De même que l'on démontre la correction d'une boucle `for` avec un **invariant de boucle**, on démontrera qu'une propriété est vraie à chaque itération de la boucle `while`. Cette propriété sera en particulier vraie à la fin de la dernière itération avant la sortie de boucle. On parle encore d'invariant de boucle. La différence est que le raisonnement, à rapprocher encore d'une récurrence, dépend moins explicitement du numéro de l'itération, puisque précisément on ne prévoit pas le nombre d'itérations lorsqu'on exécute une boucle `while`. C'est d'ailleurs son intérêt.

Exercice 13

Que renvoie la fonction `division(a,b)` lorsque `a,b` sont des entiers naturels avec $b \neq 0$?

```
1 def division(a,b):
2     q=0
3     r=a
4     while r>=b:
5         q+=1
6         r-=b
7     return q,r
```

On pourra démontrer que la propriété $a = b \times q + r$ est un invariant de boucle.

SOLUTION. La propriété est vraie lors de l'initialisation car $q = 0$ et $r = a$. On a bien $b \times q + r = b \times 0 + a = a$.

Si la propriété est vraie à la fin d'une itération alors à la fin de l'itération suivante, elle est encore vraie car q est augmenté de 1 tandis que r est diminué de b :

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

A la fin de la dernière itération, la propriété est encore vraie : $a = bq + r$ mais la propriété $r \geqslant b$ cesse d'être vraie (puisqu'il s'agit de la dernière itération).

En sortie de boucle `while`, on a bien : $a = bq + r$ avec $r \in [\![0, b - 1]\!]$ ce qui caractérise bien la division euclidienne de a par b.

□

Exercice 14

Trouver un invariant de boucle et démontrer que la fonction réalise bien le travail indiqué en préambule de l'exercice 10 puis de l'exercice 8, puis du 11.

SOLUTION.

```
1 def somme(L):
2     """renvoie la somme des éléments de L"""
3     i=0
4     s=0
5     while i<len(L):
6         s+=L[i]
7         i+=1
8     return s
```

On note $n =$ `len(L)`.

Un invariant de boucle est : `s` $= \sum_{k=0}^{i}$ `L[k]`.

La propriété est vraie lors de l'initialisation car `s` $= 0$ et $\sum_{k=0}^{0}$ `L[k]` $= 0$.

Si la propriété est vraie à la fin d'une itération, on a `s` $= \sum_{k=0}^{i}$ `L[k]`.

Alors, à la fin de l'itération suivante, on a ajouté `L[i+1]` à la variable `s` et on a augmenté `i` d'une unité.

On a donc encore `s` $= \sum_{k=0}^{i}$ `L[k]`.

La sortie de la boucle `while` s'effectue lorsqu'on a `i=n`.

La variable `s` contient donc `s` $= \sum_{k=0}^{n-1} L[k]$.

```
1 def f(n):
2     """renvoie la plus petite puissance de 2
3     supérieure ou égale à n
4     avec n entier naturel non nul"""
5     p=1
6     while p<n:
7         p=2*p
8     return p
```

On note $k \in \mathbb{N}$ tel que $2^k \leqslant n < 2^{k+1}$.

Un invariant de boucle est : p est une puissance de 2.

Cette propriété est bien sûr vraie à l'issue de l'initialisation : $p = 1 = 2^0$.

Si cette propriété est vraie à la fin d'une initialisation, alors elle est vraie à la fin de la suivante car p a été multiplié par 2.

La sortie de la boucle a lieu lorsque la condition `p<n` n'est plus vérifiée.

Cela se produit si $p = n$ ou $p > n$.

— Dans le premier cas ($p = n$) et par conséquent p est une puissance de 2, égale à n.

— Dans le deuxième cas, $p > n$ et par conséquent p est une puissance de 2, strictement supérieure à n. A l'itération précédente, on avait $p < n$ donc à la sortie de la boucle p contient bien la plus petite puissance de 2 supérieure à n.

```
def palindrome(m):
    i=0
    j=len(m)-1
    test=True
    while i<=j and test:
        if m[i]==m[j]:
            i+=1
            j-=1
        else:
            test=False
    return test
```

Un invariant de boucle est :

`m[0:i+1]=m[j:len(m)]`.

□

Exercice 15

1. Écrire une fonction de recherche dichotomique d'un élément dans une liste triée.
2. Prouver la terminaison de cette fonction.
3. Prouver la correction de cette fonction.

SOLUTION.

```
def dichotomie(L,x):
    g=0
    d=len(L)-1
    while d-g>=0:
        m=(g+d)//2
        if x==L[m]:
            return True
        elif x<L[m]:
            d=m-1
        else:
            g=m+1
    return False
```

La variable `d-g` est un variant de boucle. Sa valeur est diminuée d'une unité à chaque itération de la boucle `while`. La boucle `while` se termine.

La propriété : «l'élément `x`, s'il appartient à `L`, se trouve entre `L[g]` et `L[d]`» est un invariant de boucle.

En effet, lors de l'initialisation l'élément `x` se trouve nécessairement entre

`L[0]=L[g]` et `L[len(L)-1]=L[d]`.

Si la propriété est vraie à la fin d'une itération alors les tests conditionnels montrent que `x` se situe encore entre `L[g]` et `L[d]` à la fin de l'itération suivante.

En effet, si `x<L[m]` alors nécessairement `x` se situe entre `L[g]` et `L[m-1]`. Tandis que si `x>L[m]` alors nécessairement `x` se situe entre `L[m+1]` et `L[d]`.

La boucle `while` se termine alors de deux manières :

— **1ère possibilité :** Lors d'une itération, on a `L[m]==x` auquel cas la commande `return True` termine la boucle `while` et l'élément `x` appartient effectivement à la liste.

— **2ème possibilité :** Le test conditionnel `d-g>=0` cesse d'être vrai : cela signifie que `d<g`.
Lors de la dernière itération de la boucle `while`, il y avait encore une alternative :
  — Soit `d=m-1` dans le cas `x<L[m]`.
  L'inégalité `d<g` devient alors `m-1` $<$ `g` c'est-à-dire `m` $<$ `g+1`. On a donc `m=g`.
  Mais `x`, s'il appartient à `L`, se situe entre entre `L[g]=L[m]` et `L[d]` d'après l'invariant de boucle. Puisque dans ce cas `x<L[m]`, c'est que l'élément `x` n'appartient pas à `L`.
  — Soit `g=m+1` et on conclut encore dans ce cas `m=d` et que `x` n'appartient pas à `L`.

□