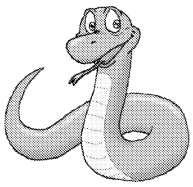
IPT : Cours 9



Preuve d'un algorithme itératif

MPSI-Schwarz : Prytanée National Militaire

Pascal Delahaye

12 janvier 2017

Il s'agit ici de présenter une méthode permettant de s'assurer :

1. qu'une boucle se termine (la terminaison)
2. qu'une boucle renvoie bien en sortie le résultat attendu (la validité)

On appelle "preuve de programme" la vérification de la terminaison et de la validité.

1 La démarche de preuve

Le principe repose sur un raisonnement de type "récurrence".

Méthodologie

Dans la boucle étudiée :

1. On commence par identifier les variables modifiées par l'itération : A, B ...
2. Puis, on énonce la propriété à démontrer sous la forme :
   "Après la ième itération, les variables A, B ... prennent les valeurs ai, bi ...
3. Enfin, on démontre par récurrence la propriété énoncée précédemment.

Ce travail permet :

1. Dans le cas d'une boucle "while" :
   (a) De vérifier que la boucle se termine
   (b) De déterminer les valeurs des différentes variables en "sortie de boucle", c'est à dire après la dernière itération.

2. Dans le cas d'une boucle "for" :
   Le nombre d'itérations étant connu, de déterminer les valeurs des différentes variables en "sortie de boucle".

On est alors en mesure de vérifier que l'objectif de notre boucle est bien rempli !

Remarque 1. Plus généralement, on peut étudier la validité d'un programme complet.
En effet, en effectuant ce travail pour l'ensemble des boucles du programme, on est en mesure de vérifier si notre programme remplit bien la fonction qu'on souhaite lui donner.

`Exemple 1.` **Cas des boucles "for"**
Prouver que le programme suivant renvoie bien la valeur a^n pour $a \in \mathbb{R}$ et $n \in \mathbb{N}^*$.

Python
```
def calc(a,n):
    resultat = 1
    for i in range(1,n+1):
        resultat = resultat*a
    print(resultat)
```

`Exemple 2.` **Cas des boucles "for"**
Que renvoie l'algorithme de Hörner, implémenté dans le programme suivant ?

Python
```
def calc(a,P):
                n = len(P)-1
    res = P[n]
    for k in range(1,n+1):
        res = res*a + P[n-k]
    print(res)
```

`Exemple 3.` **Cas des boucles "while"**
On considère le programme suivant :

Python
```
def f(x,y):                 # y étant un entier strictement positif
    r,s =0,y
    while s > 0:
          r = r+x
          s = s-1
    s = y-1
    t = r
    while s > 0:
          r = r+t
          s = s-1
    print(r)
```

1. Prouver la terminaison de ce programme.
2. Conjecturer le résultat renvoyé par le programme suivant et prouver-le.

Remarque 2. Un peu de modestie :
Malgré les méthodologies présentées ci-dessous, certains programmes ne peuvent être "prouver"...
C'est le cas par exemple de la programmation de l'algorithme de Syracus qui semble effectivement toujours se terminer mais pour lequel il n'existe actuellement aucune preuve de terminaison.
Plus généralement, on montre par l'absurde qu'il n'existe pas de méthode pour prouver la terminaison de tous les algorithmes existants.

2 Les invariants de boucle

Lorsque le contenu des variables de la boucle est difficile à expliciter, on pourra utiliser un *invariant de boucle*.
Il s'agit d'une propriété qui se substitue à la propriété de récurrence usuelle et qui permet cependant de prouver la validité du programme. Elle s'exprime souvent par une relation qui persiste entre les variables, mais pas uniquement.

Attention : un invariant de boucle ne permet pas de vérifier la terminaison d'un programme. Celle-ci doit donc être traitée de façon indépendante.

`Exemple 4.` **Algorithme d'Euclide**
Soient $a,\ b \in \mathbb{N}^*$.
Prouver que le programme suivant renvoie bien le PGCD et le PPCM de a et b.

Python
```
def euclide(a,b):
    R1=a
    R2=b
    while R2 != 0:
          R1,R2 = R2, R1%R2
    PPCM = a*b/R1
    print("Le PGCD est ", R1 , " et le PPCM est ",PPCM)
```

On pourra utiliser l'invariant de boucle suivant : "$R1 \wedge R2 = a \wedge b$*"*

`Exemple 5.` **Algorithme d'exponentiation rapide**
Soient $a, \in \mathbb{R}$ et $n \in \mathbb{N}^*$.
En utilisant l'invariant de boucle "res * $b^m = a^n$", déterminer l'objectif de la fonction `mystere` ci-dessous.

Python
```
def mystere(a,n):
    b, m = a, n
    res = 1
    while m > 0:
          if m % 2 == 1 : res = res*b
          b = b*b
          m = m // 2
    return res
```

Pouvez-vous déterminer la complexité de cet algorithme ?
Qu'en pensez-vous ?

`Exemple 6.` **Algorithme de Bezout**

Démontrer à l'aide d'un invariant de boucle bien choisi que le programme suivant renvoie bien un couple de Bezout.

Python
```
def bezout(a,b):
    R0 = a
    R1 = b
    U0 = 1
    U1 = 0
    V0 = 0
    V1 = 1
    while R1 != 0 :
        Q = R0//R1
        U0, U1 = U1, U0 - Q*U1
        V0, V1 = V1, V0 - Q*V1
        R0, R1 = R1, R0%R1
    return U0,V0
```

3 Exercices

Exercice : 1

Prouver que le programme suivant renvoie bien la valeur $n!$ où $n \in \mathbb{N}^*$.

Python
```
def fact(n):                    # n est un entier naturel non nul
    res = 1
    for i in range(1,n+1):
        res = res*i
    print(res)
```

Exercice : 2

Conjecturer le résultat renvoyé par le programme suivant.
Prouver-le à l'aide de l'invariant de boucle suivant : "$x + 2y = 3n$"

Python
```
def f(n):                     # n étant un entier positif
    x, y = n, n
    while y != 0:
          x = x+2
          y = y-1
    print(x)
```

Exercice : 3

Soient $a,\ b \in \mathbb{N}^*$.
Prouver en utilisant l'invariant de boucle `"a = bQ + R"` que le programme suivant renvoie bien la valeur le quotient et le reste de la division euclidienne de a par b.

Python
```
def DE(a,b):
    Q = 0
    R = a
    while R >= b:
          Q = Q+1
          R = R - b
    print("le quotient de la division euclidienne de ", a, " par ",
           b, " est ", Q, " et le reste est ",R)
```