Preuve de programme ou d'algorithme

Nous avons vu dans un cours précédent la multiplication égyptienne de deux entiers naturels, dont voici l'algorithme :

Algorithme 1 : Multiplication de deux entiers naturels

```
   début
   |  Données : Deux entiers naturels n et m
   |  Résultat : Le produit p de n par m
1  |  p ← 0
2  |  tant que  m > 0 faire
3  |  |  si m est pair alors
4  |  |  |  n ← n*2
5  |  |  |  m ← m//2
6  |  |  sinon
7  |  |  |  p ← p + n
8  |  |  |  m ← m - 1
9  |  |  fin
10 |  fin
   fin
```

Même si on a testé plusieurs fois l'algorithme et que les tests ont tous été positifs comment peut on être sur que :

Pour tous les entiers naturels n et m (données de cet algorithme), à la fin de l'algorithme la valeur de la variable p est bien le produit de n par m comme attendu ?

Cette question entraîne de fait deux questions :

1. La boucle `Tant que` s'arrête-t-elle **pour tous les entiers naturels n et m** (**problème de l'arrêt de la boucle**) ?
2. L'algorithme fait il ce qu'il est censé faire ?

1 Arrêt de la boucle

Focalisons nous sur la variable `m` :
A chaque tour de boucle la valeur de `m` diminue strictement car :

1. si la valeur de `m` est pair alors sa valeur est divisée par 2
2. si la valeur de `m` est impair alors sa valeur est diminuée de 1

Or d'après cette propriété mathématique "évidente" :

Toute suite d'entiers naturels strictement décroissante contient un nombre fini de termes

Donc, la boucle `tant que` de cet algorithme sera exécuté un nombre fini de fois **quelque soit la valeur de m en entrée**
Une variable comme `m` est appelée un variant de boucle

Définition 1. *Une variable* `v` *est un variant de boucle si :*

1. *Les valeurs de* `v` *sont des entiers naturels*
2. *Les valeurs de* `v` *sont strictement décroissantes*

Autrement dit l'existence d'un variant de boucle est un préalable à la terminaison de la boucle.

(encore faut il bien écrire le booléen qui contrôle l'exécution de la boucle)

2 Invariant de boucle

Pour bien écrire une boucle `tant que` il faut dans un premier temps trouver un variant de boucle et bien écrire le booléen qui contrôle l'exécution de la boucle, mais aussi trouver un **invariant de boucle**

Définition 2. *(voir démonstration par récurrence en mathématiques)*
Un invariant de boucle est une propriété (booléen) P telle que :

1. *P est vraie avant la boucle*
2. ***Si*** *P est vraie au début d'un tour de boucle quelconque* ***Alors*** *P est vraie à la fin de ce tour de boucle*

Remarque Si une des deux conditions est fausse ce n'est pas un invariant de boucle

Pour l'algorithme de la multiplication égyptienne la propriété P qui est un invariant de boucle est

`m*n + p =` $m_0 * n_0$

Que signifie cette expression ?

`m,n et p` sont des variables et les valeurs des variables évoluent au fur et à mesure du calcul.

On notera, comme dans le langage des suites, m_0 et n_0 les valeurs initiales de `m` et de `n`.

Pour faire **l'association** entre le nom de la variable, par exemple `m` et une de ses valeurs `a` à un moment donné, on va utiliser la notation des dictionnaires Python

{ `m` : `a`}

Montrons que `m*n + p =` $m_0 * n_0$ est un invariant de boucle pour l'algorithme de la multiplication égyptienne.

1. La propriété est vraie à l'état initial :
   Au début de l'algorithme { `m` : m_0, `n` : n_0, `p` :0} donc `m*n + p =` $m_0 * n_0$
2. Supposons que la propriété est vraie en début d'un tour de boucle quelconque avec { `m` : a, `n` : b, `p` : c} avec m*n + p = $m_0 * n_0$
   (a) Si `m` est pair alors <u>à la fin de la boucle</u> on a { `m` : a//2, `n` : b*2, `p` : c} et donc `m*n + p = a//2*b*2 + c = a*b + c =` $m_0 * n_0$
   (b) Si `m` est impair alors <u>à la fin de la boucle</u> on a { `m` : a - 1, `n` : b, `p` : c + b} et donc `m*n + p = (a - 1)b + c + b = ab + c =` $m_0 * n_0$

On a donc prouvé que `m*n + p =` $m_0 * n_0$ est un invariant de boucle pour l'algorithme de la multiplication égyptienne.

3 Preuve d'un algorithme

On peut maintenant **prouver** l'algorithme de la multiplication égyptienne.

On sait que la boucle s'arrête quelque soit les entrées `m` et `n` entières (naturelles), avec à la fin de l'algorithme { `m` : 0,`n` : b, `p` : c}

On sait que `m*n + p =` $m_0 * n_0$ est un invariant de boucle donc

Au début de l'algorithme `m*n + p =` $m_0 * n_0$

A la fin de l'algorithme `m*n + p = 0`$\times$ `b + c = c =` $m_0 * n_0$

Autrement dit **à la fin de l'algorithme quelque soit les valeurs entières en entrée, la valeur de la variable p est le produit des valeurs en entrée**

4 Assertions

Une **assertion** est un booléen (propriété) que l'on peut insérer dans un algorithme ou dans un programme, de telle sorte que si l'assertion est fausse cela bloque l'exécution de l'algorithme ou du programme **après l'assertion**

En Python on écrit les assertions avec le mot réservé `assert` suivi du booléen (ce n'est pas une fonction)

Cela peut servir à filtrer les entrées d'un programme et ainsi l'utilisateur est obligé de respecter les **préconditions de l'algorithme** (les conditions sur les entrées), on parle alors de programmation défensive.

Mais cela peut aussi servir à **insérer un invariant de boucle** dans la boucle.

Autrement dit si l'invariant de boucle est faux cela va bloquer l'exécution de la boucle à un moment donné sur un jeu de données.

```
def mult_egypt(m:int,n:int)->int:
"""
renvoie le produit de m par n
"""
    #filtrer les entrées
    assert m >= 0 and n >= 0

    p = 0
    #on mémorise les valeurs initiales pour l'invariant de boucle
    m_0 = m
    n_0 = n
    while m > 0:
        if (m >> 1) << 1 == m:
            n <<= 1
            m >>= 1
        else:
            p += n
            m -= 1

        #invariant de boucle
        assert m*n + p == m_0*n_0
    return p
```

5 Exercices

Ex 1

Voici à priori ci-dessous un algorithme de la division euclidienne de deux entiers naturels `a` et `b` non nuls, par soustractions successives.

Algorithme 2 : Division euclidienne de deux entiers naturels

```
début
    Données : Deux entiers naturels a >= 0 et b > 0
    Résultat : Le quotient q de a par b et le reste r tel que 0 <= r < b
1   q ← 0
2   r ← a
3   tant que  r ..... faire
4       q ← q + 1
5       r ← r - b
6   fin
fin
```

1. Quelles sont les préconditions de l'algorithme ?
2. Montrer que la variable `r` est un variant de boucle.
3. Compléter le booléen `r ......` pour que la boucle se termine dans tous les cas.

Ex 2

Voici à priori ci-dessous un algorithme qui calcule la factorielle d'un entier naturel non nul

Algorithme 3 : Factorielle d'un entier naturel

```
début
    Données : Un entier naturel n > 0
    Résultat : n ! = n × n - 1 × ... × 1
1   f ← n
2   tant que  n ..... faire
3       f ← f *n
4       n ← n - 1
5   fin
fin
```

1. Quelles sont les préconditions de l'algorithme ?
2. Montrer que la variable `n` est un variant de boucle
3. Compléter le booléen `n ......` pour que la boucle se termine dans tous les cas

Ex 3

On peut calculer la factorielle d'un entier autrement par exemple en faisant le produit

Algorithme 4 : Factorielle d'un entier naturel

début
  Données : Un entier naturel n > 0
  Résultat : n ! = n $\times$ n - 1 $\times$ ... $\times$ 1
1 f $\leftarrow$ 1
2 m $\leftarrow$ 1
3 **tant que** *m .....* **faire**
4 f $\leftarrow$ f *m
5 m $\leftarrow$ m + 1
6 **fin**
fin

On va étendre la définition du variant de boucle en posant

Définition 3. *Une* ***quantité*** `q` *est un variant de boucle si :*

1. *Les valeurs de* `q` *sont des entiers naturels*
2. *Les valeurs de* `q` *sont strictement décroissantes*

1. A partir de cette nouvelle définition montrer que la quantité `n - m` est un variant de boucle
2. Compléter le booléen `m ......` pour que la boucle se termine dans tous les cas

Ex 4

Voici un programme qui à priori calcule le nombre de chiffres d'un nombre entier positif entré au clavier par l'utilisateur.

```
n = int(input("Entrez un entier naturel"))
nb_chiffres = 0
while n > 0:
    n //= 10
    nb_chiffres += 1
print(nb_chiffres)
```

1. Trouver un variant de boucle pour ce programme.
2. Justifier que la boucle s'arrête **pour tout entier naturel entré au clavier**.
3. Que se passe-t-il si on remplace `n > 0` par `n != 0` ?

Ex 5

Voici à priori ci-dessous un algorithme de la division euclidienne de deux entiers naturels `a` et `b` non nuls, par soustractions successives.

Algorithme 5 : Division euclidienne de deux entiers naturels non nuls

```
début
    Données : Deux entiers naturels a et b non nuls
    Résultat : Le quotient q de a par b et le reste r tel que 0 <= r < b
1   q ← 0
2   r ← a
3   tant que  r > b faire
4       r ←r - b
5       q ←q + 1
6   fin
fin
```

1. Faire un tableau d'évolutions de variable pour a = 15 et b = 4 et vérifier que la propriété $a = bq + r$ reste vraie à chaque tour de boucle.
2. Montrer dans le cas général que $a = bq + r$ est un invariant de boucle.
3. Vérifier sur le contre-exemple a = 14 et b = 7 que cet algorithme n'est pas correct dans tous les cas
4. Essayer de prouver cet algorithme, et trouver la correction à effectuer.

Ex 6

Il s'agit de compléter la fonction Python ci-dessous afin qu'elle soit prouvée dans tous les cas.

```
def fact(n:int)->int:
"""
renvoie factorielle de n où n est un entier naturel
>>> fact(0)
1
>>> fact(5)
120
"""
assert ..........
n_0 = n
f = ....
while .....:
    .....
    .....
return f
```

1. Compléter le premier assert pour que les préconditions de la fonction soient vérifiées
2. **On veut construire la boucle à partir de l'invariant de boucle**
   "La valeur de la variable `f` est le produit des nombres n_0 jusqu'à `a` inclus où `a` est la valeur de la variable `n`"
   (a) Compléter la ligne pour que l'invariant de boucle soit vrai à l'état initial

(b) Si la propriété "La valeur de la variable `f` est le produit des nombres n_0 jusqu'à `a` inclus où `a` est la valeur de la variable `n`" au début d'un tour de boucle quelconque quelles instructions mettre dans la boucle pour que ce soit encore vrai à la fin de la boucle ?

(c) Ecrire correctement le booléen qui contrôle l'exécution de la boucle

3. Une fois complétée la fonction :

(a) Justifier que `n` est un variant de boucle et que la boucle s'arrête dans tous les cas.

(b) Justifier que la propriété "La valeur de la variable `f` est le produit des nombres n_0 jusqu'à `a` inclus où `a` est la valeur de la variable `n`" est un invariant de boucle

(c) Faire la preuve du programme.

Ex 7

Par définition le calcul de a^n consiste à multiplier a avec lui-même $n-1$ fois

1. Ecrire une fonction `expo(a,n)` qui implémente cet algorithme en Python
2. On peut diminuer le nombre de multiplications car cela a un coût, en utilisant **l'algorithme d'exponentiation rapide**

Algorithme 6 : Exponentiation rapide a^n

```
début
    Données : Deux entiers naturels a et n
    Résultat : a^n
1   p ← 1
2   tant que n > 0 faire
3       si n est pair alors
4           n ← n//2
5           a ← a*a
6       sinon
7           p ← p * a
8           n ← n - 1
9       fin
10  fin
fin
```

1. Calculer 2^8 avec cet algorithme en faisant le tableau d'évolution des variables. Combien de multiplications ?
2. Justifier que `n` est un variant de boucle et que la boucle s'arrête quelque soit a et n respectant les préconditions.
3. Vérifier que $p * a * *n = a_0 * *n_0$ est un invariant de boucle.
4. En déduire que dans tous les cas à la fin de l'algorithme la variable `p` a pour valeur $a_0^{n_0}$