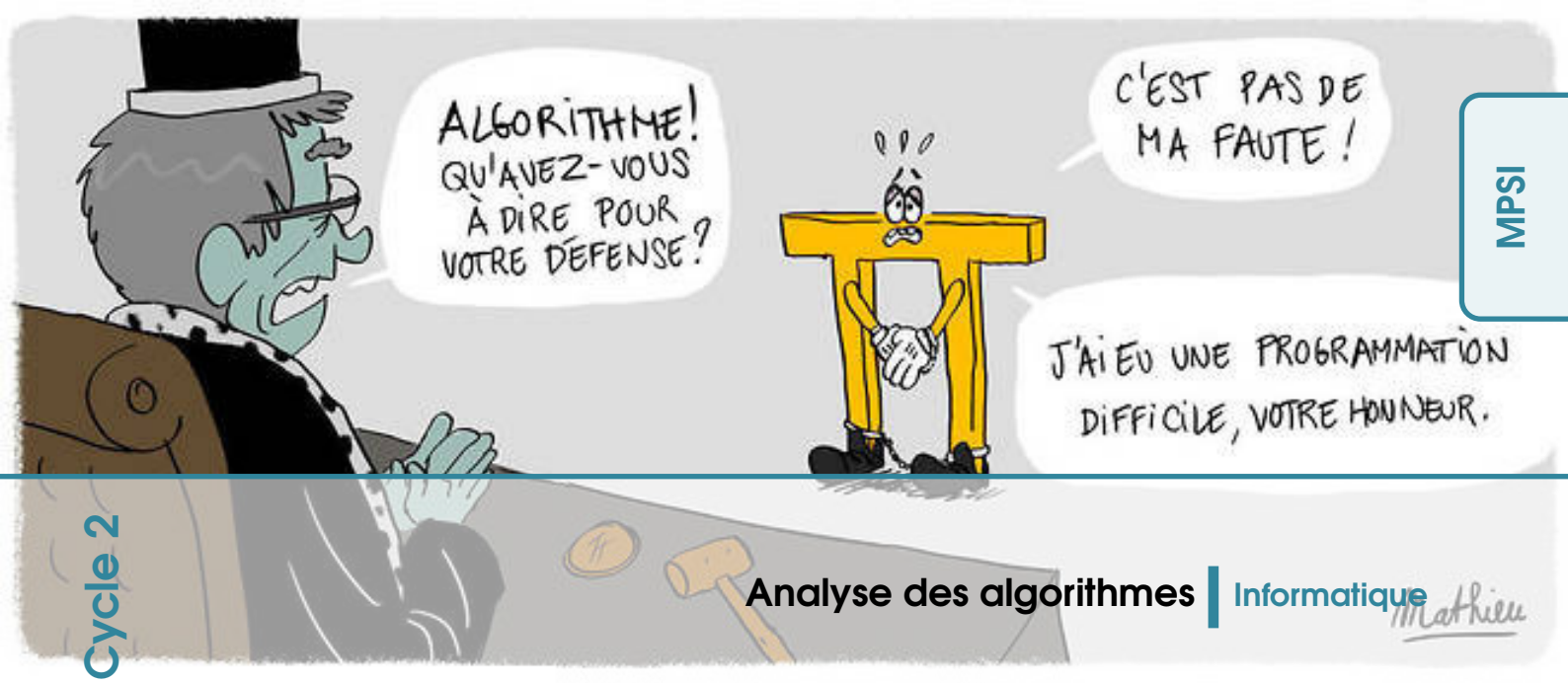

Dessin de Mathieu Vinciguerra

Cours

Chapitre 2 – 01
Invariance et variance

6 Janvier 2021

Savoirs et compétences :

- *AN.C1 : Justifier qu'une itération (ou boucle) produit l'effet attendu au moyen d'un invariant*
- *AN.C2 : Démontrer qu'une boucle se termine effectivement*

E. Durif – X. Pessoles
M. Rebout
MPSI

1 Prouver qu'un algorithme réalise donne le résultat souhaité

1.1 Invariant de boucle

Définition Un **invariant de boucle** est une proposition vérifiée à chaque tour d'une boucle. Plus précisément, on distingue les invariants d'**entrée de boucle** et ceux de **sortie de boucle**. C'est en fait une proposition de récurrence, qui doit être vrai au début de la première boucle, puis se propager de tour de boucle en tour de boucle, jusqu'au dernier.
Les invariants de boucles ont essentiellement deux intérêts :

1. **Démontrer** l'algorithme, grâce au principe de récurrence ;
2. Répondre à une question primordiale : **où commencer et où finir une boucle ?**

R Il est **fortement conseillé** de toujours indiquer les invariants de boucle en commentaire dans vos algorithmes.

1.2 Preuve de correction

Définition Une preuve de correction permet de démontrer le résultat d'un algorithme. Elle repose sur l'utilisation **d'un invariant** et d'un raisonnement par **récurrence** qui possède plusieurs étapes :

1. **Initialisation :** l'invariant de boucle est vérifié au début de la première boucle ;
2. **Supposition :** on suppose que l'invariant de boucle est vérifié au début de la $n^{ème}$ boucle ;
3. **Hérédité :** à partir de la supposition on montre que l'invariant de boucle est vérifié en fin de boucle.
4. **Conclusion :** en se basant sur l'hérédité on donne l'expression de l'invariant à la fin de la dernière boucle pour vérifier le résultat souhaité.

1.3 Exemple

▪ **Exemple** Par exemple, on souhaite calculer u_{20} où u est la suite définie par

$$\begin{aligned} u_0 &= 5 \\ \forall n \in \mathbb{N}\ u_{n+1} &= 2u_n - n - 3 \end{aligned}$$

```
x = 5            # x = u_0 (initialisation)
for k in range(20):
    # x = u_k (invariant en entrée de boucle)
    x = 2 * x - k - 3
    # x = u_{k+1} (invariant en sortie de boucle)
# après la boucle for, k vaut 19, et x vaut donc u_{20}.
```

▪

1.4 Tableau de valeurs

Définition Pour vérifier qu'un algorithme effectue bien ce qu'on lui demande, on peut utiliser un **tableau de valeurs** qui permet de déterminer en début ou fin de boucles les valeurs prises par les différentes variables.

▪ **Exemple**

```
def fonctionMystere(n) :
    if n==0 or n==1:
         return 1
    else :
        res = 1
    for i in range (2,n+1) :
        res = res * i
    return res
```

Q 1 : Dresser le tableau de valeur pour les variables `i` et `res`.
Q 2 : Quel est le nom mathématique usuel donné à la fonction `fonctionMystere` ? ▪

2 Prouver qu'un algorithme se termine

2.1 Variant de boucle

Comment démontrer un algorithme reposant sur une boucle indéfinie? Pour cela, nous utilisons encore les invariants de boucle, afin de prouver qu'après la boucle, le résultat est bien celui voulu.
Mais avant même cela, il y a un point épineux : la boucle **while** va-t-elle vraiment se finir? Il faut démontrer ce que l'on appelle la **terminaison** de l'algorithme. C'est là que réside le danger des boucles **while** : si elles sont mal écrites, la condition de la boucle ne devient jamais fausse, et la boucle est infinie.

Définition Pour montrer la terminaison d'un algorithme, on utilise cette fois un **variant** de boucle. Il consiste à mettre en avant une variable dont la valeur est modifiée au cours des tours de boucle, de telle sorte que cette modification finisse par rendre fausse la condition de la boucle.

2.2 Preuve de terminaison

Définition Une preuve de terminaison repose sur l'utilisation d'un variant de boucle qui est généralement associé à la condition donnée à la boucle `while`. La plupart des conditions associées au boucles indéfinies sont des inégalités. Il suffit alors généralement de montrer que le variant de boucle est défini par une suite d'entiers strictement croissante ou décroissante qui ne sera donc pas bornée.

2.3 Exemple

▪ **Exemple** Rechercher le premier entier n tel que la somme des entiers de 1 à n dépasse 11.

```
n = 1
s = 1                          # s = somme des i de 1 àn
while s < 11 :          # invariant : s = somme des i de 1 àn
    n = n + 1
    s = s + n           # variant : s, qui est entier et strictement croissant
```

▪

Cycle 01

TD

Sources :
exercice 3 : Recherche d'invariant exercice 4 : Recherche d'invariant exercice 5 : Recherche d'invariant

Savoirs et compétences :

- *AN.C1 : Justifier qu'une itération (ou boucle) produit l'effet attendu au moyen d'un invariant*
- *AN.C2 : Démontrer qu'une boucle se termine effectivement*

3 Applications

3.1 Test de primalité

On veut tester si un entier `n` est premier on donne l'algorithme suivant :

```
def est_premier(n):
    """ Renvoie True si n est premier, False sinon
        Préconditon : n est un entier."""
    for d in range(2,n):
        if n % d == 0:
            return False
    return True
```

Q 1 : Proposer un invariant de boucle pour démontrer cet algorithme.

3.2 Fonction mystère

On considère la fonction suivante.

```
def mystere(a,b) :
    """Précondition : a,b sont des entiers, a>0, b>1"""
    k,p = 0,1
    while a % p == 0 :
        k = k+1
        p = p*b
    return k-1
```

Q 1 : Dresser un tableau de valeurs décrivant les valeurs des variables `k` et `p` en entrée des trois premiers tours de la boucle `while` de la fonction `mystere(a,b)`.
On pourra au besoin faire intervenir les variables `a` et `b`.

Q 2 : En s'aidant de la question précédente, écrire un invariant de boucle pour la boucle `while` de la fonction `mystere(a,b)`. On justifiera la réponse.

Q 3 : Donner un variant de boucle pour la boucle `while` de la fonction `mystere(a,b)`. On justifiera la réponse.

Q 4 : Déduire des questions précédentes qu'un appel de la fonction `mystere(a,b)` renvoie un résultat et déterminer le résultat alors renvoyé. On justifiera la réponse.

3.3 Invariant pour les calculs de moyennes et de variances

Q 1 : Soit les algorithmes de calculs de moyenne ci-dessous, proposez des invariants de boucles.

```
def moyenne(t):
    """Calcule la moyenne de t
       Précondition : t est un tableau de
       nombres non vide"""
    s = 0
    for x in t:
        s = s + x
    return s/len(t)


def moyenne(t):
    """Calcule la moyenne de t
       Précondition : t est un tableau de
       nombres non vide"""
    n = len(t) # Longueur de t
    s = 0
    for i in range(n):
        s = s + t[i]
    return s/n
```

Q 2 : Soit l'algorithme de calculs de variance ci-dessous, proposez un invariants de boucles.

```
def variance(t):
    """Renvoie la variance de t
       Précondition : t est un tableau de
       nombres non vide"""
    sc = 0
    for x in t:
        sc = sc + x**2
    return sc/len(t) - moyenne(t)**2
```

3.4 Invariant pour les recherches de maximum de tableaux

Q 1 : Soit les algorithmes de recherche de maximum d'un tableau ci-dessous, proposez des invariants de boucles.

```
def maxi(t):
    """Renvoie le plus grand élément de t.
       Précondition : t est un tableau
       non vide"""
    m = t[0]
    for x in t:
        if x > m:
            m = x
    return m


def maxi(t):
    """Renvoie le plus grand élément de t.
       Précondition : t est un tableau
       non vide"""
    m = t[0]
    for i in range(1, len(t)):
        if t[i] > m:
            m = t[i]
    return m
```

Q 2 : Soit les algorithmes de recherche d'indice de maximum d'un tableau ci-dessous, proposez des invariants de boucles.

```
def indicemaxi(t):
    """Renvoie l'indice du plus grand
        élément de t.
       Précondition : t est un tableau
       non vide"""
    im = 0
    for i in range(1, len(t)):
        if t[i] > t[im]:
            im = i
    return im
```

```
def indicemaxi(t):
    """Renvoie l'indice du plus grand élément
       de t.
       Précondition : t est un tableau
       non vide"""
    im = 0
    for i, x in enumerate(t):
        if x > t[im]:
            im = i
    return im
```

3.5 Invariant pour les recherches d'occurrences dans un tableau

Q 1 : Soit les algorithmes de test d'appartenance d'un élément dans un tableau. Proposer un invariant de boucle.

```
def appartient(e, t):
    """Renvoie un booléen disant si e
    appartient àt
       Précondition : t est un tableau"""
    for x in t:
        if e == x:
            return True
    return False
```

Q 2 : Soit les algorithmes de recherche d'indice de première occurrence d'un élément dans un tableau. Proposer un invariant de boucle.

```
def ind_appartient(e,t):
    """Renvoie l'indice de la première
       occurrence de e dans t,
       None si e n'est pas dans t
       Précondition : t est un tableau"""
    for i in len(t):
        if t[i] == e:
            return i
    return None
```