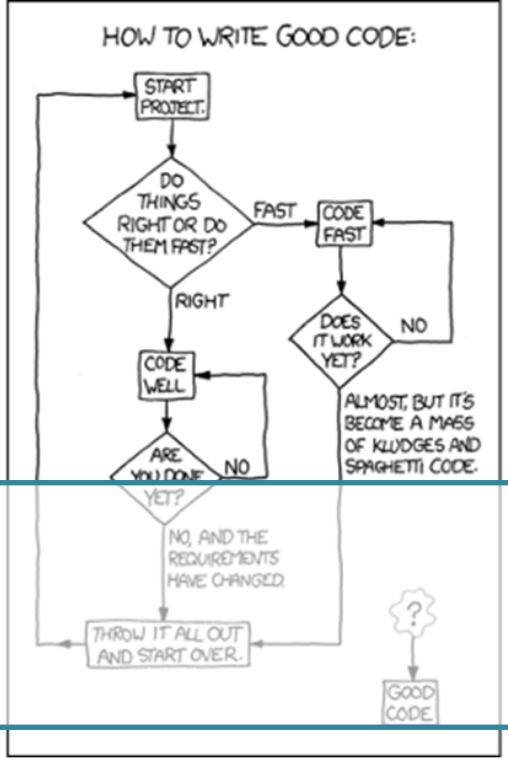

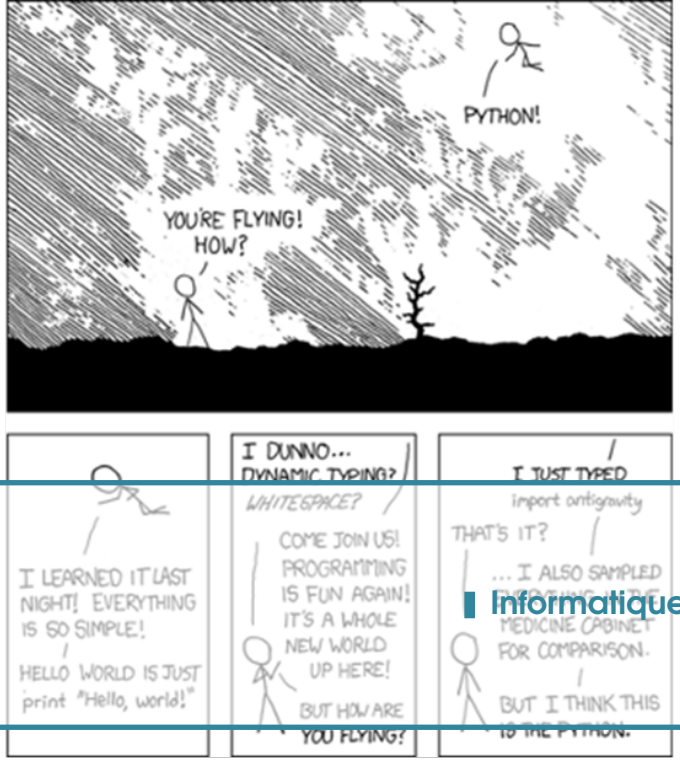

Fiche

Ch. 3
Terminaison et correction des algorithmes

1 Anlayse des algorithmes 2
1.1 Définition . . . 2
1.2 Un exemple 2
2 Terminaison d'un algorithme 3
2.1 Variant de boucle . . . 3
2.2 Un second exemple ressemblant... . . . 3
3 Correction d'un algorithme 4
3.1 Invariant de boucle . . . 4
3.2 Un « contre exemple » . . . 4

E. Durif – X. Pessoles
J.-P. Berne
MPSI

1 Anlayse des algorithmes

1.1 Définition

Définition Terminaison d'un algorithme
Prouver la terminaison d'un algorithme signifie montrer que cet algorithme se terminera en un temps fini. On utilise pour cela un **variant de boucle**.

Définition Correction d'un algorithme
Un algorithme est dit (partiellement) correct s'il est correct dès qu'il termine.

Prouver la correction d'un algorithme signifie montrer que cet algorithme fournit bien la solution au problème qu'il est sensé résoudre. On utilise pour cela un **invariant de boucle**.

Définition Invariant de boucle Un invariant de boucle est une propriété dépendant des variables de l'algorithme, qui est vérifiée à chaque passage dans la boucle.

1.2 Un exemple ...

Objectif L'objectif est ici de montrer la nécessité d'utiliser un invariant de boucle.
Pour cela, on propose la fonction suivante sensée déterminer le plus petit entier `n` strictement positif tel que $1+2+...+n$ dépasse strictement la valeur entière strictement positive `v`. Cette fonction renvoie-t-elle le bon résultat? Desfois? Toujours?

```
def foo(v:int) -> int:
    r = 0
    n = 0
    while r < v :
        n = n+1
        r = r+n
    return n
```

Montrer intuitivement que `foo(v)` *se termine pour* $v \in \mathbb{N}^*$.

L'algorithme se terminera si on sort de la boucle `while`. Il faut pour cela que la condition `r<v` devienne fausse (cette condition est vraie initialement). Pour cela, il faut que `r` devienne supérieure ou égale à `v` dont la valeur ne change jamais.

`n` étant incrémenté de 1 à chaque itération, la valeur de `r` augmente donc à chaque itération. Il y aura donc un rang `n` au-delà duquel `r` sera supérieur à `v`. L'algorithme se termine donc.

Que renvoie `foo(9)` *? Cela répond-il au besoin ?*

Début de la i^e itération	r	n	r < v
Itération 1	0	0	0 < 9 ⇒ True
Itération 2	1	1	1 < 9 ⇒ True
Itération 3	3	2	3 < 9 ⇒ True
Itération 4	6	3	6 < 9 ⇒ True
Itération 5	10	4	10< 9 ⇒ False

La fonction renvoie 4. On a $1+2+3+4=10$. On dépasse strictement la valeur 9. La fonction répond au besoin dans ce cas.

Que renvoie `foo(10)` *? Cela répond-il au besoin ?*

Début de la i^e itération	r	n	r < v
Itération 1	0	0	0 < 10 ⇒ True
Itération 2	1	1	1 < 10 ⇒ True
Itération 3	3	2	3 < 10 ⇒ True
Itération 4	6	3	6 < 10 ⇒ True
Itération 5	10	4	10 < 10 ⇒ False

La fonction renvoie 4. On a $1+2+3+4=10$. On ne dépasse pas strictement la valeur 10. La fonction ne répond pas au besoin dans ce cas.

Résultat La fonction proposée ne remplit pas le cahier des charges. Aurait-on pu le prouver formellement?

2 Terminaison d'un algorithme

2.1 Variant de boucle

Définition Variant de boucle
Un variant de boucle permet de prouver la terminaison d'une boucle conditionnelle. Un variant de boucle est une **quantité entière positive** à l'entrée de chaque itération de la boucle et qui **diminue strictement à chaque itération**.

Théorème Si une boucle admet un variant de boucle, elle termine.

Propriété Un algorithme qui n'utilise ni boucles inconditionnelles (boucle `for`) ni récursivité termine toujours. Ainsi, la question de la terminaison n'est à considérer que dans ces deux cas.

Reprenons l'exemple précédent.

```
def foo(v:int) -> int:
    r = 0
    n = 0
    while r < v :
        n = n+1
        r = r+n
    return n
```

Dans cet exemple montrons que la quantité $u_n = v - r$ est un variant de boucle :
- initialement, $r = 0$ et $v > 0$; donc $u_0 > 0$;
- à la fin de l'itération n, on suppose que $u_n = v - r > 0$ et que $u_n < u_{n-1}$;
- à l'itération $n+1$:
  - cas 1 : $r \geq v$. Dans ce cas, n et r n'évoluent pas l'hypothèse de récurrence reste vraie. On sort de la boucle `while`. L'algorithme termine,
  - cas 2 : $r < v$. Dans ce cas, à la fin de l'itération $n+1$, montrons que $u_{n+1} < u_n$: $u_{n+1} = v-(r+n+1) = u_n - n - 1$ soit $u_{n+1} = u_n - n - 1$ et donc $u_{n+1} < u_n$. L'hypothèse de récurrence est donc vraie au rang $n+1$.

Au final, $u_n = v - r$ est donc un variant de boucle et la boucle se termine.

2.2 Un second exemple ressemblant...

[https://marcdefalco.github.io/pdf/complet_python.pdf]

Considérons l'algorithme suivant qui, étant donné un entier naturel n strictement positif (inférieur à 2^{30}), détermine le plus petit entier k tel que $n \leq 2^k$.

```
def plus_grande_puissance2(n):
    k = 0
    p = 1
    while p < n:
        k = k+1
        p = p*2
    return k
```

Démonstration [1] Dans l'exemple précédent, la quantité $n - p$ est un variant de boucle :
- au départ, $n > 0$ et $p = 1$ donc $n - p \geq 0$;
- comme il s'agit d'une différence de deux entiers, c'est un entier. Et tant que la condition de boucle est vérifiée $p < n$ donc $n - p > 0$.
- lorsqu'on passe d'une itération à la suivante, la quantité passe de $n - p$ à $n - 2p$ or $2p - p > 0$ car $p \geq 1$. Il y a bien une stricte diminution.

Démonstration [2] Montrons que, la quantité $u_j = n - p$ est un variant de boucle :
- intialement, $n > 0$ et $p = 1$ donc $n - p \geq 0$;
- à la fin de l'itération j, on suppose que $u_j = n - p > 0$ et $u_j < u_{j-1}$;
- à la fin de l'itération suivante, $u_{j+1} = n - 2p = u_j - p$. p est positif donc u_{j+1} est un entier et $u_{j+1} < u_j$. Par suite, ou bien $u_{j+1} < 0$ c'est à dire que $n - p < 0$ soit $p > n$. On sort donc de la boucle. Ou bien, $u_{j+1} > 0$, et la boucle continue.

$n - p$ est donc un variant de boucle.

3 Correction d'un algorithme

3.1 Invariant de boucle

Méthode

Pour montrer qu'une propriété est un invariant de boucle dans une boucle `while` :

- le propriété doit être vérifiée avant d'entrer dans la boucle ;
- la propriété doit être vraie en entrée de boucle ;
- la propriété doit être vraie en fin de boucle.

Reprenons un des exemples précédents. Reconsidérons l'algorithme suivant qui, étant donné un entier naturel n strictement positif (inférieur à 2^{30}), détermine le plus petit entier k tel que $n \leq 2^k$.

```
def plus_grande_puissance2(n):
    k = 0
    p = 1
    while p < n:
        k = k+1
        p = p*2
    return k
```

Démonstration Montrons que la propriété suivante est un invariant de boucle : $p = 2^k$ et $2^{k-1} < n$.

- **Initialisation :** à l'entrée dans la boucle $k = 0$ et $p = 1$, $n \in \mathbb{N}^*$
  - d'une part on a bien $1 = 2^0$;
  - d'autre part $2^{-1} < n$.
- On considère que la propriété est vraie au n^e tour de boucle c'est à dire $p = 2^k$ et $2^{k-1} < n$.
- Au tour de boucle suivant :
  - **ou bien** $p >= n$. Dans ce cas, on sort de la boucle et on a toujours $p = 2^k$ et $2^{k-1} < n$ (propriété d'invariance). La propriété est donc vraie au tour $n+1$.
  - **ou bien** $p < n$. Dans ce cas, il faut montrer que $p = 2^{k+1}$ et $2^k < n$. Etant entrés dans la boucle, $p < n \Rightarrow 2^k < n$. De plus, en fin de boucle, $p \rightarrow p*2$ et $k \rightarrow k+1$. On a donc $p \leftarrow 2^k * 2 = 2^{k+1}$.

La propriété citée est donc un invariant de boucle.

3.2 Un « contre exemple »

Reprenons le tout premier exemple où on cherche le plus petit entier `n` strictement positif tel que $1+2+...+n$ dépasse strictement la valeur entière strictement positive `v` .

```
def foo(v:int) -> int:
    assert v>0
    r = 0
    n = 0
    while r <= v :
        n = n+1
        r = r+n
    return n
```

La propriété suivante est-elle un invariant de boucle : $r = \sum_{i=0}^{n} i$ et $\sum_{i=0}^{n-1} i \leq v$, $n \in \mathbb{N}^*$?

La réponse est directement NON, car la phase d'initialisation n'est pas vérifiée car $n = 0$ et $n \notin \mathbb{N}^*$. Cela signifie donc que l'algorithme proposé ne répond pas au cahier des charges.

Modifions alors l'algorithme ainsi.

```
def foo2(v:int) -> int:
    assert v>0
    r = 1
    n = 1
    while r <= v :
        n = n+1
        r = r+n
    return n
```

Montrons que la propriété suivante est un invariant de boucle : $r = \sum_{i=0}^{n} i$ et $\sum_{i=0}^{n-1} i \leq v$, $n \in \mathbb{N}^*$.

- **Initialisation :** à l'entrée dans la boucle $r = 1$ et $n = 1$, $n \in \mathbb{N}^*$
  - d'une part on a bien $r = \sum_{i=0}^{1} i = 1$;

  - d'autre part $\sum_{i=0}^{0} i = 0 < v$ et $v > 0$ (spécification de la fonction).

- On considère que la propriété est vraie au début du n^e tour de boucle c'est-à-dire $r = \sum_{i=0}^{n} i$ et $\sum_{i=0}^{n-1} i \leq v$.
- Á la fin du n^e tour de boucle, $n_{n+1} = n_n + 1$ et $r_{n+1} = r_n + n_{n+1} = r_n + n_n + 1 = \sum_{i=0}^{n}(i) + n_n + 1 = \sum_{i=0}^{n+1} i$ (car $n_n = n$).

  On a alors,
  - ou bien $r_{n+1} > v$ et on sort de la boucle ; on peut renvoyer n.
  - ou bien $r_{n+1} \leq v$ et donc $\sum_{i=0}^{n} i \leq v$.

La propriété citée est donc un invariant de boucle.