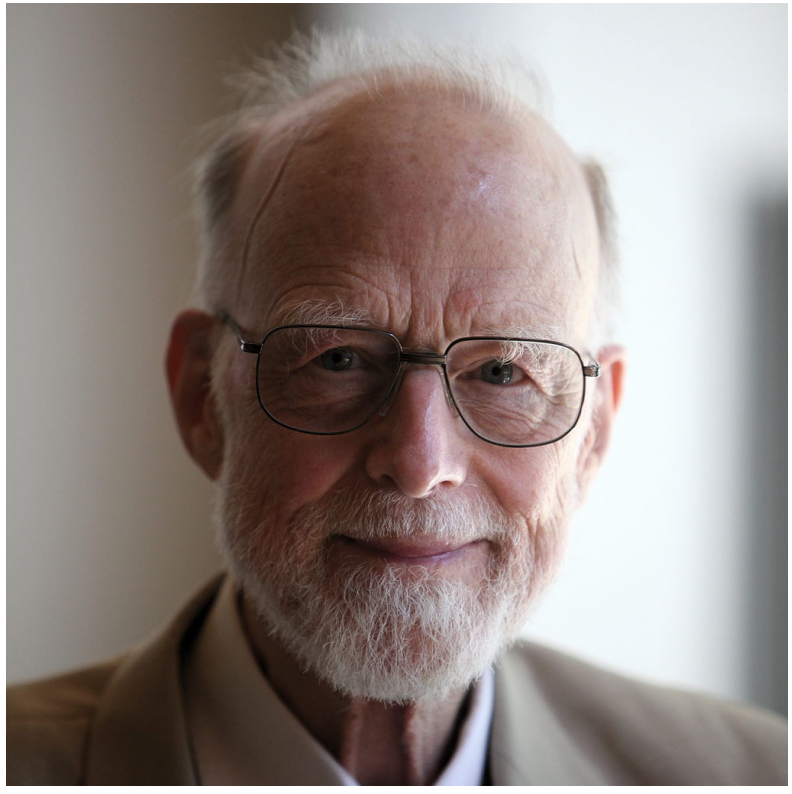
Preuve de programmes



Skander Zannad et Judicaël Courant

Lycée La Martinière-Monplaisir

2013-10-12

1 Quelques bugs célèbres

Parmi beaucoup d'autres :

1980 (NORAD) Alerte à l'attaque nucléaire.

1983 (URSS) Alerte à l'attaque nucléaire.

1978-1985 (NASA) Non détection du trou dans la couche d'ozone.

1985-1987 (Therac-25) Overdose massive de radiations (6 accidents).

1990 (AT&T, USA) Plantage en cascade du réseau téléphonique.

1994 (RAF) Crash d'un hélicoptère Chinook (29 morts).

1994 (Pentium, Intel) Bug dans la division (5×10^8€).

1996 (ESA) Vol inaugural d'Ariane 5 : boum ! (3×10^8€).

1997 (USS Yorktown) Épave flottante (remorquage au port).

1999 (NASA) Mars climate orbiter s'écrase sur Mars (3×10^8\$)

2000 (Y2K bug) 3×10^{11}\$ dépensés dans le monde pour l'éviter ; pertes causées : 2×10^{10}\$.

2003 (Northeast blackout) Coupure électrique (2 jours, 6×10^7 personnes)

2010 (Prius, Toyota) Rappel de $400\,000$ véhicules pour un problème d'ABS (pertes 3×10^9\$, procès inclus).

2012 (Knight Capital) Pertes boursières (45 minutes, 10^7\$/min).

2 Éliminer les bugs

Peut se régler à plusieurs niveaux :

1. Humain : gestion de projets, formation des programmeurs et des utilisateurs, etc.
2. Langage : choix d'un langage de programmation adapté au problème. Conceptions de langages sûrs.
3. Tests : test unitaires (*unit testing*) et tests d'intégration.
4. Preuve de programmes.

2.1 Les tests

- Sont indispensables.
- Peuvent (et devraient) être automatisés (reproductibilité). Moto-chercher « tests unitaires python ».

Limite du test :

> « Tester un programme peut montrer que des bogues sont présents, mais jamais montrer leur absence. »
>
> Edsger W. Dijkstra

2.2 Preuve de programmes

- Nécessite de comprendre comment raisonner sur les programmes.
- Peut être assistée par ordinateur (partiellement automatisable). Domaine de recherche très actif.
- Exigeant mais très efficace pour écrire des programmes corrects (au niveau prépa ou industriel — cf ligne 14 métro parisien).

3 Logique de Hoare

3.1 Introduction

- Ensemble de règles logiques permettant de raisonner sur un programme.
- Proposée par Tony Hoare (1969).
- Décrit l'évolution d'un *état* (valeurs des variables) produite par l'exécution d'un programme.
- Permet de raisonner sur un programme consistant en une liste d'instruction simples (affectation) ou complexes (conditionnelles, boucles).

3.2 Notion de précondition/postcondition

On manipule des assertions (proposition mathématiques) à propos de l'état du programme.

Exemple d'assertions :

- « La valeur de `x` est un multiple de 3 »
- « `x % 3 == 0` »
- « Toutes les valeurs du tableau `t` sont des entiers non nuls »
- « `x >= y` »

On dit qu'un programme C est *totalement correct* (resp. *partiellement correct*) vis-à-vis d'une *précondition* P et d'une *postcondition* Q, si à partir de tout état vérifiant l'assertion P, l'exécution de C termine et conduit à un état vérifiant l'assertion Q (resp. si elle termine, conduit à un état vérifiant l'assertion Q).

Le triplet (P, C, Q), noté $\{\ P\ \}\, C\, \{\ Q\ \}$ est appelé un *triplet de Hoare*. On dit qu'il est *valide* si C est partiellement correct vis-à-vis de la précondition P et de la postcondition Q.

4 Mise en œuvre sur un programme simple

On considère le programme suivant :

```
x = a
y = b
y = y - x
x = x + y
y = x - y
```

On veut montrer que le programme est correct vis-à-vis de la précondition « `a` et `b` sont des entiers » et de la postcondition « `x` vaut la même chose que `a` et `y` la même chose que `b` ».

Pour cela, il suffit de montrer qu'on peut le décorer comme par des assertions comme ceci :

```
1  # précondition : a et b sont entiers
2  x = a
3  y = b
4  # ici on a a et b entiers et x == a et y = b
5  y = y - x
6  # ici on a a, b entiers, x == a et y = b - a
7  x = x + y
8  # ici on a a, b entiers, x == b et y == b - a
9  y = x - y
10 # postcondition: x == b et y == a
```

De façon à respecter certaines conditions :

1. La première (resp. dernière) assertion est la précondition (resp. postcondition) du programme ;
2. Pour tout couple (P, Q) d'assertions consécutives, l'instruction ou la liste d'instructions C située entre ces assertions est telle que $\{\ P\ \}\, C\, \{\ Q\ \}$ est valide.

Il reste donc à vérifier cela pour les quatre couples d'assertions consécutives de notre programme...

5 Boucles inconditionnelles

On veut calculer les valeurs d'une suite u définie par $u_0 = 3$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = 3u_n + 1$.

```
x = 3
for i in range(n):
    x = 3 * x + 1
```

Montrer que le programme est correct vis-à-vis de la précondition « `n` est un entier » et la postcondition « `x` contient la valeur de u_n ».

On décore le programme en ajoutant un invariant de boucle :

```
1 # n entier
2 x = 3
3 for i in range(n):
4     # n entier, x == u(i)
5     x = 3 * x + 1
6 # x == u(n)
```

Correction d'une boucle **for** faisant itérer une variable i sur une liste `range(n)` avec $n \in \mathbb{N}$:

- Écrire un *invariant de boucle* $P(i)$ avant la première instruction du corps de la boucle (l. 4).
- Vérifier $P(0)$ avant l'entrée dans la boucle (l. 3).
- Vérifier que le triplet de Hoare $\{ P(i) \}$ corps de la boucle $\{ P(i+1) \}$ est valide pour tout $i \in [\![0, n[\![$.
- Alors la postcondition $P(n)$ est vérifiée.

Généralisation à `range(a, b)` avec $(a, b) \in \mathbb{N}^2$ et $a \leq b$:

- Écrire un *invariant de boucle* $P(i)$ avant la première instruction du corps de la boucle (l. 4).
- Vérifier $P(a)$ avant l'entrée dans la boucle (l. 3).
- Vérifier que le triplet de Hoare $P(i)$, corps de la boucle, $P(i+1)$ est valide pour tout $i \in [\![a, b[\![$.
- Alors la postcondition $P(b)$ est vérifiée.

5.1 Suite de Fibonacci

Exercice : montrer que si n est un entier naturel, alors après l'exécution de ce programme, x vaut F_n, où $(F_n)_{n\in\mathbb{N}}$ est la suite de Fibonacci.

```
x = 0
y = 1
for i in range(n):
    x, y = y, x+y
```

Solution : en étiquetant le programme ainsi

```
1 # n est un entier
2 x = 0
3 y = 1
4 for i in range(n):
5     # x == F(i) et y == F(i+1)
6     x, y = y, x+y
7 # x == F(n)
```

il suffit de constater :

- À l'entrée de la boucle, on a $x = F_0$ et $y = F_1$.
- Sous la précondition $x = F_i$ et $y = F_{i+1}$, l'instruction `x, y = y, x+y` conduit à la postcondition $x = F_{i+1}$ et $y = F_{i+2}$.
- À la sortie de la boucle, on a donc $x = F_n$ et $y = F_{n+1}$.

5.2 Exemple : calcul du maximum

Précondition : t est un tableau non vide contenant des entiers.

Montrer que sous cette précondition, après l'exécution du programme suivant, j contient l'indice d'un élément de t qui est maximum.

```
1 j = 0
2 for i in range(1, len(t)):
3     if t[i] > t[j]:
4         j = i
```

Dire que j contient l'indice d'un élément de t qui est maximum est dire qu'on a $j \in [\![0, n[\![$ et $\forall i \in [\![0, n[\![\ t[i] \leq t[j]$ où n est la longueur du tableau t.

Pour $i \in [\![1, n+1[\![$, notons $P(i)$ l'assertion

$$j \in [\![0, n[\![\text{ et } \forall k \in [\![0, i[\![\ t[k] \leq t[j]$$

Montrons que, ligne 3, l'invariant de boucle $P(i)$ est vérifié :

- Ligne 2, on a $j = 0$ or $n > 0$ (précondition t non vide), donc $j < n$. De plus $[\![0, 1[\![= \{ 0 \}$, donc $P(1)$ est vraie.

– Pour tout $i \in [\![1, n[\![$, montrons que sous la précondition $P(i)$, le corps de la boucle (lignes 3-4) conduit à la postcondition $P(i+1)$.
Soit $i \in [\![1, n[\![$. Supposons que la précondition est vérifiée.

Notons j_0 la valeur de j au début du corps de boucle. On a $\forall k \in [\![0, i[\![\ t[k] \leq t[j_0]$.

Si $t[i] \leq t[j_0]$, alors le test de l'instruction conditionnelle ligne 3 vaut faux, donc la ligne 4 n'est pas exécutée donc après l'exécution du corps de la boucle j n'a pas changé, on a toujours $j \in [\![0, n[\![$ et on a $\forall k \in [\![0, i+1[\![\ t[k] \leq t[j]$, c'est-à-dire $P(i+1)$.

Si en revanche, $t[i] > t[j_0]$, alors pour tout $k \in [\![0, i[\![$, on a $t[k] \leq t[j_0] < t[i]$. De plus, on a évidemment $t[i] \leq t[i]$, donc pour tout $k \in [\![0, i+1[\![$, on a $t[k] \leq t[i]$. Par ailleurs, le test ligne 3 vaut vrai,

donc on exécute la ligne 4. À la fin du corps de la boucle, j vaut donc i et de plus $i \in [\![1, n[\![$ donc $j \in [\![0, n[\![$. On a donc $P(i+1)$.

Dans les deux cas, on a $P(i+1)$ à la fin du tour de boucle.

– On en déduit qu'à la fin de la boucle, on a $P(n)$, qui est exactement ce qu'on veut démontrer.

6 Boucles conditionnelles

Exemple : $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 0$ et $\forall n \in \mathbb{N}\ u_{n+1} = u_n + 1/(n+1)^2$.
Trouver $n \in \mathbb{N}$ tel que $u_n \geq k$, où k est une valeur fixée.

Montrer qu'à l'issue de l'exécution du programme suivant, sous la précondition k est un nombre flottant, n contient un entier tel que $u_n \geq k$.

```
n = 0
x = 0
while x < k:
    x += 1. / (n+1)**2
    n += 1
```

On décore le programme en ajoutant un invariant de boucle :

```
1  # k nombre flottant
2  n = 0
3  x = 0
4  while x < k:
5      # x == u(n)
6      x += 1. / (n+1)**2
7      n += 1
8  # u(n) >= k
```

Correction d'une boucle **while** portant sur une condition B :

- Écrire un *invariant de boucle* P avant la première instruction du corps de la boucle (l. 5)
- Vérifier P avant l'entrée dans la boucle (l. 4)
- Vérifier que le triplet de Hoare $\{ P \wedge B \}$ corps de la boucle $\{ P \}$ est valide.
- Alors la postcondition $P \wedge \neg B$ est vérifiée.

6.1 Attention

Cette méthode ne montre que la correction partielle !

Sur le programme précédent : Pour k supérieur à une certaine valeur L, le programme ne termine pas (avec $L = \pi^2/6$ aux erreurs d'arrondi près).

7 Correction totale

Correction totale : correction partielle et terminaison.

Seule source de non-terminaison : les boucles conditionnelles.

Pour montrer la terminaison : on introduit un *variant* de boucle.

8 Exemple : recherche linéaire

Précondition : t est un tableau. Montrer que si la valeur x apparaît dans le tableau, après l'exécution du programme suivant, i contient un entier tel que $t[i] = x$ et que si la valeur x n'apparaît pas, après l'exécution du programme $i \geq \texttt{len}(t)$.

```
i = 0
while i < len(t) and t[i] != x:
    i += 1
```

On peut décorer le programme comme suit :

```
1  i = 0
2  while i < len(t) and t[i] != x:
3      # pour tout k tq 0<=k<i, t[k] != x
4      # variant : len(t) - i
5      i += 1
```

Correction partielle :

- Avant l'exécution de la boucle, on a $\forall k \in [\![0, 0[\![\ t[k] \neq x$.

- Montrons que, sous la précondition $\forall k \in [\![0, i[\![\ t[k] \neq x$ et $i <$ `len`(t) et $t[i] \neq x$, l'exécution du corps de la boucle conduit à la postcondition $\forall k \in [\![0, i[\![\ t[k] \neq x$.

  Notons α la valeur de i au début du corps de la boucle. Alors on a $\forall k \in [\![0, \alpha[\![\ t[k] \neq x$ et $\alpha <$ `len`(t) et $t[\alpha] \neq x$, donc $\forall k \in [\![0, \alpha + 1[\![\ t[k] \neq x$.

  De plus, après l'exécution du corps de la boucle, i vaut $\alpha + 1$, on a donc bien la condition $\forall k \in [\![0, i[\![\ t[k] \neq x$.

- Après l'exécution du corps de la boucle, on a donc $\forall k \in [\![0, i[\![\ t[k] \neq x$ et ($i \geq$ `len`(t) ou $t[i] = x$). Si $i \geq$ `len`(t), alors on a $\forall k \in [\![0, \texttt{len}(t)[\![\ t[k] \neq x$ et la postcondition est bien vérifiée. Si $t[i] = x$, elle est vérifiée également.

Terminaison : on introduit un variant (l. 4) : quantité entière positive ou nulle qui décroît strictement à chaque tour de boucle.

Plus précisément, ici

- Sous la précondition «l'invariant de boucle et la condition de la boucle sont vérifiés», $\texttt{len}(t) - i$ est un entier naturel.
- Sous cette même précondition, la valeur de $\texttt{len}(t) - i$ à la fin du corps de la boucle est strictement plus petite qu'au début.

Donc le programme termine.

9 Chiffres d'un nombre

Précondition : n est un entier naturel. Postcondition : t est un tableau contenant la liste des chiffres de n, en commençant par les chiffres de poids faibles (ce tableau est vide si $n = 0$).

```
1  t = []
2  k = n
3  while k > 0:
4      # n == k * 10**p
5      #       + somme(t[i]*10**i pour i dans range(p))
6      # où p désigne len(t)
7      # variant : k
8      c = k % 10
9      t.append(c)
10     k = k // 10
```

NB : on se contente de montrer la postcondition $n = \sum_{i \in [\![0,p[\![} t[i] \times 10^i$ (où p désigne la longueur de t). On ne montre pas que le dernier chiffre de t est non nul (dans le cas $n \neq 0$).

Correction partielle :

- À l'entrée dans la boucle, $t = []$, donc $\texttt{len}(t) = 0$, $\sum_{i \in [\![0,\texttt{len}(t)[\![} t[i] \times 10^i = 0$ et $n = k$ donc l'invariant est vérifié.

– Notons p_0 la longueur de t avant l'exécution du corps de la boucle, $x_0, \ldots\ x_{p_0-1}$ les éléments de t, et k_0 la valeur de k. Après l'exécution de ce corps de boucle, notons x_{p_0} le nouvel élément de t , c_1 la valeur de c et k_1 la valeur de k et p_1 la longueur de t. Sous la précondition $k_0 > 0$ et $n = k_0 \times 10^{p_0} + \sum_{i \in [\![0,p_0[\![} x_i \times 10^i$, on a $10k_1 + c_1 = k_0$ et $x_p = c_1$.

  Donc $k_0 \times 10^{p_0} = k_1 \times 10^{p_0+1} + x_p \times 10^p$, donc $n = k_1 \times 10^{p_1} + \sum_{i \in [\![0,p_1[\![} x_i \times 10^i$.

Terminaison :

– Clairement, k est un entier naturel.

– En notant k_0 la valeur de k avant l'exécution du corps de la boucle et k_1 sa valeur après l'exécution, sous la précondition $k_0 > 10$, on a $k_1 = \lfloor k_0/10 \rfloor \leq k_0/10 < k_0$. Donc k décroît strictement à chaque exécution du corps de la boucle.

10 Recherche dichotomique

Précondition : t tableau de flottants triés par ordre croissant, x flottant apparaissant dans t.

On notera R l'ensemble des indices k du tableau pour lesquels $t[k] = x$.

Justifier qu'on a la postcondition : « $t[i] = x$ si $R \neq \emptyset$ » (ce n'est pas la même chose que la postcondition « $t[i] = x$ » sous la précondition additionnelle « $R \neq \emptyset$ »)

```
1  # t trié par ordre croissant
2  b, e = 0, len(t)
3  while b<e and t[(b+e)//2] != x:
4    # R inclus dans range(b,e)
5    # et t trié par ordre croissant
6    # et 0<=b et e<=len(t)
7    # variant : e-b
8    if t[(b+e)//2] > x:
9      e = (b+e) // 2
10   else:
11     b = 1 + (b+e) // 2
12 i = (b+e) // 2
```

Correction partielle.

- À l'entrée de la boucle, l'invariant est clairement vérifié.

– Montrons que sous la précondition l'invariant de boucle est vérifié et $b < e$ et $t[(b+e)/2] \neq x$ l'exécution du corps de la boucle mène à la postcondition l'invariant de boucle est vérifié.

Notons b_0 et e_0 les valeurs de b et e au début du corps de la boucle et m_0 la valeur $\lfloor (b_0 + e_0)/2 \rfloor$. Alors $0 \leq b_0 < e_0 \leq n$, ce qui assure $0 \leq m_0 < n$.

Si $t[m_0] > x$, alors, t étant trié par ordre croissant, $R \subset [\![0, m_0[\![$. Or $R \subset [\![b_0, e_0[\![$, donc $R \subset [\![b_0, m_0[\![$. Le test ligne 8, conduit à l'exécution de la ligne 9, après laquelle on a $e = m_0$, donc $e \leq n$, $R \subset [\![b, e[\![$ et par ailleurs t est toujours trié et $0 \leq b$.

Si $t[m_0] \leq x$, alors, comme $t[m_0] \neq x$, on a $R \subset [\![m_0 + 1, n[\![$. Or $R \subset [\![b_0, e_0[\![$, donc $R \subset [\![m_0 + 1, e_0[\![$. Le test ligne 8 conduit à

l'exécution de la ligne 11, après laquelle on a $b = m_0 + 1$, donc $0 \leq b$, $R \subset [\![b, e[\![$ et par ailleurs t est toujours trié et $e \leq n$.

– À la sortie de la boucle, on a l'invariant de boucle $R \subset [\![b, e[\![$ et ($b \geq e$ ou $t[\lfloor (b+e)/2 \rfloor] = x$). Dans le premier cas, $R = \emptyset$, dans le second, après l'instruction ligne 12, on a $t[i] = x$.

Donc sous la précondition t est trié par ordre croissant, on a la postcondition si $R \neq \emptyset$, $t[i] = x$.

Terminaison :

– Clairement, $e - b$ est toujours un entier.

– En notant comme précédemment b_0 et e_0 les valeurs respectives de b et e avant l'exécution du corps de la boucle, et b_1 et e_1 leur valeur après l'exécution, on a clairement $e_1 - b_1 = \lfloor (b_0 + e_0)/2 \rfloor - b_0$ ou $e_1 - b_1 = e_0 - \lfloor (b_0 + e_0)/2 \rfloor - 1$. Or $(b_0 + e_0)/2 - 1 < \lfloor (b_0 + e_0)/2 \rfloor \leq (b_0 + e_0)/2$, d'où $e_1 - b_1 \leq (e_0 - b_0)/2$ dans le premier cas et $e_1 - b_1 < (e_0 - b_0)/2$ dans le second. Or sous la précondition $b_0 < e_0$, on a $(e_0 - b_0)/2 < e_0 - b_0$. D'où $e_1 - b_1 < e_0 - b_0$. Donc $e - b$ décroît strictement à chaque tour de boucle.