Algorithmes : terminaison et correction

Florent Foucaud - Pascal Lafourcade - Malika More
(malika.more@uca.fr)

3A - BUT INFO - UCA

Qualité algorithmique

Trois questions

Face à un algorithme

- Est-ce-qu'il donne toujours un résultat ?
  ou bien est-ce-que parfois il ne s'arrête jamais ?

  Terminaison
- Est-ce-qu'il donne toujours le résultat attendu ?
  ou bien est-ce-qu'il calcule parfois n'importe quoi ?

  Correction
- Est-ce-qu'il donne toujours un résultat en un temps raisonnable ?
  ou bien est-ce-qu'il faut parfois attendre plusieurs siècles ?

  (Complexité)

Parfois c'est simple

On est sûr qu'un algorithme se termine toujours :

- Lorsque le nombre d'instructions à effectuer est connu à l'avance

On est sûr qu'un algorithme est correct :

- Lorsque l'algorithme reproduit directement la spécification

L'âge de Toutou

```
def AgeHumain(x):
    if x>=19:
        y=10*x-90
    elif x>=17:
        y=6*x-14
    elif x>=2:
        y=4*x+20
    else :
        y=8*x+12
    return y
```

Un être humain et un chien ne vieillissent pas de la même manière. Pour déterminer l'équivalent humain de l'âge d'un chien de moins de 15 kg, on utilise le tableau ci-dessous, où x représente l'âge réel de l'animal (en années), et y l'équivalent humain en terme de vieillissement.

x	y
moins de 2	$8x + 12$
de 2 à 17	$4x + 20$
de 17 à 19	$6x - 14$
19 et plus	$10x - 90$

L'âge de Toutou

```
def AgeHumain(x):
    if x>=19:
        y=10*x-90
    elif x>=17:
        y=6*x-14
    elif x>=2:
        y=4*x+20
    else :
        y=8*x+12
    return y
```

Terminaison

- C'est un « algorithme-calcul »
- Il se termine simplement quand toutes les instructions sont effectuées

Correction

- L'algorithme correspond à la formule
- La formule est très simple

Somme des entiers jusqu'à n

```
def Somme(n):
    S=0
    for i in range(n+1):
        S=S+i
    return S
```

Somme des entiers jusqu'à n

```
def Somme(n):
    S=0
    for i in range(n+1):
        S=S+i
    return S
```

Terminaison

- Il y a une boucle, mais le nombre de passages dans la boucle est connu *a priori*
- Donc l'algorithme se termine toujours

Somme des entiers jusqu'à n

```
def Somme(n):
    S=0
    for i in range(n+1):
        S=S+i
    return S
```

Correction

- On vérifie que l'initialisation de S est correcte
- On vérifie que l'itération est correcte
- On vérifie que le nombre d'itérations est correct
- Donc l'algorithme est correct

Souvent c'est moins simple

Terminaison :

- On ne connaît pas *a priori* le nombre d'instructions effectuées (boucle `while`, récursivité, ...)

Correction :

- La spécification est compliquée
- La spécification ne dit pas comment obtenir le résultat

Calcul du *PGCD* de deux entiers

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

Calcul du *PGCD* de deux entiers

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

L'algorithme se termine si la condition d'arrêt de la boucle se réalise

- C'est-à-dire si y s'annule
- Dans la boucle, y est remplacé par un reste $\%y$
- À chaque passage, y décroît strictement, tout en restant ≥ 0
- Par conséquent, y finit par atteindre 0
- Et on sort de la boucle
- Conclusion : l'algorithme se termine

Calcul du *PGCD* de deux entiers

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

L'algorithme se termine si la condition d'arrêt de la boucle se réalise

- C'est-à-dire si y s'annule
- Dans la boucle, y est remplacé par un reste $\%y$
- À chaque passage, y décroît strictement, tout en restant ≥ 0
- Par conséquent, y finit par atteindre 0
- Et on sort de la boucle
- Conclusion : l'algorithme se termine

Calcul du *PGCD* de deux entiers

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

Diviseurs de 70 :
1, 2, 5, 7, 10, 14, 35, 70

Diviseurs de 35 :
1, 5, 25

$PGCD(70, 25) = 5$

`Euclide(70,25)`

x	y
70	25
25	20
20	5
5	0

renvoie 5

Calcul du *PGCD* de deux entiers

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

L'algorithme est supposé calculer $PGCD(a, b)$

- Au départ, on a $x = a$ et $y = b$
- Dans la boucle, on remplace (x, y) par $(y, x\%y)$
- On sait que $PGCD(x, y) = PGCD(y, x\%y)$
- Donc à chaque étape, $PGCD(x, y) = PGCD(a, b)$
- En sortie de boucle, on a $y = 0$ et on renvoie x
- On sait que $PGCD(x, 0) = x$
- Conclusion : l'algorithme calcule bien $PGCD(a, b)$

Calcul du *PGCD* de deux entiers

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

L'algorithme est supposé calculer $PGCD(a, b)$

- Au départ, on a $x = a$ et $y = b$
- Dans la boucle, on remplace (x, y) par $(y, x\%y)$
- On sait que $PGCD(x, y) = PGCD(y, x\%y)$
- Donc à chaque étape, $PGCD(x, y) = PGCD(a, b)$
- En sortie de boucle, on a $y = 0$ et on renvoie x
- On sait que $PGCD(x, 0) = x$
- Conclusion : l'algorithme calcule bien $PGCD(a, b)$

Outil pour la terminaison

Définition

On appelle convergent une quantité qui prend ses valeurs dans un ensemble bien fondé et qui diminue strictement à chaque passage dans une boucle.

Outil pour la terminaison

Définition

On appelle convergent une quantité qui prend ses valeurs dans un ensemble bien fondé et qui diminue strictement à chaque passage dans une boucle.

Remarques

- Un ensemble *bien fondé* est un ensemble totalement ordonné dans lequel il n'existe pas de suite infinie strictement décroissante.
- En particulier, $\mathbb{N}$, ou $\mathbb{N}^k$ muni de l'*ordre lexicographique*, sont des ensembles bien fondés.
- On a $(a, b) < (c, d)$ pour l'ordre lexicographique lorsque $a < c$ ou $a = c$ et $b < d$.
- Un exemple d'ensemble qui n'est pas bien fondé est $\mathbb{Z}$: $\ldots < -3 < -2 < -1 < 0 < 1 < 2 < 3$ est une suite infinie strictement décroissante.

Outil pour la terminaison

Définition

On appelle convergent une quantité qui prend ses valeurs dans un ensemble bien fondé et qui diminue strictement à chaque passage dans une boucle.

Propriété

L'existence d'un convergent pour une boucle garantit que l'algorithme finit par en sortir.

Terminaison de l'algorithme d'Euclide

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

On a vu que y est un convergent

Outil pour la correction

Définition

On appelle invariant de boucle une propriété qui, si elle est vraie avant l'entrée dans une boucle, reste vraie après chaque passage dans cette boucle, et donc est vraie aussi à la sortie de cette boucle.

Outil pour la correction

Définition

On appelle invariant de boucle une propriété qui, si elle est vraie avant l'entrée dans une boucle, reste vraie après chaque passage dans cette boucle, et donc est vraie aussi à la sortie de cette boucle.

Remarque

Analogie évidente avec une preuve par récurrence :

- Entrée de boucle $\longrightarrow$ initialisation
- Passage dans la boucle $\longrightarrow$ hérédité
- Sortie de boucle $\longrightarrow$ conclusion

Outil pour la correction

Définition

On appelle invariant de boucle une propriété qui, si elle est vraie avant l'entrée dans une boucle, reste vraie après chaque passage dans cette boucle, et donc est vraie aussi à la sortie de cette boucle.

Propriété

La mise en évidence d'un invariant de boucle adapté permet de prouver la correction d'un algorithme.

Correction de l'algorithme d'Euclide

On a vu que
$P(i) : pgcd(x_i, y_i) = pgcd(a, b)$
est un invariant de boucle

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

Correction de l'algorithme d'Euclide

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

- *Initialisation* : Avant la boucle, on a $x = a$ et $y = b$, donc
  $P(0) : pgcd(x_0, y_0) = pgcd(a, b)$
  est vérifiée

Correction de l'algorithme d'Euclide

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

- *Hérédité* : Supposons que $P(i) : pgcd(x_i, y_i) = pgcd(a, b)$ soit vérifiée après le $i$ème passage dans la boucle
  - Dans la boucle, on calcule $x_{i+1} = y_i$ et $y_{i+1} = x_i \% y_i$
  - Or $PGCD(u, v) = PGCD(v, u\%v)$
  - Donc $PGCD(x_{i+1}, y_{i+1}) = PGCD(x_i, y_i) = PGCD(a, b)$
  - Après le $i + 1$ième passage dans la boucle : $P(i + 1) : pgcd(x_{i+1}, y_{i+1}) = pgcd(a, b)$ est vérifiée

Correction de l'algorithme d'Euclide

```
def Euclide(a,b):
    x = a
    y = b
    while y > 0:
        temp = y
        y = x % y
        x = temp
    return x
```

- *Conclusion* : En sortie de boucle, $P(f) : PGCD(x_f, y_f) = PGCD(a, b)$ est vérifiée
- Or $y_f = 0$ et on renvoie x_f
- On sait que $PGCD(u, 0) = u$
- Conclusion : l'algorithme calcule bien $PGCD(a, b)$

Algorithme « aléatoire »

```
import random

def Aleatoire(n,p):
    x=n
    y=p
    while (x>0 or y>0):
        if y>0:
            y=y-1
        else :
            if x>0 :
                x=x-1
                y=random.randint(0,100)
    return (x,y)
```

Algorithme « aléatoire »

```
import random

def Aleatoire(n,p):
    x=n
    y=p
    while (x>0 or y>0):
        if y>0:
            y=y-1
        else :
            if x>0 :
                x=x-1
                y=random.randint(0,100)
    return (x,y)
```

```
Aleatoire(3,3)
```

$(3,3)\ (3,2)\ (3,1)\ (3,0)$
$(2,5)\ (2,4)\ (2,3)\ (2,2)\ (2,1)\ (2,0)$
$(1,3)\ (1,2)\ (1,1)\ (1,0)$
$(0,4)\ (0,3)\ (0,2)\ (0,1)\ (0,0)$

Algorithme « aléatoire »

```
import random

def Aleatoire(n,p):
    x=n
    y=p
    while (x>0 or y>0):
        if y>0:
            y=y-1
        else :
            if x>0 :
                x=x-1
                y=random.randint(0,100)
    return (x,y)
```

Cet algorithme se termine-t-il toujours ?

- C'est l'occasion de découvrir un convergent plus sophistiqué !
  $\rightsquigarrow$ (x, y)

Algorithme « aléatoire »

```
import random

def Aleatoire(n,p):
    x=n
    y=p
    while (x>0 or y>0):
        if y>0:
            y=y-1
        else :
            if x>0 :
                x=x-1
                y=random.randint(0,100)
    return (x,y)
```

Cet algorithme se termine-t-il toujours ?

- C'est l'occasion de découvrir un convergent plus sophistiqué !
  $\rightsquigarrow$ (x, y)

Algorithme « aléatoire »

```
import random

def Aleatoire(n,p):
    x=n
    y=p
    while (x>0 or y>0):
        if y>0:
            y=y-1
        else :
            if x>0 :
                x=x-1
                y=random.randint(0,100)
    return (x,y)
```

Cet algorithme se termine-t-il toujours ?

- La sortie de boucle se produit lorsque $(x, y) = (0, 0)$
- $(x, y) \in \mathbb{N}^2$ diminue strictement à chaque passage dans la boucle pour l'ordre lexicographique :
  - si $y > 0 \rightsquigarrow (x, y) < (x, y - 1)$
  - si $y = 0 \rightsquigarrow (x, 0) < (x - 1, 0)$
- $(0, 0)$ sera atteint un jour car $\mathbb{N}^2$ est *bien fondé*

Algorithme « aléatoire »

```
import random

def Aleatoire(n,p):
    x=n
    y=p
    while (x>0 or y>0):
        if y>0:
            y=y-1
        else :
            if x>0 :
                x=x-1
                y=random.randint(0,100)
    return (x,y)
```

Cet algorithme se termine-t-il toujours ?

OUI

car (x, y) est un convergent pour la boucle

Suite de Syracuse

```
def Syracuse(n):
    x=n
    while x>1:
        if x%2==0:
            x=x//2
        else:
            x=3*x+1
    return x
```

Suite de Syracuse

```
def Syracuse(n):
    x=n
    while x>1:
        if x%2==0:
            x=x//2
        else:
            x=3*x+1
    return x
```

```
Syracuse(18)
18      9
28      14      7
22      11
34      17
52      26      13
40      20      10      5
16      8       4       2
1
```

Suite de Syracuse

```
def Syracuse(n):
    x=n
    while x>1:
        if x%2==0:
            x=x//2
        else:
            x=3*x+1
    return x
```

On ne connaît pas de convergent et personne ne sait si cet algorithme se termine toujours !

Exponentielle rapide

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

Exponentielle rapide

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

Terminaison :

- On se convainc facilement que m est un convergent
- Conclusion : l'algorithme se termine

Exponentielle rapide

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

Exemple : $a = 5$ et $n = 13$
Retourne-t-on $a^n = 5^{13}$?

- Avant la boucle
  - $p = 1$, $b = 5$, $m = 13$
- Boucle : $m = 13$ (impair)
  - $p = 5$, $b = 5^2$, $m = 6$
- Boucle : $m = 6$ (pair)
  - $p = 5$, $b = 5^4$, $m = 3$
- Boucle : $m = 3$ (impair)
  - $p = 5^5$, $b = 5^8$, $m = 1$
- Boucle : $m = 1$ (impair)
  - $p = 5^{13}$, $b = 5^{16}$, $m = 0$
- $m = 0$: Sortie de boucle
- On retourne bien $p = 5^{13}$

Exponentielle rapide

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

Correction :

- On vérifie que
  $P(i) : p_i \times b_i^{m_i} = a^n$
  est un invariant de boucle

Exponentielle rapide

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

Correction :

- On vérifie que
  $P(i) : p_i \times b_i^{m_i} = a^n$
  est un invariant de boucle
- *Initialisation*
  Avant la boucle :
  $p_0 = 1$, $b = a$ et $m_0 = n$
  et $1 \times a^n = a^n$, donc
  $P(0) : p_0 \times b_0^{m_0} = a^n$
  est vérifiée

Exponentielle rapide

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

Correction :

- On vérifie que
  $P(i) : p_i \times b_i^{m_i} = a^n$
  est un invariant de boucle
- *Hérédité* Supposons que
  $P(i) : p_i \times b_i^{m_i} = a^n$
  soit vérifiée après le $i$ème passage dans la boucle
  - Si $m_i = 2k$:
    $p_{i+1} = p_i$, $b_{i+1} = b_i^2$ et $m_{i+1} = k$
    donc $p_{i+1} \times b_{i+1}^{m_{i+1}} = p_i \times (b_i^2)^k =$
    $p_i \times b_i^{2k} = p_i \times b_i^{m_i} = a^n$
    donc $P(i+1) : p_{i+1} \times b_{i+1}^{m_{i+1}} = a^n$
    est vérifiée après le $i+1$ième passage dans la boucle

Exponentielle rapide

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

Correction :

- On vérifie que
  $P(i) : p_i \times b_i^{m_i} = a^n$
  est un invariant de boucle
- *Hérédité* Supposons que
  $P(i) : p_i \times b_i^{m_i} = a^n$
  soit vérifiée après le $i$ème passage dans la boucle
  - Si $m_i = 2k + 1$:
    $p_{i+1} = p_i \times b_i$, $b_{i+1} = b_i^2$ et
    $m_{i+1} = k$ donc
    $p_{i+1} \times b_{i+1}^{m_{i+1}} = (p_i \times b_i) \times (b_i^2)^k =$
    $p_i \times b_i^{2k+1} = p_i \times b_i^{m_i} = a^n$
    donc $P(i+1) : p_{i+1} \times b_{i+1}^{m_{i+1}} = a^n$
    est vérifiée après le $i + 1$ième passage dans la boucle

Exponentielle rapide

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

Correction :

- On vérifie que
  $P(i) : p_i \times b_i^{m_i} = a^n$
  est un invariant de boucle
- *Conclusion*
  En sortie de boucle,
  $P(f) : p_f \times b_f^{m_f} = a^n$ est vérifiée

Exponentielle rapide

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

Correction :

- On a vérifié que
  $P(i) : p_i \times b_i^{m_i} = a^n$
  est bien un invariant de boucle
- En sortie de boucle, on a
  $p_f \times b_f^{m_f} = a^n$
  Or $m_f = 0$ et on renvoie p_f
  On sait que $u^0 = 1$
  donc $p_f \times b_f^{m_f} = p_f \times b_f^0 = p_f = a^n$
- Conclusion :
  l'algorithme calcule bien a^n

FIN