Chapitre 1 : correction et terminaison des algorithmes

12 janvier 2024

I Un exemple d'introduction : la division Euclidienne

On donne ci-dessous un exemple d'algorithme calculant le quotient et le reste de la division euclidienne d'un entier a par un entier b par soustractions successives.

```
def div(a,b):
    q=0
    r=a
    while r>=b:
        q=q+1
        r=r-b
    return q,r
```

Problème. Cet algorithme s'arrête-t-il toujours ? Donne-t-il le bon résultat ?

II Préconditions, postconditions et correction des programmes

Définition. Soit P un problème instancié par une donnée D et fournissant un résultat R. Une spécification du problème est la donnée :

1. D'un propriété $P(D)$ portant sur les donnée D appelée **précondition**.
2. D'une propriété $Q(D,R)$ portant sur les données D et le résultat R appelée **postcondition**.

Définition.

1. Un programme est dit partiellement correct par rapport à une spécification si pour toute donnée D vérifiant la précondition $P(D)$ et telle que le programme s'arrête, le résultat vérifie la post-condition $Q(D,R)$.
2. Le programme est dit correct si pour toute donnée D vérifiant la précondition $P(D)$, le programme s'arrête et le résultat vérifie la post-condition $Q(D,R)$.

On peut résumer les conditions à vérifier pour la correction ainsi :

correction partielle et terminaison $\Rightarrow$ correction

III Terminaison et variant de boucle

1 Boucles non conditionnelle

Pour vérifier la terminaison d'un programme, il faut vérifier que toutes ses boucles terminent. Les boucles non conditionnelles (ou boucles **for**) ne posent a priori pas de problème, le nombre d'exécutions étant a priori fixé à l'avance. Il faut toutefois éviter de modifier l'indice de boucle à l'intérieur de celle-ci, comme dans l'exemple ci-dessous.

```
for i in range(10):
        i=i-1
```

Remarque. Cela ne crée en fait pas une boucle infinie en Python.

2 Boucle non conditionnelle

Les boucles conditionnelles (ou boucles **while**) posent plus de problèmes : elle ne s'arrêtent que lorsque la condition d'entrée dans la boucle devient fausse. Comment vérifier que c'est le cas ?

Une méthode pratique est d'utiliser un **variant de boucle**. Un tel variant est une quantité entière qui est positive et décroît strictement à chaque itération de la boucle.

Une suite d'entiers positifs strictement décroissante ne pouvant pas être infinie, l'existence d'un tel variant garantit la terminaison de la boucle.

IV Correction et invariant

1 Notion d'invariant de boucle

En admettant avoir prouvé que le programme termine, il reste à prouver qu'il vérifie les spécifications données.

Une méthode usuelle est d'associer à chaque boucle un **invariant de boucle**, c'est-à-dire une propriété vérifiée à chaque passage dans celle-ci.

La conjonction de l'invariant de boucle et de la condition de terminaison de celle-ci (c'est-à-dire de la négation de sa condition de continuation) permet généralement de prouver le résultat attendu en sortie de boucle.

2 Cas d'une boucle for : calcul d'une somme

On donne maintenant l'algorithme ci-dessus, qui prend en entrée un entier naturel n et doit retourner la somme $\sum_{k=0}^{n} k$.

```
def somme(n):
    s=0
    for i in range(n+1):
        s=s+i
    return s
```

3 Un exemple plus complexe : recherche d'un élément dans un tableau

On désire un algorithme prenant en entrée une liste d'entiers T et un entier elt et désirant tester si elt appartient à T, en s'arrêtant éventuellement dès que elt a été trouvé. Une idée est de partir d'un indice i égal à 0, et, tant que $T[i]$ n'est pas l'élément recherché et que i n'a pas atteint $len(T)$, d'incrémenter i. On a lors trouvé l'élément si et seulement si la boucle s'interrompt avant que tous le tableau ait été parcouru. Cela est résumé par l'algorithme suivant.

```
def recherche(T,elt):
    i=0
    while i<len(T) and T[i]!=elt:
        i=i+1
    return i!=len(T)
```