Université Clermont Auvergne
ISIMA FISA 1
Algorithmique

Florent Foucaud

TD2 - Terminaison et correction d'un algorithme

VERSION AVEC SOLUTION

Les exercices commencent à la page 2

1 Rappel de cours

1.1 Terminaison d'un algorithme

Pour montrer qu'un algorithme se termine (c'est-à-dire que l'exécution de l'algorithme sur n'importe quelle entrée s'arrête au bout d'un temps fini), il faut montrer qu'une condition de terminaison est obligatoirement vérifiée à un moment donné de l'exécution. Par exemple, qu'on va sortir de la boucle principale, ou bien qu'on ne fait pas une infinité d'appels récursifs.

Pour une boucle, on peut utiliser un *convergent* pour montrer que la suite des valeurs de la variable principale (celle du test de terminaison de la boucle) converge vers la valeur qui interrompt la boucle.

Pour un algorithme récursif, on peut montrer que la suite des valeurs des appels récursifs converge vers le cas de base de la fonction récursive.

1.2 Correction d'un algorithme : les invariants de boucle

Il s'agit de montrer que l'algorithme est correct, c'est à dire qu'il calcule bien toujours le résultat attendu.

Pour montrer qu'un algorithme utilisant une boucle principale est correct, on peut utiliser un *invariant de boucle* : en notant par j le numéro d'itération de la boucle, c'est une propriété $P(j)$ qui est vraie avant la boucle ($j = 0$) ou bien après le premier tour de boucle ($j = 1$), qui reste vraie après chaque itération de la boucle (on passe de $P(j)$ à $P(j+1)$), et qui, pour la dernière itération, montre que le résultat calculé par l'algorithme est correct. On montre le passage de $P(j)$ à $P(j+1)$ généralement par récurrence sur j.

Pour un algorithme récursif, on utilise naturellement une preuve par récurrence sur les appels récursifs.

2 Exercices

Exercice 1 (Terminaisons d'algorithmes itératifs)**.**
Pour chacun des quatre algorithmes suivants, dites s'il se termine ou pas. Si oui, mettez en évidence un convergent et prouvez la terminaison. Si non, précisez les cas particuliers dans lesquels l'algorithme se termine.

```
Algo1(n)
Entrée : un entier n
Sortie : un entier
  • Tant que n > 0 faire :
      ⋆ Si n est pair alors :
        n = 2n
      ⋆ Sinon :
        n = n − 1
  • Retourner n
```

```
Algo2(n)
Entrée : un entier n
Sortie : un entier
  • Tant que n > 1 faire :
      ⋆ Si n est pair alors :
        n = n/2
      ⋆ Sinon :
        n = n − 1
  • Retourner n
```

```
Algo3(n)
Entrée : un entier n
Sortie : un entier
  • Tant que n ≠ 11 faire :
      ⋆ Si n est pair alors :
        n = n/2
      ⋆ Sinon :
        n = n − 2
  • Retourner n
```

```
Algo4(n)
Entrée : un entier n
Sortie : un entier
  • Tant que n > 1 faire :
      ⋆ Si n est pair alors :
        n = n/2
      ⋆ Sinon :
        n = n + 1
  • Retourner n
```

Solution.

1. Algo1 : Non, car la suite des valeurs de n croît vers $+\infty$, par exemple si n est pair et non nul, n est doublé à chaque étape. Si n est impair et $n \neq 1$, la valeur suivante est paire et donc on se retrouve dans le cas précédent. Néanmoins, si $n \in \{0, 1\}$, l'algorithme s'arrête immédiatement.
2. Algo2 : Oui, car la suite des valeurs de n décroît strictement à chaque passage dans la boucle, et finit par réaliser la condition d'arrêt. C'est donc un convergent qui prouve la terminaison de cet algorithme.
3. Algo3 : Non, car la terminaison dépend de n : si la suite des valeurs de n décroît strictement, elle ne réalise cependant pas forcément la condition d'arrêt ($n = 11$). Si la valeur initiale de n est de la forme $2^k p$ où p est impair et supérieur ou égal à 11, l'algorithme s'arrête, sinon il ne s'arrête pas (la valeur de n décroît vers $-\infty$).
4. Algo4 : On voit bien que oui, mais ce n'est pas si simple de trouver un convergent. Pour cela, on considère la suite des valeurs de n, sauf celles où la valeur précédente était impaire. Cette suite est décroissante et converge vers $n = 1$, qui est la valeur d'arrêt.

Exercice 2 (Terminaisons d'algorithmes récursifs)**.**
Dans le cas de la terminaison d'algorithmes récursifs, la notion de *convergent* s'adapte facilement : il s'agit d'une quantité qui diminue strictement *à chaque appel récursif* et qui prend ses valeurs dans un ensemble bien fondé.

Pour chacun des deux algorithmes suivants, dites s'il se termine ou pas. Si oui, mettez en évidence un convergent et prouvez la terminaison. Si non, précisez les cas particuliers dans lesquels l'algorithme se termine.

Algo5(n)

Entrée : un entier n

Sortie : un entier

- Retourner $n + Algo5(n-1)$

Algo6(n)

Entrée : un entier n

Sortie : un entier

- Si $n == 10$ alors :
  Retourner 1
- Retourner $n - Algo6(n-1)$

Algo7(n)

Entrée : un entier n

Sortie : un entier

- Si $n == 10$ alors :
  Retourner 1
- Si $n < 10$ alors :
  Retourner $n - Algo7(11)$
- Retourner $n - Algo7(n-1)$

Solution.

1. Algo5 : Non, car on effectue des appels récursifs indéfiniment (pas de test d'arrêt) et n décroît vers $-\infty$ pour tout n.
2. Algo6 : Non, car cela dépend de n. Si $n \geq 10$, le calcul se termine car la suite des arguments des appels récursifs converge vers $n = 10$, qui est une condition d'arrêt. Par contre, si $n < 10$, on a le même problème que pour Algo5 et le calcul ne se termine jamais.
3. Algo7 : Oui car si $n < 10$, l'appel récursif se fait avec $11 > 10$, donc on passe dans le cas où le calcul se termine.

Exercice 3 (Dans la vraie vie : le bug de Zune)**.**
Les baladeurs MP3 Zune 30 de Microsoft ont été commercialisés à partir de 2006. Le 31 décembre 2008, ils ont été paralysés par un bug (cf. `https://fr.wikipedia.org/wiki/Zune`). Ce bug provenait d'un bout de code qui convertissait un nombre de jours absolu (compté depuis le 1er janvier 1980) en un couple formé d'une année, et d'un nombre de jours relatif (depuis le 1er janvier l'année en cours). Par exemple : $380 \mapsto (1981, 14)$. En effet, 1980 était une année bissextile (366 jours) donc la décomposition est 380 = 366 (1 an) + 14 (jours). On suppose aussi que le 1er jour de l'année vaut 1 (pas 0), donc $367 \mapsto (1981, 1)$.

Le code en question (traduit en python) est le suivant :

```
def jour_de_l_annee(jour, annee):
  while jour > 365:
    if bissextile(annee):
      if jour > 366:
        jour = jour - 366
        annee = annee + 1
    else:
```

```
    jour = jour - 365
    annee = annee + 1
return (annee, jour)
```

1. Expliquez le bug.
2. Proposez une correction et prouvez la terminaison du programme corrigé.

Solution.

1. Que se passe-t-il le 31 décembre d'une année bissextile ?

   `jour` = 366 et `bissextile(annee)` = True

   La variable `jour` n'est pas modifiée et le programme boucle. Autrement dit, il n'y a pas de convergent possible pour la boucle `while`.

   Cela se produit pour la première fois avec l'appel `jour_de_l_annee(366)` (1980 bissextile), et donc aussi avec l'appel `jour_de_l_annee(10593)` (31 décembre 2008), puisque 2008 est la première année bissextile suivant la commercialisation de Zune 30.

   Dans les autres cas, tout fonctionne bien.

2. Voici une correction possible :

```
def jour_de_l_annee(jour, annee=1980):
  while jour > 365:
    if bissextile(annee):
      if jour > 366:
        jour = jour - 366
        annee = annee + 1
      else:
        return (annee, jour)
    else:
      jour = jour - 365
      annee = annee + 1
return (annee, jour)
```

   La valeur de `jour` est un convergent : on n'entre dans la boucle que si `jour` > 365 ; si on re-rentre dans la boucle, sa valeur a été décrémentée strictement (de 365 ou de 366). Au final, la condition `jour<=365` finit par être atteinte et le programme se termine.