Terminaison
Correction

Autres questions théoriques :
terminaison et correction d'un algorithme

Aujourd'hui :
des notions nouvelles et importantes

Une introduction à des notions théoriques importantes
- terminaison : variants de boucles
- correction : invariant de boucles

Motivations

Contexte
- un problème à résoudre
  - un problème = une question + des paramètres (données "en entrée")
- une instance de ce problème qui admet au moins une solution
  - une instance = un choix des données d'entrée
- un algorithme qui calcule sa solution
  - un ou même plusieurs algorithmes

Questions déjà vues
- De combien de temps a besoin l'algorithme pour calculer cette solution ?
- De combien d'espace-mémoire pour que l'algorithme calcule cette solution ?

Questions du jour
- **Est-ce que l'algorithme termine et calcule une réponse ?**
- **Est-ce que l'algorithme calcule effectivement la solution ?**

Je calcule la somme de n valeurs

Une instance : un choix de n et des n valeurs du tableau t

Principe d'un algorithme : je parcours le tableau, "du début à la fin", je lis chaque valeur, je l'accumule dans une variable (initialement mise à zéro), je retourne cette variable.

```
fonction sommer(t : tableau d'entiers, n : entier) retourne entier
  res : entier = 0        // j'accumule dans res
  i : entier              // iterateur
debut
  pour i de 0 à n-1 faire
     res = res + t[i]
  fin pour
  retourne res
fin fonction
```

Terminaison

Est-ce que l'algorithme termine ?

Je calcule la somme de n valeurs

Est-ce que cet algorithme termine ?

Par construction :

- la boucle pour commence 0 et **FINIT** à n-1
- son indice est incrémenté et prend toutes les valeurs entre 0 et n-1 dans le corps de la boucle (modulo le pas de l'incrément).

Remarque : la dernière valeur prise par l'indice de boucle est n. Le corps de la boucle n'est alors pas exécuté.

Ici :

. Cas 1 : si le tableau contient au moins une valeur alors n ≥ 1.
Et la boucle fait au moins 1 itération : de 0 à 0.
res est remis à jour à chaque itération.
. Cas 2 : si le tableau ne contient aucune valeur alors n=0.
L'indice de fin (n-1=-1) est strictement inférieur à l'indice du début (0) donc aucune itération de la boucle n'est effectuée.
res est inchangé : il vaut 0 (ce qui est peut-être dommage)

Dans les deux cas :

- la boucle termine (par construction)
- res à une valeur qui est renvoyée par la fonction

donc somme termine.

Je calcule la somme de n valeurs

Est-ce que cet algorithme termine ?

1ère variante tant que.

Implicitement, la difficulté est prise en charge par la boucle pour.

Qu'en est-il si je recode cette boucle avec un tant que ?

```
i = 0
tant que ( i < n ) répéter
  res = res + t[i]
  i = i + 1
fin tant que
```

Preuve :

- i = 0 avant la première itération, puis i est incrémenté de 1 à chaque itération donc i est une suite strictement croissante.
- La boucle tant que se termine quand i ≥ n, ce qui arrive ici quand i = n

- Le raisonnement sur res est identique

Je calcule la somme de n valeurs

Est-ce que cet algorithme termine ?

2ème variante tant que. Qu'en est-il si je recode cette boucle avec un tant que ?

Principe : j'explicite une suite d'indices décroissante et minorée.

```
i = n-1
tant que ( i ≥ 0 ) répéter
  res = res + t[i]
  i = i - 1
fin tant que
```

Preuve :

- i = n-1 avant la première itération donc i ≥ 0 dès que le tableau est non vide
- Puis i est décrémenté de 1 à chaque itération donc i décroît strictement à chaque itération.
- La boucle tant que n'est plus exécutée quand i = -1 ; elle se termine alors).
- Toute suite décroissante est minorée. Son premier terme i ≥ 0 > -1 = minorant

Donc i atteint ce minorant et la boucle se termine alors.

- Le raisonnement sur res est identique

Je calcule la somme de n valeurs

Est-ce que cet algorithme termine ?

3ème variante tant que. Qu'en est-il si je recode cette boucle avec un tant que ?
Principe : j'exploite "je dois faire n traitements exactement".

```
i = n
tant que ( i > 0 ) répéter
  i = i – 1
  res = res + t[i]  // parce que les tableaux sont numérotés de 0 à n-1 ...
fin tant que
```

Preuve :

- La boucle se répète tant que i prend des valeurs strictement positives.
- Avant la première itération, i = n = nombre de valeurs du tableau

donc i > 0 dès que le tableau est non vide.

- Puis i décroît strictement à chaque itération.
- Il n'existe pas de suite infinie strictement décroissante d'entiers strictement positifs.

Donc il ne peut y avoir qu'un nombre fini d'itérations.

- Le raisonnement sur res est identique

Je calcule la somme de n valeurs

Est-ce que cet algorithme termine ?

1. Dans les deux dernières versions précédentes (tant que),
   l'indice de boucle est un variant, c.a.d. ici :
   - une valeur initialement positive,
   - qui reste positive à chaque itération,
   - et qui décroit strictement à chaque itération.

2. Dans les deux premières versions (pour et tant que similaires),
   l'indice de boucle est aussi un variant :
   - initialement nul, strictement croissant et majoré par la condition d'arrêt (et la taille du tableau).

3. Une fonction se termine lorsqu'elle renvoie une valeur
   - c'est la dernière instruction de la fonction ici.
   - attention si la fonction est composée de si .. sinon ..

4. Il existe des algorithmes qui ne terminent pas !
   - bug : variant non initialisé → à corriger
   - voulu : la boucle extérieure d'un logiciel embarqué

Pour conclure sur le variant

- Le variant permet d'expliciter la terminaison de la boucle
- Le variant est une expression, le plus souvent le contenu d'une variable
- Le variant est très souvent l'itérateur de la boucle analysée

Remarque

- On veille à ce que l'accès aux éléments du tableau ait un sens quelque soit la valeur de l'indice : t[i] avec i = 0..n-1
  exemple : ni t[-1], ni t[n] n'existent

Correction

Est-ce que l'algorithme réalise le traitement attendu ?

Je calcule la somme de n valeurs

Est-ce que cet algorithme est correct ?

Définition :

un algorithme est correct si il réalise le traitement attendu.

Exemple :

sommer calcule la somme des n valeurs stockées dans le tableau t

Approche empirique : va bien pour le débuggage classique

- **tester** l'algorithme sur certains jeux de valeurs d'entrée
- bien adapté aux valeurs particulières

Approche formelle :

- **prouver** que l'algorithme est correct quelques soient les entrées admissibles
- identifier et prouver **un invariant de boucle**

Un invariant de boucle

Qu'est ce que c'est ?

Un **invariant de boucle** est une propriété vérifiée tout au long de l'exécution d'une boucle et qui exhibe la correction de l'algorithme à la terminaison de la boucle.

Comment le sait-on ?

Soit P une propriété. On prouve que P est invariant de boucle en 3 temps :

1. initialisation :
   P est vraie *avant la première itération* du corps de la boucle
2. conservation :
   On suppose P vraie *avant la i-ème itération.* On montre que la i-ème itération conserve P.
   C.a.d. que P est encore vraie avant la (i+1)-ème itération.
3. terminaison
   P est vraie après la dernière itération. Ce qui prouve le traitement attendu.

Un exemple : la boucle pour de `sommer`

Avant l'itération i, `res` a pour valeur la somme des (i-1) premières valeurs du tableau t

Un invariant de boucle

Remarques

- initialisation et conservation sont similaires à une preuve par récurrence
- la terminaison prouve la correction, et aussi la terminaison de l'algorithme si besoin
- identifier l'invariant est beaucoup plus difficile que le prouver
- identifier et prouver un invariant permet :
  - de comprendre *en profondeur* l'algorithme,
  - d'identifier les cas exceptionnels et leurs traitements,
  - de préparer l'analyse de la complexité de l'algorithme

La boucle pour dans `sommer`

Prouvons l'invariant suivant :

Avant l'itération d'indice i, `res` *a pour valeur la somme des i premières valeurs du tableau t, soit* `res` *= t[0]+...+t[i-1].*

```
1. fonction sommer(t : tableau d'entiers, n : entier) retourne entier
2.   res : entier = 0        // j'accumule dans res
3.   i : entier              // iterateur
4. debut
5.   pour i de 0 à n-1 faire
6.     res = res + t[i]
7.   fin pour
8.   retourne res
9. fin fonction
```

La boucle pour dans sommer

Prouvons l'invariant suivant :

(P) : Avant l'itération d'indice i, `res` *a pour valeur la somme des i premières valeurs du tableau t, soit* `res` *= t[0]+ ... +t[i-1]*

- Initialisation : avant la première exécution du corps de la boucle pour,
  - i = 0 : on suppose en effet que la mise à jour de l'indice de boucle a été réalisée. C.a.d. on se place juste avant la première instruction du corps de la boucle, au début de la ligne L6.
  - `res` = 0 depuis la ligne L2.
  - Les 0 premières valeurs du tableau = le tableau vide
  - La somme des valeurs du tableau vide = 0 = `res`

Donc la propriété (P) vérifie l'initialisation.

La boucle pour dans sommer

Prouvons l'invariant suivant :

(P) : Avant l'itération d'indice i, `res` *a pour valeur la somme des i premières valeurs du tableau t, soit* `res` *= t[0]+ ... +t[i-1]*

- Conservation:

Supposons (P) avant l'itération i : `res` = t[0]+ ... +t[i-1].

Exécutons l'itération d'indice i :

L6 : `res` = `res` + t[i] = t[0]+ ... +t[i-1] + t[i].

Le corps de boucle est achevé ; on entame l'itération suivant en mettant à jour i qui vaut maintenant i+1.

Et `res` a bien pour valeur la somme des (i+1) premières valeurs du tableau t : t[0]+ ... +t[i-1] + t[i].

Donc la propriété (P) est vérifiée avant l'itération d'indice i+1,

ce qui prouve la conservation.

La boucle pour dans sommer

Prouvons l'invariant suivant :

(P) : Avant l'itération d'indice i, `res` *a pour valeur la somme des i premières valeurs du tableau t, soit* `res` *= t[0]+ ... +t[i-1]*

- Terminaison :

Après la dernière itération du corps de la boucle, l'indice est mis à jour et vaut maintenant n.

Le corps de la boucle n'est donc pas exécuté donc la valeur de `res` demeure inchangée. D'après (P), `res` a pour valeur la somme des n premières valeurs du tableau t, soit :

`res` = t[0]+ ... + t[n-1].

Le tableau t est de longueur n et est composé des valeurs t[0] ... t [n-1].

La fonction `sommer` termine et renvoie `res` dont la valeur est la somme des n valeurs du tableau t. Ce qu'elle était censé faire !

Ce qui achève la preuve de la correction de `sommer`.

Conclusion sur la correction

- Invariant de boucle : notion essentielle
- Objectif 10 : savoir prouver un invariant de boucle
- Objectif 20 : savoir expliciter un invariant de boucle
- Permet de prouver la correction d'un algorithme
- A mettre en relation avec l'analyse de la complexité de l'algorithme

Synthèse de la séance

Des notions nouvelles et importantes

Une sensibilisation aux notions théoriques importantes

1. terminaison : variants de boucles ... facile

2. correction d'un algorithme itératif : invariant de boucles

- objectif 10 : savoir les prouver
- objectif 20 : savoir les identifier et les prouver