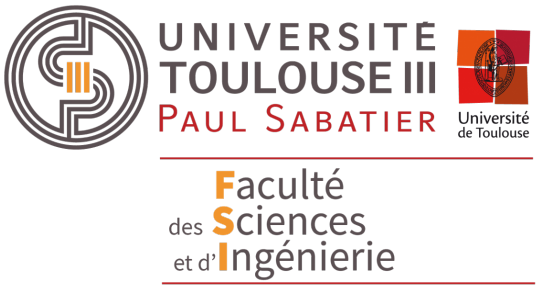
Méthodologie: Spécification

Équipe pédagogique du L2 Algorithmique

L2 Informatique, Université de Toulouse



D[i] | Département Informatique

Plan

Introduction

- Propriétés d'un algorithme
- Programmation impérative
- Démarche Méthodologique
- Analyser, comprendre : objectif
- Spécification du programme
- Modèle de Solution, Complexité, Stratégie et développement avec vérification (Chapitres suivants)
- Exercices de spécification

Propriétés d'un Algorithme

Donald Knuth lista les cinq propriétés suivantes comme étant les prérequis d'un algorithme :

1. *Finitude* : un algorithme doit toujours se terminer après un nombre fini d'étapes.
2. *Définition précise* : chaque étape d'un algorithme doit être définie précisément, les actions à transposer doivent être spécifiées rigoureusement et sans ambiguïté pour chaque cas.
3. *Entrées* : des quantités qui lui sont données avant qu'un algorithme ne commence. Ces entrées sont prises dans un ensemble d'objets spécifié.
4. *Sorties* : des quantités ayant une relation spécifiée avec les entrées.
5. *Rendement* : toutes les opérations que l'algorithme doit accomplir doivent être suffisamment basiques pour pouvoir être en principe réalisées dans une durée finie par un homme utilisant un papier et un crayon.

Programmation Impérative

Les notions, selon l'approche de N. Wirth pour les langages dits impératifs les plus répandus (Pascal, ADA C, etc.), appartiennent à deux classes :

- les structures de contrôle
  - séquence
  - conditionnelle
  - boucle
- les structures de données
  - constantes
  - variables
  - tableaux
  - structures récursives (listes, arbres, graphes) étudiées en S4

Démarche Méthodologique

Consiste à développer un programme avec vérification

1. Comprendre le problème : l'analyser et modéliser ses entrées et sorties.
2. Spécifier le programme à développer à l'aide d'un triplet de Hoare :

```
/* Precondition*/
Abstraction du programme (entrees, sorties)
/*Postcondition*/
```

3.
   - Proposer des modèles abstraits de solution selon différentes stratégies,
   - Choisir parmi eux le modèle qui permettra de répondre autant que possible aux exigences de performance.
   - Le choix d'une solution se fait en fonction de sa complexité en temps (nombre d'étapes), en espace mémoire...
4. Implémenter avec vérification le modèle de solution choisi et effectuer des mesures de performance.

Implémenter avec vérification

- Construction de programme impératif par contrat (spécification en triplet).
- Sa vérification se fait au fur et à mesure de sa construction.

Il s'agit de traiter les programmes comme des objets mathématiques à part entière. Une activité qui a été identifiée, dès le départ, par Turing à travers la notion d'assertion qui associe un prédicat logique à un point de contrôle du programme (pré- et postcondition associées à un morceau de code). Deux étapes ont été identifiées :

1. **Correction Partielle** Écrire un algorithme dit *correct*, c'est-à-dire satisfaisant les assertions qu'il contient, sans tenir compte de sa terminaison.
2. **Terminaison** Pour vérifier la terminaison d'une boucle après la vérification de sa correction partielle, on cherche à construire un ordre bien fondé [1] associé à la boucle qui ne pourra donc pas s'exécuter indéfiniment.

1. n'admettant pas de chaîne infinie strictement décroissante

Implémenter avec vérification

Cette approche a été perfectionnée par Floyd, Hoare et Dijkstra (triplet de Hoare, Invariant de boucle, plus faible précondition ou **wp**[2]).

1. contrat : P et Q sont des formules

```
/* P*/
morceau de code
/*Q*/
```

2. Correction Partielle : vérifier que la formule suivante est $\top$:

$$P \rightarrow \text{wp("morceau de code", Q)}$$

3. Terminaison : si "morceau de code" est une boucle, vérifiez sa terminaison

Des assistants de preuves ont été construits : **FramaC**, l'outil de la méthode **B** etc...

- **autres approches** :réécriture de termes pour l'optimisation de programmes et leur vérification formelle (J. McCarthy).

2. weakest precondition

Analyser, comprendre : objectif

Nous sommes dans une démarche déclarative. Il s'agit d'identifier avec précision, à partir d'un problème calculable, ses différentes propriétés. D'une façon pratique :

- identifier le domaine du problème
- identifier les données du problèmes et décrire leurs propriétés
- identifier l'objectif du problème en décrivant les propriétés des résultats attendus
- Souvent pour comprendre le problème nous avons besoin de l'étudier au travers d'exemples.
- Nous cherchons les cas limites ainsi que par exemple, pour des problèmes de décision, les cas où la réponse et $\top$ et les cas où la réponse est $\bot$. Le choix des exemples est souvent problématique et insuffisant et dépend de l'analyse du problème, mais permettra d'approcher les propriétés de l'objectif.

Exemple

Ecrire un programme qui calcule $N!$

- Domaine : arithmétique
- Données : $N!$ est défini pour N naturel
  - 1 si $N = 0$ et $N * (N - 1)!$ si $N > 0$
- Entrée : N doit être une entrée de taille contrainte par la plateforme
- Résultat : $N!$ (si la valeur est représentable)

Plan

Spécification du programme

Il s'agit de décrire avec précision les propriétés identifiées dans l'étape précédente. Pour ce faire, on utilise un langage mathématique fondé sur la logique classique, l'arithmétique et la théorie des ensembles : les **triplets de Hoare**.

```
/* Precondition*/
 Action(E,s)
/*Postcondition*/
```

Spécification du programme

```
/* Precondition*/
 Action(E,s)
/*Postcondition*/
```

- La *précondition* exprime, à l'aide d'une formule logique, les propriétés que doivent satisfaire les données fournies par l'utilisateur du programme

Spécification du programme

```
/* Precondition*/
 Action(E,s)
/*Postcondition*/
```

- Action (E, s) exprime, dans un langage de programmation, une abstraction de l'algorithme où **E** est sa liste d'entrées et **s** sa liste de sorties.

Convention

- les entrées seront en majuscules,
- et les sorties en minuscules (précédées par & pour le C)

Spécification du programme

```
/* Precondition*/
 Action(E,s)
/*Postcondition*/
```

- La *postcondition* exprime, à l'aide d'une formule logique, les propriétés satisfaites par les variables du programme à la fin de l'exécution de celui-ci.

Spécification du programme

```
/* Precondition*/
 Action(E,s)
/*Postcondition*/
```

- Les pré- et postconditions sont écrits en commentaires dans le code du programme.

Spécification du programme

Le contrat

```
/* Precondition*/
 Action(E,s)
/*Postcondition*/
```

exprime que :

si la précondition est satisfaite, alors Action(E,s) se termine en satisfaisant la postcondition

Rappel sur les quantificateurs : Notations

Nous utilisons les notations suivantes :

- $\top$ pour vrai
- $\bot$ pour faux
- $\emptyset$ pour dénoter un ensemble vide

Quantificateur universel

- $(\forall I : I \in E \rightarrow P(I))$
- Pour tout I, si I appartient à l'ensemble E, alors $P(I)$ est $\top$.
- **Si E est vide, cette formule est** $\top$. $\bot \rightarrow P$ est $\top$ (avec P quelconque).

Exemple

1. $E = \{a, b, c\} : (\forall\ I : I \in E \rightarrow I > 0)$ signifie que $(a > 0 \wedge b > 0 \wedge c > 0)$
2. $E = \{1, 2, 3\} : (\forall\ I : I \in E \rightarrow I > 0)$ est $\top$ car 1, 2 et 3 sont des nombres strictement positifs
3. $E = \emptyset : (\forall\ I : I \in E \rightarrow P(I))$ est $\top$, pour P quelconque, car la prémisse de $I \in E$ est $\bot$

Quantificateur existentiel

- $\exists\ I : I \in E \wedge P(I)$
- Il existe au moins un I dans E qui vérifie $P(I)$.
- **Si $E = \emptyset$, cette formule est $\bot$.** $\exists I : I \in \bot \wedge P(I)$ est $\bot$

Exemple

- $E = \{a, b, c\} : (\exists I : I \in E \wedge I > 0)$ signifie que $(a > 0 \vee b > 0 \vee c > 0)$
- $E = \{-1, 2, 3\} : (\exists I : I \in E \wedge I > 0)$ est $\top$ car $2 \in E$
- $E = \emptyset : (\exists I : I \in E \wedge P(I)$ est $\bot$ car $I \in \emptyset$ est $\bot$

Quantificateur numérique

On se donne un ensemble fini E.

- $(\nu I : I \in E \wedge P(I))$
- Le nombre de I dans E qui vérifient P(I).
- si E est $\emptyset$, $(\nu \mathbf{I} : \mathbf{I} \in \mathbf{E} \wedge \mathbf{P(I)}) = \mathbf{0}$

Exemple

Pour $E = \{1, 2, 3\}$

- $(\nu I : I \in E \wedge I < 3) = 2$
- $(\nu I : I \in E \wedge I > 3) = 0$

Somme et produit

- $N \in \mathbb{N}$
- $\sum_{I=1}^{N} I$ est la somme des valeurs de 1 à N.
- $\sum_{I=1}^{N} I = 0$ si $N < 1$
- Pour un tableau int T[N], la somme des élements du tableau T est exprimée par : $\sum_{I=0}^{N-1} T[I]$

- $N \in \mathbb{N}$
- $\prod_{I=1}^{N} I$ est le produit des valeurs de 1 à N $= N!$.
- $\prod_{I=1}^{N} I = 1$ si $N < 1$
- Pour un tableau int T[N], le produit des élements du tableau T est exprimé par : $\prod_{I=0}^{N-1} T[I]$

Spécification Informelle/Formelle

- **Spécification informelle**

```
/* N est un entier naturel > 1 */
PgPuis2Inf(N, &pgn)
/* (pgn est la plus grande puissance de 2 inférieure à N) */
```

- **spécification formelle**

```
/* N ∈ ℕ ∧ N > 1 */
PgPuis2Inf(N, &pgn)
/* (∃I : I ∈ ℕ ∧ pgn = 2^I ∧ 2^I < N ≤ 2^(I+1)) */
```

Convention

- Constante ou valeur en entrée : **majuscule**
- Variable du programme ou une sortie : **minuscule**.
- Dans les commentaires les types des valeurs et des variables sont vérifiés implicitement.

Spécifier un programme

Exemple

Écrire la spécification d'un programme qui trie, dans l'ordre croissant, un tableau t de N entiers, avec $N \geq 0$.

- t est un tableau
- A représente la liste des valeurs initiales de int $t[N]$
- Nous supposons l'existence d'un prédicat permutation(t,A) qui donne $\top$ si t est une permutation de A, sinon il donne $\bot$

- **Comprendre**
  - l'objectif est que t soit trié dans l'ordre croissant à l'arrêt du programme. La propriété suivante doit dont être satisfaite : **Tout élément du tableau t est $\leq$ à son suivant dans le tableau**.
  - . . . sauf le dernier qui n'a pas de suivant
  - NB : si N=0, càd si t est un tableau vide, alors t est trié
  - Les éléments du tableau sont préservés par le tri

Spécifier un programme

Exemple

- **Spécification informelle**

/* $N \geq 0$, A est un tableau de N entiers, t est égal à A */
trier(t) ;
/* (tout élément de t $\leq$ à son suivant dans t) et (t est une permutation de A*/

- **Spécification formelle**

/* $N \geq 0 \wedge (\textbf{int}\ A[N]) \wedge \forall I : 0 \leq I < N \rightarrow t[I] = A[I]$ */
trier(t) ;
/* $\forall I : 0 \leq I < N - 1 \rightarrow t[I] \leq t[I+1] \wedge \text{permutation}(t, A)$ */

Remarque

La liste A des valeurs initales de t est utilisée seulement dans les assertions

Evaluez les formules suivantes

1.
   - $(\forall I : 0 \leq I < 0 \rightarrow I = I)$
   - $(\forall I : 0 \leq I < 0 \rightarrow I \neq I)$
   - $(\exists I : 0 \leq I < 0 \wedge I = I)$
   - $(\exists I : 0 \leq I \leq 0 \wedge I = I)$
   - $(\nu I : 0 \leq I < 0 \wedge I = I) = ?$
   - $(\nu I : 0 \leq I \leq 0 \wedge I = I) = ?$
   - $\sum_{I=0}^{50} I = ?$
   - $(\forall I : 0 \leq I < 5 \wedge I \neq 2 \rightarrow T[I] > 0)$ avec int T[5]={1, 2, 0, 1, 4}

Evaluez les formules suivantes :

2

- $(\forall X : X \in \emptyset \rightarrow P(X)) \equiv \ ?$
- $(\exists X : X \in \emptyset \wedge P(X)) \equiv \ ?$
- $(\forall X : X \in E \rightarrow P(X) \vee Q(X)) \stackrel{?}{\equiv} (\forall X : X \in E \rightarrow P(X)) \vee (\forall X : X \in E \rightarrow Q(X))$
- $(\forall X : X \in E \rightarrow P(X) \wedge Q(X)) \stackrel{?}{\equiv} (\forall X : X \in E \rightarrow P(X)) \wedge (\forall X : X \in E \rightarrow Q(X))$
- $(\exists X : X \in E \wedge (P(X) \wedge Q(X))) \stackrel{?}{\equiv} (\exists X : X \in E \wedge P(X)) \wedge (\exists X : X \in E \wedge Q(X))$
- $(\exists X : X \in E \wedge (P(X) \vee Q(X))) \stackrel{?}{\equiv} (\exists X : X \in E \wedge P(X)) \vee (\exists X : X \in E \wedge Q(X))$

Evaluez les formules suivantes :

3

- $(\exists X : X \in E_1 \cup E_2 \wedge P(X)) \stackrel{?}{\equiv} (\exists X : X \in E_1 \wedge P(X)) \wedge (\exists X : X \in E_2 \wedge Q(X))$
- $(\exists X : X \in E_1 \cup E_2 \wedge P(X)) \stackrel{?}{\equiv} (\exists X : X \in E_1 \wedge P(X)) \vee (\exists X : X \in E_2 \wedge Q(X))$
- $(\forall X : X \in E_1 \cup E_2 \rightarrow P(X)) \stackrel{?}{\equiv} (\forall X : X \in E_1 \rightarrow P(X)) \vee (\forall X : X \in E_2 \rightarrow Q(X))$
- $(\forall X : X \in E_1 \cup E_2 \rightarrow P(X)) \stackrel{?}{\equiv} (\forall X : X \in E_1 \rightarrow P(X)) \wedge (\forall X : X \in E_2 \rightarrow Q(X))$
- $(\nu X : X \in E_1 \cup E_2 \wedge P(X)) \stackrel{?}{=} (\nu X : X \in E_1 \wedge P(X)) + (\nu X : X \in E_2 \wedge P(X))$

Calculez les expressions suivantes :

4.
- $\Sigma_{i \in \emptyset} f(i) = ?$
- $\Sigma_{i \in I_1 \cup I_2} f(i) \stackrel{?}{=} \Sigma_{i \in I_1} f(i) + \Sigma_{i \in I_2} f(i)$
- $\Sigma_{i \in I} f(i) + g(i) \stackrel{?}{=} \Sigma_{i \in I} f(i) + \Sigma_{i \in I} g(i)$
- $\Sigma_{i \in 1..N} i = ?$
- $\Pi_{i \in \emptyset} f(i) = ?$
- $\Pi_{i \in 1..N} i = ?$
- $\Pi_{i \in I} f(i) \times g(i) \stackrel{?}{=} \Sigma_{i \in I} f(i) \times \Sigma_{i \in I} g(i)$
- $\Pi_{i \in I_1 \cup I_2} f(i) \stackrel{?}{=} \Pi_{i \in I_1} f(i) \times \Pi_{i \in I_2} f(i)$

Ecrire la spécification des programmes suivants :

5. 1. qui reçoit en entrée un tableau non vide de N entiers et retourne dans npos le nombre de cases de ce tableau contenant des valeurs strictement positives.
   2. qui cherche la première position, à partir de 0, du tableau int X[2] dans le tableau int Y[N]. En sachant que Y contient au moins une occurrence de X.
   3. qui calcule la longueur du plus grand plateau dans tableau T, qui n'est pas vide, de N entiers trié dans l'ordre croissant. Un plateau de T est une tranche du tableau T dont les éléments sont tous égaux.
   4. qui reçoit un tableau T de N entiers tous différents, et calcule dans u et v les indices des deux plus grandes valeurs de t telles que : t[u] $<$ t[v].

Ecrire la spécification des programmes suivants :

6
1. qui reçoit un tableau t de $N \geq 0$ éléments parmi (bleu = 0, blanc = 1, rouge = 2). L'idée est de trier ce tableau en le traversant une seule fois. Il faut ranger le tableau t dans l'ordre (bleu, blanc, rouge) en précisant dans une variable i le début des blancs et dans une variable j le début des rouges. Le tableau ne contient pas forcément toutes les couleurs.
2. qui à partir d'un tableau de N entiers int t[N] avec $N > 0$, modifie chaque case t[i], en lui affectant, le nombre de cases t[j], qui la précède si la valeur initiale de t[j] est $<$ à la valeur initiale de t[i].

Entraînement

Soit :
/* 0 <= M <= N */
int T[N] ; int R[M] ;
Formaliser les propriétés suivantes :

1. Tous les éléments de T[N] qui sont impairs sont > 0
2. Tous les éléments d'indice impair de T[N] sont < 0
3. Toutes les sous-séquences à deux éléments sont constituées de deux éléments différents.
4. R est une sous-séquence de T
5. Les éléments de R sont présents dans T (sans prendre en compte l'ordre ou le nombre d'occurrences)
6. Les éléments de R sont présents dans T avec le même nombre d'occurrences
7. Toutes les sous-séquences adjacentes de T sont différentes