BUT Informatique - R3.02
Conception d'algorithmes corrects et efficaces
Spécification et construction des algorithmes

J. Christian Attiogbé

Novembre 2023

Nantes Université

Contenu prévisionnel

Plan de la suite (adaptation selon contraintes)

1. Spécifications - Construction de programmes
2. Triplet de Hoare
3. Terminaison
4. Construction inductive des algorithmes et programmes
5. Construction des itérations/boucles

Introduction

Le couple (précondition, postcondition) est appelé la *spécification* d'un algorithme ou d'un programme.

spécification ≡ ***(précondition, postcondition)***

C'est un **contrat** ;
précondition honorée par l'utilisateur du programme ;
postcondition honorée par le programme.

Des **méthodes rigoureuses** existent pour guider les différentes **étapes de construction** de telle sorte que : étant donné la précondition, le programme établit, lorsqu'il est terminé, la postcondition.

Triplet de Hoare

La logique de Hoare est un outil formel qui permet d'effectuer des **preuves de correction partielle des programmes**.

Le **triplet de Hoare** exprime la **spécification de la correction partielle** d'un programme.

Triplet de Hoare

Un triplet de Hoare est une expression de la forme **{P} S {Q}** où S est un programme et P (précondition) et Q (postcondition) sont des formules logiques (prédicats).

Validité du triplet de Hoare

Triplet valide

Le triplet $\{P\}\ \ S\ \ \{Q\}$ **est valide** (i.e. toujours vrai) si et seulement si **toute exécution finie de** S **commençant par un état initial qui rend vraie** $\{P\}$ se termine par un **état final qui vérifie** $\{Q\}$.

La démonstration de la validité de $\{P\}\ \ S\ \ \{Q\}$ prouve la correction du programme S. La logique de Hoare, ramène toutes les démonstrations à la preuve de la validité d'**une suite d'expressions de la forme** $\{P\}\ \ S\ \ \{Q\}$.

Preuve de validité du triplet de Hoare

Les preuves se font en utilisant des **règles précises de déduction** qui sont notées comme suit :

$$\frac{\{P\}\ \ S\ \ \{Q\}}{\{P'\}\ \ S'\ \ \{Q'\}}$$

Cette notation se lit : si $\{P\}\ \ S\ \ \{Q\}$ est valide alors $\{P'\}\ \ S'\ \ \{Q'\}$ l'est aussi.

Structure d'un programme et validité du triplet

☞ Tout algorithme/programme structuré se ramène aux instructions de base (et leurs équivalences) :
- séquence,
- conditionnelle "Si Alors Sinon" / IF THEN ELSE,
- répétitive "Tantque" / WHILE.

Règles déduction pour chacune des structures de contrôle de base :
séquence,
la conditionnelle "Si Alors Sinon",
la répétitive "Tantque".

☞ Validité/correction de tout programme structuré se ramène aux règles de base de ces structures.

Preuve de validité du triplet de HOARE

Règle de déduction pour l'opération neutre (skip) :

$$\frac{}{\{P\} \quad skip \quad \{P\}}$$

Règle de déduction pour l'opération élémentaire (axiome) : **affectation**

$$\frac{}{\{P[e/v]\} \quad v \leftarrow e \quad \{P\}}$$

Attention à cette règle (axiome)!

Validité du triplet de HOARE

Règle de la pré-condition (**renforcement**)

$$\frac{P' \Rightarrow P \quad \{P\}\ S\ \{Q\}}{\{P'\}\ S\ \{Q\}}$$

Une précondition $note > 10$
est satisfaite par la condition $note > 15$

Validité du triplet de HOARE

Règle de la post-condition (**affaiblissement**)

$$\frac{\{P\}\ S\ \{Q\} \qquad Q \Rightarrow Q'}{\{P\}\ S\ \{Q'\}}$$

Par

exemple, $x > 4 \quad \Rightarrow \quad x > 0$
donc une postcondition $x > 4$ satisfait aussi $x > 0$

Validité du triplet de HOARE

Règle de la **séquence**

$$\frac{\{P\}\ \ S_1\ \ \{Q\} \qquad \{Q\}\ \ S_2\ \ \{R\}}{\{P\}\ \ S_1; S_2\ \ \{R\}}$$

Règle de la **conditionnelle**

$$\frac{\{P \wedge B\}\ \ S_1\ \ \{Q\} \qquad \{P \wedge \neg B\}\ \ S_2\ \ \{Q\}}{\{P\}\ \ \mathit{IF}\ \ (B)\ \ \mathit{THEN}\ \ S_1\ \ \mathit{ELSE}\ \ S_2\ \ \{Q\}}$$

Validité du triplet de HOARE

Règle de la **répétition (tantque)**

$$\frac{\{P \wedge B\}\ \ S_1\ \ \{P\}}{\{P\}\ \ \mathit{WHILE}\ \ (B)\ \ \mathit{DO}\ \ S_1\ \ \mathit{END}\ \ \{P \wedge \neg B\}}$$

B : **Condition** de boucle
P : **Invariant** de boucle (attention ici !)

Exemple - utilisation du triplet de HOARE

Produit(multiplication) de deux entiers par sommes successives

```
Algo multipliAB(a : NAT, b : NAT, p : NAT)
variables     n : NAT // nombre d'additions
début
    {a ≥ 0   ∧   b ≥ 0}
    n := b ; p := 0
    { (p + n × a) = (a × b)}      // ⚠ invariant
    tantque n ≠ 0 faire
        { (p + n × a) = (a × b)   ∧   n ≠ 0 }
        p := p + a ; n := n -1
        { (p + n × a) = (a × b) }     // invariant
    fintantque
    { (p + n × a) = (a × b)   ∧   n = 0 }
finAlgo
```


Prouver la terminaison

La validité du triplet de Hoare $\{P\}\ S\ \{Q\}$ **suppose, mais ne garantit pas que le programme S termine**. Il se peut que le programme s'exécute indéfiniment. $\Longrightarrow$ Obligation de preuve.

En excluant la récursivité et le branchement inconditionnel (instruction goto qui n'a pas sa place dans le contexte des algorithmes structurés) le **non arrêt d'un programme ne peut provenir que d'une répétitive**.

Prouver la terminaison d'un programme revient donc à **prouver que chaque boucle du programme ne peut s'exécuter qu'un nombre fini de fois**.

Prouver la terminaison

Pour prouver la terminaison d'une boucle il faut :
exhiber une suite s, fonction des variables intervenant dans la boucle et vérifiant les propriétés suivantes :

- s prend ses valeurs sur un domaine scalaire muni d'une relation d'ordre ;
- s est strictement monotone (i.e. strictement croissante ou strictement décroissante) ;
- s est **majorée** si elle est croissante,
  minorée si elle est décroissante.

Généralement, s une suite décroissante minorée par zéro.

Exemple - Terminaison

```
Algo multipliAB(a : NAT, b : NAT, p : NAT)
variables     n : NAT // nombre d'additions
début
    {a ≥ 0   ∧   b ≥ 0}
    n := b ; p := 0
    { (p + n × a) = (a × b)}
    tantque n ≠ 0 faire
        { (p + n × a) = (a × b)   ∧   n ≠ 0 }
        p := p + a ; n := n -1
        { (p + n × a) = (a × b) }
        suite décroissante : { n }
    fintantque
    { (p + n × a) = (a × b)   ∧   n = 0 }
finAlgo
```

Construction des algorithmes structurés

Programme séquentiel, impératif : construit à l'aide d'instructions élémentaires et d'instructions composées à l'aide de structures de contrôle. Ces structures guident la construction.

- **Affectation :** affecter une valeur/expression à une variable.
- **Séquence :** enchaînement (souvent noté ";") de deux/plusieurs instructions élémentaires ou non.
- **Conditionnelle :** faire dépendre l'exécution d'un bloc d'instructions T1 / T2, d'une condition C (un prédicat) ;
- **Itération :** répéter/itérer une suite d'instructions T aussi longtemps qu'une certaine condition donnée/voulue C est vraie. (Tantque C faire T finfaire).

☞ Parmi ces instructions et structures de contrôle, la construction des itérations mérite une attention particulière.

Construction des boucles

Une boucle dépend de sa spécification (précondition, postcondition).

La **construction de boucle est systématisée par** les constituants :

1. Invariant (de boucle) : prédicat représentant une propriété caractéristique du problème traité (ici par la boucle).
2. Condition d'arrêt : également un prédicat, qui exprime la condition d'arrêt de la boucle
3. Progression : une action/instruction ou un bloc d'actions.
4. Initialisation : une action/instruction ou un bloc d'actions, effectué une seule fois au début.
5. Expression de Terminaison : une expression à valeur entière (qui va croître tout le temps ou bien décroître tout le temps).

Exemple - Boucle+Terminaison

```
Algo multipliAB(a : NAT, b : NAT, p : NAT)
variables     n : NAT // nombre d'additions
début
    {a ≥ 0   ∧   b ≥ 0}
    n := b ; p := 0
    Invariant : { (p + n × a) = (a × b)}
    tantque n ≠ 0 faire
        { (p + n × a) = (a × b)   ∧   n ≠ 0 }
        p := p + a ; n := n -1
        Invariant : { (p + n × a) = (a × b) }
        Expr Terminaison : n
    fintantque
    { Invariant: (p + n × a) = (a × b) } ∧ n = 0}
finAlgo
```


Relations entre les constituants d'une boucle

R1 La conjonction de l'invariant et de la condition d'arrêt conduit au but recherché par le programme :

(**Invariant** et condition d'arrêt) ⇒ post-condition

R2 La progression doit

1. **conserver l'invariant**. Avant l'exécution de la progression, le prédicat invariant et non condition d'arrêt est vrai. **Avant et après l'exécution de la progression, l'invariant doit être vrai**.
2. faire décroître strictement l'expression de terminaison. Ainsi à chaque tour de boucle, l'expression est positive ou nulle, et strictement inférieure à sa valeur précédente ; la condition d'arrêt sera donc atteinte au bout d'un temps fini.

R3 L'initialisation doit instaurer l'invariant. Notons que la pré-condition de l'initialisation est la pré-condition du programme et sa post-condition est l'**invariant de la boucle**.

Relations entre les constituants d'une boucle

A travers **R1, R2, R3**, on note que l'invariant est un constituant fondamental pour la conception/construction des boucles.

Il est raisonnable de commencer la construction de boucle par l'identification de l'invariant.