Mémo n. 2

Sémantique des instructions

I32 Preuves et Analyses d'algorithmes

P. Véron

But : Associer aux variables de l'algorithme certaines propriétés et étudier l'évolution de ces propriétés après chaque instruction afin de controler le bon fonctionnement de l'algorithme.

1. Assertion, Pré-assertion, Post-assertion

Définition : *Une assertion est une expression logique liant certaines variables d'un algorithme entre elles ou à des éléments de leurs domaines respectifs.*

Exemple : Si a et b sont des entiers, alors $\{a \leqslant b\}$ et $\{a \geqslant 3\}$ sont des assertions.

Définition : *Une pré-assertion est une assertion qui précède une instruction. Elle précise l'environnement dans lequel on se trouve et les hypothèses sur les variables avant l'exécution de l'instruction. Si aucune hypothèse ne peut être faite, on utilise l'assertion vide notée* $\{\}$.

Exemple : Soient les instructions suivantes :

$a \leftarrow 3$
$b \leftarrow 5$
$a \leftarrow a + b$

alors $\{a = 3, b = 5\}$ est une pré-assertion de l'instruction $a \leftarrow a + b$.

Définition : *Une post-assertion est une assertion qui suit une instruction. Elle précise l'effet de l'instruction sur l'environnement courant de l'algorithme. C'est en général la dernière post-assertion de l'algorithme qui détermine le résultat de l'algorithme.*

Exemple : En considérant les instructions précédentes $\{a = 8\}$ est une post-assertion de l'instruction $a \leftarrow a + b$.

Définition : Un schéma pré-post assertion est formé d'une pré-assertion $\{A\}$, d'une instruction I et d'une post-assertion $\{B\}$, on note $\{A\}I\{B\}$. Ce schéma est dit correct si $\{B\}$ est une conséquence de $\{A\}$ après l'instruction I.

Exemple : $\{i < n\}i \leftarrow i + 1\{i \leqslant n\}$ est un schéma correct.

2. Schéma associé à l'affectation

Définition : *Soit* $\{A\}$ *une pré-assertion de l'instruction* $x \leftarrow E$ *(*E *une expression), alors* $\{A\}x \leftarrow E\{B\}$ *est un schéma correct si* B *est une post-assertion déduite de* A *en remplaçant chaque occurence de* E *par* x.

Exemple : $\{S = nx\}x \leftarrow x + 1\{S = n(x - 1)\}$ est un schéma correct. En effet $\{S = nx\} \Rightarrow \{S = n(x + 1) - n\}$ l'assertion $\{S = n(x - 1)\}$ est alors obtenue en remplaçant $x + 1$ par x.

3. Schéma associé à un choix simple

Définition : Soit l'instruction I suivante :

si C **alors**
S

finsi

et $\{A\}$ une pré-assertion, alors $\{A\}I\{B\}$ est correct ssi :
- $\{A$ et $C\}$ S $\{B\}$ est un schéma correct,
- $\{\neg C$ et $A\} \Rightarrow B$.

Exemple : Soit l'instruction I suivante :

si $a < 0$ **alors**
$a \leftarrow -a$
finsi

Le schéma $\{a \in \mathbb{Z}\}I\{a \geqslant 0\}$ est correct. En effet :
- $\{a \in \mathbb{Z}$ et $a < 0\} \Rightarrow \{-a \geqslant 0\}$ $a \leftarrow -a$ $\{a \geqslant 0\}$,
- $\{a \in \mathbb{Z}$ et $a \geqslant 0\} \Rightarrow \{a \geqslant 0\}$

4. Schéma associé à un choix avec alternative

Définition : *Soit l'instruction I suivante :*

si C **alors**
S
sinon
S'
finsi

et $\{A\}$ une pré-assertion, alors $\{A\}I\{B\}$ est correct ssi :
- $\{A$ et $C\}$ S $\{B\}$ est un schéma correct,
- $\{A$ et $\neg C\}$ S' $\{B\}$ est un schéma correct.

Exemple : Soit l'instruction I suivante :

si $a > b$ **alors**
$u \leftarrow a$
sinon
$u \leftarrow b$
finsi

Le schéma $\{a = \alpha \in \mathbb{N}, b = \beta \in \mathbb{N}\}$ I $\{u = \max(\alpha, \beta)\}$ est correct. En effet :
- $\{a = \alpha \in \mathbb{N}, b = \beta \in \mathbb{N}$ et $a > b\} \Rightarrow \{a = \max(\alpha, \beta)\}$ $u \leftarrow a$ $\{u = \max(\alpha, \beta)\}$,
- $\{a = \alpha \in \mathbb{N}, b = \beta \in \mathbb{N}$ et $a \leqslant b\} \Rightarrow \{b = \max(\alpha, \beta)\}$ $u \leftarrow b$ $\{u = \max(\alpha, \beta)\}$.

5. Schéma associé à l'itération

Définition : *Soit l'instruction I*

tant que C **faire**
J
fintq

et $\{A\}$ une pré-assertion. Si $\{A$ et $C\}$ J $\{A\}$ est un schéma correct, alors A est appelé invariant de boucle. Le schéma $\{A\}$ I $\{A\}$ est alors correct.

Définition : *Soit I l'instruction :*

tant que C **faire**
J
fintq

et $\{A\}$ un invariant de boucle, on dit que $\{A\}$ I $\{A$ et $\neg C\}$ est partiellement correct.

6. Cas particulier de l'instruction `lire`

Soit $\mathcal{D}$ le domaine de définition de la variable v, après l'instruction `lire(v)` on a la post-assertion $\{v = \alpha, \alpha \in \mathcal{D}\}$

7. Preuve d'arrêt

Soit I l'instruction :

tant que C **faire**
J
fintq

et $\{A\}$ un invariant de boucle, on dit que $\{A\}\ I\ \{A$ et $\neg C\}$ est **partiellement correct**. Pourquoi utiliser une telle terminologie ? La raison en est simple : A étant un invariant de boucle, par définition il est encore vrai après la sortie de la boucle, par contre la proposition $\neg C$ n'est vraie qu'à la condition que le test de boucle C soit faux, i.e que la boucle s'arrête. Ainsi si on démontre que la boucle se termine alors le schéma $\{A\}\ I\ \{A$ et $\neg C\}$ sera **correct**.
Exemple :

```
1 Algo infini
2 données n : entiers
3   début
4     n ← 1
5     tant que n ≠ 0 faire
6                n ← 2n
7     fintq
8     ecrire(n)
9   fin
```

Sur cet exemple simple, il est facile de voir que l'algorithme ne s'arrête jamais. On peut montrer sans difficulté que $\{n \geqslant 0\}$ est un invariant de boucle. On comprend alors mieux le sens de l'énoncé :
$\{n \geqslant 0\}$ `Tant que ... fintq` $\{n \geqslant 0$ et $n = 0\}$ *est un schéma partiellement correct.*
En effet si ce dernier était correct nous aurions démontrer qu'en sortie de boucle $n = 0$!
En fait l'énoncé doit être interprété de la façon suivante : *si la boucle s'arrête alors on a en sortie de boucle* $\{n \geqslant 0$ et $n = 0\}$. On voit donc qu'il est indispensable de faire une preuve d'une terminaison de boucle pour réaliser une preuve de programme.

Prouver la terminaison d'une boucle `Tant que C faire J fintq` n'est pas un travail facile. Dans cette introduction à la preuve de programmes, nous nous contenterons d'étudier des algorithmes simples dont nous prouverons la terminaison en exhibant une expression entière m (dite *variant de boucle*) telle que :

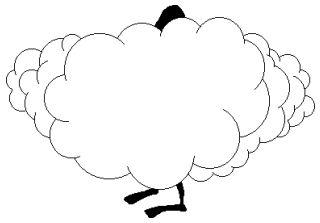
a) $\{m \geqslant 0\}$ soit un invariant de la boucle `Tant que C faire J fintq`,
b) $\{m = m_0$ et $C\}\ J\ \{m < m_0\}$

Le point a) revient à démontrer que m est une quantité toujours positive, i.e. le signe de m n'est pas affecté par l'exécution de la boucle. Le point b) signifie que, si à une étape donnée, la valeur de m est m_0, alors après un passage dans la boucle la nouvelle valeur de m est strictement inférieure à m_0. Autrement dit, si on observe la suite des valeurs prises par m, on obtient une suite strictement décroissante. Ainsi si la boucle était infinie, au bout d'un moment m devrait être négatif ! Or d'après a) m reste toujours positif, ceci signifie que la boucle s'arrête à une certaine étape.

8. Un exemple concret

Soit l'algorithme suivant :

```
1  Algo somme
2  données i, s, n : entiers positifs
3    début
4      lire(n)
5      i ← 1
6      s ← 0
7      tant que i ⩽ n faire
8                  s ← s + i
9                  i ← i + 1
10     fintq
11     ecrire(s)
12   fin
```



Pour démontrer que cet algorithme calcule la somme des n premiers entiers, montrons que $\{s = \sum_{j=1}^{i-1} j \text{ et } 0 \leqslant i \leqslant n+1\}$ est un invariant de boucle.

. Avant la boucle , l'assertion est vraie car $i = 1$ et $s = 0$ et $n \in \mathbb{N}$.
. Voici ensuite les différentes assertions obtenues après exécution de la $i^{\text{ème}}$ ligne :

(7) $\{s = \sum_{j=1}^{i-1} j \text{ et } 0 \leqslant i \leqslant n+1\}$ et $\{0 \leqslant i \leqslant n\} \Rightarrow \{s + i = \sum_{j=1}^{i} j \text{ et } 0 \leqslant i \leqslant n\}$

(8) $\{s = \sum_{j=1}^{i} j \text{ et } 0 \leqslant i \leqslant n\} \Rightarrow \{s = \sum_{j=1}^{i+1-1} j \text{ et } 0 \leqslant i+1 \leqslant n+1\}$

(9) $\{s = \sum_{j=1}^{i-1} j \text{ et } 0 \leqslant i \leqslant n+1\}$

$\{s = \sum_{j=1}^{i-1} j \text{ et } 0 \leqslant i \leqslant n+1\}$ est donc un invariant de boucle.

En sortant de la boucle on a alors :

$\{s = \sum_{j=1}^{i-1} j \text{ et } 0 \leqslant i \leqslant n+1 \text{ et } i > n\}$, soit $s = \sum_{j=1}^{n} j$.

Cette dernière affirmation n'est valable que si l'on démontre la terminaison de la boucle. On cherche donc une valeur positive qui décroit strictement. Le compteur de boucle i croit strictement vers la valeur $n+1$. On en déduit donc que $n+1-i$ est certainement la valeur recherchée.

• Montrons que $\{n + 1 - i \geqslant 0\}$ est un invariant de boucle. Avant la boucle l'assertion est vraie car $n \geqslant 0$. Voici ensuite les différentes assertions obtenues après exécution de la $i^{\text{ème}}$ ligne :

(7) $\{n + 1 - i \geqslant 0 \text{ et } i \leqslant n\} \Rightarrow \{n + 1 - i \geqslant 1\} \Rightarrow \{n + 1 - (i + 1) \geqslant 0\}$

(8) $\{n + 1 - (i + 1) \geqslant 0\}$ (car l'assertion est indépendante de l'instruction)

(9) $\{n + 1 - i \geqslant 0\}$

$\{n + 1 - i \geqslant 0\}$ est donc un invariant de boucle.

• Montrons que $\{n + 1 - i = n_0 \text{ et } i \leqslant n\} s \leftarrow s + i \quad i \leftarrow i + 1 \; \{n + 1 - i < n_0\}$.

$$\begin{array}{lll} \{n+1-i=n_0 \text{ et } i \leqslant n\} & \Rightarrow & \{n+1-(i+1)<n_0\} \\ & & s \leftarrow s+i \\ & & \{n+1-(i+1)<n_0\} \\ & & i \leftarrow i+1 \\ & & \{n+1-i<n_0\} \end{array}$$

Ceci termine la preuve d'arrêt et la preuve totale de l'algorithme `somme`.