Examen programmation rigoureuse

Pierre Courtieu

6 février 2008
2h30 – Documents autorisés

1 Sémantique - 2 points

Construire un arbre de dérivation pour la configuration suivante où $\sigma(b) = 3, \sigma(n) = 2$:

$$\langle \mathtt{x} := \mathtt{1}; \mathtt{while}\ (\mathtt{1} \leq \mathtt{n})\ \mathtt{do}\ (\mathtt{x} := \mathtt{x} * \mathtt{b}; \mathtt{n} := \mathtt{n} - \mathtt{1}), \sigma \rangle$$

2 Sémantique - 2 points

Montrer un sens de l'équivalence $I_1;(I_2;I_3) \sim (I_1;I_2);I_3$

3 Sémantique - 3 points

Les règles de syntaxe suivante permettent la définition de variables avec leurs valeurs initiales :

$$D_V \quad ::= \quad \mathtt{var}\, x\ := a\ ;\ D_V \quad | \quad \varepsilon$$

où D_V est une déclaration de variable, x correspond à un identificateur quelconque, a est une expression arithmétique, et ε correspond à la déclaration vide. Par exemple, la déclaration suivante est valide dans cette syntaxe (**Remarque** : notez que la valeur initiale d'une variable peu dépendre des variables précédemment déclarées.) :

```
var y:= 2; var z:= y+1; var w:= y*z;   (A)
```

La règle suivante permet de donner une sémantique aux programmes formés par les déclarations dans cette syntaxe.

$$\frac{< a, s > \rightsquigarrow v \qquad < D_V, s[x \mapsto v] > \rightarrow s'}{< \mathtt{var}\ \mathtt{x} := \mathtt{a}\ ; D_V, s > \rightarrow s'} \quad (1)$$

où la notation $s[x \mapsto v]$ signifie "s étendu pas la liaison $x \mapsto v$", et $< a, s > \rightsquigarrow v$ est un jugement de la sémantique pour les expressions arithmétiques vue en cours.

1. Donnez un arbre de dérivation pour la suite des déclarations données dans (A), en partant d'un état initial σ_0.
2. Modifiez la règle (1) de manière à interdire les déclarations multiples d'un même nom de variable. En particulier, vous devrez ajouter une ou plusieurs règles pour ce cas d'erreur (et seulement celui-ci). Ainsi, la déclaration suivante doit donner lieu à une erreur (s'évaluer dans `err`) :

   ```
   var y:= 2; var z:= y+1; var y:= 3;
   ```

4 Preuve de programme - 3 points

Démontrez les triplets de Hoare suivants en construisant un arbre de preuve (il s'agit de correction partielle) :

1. *{}* **if** x<=0 **then** y:=x-1 **else** y:=x+1 **end** *{ |y| > |x| }* où *|x|* signifie *valeur absolue de x*.

Solution:

$$\textsc{If}\ \dfrac{\textsc{Conseq}\ \dfrac{\textsc{Aff}\ \dfrac{}{\{x<=0\}\mathtt{y:=x-1}\{x<=0\wedge y=x-1\}}}{\{x<=0\}\mathtt{y:=x-1}\{|y|>|x|\}} \quad \dfrac{\dfrac{}{\{x>0\}\mathtt{y:=x+1}\{x>0\wedge y=x+1\}}\ \textsc{Aff}}{\{x>0\}\mathtt{y:=x+1}\{|y|>|x|\}}\ \textsc{Conseq}}{\{\}\mathbf{if}\ \mathtt{x<=0}\ \mathbf{then}\ \mathtt{y:=x-1}\ \mathbf{else}\ \mathtt{y:=x+1}\ \mathbf{end}\{|y|>|x|\}}$$

2. *{x>0}* **if** x<=0 **then** y:=x-1 **else** y:=x+1 **end** *{ y > x }*

Solution:

$$\textsc{If}\ \dfrac{\textsc{Conseq}\ \dfrac{\textsc{Aff}\ \dfrac{}{\{false\}\mathtt{y:=x-1}\{false\}}}{\{x>0\wedge x<=0\}\mathtt{y:=x-1}\{y>x\}} \quad \dfrac{\dfrac{}{\{true\}\mathtt{y:=x+1}\{y=x+1\}}\ \textsc{Aff}}{\{x>0\wedge x>0\}\mathtt{y:=x+1}\{y>x\}}\ \textsc{Conseq}}{\{x>0\}\mathbf{if}\ \mathtt{x<=0}\ \mathbf{then}\ \mathtt{y:=x-1}\ \mathbf{else}\ \mathtt{y:=x+1}\ \mathbf{end}\{y>x\}}$$

3. *{y=x ∧ x>0}* **while** x<0 **do** x:=x+1 **done** *{ y=x }*

Solution:

$$\textsc{Conseq}\ \dfrac{\dfrac{\textsc{Conseq}\ \dfrac{\textsc{Aff}\ \dfrac{}{\{false\}x:=x+1\{false\}}}{\{(y=x\wedge x>0)\wedge x<0\}x:=x+1\{(y=x\wedge x>0)\}}}{\{(y=x\wedge x>0)\}\mathbf{while}\ \mathtt{x<0}\ \mathbf{do}\ \mathtt{x:=x+1}\ \mathbf{done}\{(y=x\wedge x>0)\}}\ \textsc{While}}{\{y=x\wedge x>0\}\mathbf{while}\ \mathtt{x<0}\ \mathbf{do}\ \mathtt{x:=x+1}\ \mathbf{done}\{y=x\}}$$

5 Preuve de programme - 3 points

Dans cet exercice, l'instruction : **for** i = e1 **to** e2 **do** P **done** est équivalente à :
i=e1; **while** i<=e2 **do** P; i:=i+1 **done**.

1. Proposez une règle de Hoare (correction partielle) pour cette instruction sans la transformer en un **while**.

Solution:

$$\textsc{For}\ \dfrac{\{I\wedge \mathtt{i}\le \mathtt{e2}\}\ \mathtt{P;i:=i+1}\ \{I\}}{\{I\}\ \mathbf{for}\ \mathtt{i\ =\ e1}\ \mathbf{to}\ \mathtt{e2}\ \mathbf{do}\ \mathtt{P}\ \mathbf{done}\ \{I\wedge \mathtt{i}>\mathtt{e2}\}}$$

2. Proposez une règle de correction totale pour cette même instruction. Notez qu'à priori la variable `i` peut être modifiée par le programme `P`, ainsi que la valeur des expressions `e1` et `e2`. (Remarque : plusieurs solutions sont possibles).

Solution:

Il faut contrôler que la variable `i` se rapproche bien de `e2` après l'incrémentation implicite de fin de boucle :

$$\textsc{For}\ \frac{\langle I \wedge i <= e2 \wedge e2 - i = V_0 \wedge V_0 >= 0\rangle\ P\texttt{;}\ i\texttt{:=}i\texttt{+}1\ \langle I \wedge e2 - i < V_0\rangle}{\langle I\rangle\ \textbf{for}\ i\ =\ e1\ \textbf{to}\ e2\ \textbf{do}\ P\ \textbf{done}\ \langle I \wedge i > e2\rangle}$$

6 Preuve de programme - 7 points

Dans cet exercice on suppose une bibliothèque sur la structure de donnée `ensemble`. Vous pouvez utiliser dans les annotations les notations suivantes sur les ensembles : $x \in$ e et $|$e$|$. À titre d'exemple, voici deux annotations (sans rapport avec les exercices) autorisées :

- ($\forall x \in$ e$, x$ est pair) signifie que tout x appartenant à l'ensemble e est pair,
- $|$e$| \leq |$e$'|$ signifie que le nombre d'éléments de `e` est inférieur ou égal à celui de `e'`.

  Par ailleurs vous pouvez utiliser les connecteurs logiques habituels : $\wedge, \vee, \forall, \exists \ldots$

  Enfin, Les fonctions suivantes sont supposées déjà définies et correctes : `appartient(x,e)` qui retourne **`true`** si `x` est dans l'ensemble `e` et **`false`** sinon, `vide()` qui retourne l'ensemble vide, `choisir(e)` qui retourne un élément quelconque de `e`, `retire(x,e)` qui retourne l'ensemble `e` privé de `x` (`e` n'est pas modifié), `ajoute(x,e)` qui retourne l'ensemble `e` auquel est ajouté `x` (idem).

Le programme `sousEnsemble` suivant construit (dans `res`) le sous-ensemble `res` de `e` contenant tous les éléments de `e` plus petits que l'entier `v`.

```
res := vide();
aux := e;
while not (aux = vide()) do
  x := choisir(aux);
  if x <= v then res := ajoute (x , res);
  aux := retire (x , aux);
done
```

1. Annotez ce programme avec les pré- et post-conditions exprimant que la fonction se comporte comme énoncé.

Solution:

$\{\ \textbf{true}\ \}\ \mathit{sousEnsemble}\ \{\ \forall n \in \mathbb{N},\ (n \in e \wedge n \leq v) \leftrightarrow n \in res\ \}$

2. Trouvez un variant et un invariant de boucle suffisants pour prouver que le triplet de Hoare ainsi formé est correct (correction total donc). Il ne vous est pas demandé de construire l'arbre de preuve.

Solution:

Variant : $|aux|$.

Invariant : Les éléments déjà traités ($n \in e\ \wedge\ n \notin aux$) et inférieurs à `v` sont dans `res` :

$\forall n \in \mathbb{N},\ (n \in e\ \wedge\ n \leq v\ \wedge\ n \notin aux)\ \leftrightarrow\ n \in res$

3. Écrivez un programme qui calcule (dans une variable `res`) l'intersection de *trois* ensembles d'entiers. Annotez-le avec pré-condition, post-condition, variant(s) et invariant(s) de boucle(s). Vous pouvez utilisez la syntaxe C (`Caduceus`) si vous voulez.

Solution:

(*Invariant meilleur :* $res = (e1 \cap e2 \cap e3) \cup aux$)

```
{ }
res := e1;
aux := e1;
while not (aux = vide()) do {variant |aux|
        invariant ∀x∈ℕ, (x∈ e1 ∧ x ∈ e2 ∧ x ∈ e3 ∧ x∉aux) ↔ x∈res }
  x := choisir(aux);
  if not (appartient(x, e2)) or not (appartient(x,e3))
  then res := retire (x , res);
  aux := retire (x , aux);
done
{ ∀x∈ℕ,x∈ res ↔ x∈ e1 ∧ x ∈ e2 ∧ x ∈ e3}
```

4. Proposez des triplets de Hoare exprimant que les fonctions auxiliaires `vide()`, `choisir(e)`, `retire(x,e)` et `ajoute(x,e)` sont correctes. Vous utiliserez `\result` pour désigner la valeur de retour de la fonction. Par exemple un triplet possible pour `plus` serait : $\{\}$`plus(x,y)`$\{\backslash result = x+y\}$.

Solution:

- $\{\}$ *vide()* $\{\forall\ x \in \mathbb{N}\ ,\ x \notin \backslash result\ \}$
- $\{\exists\, x \in e\}$ *choisir(e)* $\{\backslash result\ \in\ e\}$
- $\{\}$ *retire(x,e)* $\{\forall y,\ (y \in e\ \wedge\ y \neq x)\ \leftrightarrow\ (y \in \backslash result)\}$
- $\{\}$ *ajoute(x,e)* $\{\forall y,\ (y \in e\ \vee\ y = x)\ \leftrightarrow\ (y \in \backslash result)\}$

Récapitulatif des règles de Hoare

$$\textsc{Seq}\ \frac{\{P\}\,\mathtt{C}_1\,\{Q\} \qquad \{Q\}\,\mathtt{C}_2\,\{R\}}{\{P\}\,\mathtt{C}_1\ \mathtt{;}\ \mathtt{C}_2\,\{R\}}$$

$$\textsc{Aff1}\ \frac{}{\{P\,[x \leftarrow E\,]\}\,\mathtt{x:=}E\,\{P\}}$$

$$\textsc{Aff}\ \frac{}{\{P\}\quad \mathtt{x:=}E \quad \{P\,[x \leftarrow x_0\,] \ \wedge\ x{=}E\,[x \leftarrow x_0\,]\}}$$

$$\textsc{Cond}\ \frac{\{\,\{P\wedge B\}\quad I_1 \quad \{Q\} \qquad \{P\wedge\neg B\}\quad I_2 \quad \{Q\}}{\{P\}\quad \mathtt{if}\ B\ \mathtt{then}\ I_1\ \mathtt{else}\ I_2 \quad \{Q\}}$$

$$\textsc{Conseq}\ \frac{P \Rightarrow P' \qquad \{P'\}C\{Q'\} \qquad Q' \Rightarrow Q}{\{P\}C\{Q\}}$$

$$\textsc{While}\ \frac{\{P\wedge B\}\quad C \quad \{P\}}{\{P\}\quad \mathtt{while}\ B\ \mathtt{do}\ C\ \mathtt{done} \quad \{P \wedge \neg B\}}$$

Correction totale :

$$\textsc{WhileT1}\ \frac{\forall n \in \mathbb{N}, \qquad P(n+1) \Rightarrow B \qquad \langle P\,(n{+}1)\rangle\ C\ \langle P\,(n)\rangle \qquad P(0) \Rightarrow B}{\langle \exists n \in \mathbb{N}.P(n)\rangle \quad \mathtt{while}\ B\ \mathtt{do}\ C\ \mathtt{done} \quad \langle P(0)\rangle}$$

$$\textsc{WhileT}\ \frac{\langle P\ \wedge\ B\ \wedge\ E = n\ \wedge\ E \geq 0\rangle\quad C \quad \langle\ P\ \wedge\ E\ <\ n\ \wedge\ E \geq 0\ \rangle}{\langle P\rangle \quad \mathtt{while}\ B\ \mathtt{do}\ C\ \mathtt{done} \quad \langle P\wedge\neg B\rangle}$$