Cours Composant
7. Logique de Hoare 2

©2005-2008 Frédéric Peschanski

UPMC Paris Universitas

4 mars 2013

Plan du cours

Logique de Hoare II

- Rappel des règles
- J-While avec boucles et tableaux
- Boucles et conception par contrat
- Logique de Hoare - règles supplémentaires
  - Blocs lexicaux
  - Affectations dans des tableaux
  - Boucles

Rappels (I)

Triplet de Hoare

{P} prog {Q}

Axiome d'affecation

$$\frac{}{\{Q[\mathrm{expr}/V]\}\ \mathtt{V} = \mathtt{expr}\ \{Q\}}\ (\mathit{aff})$$

Règle de séquencement

$$\frac{\{P\}\ \mathtt{C}_1\ \{Q_1\} \quad \{Q_1\}\ \mathtt{C}_2\ \{Q_2\} \quad \ldots \quad \{Q_{n-1}\}\ \mathtt{C}_n\ \{Q\}}{\{P\}\ \mathtt{C}_1; \ldots; \mathtt{C}_n\ \{Q\}}\ (\mathit{seq})$$

Rappels (II)

Règles du modus-ponens

$$\frac{P \implies P' \quad \{P'\}\ \mathtt{c}\ \{Q'\} \quad Q' \implies Q}{\{P\}\ \mathtt{c}\ \{Q\}}\ (mp)$$

$$\frac{P \implies P' \quad \{P'\}\ \mathtt{c}\ \{Q\}}{\{P\}\ \mathtt{c}\ \{Q\}}\ (mp\text{-}pre)$$

$$\frac{\{P\}\ \mathtt{c}\ \{Q'\} \quad Q' \implies Q}{\{P\}\ \mathtt{c}\ \{Q\}}\ (mp\text{-}post)$$

Rappels (III)

Règle de conjonction

$$\frac{\{P_1\}\ \mathtt{C}\ \{Q_1\} \quad \{P_2\}\ \mathtt{C}\ \{Q_2\}}{\{P_1 \wedge P_2\}\ \mathtt{C}\ \{Q_1 \wedge Q_2\}}\ (conj)$$

Règles de disjonction

$$\frac{\{P_1\}\ \mathtt{C}\ \{Q_1\}}{\{P_1 \vee P_2\}\ \mathtt{C}\ \{Q_1 \vee Q_2\}}\ (disj_1) \qquad \frac{\{P_2\}\ \mathtt{C}\ \{Q_2\}}{\{P_1 \vee P_2\}\ \mathtt{C}\ \{Q_1 \vee Q_2\}}\ (disj_2)$$

Rappels (IV)

Règle des alternatives

$$\frac{\{B \wedge P\}\ \mathtt{C}_1\ \{Q\} \quad \{\neg B \wedge P\}\ \mathtt{C}_2\ \{Q\}}{\{P\}\ \mathtt{if(B)}\ \mathtt{C}_1\ \mathtt{else}\ \mathtt{C}_2\ \{Q\}}\ (alt)$$

Technique de preuve

1. Chercher P_1 telle que $\{P_1\}\mathtt{C}_1\{Q\}$
2. Chercher P_2 telle que $\{P_2\}\mathtt{C}_2\{Q\}$
3. La précondition recherchée est $P \stackrel{def}{=} (B \implies P_1) \wedge (\neg B \implies P_2)$

Langage J-While : version très simplifiée de Java

- « Programme » = Corps d'une méthode
- pas d'invocation
- types booléens, entiers et tableaux
- expressions arithmétiques et logiques de base
- instructions : affectations, séquencement, alternatives et boucles while

J-While : extensions

Blocs lexicaux

{ **var** x ;**var** y ; *<corps>* }

J-While : extensions

Blocs lexicaux

{ **var** x ;**var** y ; *<corps>* }

Tableaux

Declaration	**var** t[N]
Accès	x=t[4]+2 // *expression*
Affectation	t[5+3]=x-1 // *instruction*

J-While : extensions

Blocs lexicaux

{ **var** x ;**var** y ; *<corps>* }

Tableaux

Declaration	**var** t[N]
Accès	x=t[4]+2 // *expression*
Affectation	t[5+3]=x-1 // *instruction*

Boucles

while(*<cond>*) *<instr>*

Conception par contrat

- Invariants de classe
- Préconditions et postconditions de méthodes
- Boucles

  Invariant expression logique vraie avant et après chaque tour de boucle

  Variant expression logique montrant une décroissance stricte sur un ordre bien fondé

Exemple

Programme

```
i = 0;
max=Integer.MIN_INT;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
}
```

Précondition tous les éléments du tableau sont des entiers, max vaut Integer.MIN_INT

Postcondition la variable max contient le plus grand de ces éléments

Invariant

Programme

```
i = 0;
max=Integer.MIN_INT;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
}
```

Invariant

Programme

```
i = 0;
max=Integer.MIN_INT;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
}
```

Invariant

- expression logique vraie avant et après chaque tour de boucle
- objectif : caractériser de façon cohérente et complète la sémantique de la boucle $\Rightarrow$ invariant “utile”

Programme

```
i = 0;
max=Integer.MIN_INT;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
}
```

Invariant

- expression logique vraie avant et après chaque tour de boucle
- objectif : caractériser de façon cohérente et complète la sémantique de la boucle ⇒ invariant “utile”

Contre-exemple : *true* “marche” mais n’est pas utile

Invariants de boucles : techniques

Question : comment trouver un bon invariant de boucle ?

Invariants de boucles : techniques

Question : comment trouver un <u>bon</u> invariant de boucle ?
Réponse : réflêchir, imaginer, ruminer, cogiter, méditer ...

Invariants de boucles : techniques

Question : comment trouver un <u>bon</u> invariant de boucle ?
Réponse : réflêchir, imaginer, ruminer, cogiter, méditer . . .
Techniques :

- illumination
- simulation
  - Sélectionner les variables "importantes"
  - Simuler le fonctionnement de la boucle
  - Etudier l'évolution des valeurs de variables
  - Déduire un invariant
- analyse symbolique des préconditions/postconditions (utilisation d'un outil)

Invariant : simulation

Programme

```
i = 0;
max=Integer.MIN_INT;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
}
```

Invariant : simulation

Programme

```
i = 0;
max=Integer.MIN_INT;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
}
```

tour	i	max
1	0	tab[0]
2	1	max(tab[0],tab[1])
3	2	max(tab[0],tab[1],tab[2])
4	3	max(tab[0]...tab[3])
		etc...

Invariant : candidat

tour	i	max
1	0	tab[0]
2	1	max(tab[0],tab[1])
3	2	max(tab[0],tab[1],tab[2])
4	3	max(tab[0]...tab[3])
		etc...

Candidat invariant

Tour de boucle k : max = max(tab[0],...,tab[k-1])

Question : comment trouver un <u>bon</u> variant de boucle ?

Variants de boucles : techniques

Question : comment trouver un <u>bon</u> variant de boucle ?

Réponse : réflêchir, imaginer, ruminer, cogiter, méditer . . . (mais moins)

Variants de boucles : techniques

Question : comment trouver un <u>bon</u> variant de boucle ?

Réponse : réflêchir, imaginer, ruminer, cogiter, méditer . . . (mais moins)

Programme

```
i = 0;
max=Integer.MIN_INT;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
}
```

Remarque : i augmente à chaque tour, donc -i diminue, une borne inf est -tab.length (fixée)

Variants de boucles : techniques

Question : comment trouver un <u>bon</u> variant de boucle ?
Réponse : réflêchir, imaginer, ruminer, cogiter, méditer ... (mais moins)

Programme

```
i = 0;
max=Integer.MIN_INT;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
}
```

Remarque : i augmente à chaque tour, donc -i diminue, une borne inf est -tab.length (fixée)
Candidat : -i ou tab.length-i

En Java : utilisation de **assert** + méthodes de "test embarqué"

Test d'invariant

```
boolean loopInv(tab[],n,max) {
  j=0;max2=Integer.MIN_INT;
  while(j<=n) {
    if(max2>tab[j]) max2=tab[j] else max2=max2;
    j=j+1;
  }
  return max==max2;
}
```

Programme “vérifié”

Programme+assertions

```
//précondition
loopPre(tab);
// exercice i = 0;
max=Integer.MIN_INT;
assert(loopInv(tab,i-1,max)); // invariant avant
variant=tab.length-i;
while(i<tab.length) {
  // traitement
  if(max<=tab[i])
    max=tab[i];
  else noop;
  i++;
  assert(loopInv(tab,i-1,max)); // invariant de tour
  assert(tab.length-i<variant); // variant
  variant = tab.length-i;
}
assert(loopInv(tab,i-1,max)); // invariant après
// postcondition
assert(max = maxTab(tab)); // exercice
```

Question : comment être sûr que les invariants et variants sont corrects (idem pré/postconditions) ?

Contrats et logique de Hoare

Question : comment être sûr que les invariants et variants sont corrects (idem pré/postconditions) ?
Réponse : logique de Hoare

Blocs lexicaux

Règle des blocs lexicaux

$$\frac{\{P\}\ \mathtt{C}\ \{Q\} \quad v_1, \ldots, v_n \notin P \cup Q}{\{P\}\ \{\mathtt{varv_1}; \ldots; \mathtt{varv_n}; \mathtt{C}\}\ \{Q\}}\ (let)$$

Blocs lexicaux

Règle des blocs lexicaux

$$\frac{\{P\}\ \mathtt{C}\ \{Q\} \quad v_1,\ldots,v_n \notin P \cup Q}{\{P\}\ \{\mathtt{var v_1};\ldots;\mathtt{var v_n};\mathtt{C}\}\ \{Q\}}\ (let)$$

Attention : renommage manuel des variables en collision

{ var x ; x=1 ; { var x ; x=2 } z=x }

Blocs lexicaux

Règle des blocs lexicaux

$$\frac{\{P\}\ \mathtt{C}\ \{Q\} \quad v_1, \ldots, v_n \notin P \cup Q}{\{P\}\ \{\mathtt{var} \mathtt{v_1}; \ldots; \mathtt{var} \mathtt{v_n}; \mathtt{C}\}\ \{Q\}}\ (let)$$

Attention : renommage manuel des variables en collision

{ var x ; x=1 ; { var x ; x=2 } z=x }

→ { var x ; x=1 ; { var y ; y=2 } z=x }

Blocs lexicaux

Règle des blocs lexicaux

$$\frac{\{P\}\ \mathtt{C}\ \{Q\} \quad v_1, \ldots, v_n \notin P \cup Q}{\{P\}\ \{\mathtt{varv_1}; \ldots; \mathtt{varv_n}; \mathtt{C}\}\ \{Q\}}\ (let)$$

Attention : renommage manuel des variables en collision
{ var x ; x=1 ; { var x ; x=2 } z=x }
$\rightarrow$ { var x ; x=1 ; { var **y** ; **y**=2 } z=x }

Exercice :
montrer $\{x = a \wedge y = b\}$ { var r ; r=x ; x=y ; y=r} $\{x = b \wedge y = a\}$

Affectation dans les tableaux

Syntaxe : tab[$<expr_1>$]=$<expr_2>$

Affectation dans les tableaux

Syntaxe : tab[<$expr_1$>]=<$expr_2$>

Axiome d'affectation dans un tableau

$$\frac{}{\{Q[tab(expr_1 \leftarrow expr_2)/tab]\}\ \mathtt{tab}[expr_1]{=}expr_2\ \{Q\}}\ (tab)$$

avec

$$\begin{cases} tab(expr_1 \leftarrow expr_2)[expr_1] = expr_2 \\ \forall expr_2 \neq expr_1,\ \ tab(expr_1 \leftarrow expr_2)[expr_3] = tab[expr_3] \end{cases}$$

Exercice montrer :

$\{x \neq y \wedge A[y] = 0\}$ A[x]=1 $\{A[y] = 0 \wedge A[x] = 1\}$

Boucles while

Règle de la boucle while

$$\frac{\{P \wedge S\}C\{P\}}{\{P\}\texttt{while(S) C}\{Q\}}\ (\textit{while})$$

Deux interprétations possibles :

Correction partielle si la boucle se termine et l'hypothèse alors la conclusion

Correction totale on prouve que la boucle se termine et si l'hypothèse alors la conction

Règle de la boucle while

$$\frac{\{P \wedge S\}C\{P\}}{\{P\}\texttt{while(S) C}\{Q\}}\ (\textit{while})$$

Technique (correction partielle) :

1. trouver l'invariant de boucle P
2. prouver $P \wedge \neg S \implies Q$
3. déduire P' la plus faible précondition telle que $\{P'\}\texttt{C}\{P\}$
4. prouver $P \wedge S \implies P'$

   $\Rightarrow$ P est la précondition recherchée

Correction partielle de boucle : Exercice

Exercice Montrer :

$\{x \geq 0\}$
```
y=1 ;z=0 ;
while(z !=x) {
  z=z+1 ;
  y=y*z ;
}
```
$\{y = x!\}$

Exercice Montrer :

$\{x \geq 0\}$
y=1 ;z=0 ;
while(z !=x) {
  z=z+1 ;
  y=y*z ;
}
$\{y = x!\}$

1. trouver l'invariant de boucle P
2. prouver $P \wedge \neg S \implies Q$
3. déduire P' la plus faible précondition telle que $\{P'\}\texttt{C}\{P\}$
4. prouver $P \wedge S \implies P'$
   $\Rightarrow$ P est la précondition recherchée