La logique de Hoare

Colette Johnen
johnen@labri.fr
www.labri.fr/~johnen/

Plan

- Syntaxe
- Système déductifs
- Correction de programme

Introduction (1)

Le but de la logique de Hoare est de formaliser la preuve de la correction des programmes.

Rappelons que notre souhait est de prouver des choses du type :

<pré-condition>
programme
<post-condition>

Signification : si la pré-condition est vraie alors, après exécution du programme, la post-condition est vraie.

Introduction (2)

Pour cela, nous allons utiliser des règles de déduction qui, étape par étape (i.e. instruction par instruction), nous amènerons de la pré-condition à la post-condition.

Les formules (1)

Les formules ont la forme suivante :

{p} S {q}

où **p** et **q** sont deux prédicats (exprimés dans la logique des prédicats du premier ordre, avec **X = {variables du programme}**) et **S** un « extrait de programme »...

Signification intuitive. Si **p** est vrai alors, après exécution de l'extrait de programme **S**, **q** est vraie.

Les formules - Exemples

Des formules vraies de la logique de Hoare :

1. $\{x \geq 0\}$ $x \leftarrow x + 1$; $\{ x > 0 \}$

2. $\{x > 0\}$ $x \leftarrow 5$; $\{ x = 5 \}$

3. $\{(x > 0) \wedge (y \geq 0)\}$ $z \leftarrow y/x$
   $\{(x > 0) \wedge (y \geq 0) \wedge (z \geq 0)\}$

4. **si (x=0) alors** $y \leftarrow 1$;
   sinon $y \leftarrow x$;
   $\{(y \neq 0)\}$

Les formules (2)

Remarque 1 : Pour des raisons de lisibilité, on s'autorise l'écriture de « (x > 0) », plutôt que « sup(x,0) ».

Remarque 2 : On ne cherche pas à exprimer « tout ce qui est vrai », mais seulement ce qui nous intéresse...

Restriction : On ne considèrera que des extraits de programme ne permettant pas la synonymie de variables (pas de pointeurs par exemple). En effet, les règles de la logique de Hoare ne fonctionnent pas dans ce cas. Cette restriction est effective dans l'industrie !

7

Les règles de déduction

Nous allons utiliser des règles de déduction qui auront toutes le format suivant :

```
   prémisses
--------------
  conclusion
```

« si toutes les prémisses sont vraies, la conclusion est vraie » (cf. séquents)

Les prémisses et la conclusion sont des formules de la logique de Hoare.

8

Le système déductif (1)

Règle (a). Axiome

$$\frac{}{\{ p(t) \}\quad x \leftarrow t;\quad \{ p(x) \}}$$

Intuition : Si une propriété est vraie pour **t** alors, après exécution de « **x ← t** », la propriété est vraie pour **x**.

9

Le système déductif (2)

Règle (b). Composition

$$\frac{\{ p \}\ S1\ \{ q \} \quad \text{et} \quad \{ q \}\ S2\ \{ r \}}{\{ p \}\quad S1\ ;\ S2\quad \{ r \}}$$

Intuition : une règle naturelle pour l'enchaînement séquentiel de deux portions de programme...

10

Le système déductif (3)

Règle (c1). Conditionnelle 1

$$\frac{\{(p \wedge b)\}\ S\ \{q\} \quad \text{et} \quad \{((p \wedge \neg b) \Rightarrow q)\}}{\{p\}\quad \text{si b alors S fsi}\quad \{q\}}$$

Intuition : dans tous les cas, le prédicat **q** est vrai.

11

Le système déductif (4)

Règle (c2). Conditionnelle 2

$$\frac{\{(p \wedge b)\}\ S1\ \{q\} \quad \text{et} \quad \{(p \wedge \neg b)\}\ S2\ \{q\}}{\{p\}\quad \text{si b alors S1 sinon S2 fsi}\quad \{q\}}$$

Intuition : Dans tous les cas, le prédicat **q** est vrai.

12

Le système déductif (5)

Règle (d). Conséquence

$$\frac{\{(p \Rightarrow q)\} \text{ et } \{q\}\ S\ \{r\} \text{ et } \{(r \Rightarrow s)\}}{\{p\}\ S\ \{s\}}$$

$$\frac{\{(p \Rightarrow q)\} \text{ et } \{q\}\ S\ \{r\}}{\{p\}\ S\ \{r\}}$$

13

Intérêt de la règle « conséquence »

Nous voulons prouver :

$\{entier(a)\}$
si (a < 0) alors a ← -a; fsi
$\{a \geq 0\}$

selon la règle (c) (conditionnelle), nous avons :

$\{ entier(a) \wedge (a \geq 0) \} - \{ a \geq 0 \}$

OK mais

$\{entier(a) \wedge (a<0)\}$ **a ← -a** $\{a > 0\}$

AIE....

règle (d) : $(a > 0) \Rightarrow (a \geq 0)$, on peut conclure !

14

Boucle (1)

Technique de preuve de la correction partielle (cas des boucles).

Format général :

$\{$ **<pré-condition>** $\}$
Tant que <test> faire
<corps>
Fin tant que
$\{$ **<post-condition>** $\}$

15

Boucle (2)

Technique de preuve de la correction partielle (cas des boucles).

Idée: On exhibe un invariant de boucle, c'est-à-dire un prédicat **p** telle que :

1. **(<pré-condition>** $\Rightarrow$ **p)**
2. **p** est vrai après chaque itération du **<corps>**
3. **((p** $\wedge$ $\neg$**<test>)** $\Rightarrow$ **<post-condition>)**

16

Preuve de Correction - Exemple

pré-condition

$\{ (x \text{ et } y \text{ entiers}) \wedge (x \geq 0) \wedge (y \geq x) \wedge (b = false) \wedge (xx = x) \wedge (yy = y)\}$

boucle

```
tant que ( xx ≠ yy ) faire
    si b alors xx ← xx + 1
    sinon yy ← yy – 1
    fsi
    b ← non b
fin tant que
```

post-condition

$\{ (xx = (x + y) \text{ div } 2) \}$

$((xx+yy+b)=(x+y))$ est l'invariant de boucle.

17

Le système déductif (5)

Règle (e). Invariant de boucle

$$\frac{\{(p \wedge b)\}\ S\ \{p\}}{\{p\}\ \textbf{tant-que b faire S fin-tq}\ \{(p \wedge \neg b)\}}$$

18

Code à vérifier

On veut prouver :

```
{ (val ≥ 0) }
        cpt ← val; res ← 0 ;
        tant que ( cpt > 0 ) faire
            res ← res + val;
            cpt ← cpt – 1 ;
        fin tant que
{ (res = val²) }
```

19

Début

{(val≥0)} cpt ← val
{((cpt≥0) ∧ (cpt=val))} // axiome

{((cpt≥0) ∧ (cpt=val))} res ← 0
{(((res=0) ∧ (cpt≥0)) ∧ (cpt=val))}
// axiome

{(val≥0)} cpt ← val ; res ← 0 ;
{(((res=0) ∧ (cpt≥0)) ∧ (cpt=val))}
// composition

20

Invariant de boucle - A prouver

Il faut prouver que :

1. avant la boucle, le prédicat **p** est vérifié :
   p ≡((res = val.val – val.cpt) ∧(cpt≥0))
2. {(p ∧ b)} S {p}

---?????A PROUVER ????????

{ (res = val.val - val.cpt) ∧ (cpt>0) }
res ← res + val; cpt ← cpt – 1;
{ ((res = val.val – val.cpt) ∧ (cpt≥0))}

21

Invariant de boucle - Avant la boucle

Avant la boucle, le prédicat **p** est vérifié :

```
{ (val≥0) }
  cpt ← val ; res ← 0 ;
{ (((res=0) ∧ (cpt≥0)) ∧ (cpt=val))};
{ ( (((res=0) ∧ (cpt≥0)) ∧ (cpt=val)) ⇒
((res = val.val – val.cpt) ∧ (cpt≥0)) )}
----------------------------------------
{ (val≥0) }
  cpt ← val ; res ← 0 ;
{((res = val.val – val.cpt) ∧(cpt≥0))}
// conséquence
```

22

{(p ∧ b)} S {p} (1)

{ ((res = val.val - val.cpt) ∧ (cpt>0)) }
res ← res + val
{ ((res = val.val - val.cpt + val) ∧ (cpt>0)) }
// axiome

23

{(p ∧ b)} S {p} (2)

```
{ ((res = val.val - val.cpt) ∧ (cpt>0)) }
    res ← res + val
{ ((res = val.val - val.cpt + val) ∧ (cpt>0)) }
{ ((res = val.val - val.cpt + val) ∧ (cpt>0))
          ⇒
  ((res = val.val - val.(cpt -1)) ∧ (cpt>0))) }
-------------------------------------
{ ((res = val.val - val.cpt) ∧ (cpt>0)) }
    res ← res + val
{ ((res = val.val - val.(cpt-1)) ∧ (cpt>0)) }
// conséquence
```

24

{(p ∧ b)} S {p} (3)

```
-----------------------------------------
  {((res = val.val - val.(cpt-1)) ∧ (cpt>0)) }
   cpt ← cpt -1
  { ((res = val.val - val.cpt) ∧ (cpt≥0)) }
// axiome
```

25

{(p ∧ b)} S {p} (4)

```
 { ((res = val.val - val.cpt) ∧ (cpt>0)) }
     res ← res + val
 { ((res = val.val - val.(cpt-1)) ∧ (cpt>0))} ;
 { ((res = val.val - val.(cpt-1)) ∧ (cpt>0)) }
     cpt ← cpt -1
 { ((res = val.val - val.cpt) ∧ (cpt≥0)) }
-----------------------------------------
 { ((res = val.val - val.cpt) ∧ (cpt>0)) }
     res ← res + val ; cpt ← cpt -1;
 { ((res = val.val - val.cpt) ∧ (cpt≥0))
// composition
```

26

Invariant de boucle- Fin

On a :

p ≡ ((res = val.val - val.cpt) ∧ (cpt≥0))

b ≡ (cpt ≠ 0)

S : res ← res + val; cpt ← cpt – 1;

```
 { ((res = val.val - val.cpt) ∧ (cpt > 0)) }
     res ← res + val; cpt ← cpt – 1 ;
 { ((res = val.val - val.cpt) ∧ (cpt≥0)) }
 ----------------------------------------
 { ((res = val.val - val.cpt) ∧ (cpt≥0)) }
     tant que (cpt > 0) faire
         res ← res + val; cpt ← cpt – 1 ;
      fin tant que
 { ((res = val.val - val.cpt) ∧ (cpt = 0)) }
```

27

Exemple

```
 { (val≥0)}
     cpt ← val ; res ← 0 ;
 { ((res = val.val – val.cpt) ∧(cpt≥0)) } ;
 { ((res = val.val - val.cpt) ∧ (cpt≥0)) }
     tant que (cpt > 0) faire
         res ← res + val; cpt ← cpt – 1 ;
       fin tant que
 { ((res = val.val - val.cpt) ∧ (cpt = 0)) }
-------------------------------------
 { (val ≥ 0) }
     cpt ← val; res ← 0 ;
     tant que ( cpt > 0 ) faire
                 res ← res + val;cpt ← cpt – 1 ;
     fin  tant que
 { ((res = val.val - val.cpt) ∧ (cpt = 0)) }
// composition
```

28

Exemple - Fin

```
 { (val ≥ 0) }
     cpt ← val; res ← 0 ;
     tant que ( cpt > 0 ) faire
                 res ← res + val;cpt ← cpt – 1 ;
     fin  tant que
 {((res = val.val - val.cpt) ∧(cpt=0))};
 { (((res = val.val - val.cpt) ∧(cpt=0))
      ⇒ (res = val.val)) }
-------------------------------------
 { (val ≥ 0) }
     cpt ← val; res ← 0 ;
     tant que ( cpt > 0 ) faire
                 res ← res + val;cpt ← cpt – 1 ;
     fin  tant que
 { (res = val.val) } // conséquence
```

29

Remarque : autres structures ?

Certains langages de programmation permettent l'utilisation d'autres structures de contrôle. Il est cependant toujours possible de se ramener aux structures prises en compte dans notre système :

- Si-Alors-Sinon imbriqués
- boucle « tant que »

30

Cohérence du système déductif

On peut montrer que le système déductif présenté est

cohérent (tout ce qu'il prouve est vrai)

il n'est pas complet (certaines formules vraies ne sont pas prouvables avec les règles de ce système...)

31

Correction totale (1)

Le système déductif présenté permet de prouver la correction partielle d'un programme : si celui-ci termine, alors le résultat produit est correct.

Pour obtenir la correction totale d'un programme, il est également nécessaire de prouver sa terminaison.

Pour cela, il faut examiner précisément :
⇒ les structures itératives (boucles)
⇒ la récursivité

32

Terminaison structure de « boucle »

En toute généralité, le problème de la terminaison d'un programme est un problème indécidable.

En pratique, nous sommes très souvent dans le cas où il est possible de prouver qu'un programme termine...

33

Preuve de Terminaison (1)

Technique de preuve de terminaison (cas des boucles).

Format général :

`Tant que <test> faire <corps>`

Idée : On exhibe une variable entière **v**, appelée variant de boucle, telle que :

- **v ∈ N⁺** (ayant une valeur avant le « **tant que** », et après chaque itération)
- **v** diminue à chaque itération

34

Preuve de Terminaison (2)

Technique de preuve de terminaison (cas des boucles).

v diminue à chaque itération. c'est-à-dire :

1. Si **`<test>`** est vrai alors la valeur de **v** est strictement positive
2. On nomme **v** la valeur du variant, et **v'** la valeur du variant après l'exécution du **`<corps>`**. On doit avoir **v > v'**

35

Preuve de Terminaison - Exemple

pré-condition.

```
(x, y entiers) et (x ≥ 0) et (y ≥ x)
et (b booléen)
```

programme.

```
xx ← x ; yy ← y ; b ← false;
tant que ( xx ≠ yy )faire
    si b alors xx ← xx + 1 ;
    sinon yy ← yy – 1;
    fsi
    b ← b+1;
fin tant que
```

yy-xx est-il le variant de boucle?...

36