Sémantique des langages

Logique de Hoare

ENSIIE S5

Sémantique axiomatique

Comme pour la sémantique opérationnelle, l'exécution du programme modifie l'état du système

Ici, l'état du système est abstrait par les propriétés logiques qu'il satisfait

- adapté à la preuve de programme

Défini par Hoare (inventeur de QuickSort) en 1969

Pour les langages impératifs (IMP)

Langages des assertions

L'état du système est décrit par des propositions de la logique des prédicats (logique du premier ordre) avec arithmétique

- $P ::= b \mid P \wedge P \mid P \vee P \mid \neg P \mid P \Rightarrow P \mid \forall x.\ P \mid \exists x.\ P$
- b peut être n'importe quelle expression arithmétique du langage IMP
- les variables du programme et celles de la logique sont identifiées
- implicitement, les variables prennent des valeurs entières

Exemple :

$$\forall z.\ x \leq z \Rightarrow y + 2 = z$$

Triplet de Hoare

$$\{P\}\mathtt{c}\{Q\}$$

avec P et Q des assertions logiques et c un programme

- P : précondition
- Q : postcondition
- si x est une variable du programme, dans P et Q x représente la valeur de x à ce point du programme

Lire : si la propriété P est vraie avant l'exécution de c et si l'exécution termine, alors la propriété Q est vraie après l'exécution de c
(Correction partielle, pas de preuve de terminaison)

Exemple

$\{x = n \wedge n > 0\}$
`y := 1; while not (x = 1) do (y := y * x; x := x - 1)`
$\{y = n! \wedge n > 0\}$

Ici, n est une variable logique, elle permet de se référer à la valeur de x au début du programme

Validité des triplets de Hoare

Tous les triplets de Hoare ne sont pas valides.

Intuitivement, on veut que $\{P\}\mathsf{c}\{Q\}$ soit valide si quand P est vrai avant d'exécuter le programme, alors Q est vrai après l'exécution.

Exemple de triplets valides :

- $\{\mathsf{true}\}\mathtt{x} := 5\{\mathsf{false}\}$
- $\{x = y\}\mathtt{x} := \mathtt{x} + 3\{x = y + 3\}$
- $\{x > 0\}\mathtt{x} := 2 * \mathtt{x}\{x > -2\}$
- $\{x = a\}\mathsf{if}\ \mathtt{x} < 0\ \mathsf{then}\ \mathtt{x} := -\mathtt{x}\ \mathsf{else}\ \mathsf{skip}\{x = |a|\}$

Mais $\{x < 0\}\mathtt{x} := \mathtt{x} - 3\{x > 0\}$ n'est pas valide

Règles d'inférence

La validité des triplets est définie par un système d'inférence dont les jugements sont les triplets

skip $\dfrac{}{\{P\}\mathsf{skip}\{P\}}$ aff $\dfrac{}{\{\{e/x\}P\}\mathtt{x} := \mathtt{e}\{P\}}$

seq $\dfrac{\{P\}\mathsf{c1}\{Q\} \qquad \{Q\}\mathsf{c2}\{R\}}{\{P\}\mathsf{c1}; \mathsf{c2}\{R\}}$

if $\dfrac{\{P \wedge b\}\mathsf{c1}\{Q\} \qquad \{P \wedge \neg b\}\mathsf{c2}\{Q\}}{\{P\}\mathsf{if\ b\ then\ c1\ else\ c2}\{Q\}}$

while $\dfrac{\{I \wedge b\}\mathsf{c}\{I\}}{\{I\}\mathsf{while\ b\ do\ c}\{I \wedge \neg b\}}$

consG $\dfrac{P \Rightarrow P' \qquad \{P'\}\mathsf{c}\{Q\}}{\{P\}\mathsf{c}\{Q\}}$ consD $\dfrac{Q' \Rightarrow Q \qquad \{P\}\mathsf{c}\{Q\}'}{\{P\}\mathsf{c}\{Q\}}$

Skip

$$\text{skip}\ \frac{}{\{P\}\text{skip}\{P\}}$$

skip ne change pas l'état, si P est vraie avant, P est vraie après

Séquence

$$\text{seq}\ \frac{\{P\}\texttt{c1}\{Q\} \qquad \{Q\}\texttt{c2}\{R\}}{\{P\}\texttt{c1};\texttt{c2}\{R\}}$$

c1 ; c2 exécute c1 puis c2

Exemple :

$$\text{seq}\ \frac{\{x > 2\}\texttt{x} := \texttt{x} + 1\{x > 3\} \qquad \{x > 3\}\texttt{x} := \texttt{x} + 2\{x > 5\}}{\{x > 2\}\texttt{x} := \texttt{x} + 1; \texttt{x} := \texttt{x} + 2\{x > 5\}}$$

Affectation

$$\text{aff}\ \frac{}{\{\{e/x\}P\}\mathtt{x} := \mathtt{e}\{P\}}$$

Rappel : $\{e/x\}P$ est la formule P où toutes les occurrences libres de x ont été substituées par e, par exemple $\{x+1/x\}(x > 1)$ vaut $x + 1 > 1$

- L'affectation change l'état, il faut donc que la logique de Hoare reflète ce changement
- De droite à gauche :
  Pour que P soit vrai après l'exécution, il fallait qu'il soit vrai avant mais en donnant à x sa nouvelle valeur

Exemple : $\{x + 1 > 5\}\mathtt{x} := \mathtt{x} + 1\{x > 5\}$

Exemple

Soit le programme c suivant :

```
tmp := x;
x := y;
y := tmp
```

Montrer que $\{x = a \wedge y = b\}\mathsf{c}\{y = a \wedge x = b\}$ est valide

Conditionnelle

$$\text{if}\ \frac{\{P \wedge b\}\texttt{c1}\{Q\} \qquad \{P \wedge \neg b\}\texttt{c2}\{Q\}}{\{P\}\texttt{if b then c1 else c2}\{Q\}}$$

- Quand la conditionnelle est exécutée, soit c1 soit c2 est exécuté
- Pour établir que Q est une post-condition, il faut donc que ce soit une post-condition des deux branches
- De la même façon, P est une pré-condition des deux branches
- Dans la branche c1, on sait que la condition b est vraie, on peut donc l'ajouter en pré-condition
- Similairement, on ajoute $\neg b$ comme pré-condition de c2

Utilisation

Prenons $\{x > 2\}$if x > 2 then y $:= 1$ else y $:= -1\{y > 0\}$

En utilisant la règle if, il faut donc montrer
$\{x > 2 \wedge x > 2\}$y $:= 1\{y > 0\}$ et
$\{x > 2 \wedge \neg x > 2\}$y $:= -1\{y > 0\}$

ce que l'on peut simplifier en
$\{x > 2\}$y $:= 1\{y > 0\}$ et
$\{$false$\}$y $:= -1\{y > 0\}$

Or à partir de la règle aff, on ne peut dériver que :
$\{1 > 0\}$y $:= 1\{y > 0\}$ et
$\{-1 > 0\}$y $:= -1\{y > 0\}$

Les règles structurelles ne suffisent pas, il faut utiliser des règles logiques

Conséquence gauche

Une condition P est dite plus forte qu'une condition Q si $P \Rightarrow Q$ est valide logiquement
Similairement, Q est dite plus faible que P.

$$\text{consG}\ \frac{P \Rightarrow P' \qquad \{P'\}\mathsf{c}\{Q\}}{\{P\}\mathsf{c}\{Q\}}$$

Il est correct de rendre une précondition plus spécifique (plus forte). Cette règle permet de renforcer les préconditions.
Exemple :

$$\text{consG}\ \frac{x = 4 \Rightarrow x > 2 \qquad \{x > 2\}\mathtt{x} := \mathtt{x} + 1\{x > 3\}}{\{x = 4\}\mathtt{x} := \mathtt{x} + 1\{x > 3\}}$$

Conséquence droite

De même, il est correct d'affaiblir une post-condition :

$$\text{consD}\ \frac{Q' \Rightarrow Q \qquad \{P\}\mathsf{c}\{Q\}'}{\{P\}\mathsf{c}\{Q\}}$$

Exemple :

$$\text{consG}\ \frac{x > 3 \Rightarrow x > 1 \qquad \{x > 2\}\mathtt{x} := \mathtt{x} + 1\{x > 3\}}{\{x > 2\}\mathtt{x} := \mathtt{x} + 1\{x > 1\}}$$

Retour sur l'exemple

On a $x > 2 \Rightarrow 1 > 0$ donc à partir de $\{1 > 0\}\mathtt{y} := 1\{y > 0\}$ on déduit $\{x > 2\}\mathtt{y} := 1\{y > 0\}$

De même false $\Rightarrow -1 > 0$ donc à partir de $\{-1 > 0\}\mathtt{y} := -1\{y > 0\}$ on déduit $\{\mathsf{false}\}\mathtt{y} := -1\{y > 0\}$

On peut donc finaliser la dérivation qui montre que $\{x > 2\}\mathsf{if}\ \mathtt{x} > 2\ \mathsf{then}\ \mathtt{y} := 1\ \mathsf{else}\ \mathtt{y} := -1\{y > 0\}$ est valide

Boucle

$$\text{while}\ \frac{\{I \wedge b\}\mathsf{c}\{I\}}{\{I\}\text{while b do c}\{I \wedge \neg b\}}$$

- I est appelé l'invariant de la boucle
- I est vrai à chaque tour de boucle (mais pas nécessairement au milieu d'un tour)
- si la boucle s'arrête, la condition b est fausse et sa négation peut donc être ajoutée en post-condition
- le corps de la boucle n'est exécuté que si la condition b est vrai, donc pour vérifier que l'invariant est conservé après l'exécution de la boucle, on peut ajouter b en précondition

Exercice

Montrez que $\{x \leq 0\}$while $\mathtt{x} \leq 5$ do $\mathtt{x} := \mathtt{x} + 1\{x = 6\}$ est valide.

Indication : vérifier que $x \leq 6$ est un invariant de la boucle.

Exercice

Soit c le programme :

```
i := 0;
s := 0;
while not (i = n) do (
  i := i + 1;
  s := s + 2 * i - 1 )
```

Vérifier que $s = i * i$ est un invariant de la boucle.

Montrer que $\{\text{true}\}\text{c}\{s = n * n\}$ est valide (la somme des n premiers nombres impairs est égale à n^2).

Exercice

1. Montrer que pour tout P et tout c, le triplet de Hoare $\{P\}\mathsf{c}\{\mathsf{true}\}$ est valide.

2. Montrer que pour tout P, Q et tout c, le triplet de Hoare $\{P\}\mathsf{while\ true\ do\ c}\{Q\}$ est valide.

Lien avec la sémantique opérationnelle

Une valuation (qui associe une valeur entière aux variables du programme) satisfait une assertion P si l'interprétation de P est vraie lorsque les variables libres de P sont interprétées par leur valeur dans la valuation.

Proposition (Correction de la logique de Hoare) :
Si un triplet $\{P\}\mathrm{c}\{Q\}$ est valide alors pour toute valuation σ, σ', si $\langle c, \sigma \rangle \to \sigma'$, si σ satisfait P alors σ' satisfait Q.

Automatisme

Pour la plupart, les règles de la logique de Hoare sont dirigées par la syntaxe du programme,
mais

- Quand faut-il appliquer les règles de conséquence ?
- Comment trouver les invariants ?
- Comment prouver les implications dans les règles de conséquence ?

Comment utiliser la démonstration automatique et interactive pour nous aider ?

Plus faible précondition (*Weakest precondition*)

Définition

Soit c *un programme et* Q *une formule, on note* $\mathsf{WP}(\mathsf{c}, Q)$ *la plus faible précondition telle que si* c *se termine, alors* c *se termine dans un état satisfaisant* Q*.*

Autrement dit :

- C'est une précondition qui mène à Q :
  $\{\mathsf{WP}(\mathsf{c}, Q)\}\mathsf{c}\{Q\}$ est valide
- c'est la plus faible de celles-ci :
  si $\{P\}\mathsf{c}\{Q\}$ est valide, alors $P \Rightarrow \mathsf{WP}(\mathsf{c}, Q)$

Théorème (Correction partielle avec WP)

Pour montrer $\{P\}\mathsf{c}\{Q\}$*, il suffit de montrer* $P \Rightarrow \mathsf{WP}(\mathsf{c}, Q)$*.*

Calcul de WP

Sans les boucles, WP peut être calculée automatiquement :

- $\mathsf{WP}(\mathsf{skip}, Q) = Q$
- $\mathsf{WP}(\mathtt{x} := \mathsf{a}, Q) = \{a/x\}Q$
- $\mathsf{WP}((\mathsf{c1}; \mathsf{c2}), Q) = \mathsf{WP}(\mathsf{c1}, \mathsf{WP}(\mathsf{c2}, Q))$
- $\mathsf{WP}(\text{if } \mathtt{b} \text{ then } \mathsf{c1} \text{ else } \mathsf{c2}, Q) =$
  $(b \Rightarrow \mathsf{WP}(\mathsf{c1}, Q)) \wedge (\neg b \Rightarrow \mathsf{WP}(\mathsf{c2}, Q))$

Exercice

Calculer :

- WP($\texttt{x} := \texttt{x} + 1; \texttt{y} := \texttt{y} - 1, x > y$)
- WP(if $\texttt{x} > \texttt{y}$ then $\texttt{x} := \texttt{x} - \texttt{y}$ else $\texttt{y} := \texttt{y} - \texttt{x}, x > 0 \wedge y > 0$)

Montrer en utilisant WP que
$\{x = n\}$if $\texttt{pair}(\texttt{x})$ then $\texttt{x} := \texttt{x} + 2$ else $\texttt{x} := \texttt{x} + 1\{x > n \wedge pair(x)$

Cas de la boucle

WP(while b do c, Q) = pas de formule simple !

En effet while b do c est (sémantiquement) équivalent à
if b then $(c;$ while b do $c)$ else skip

Donc WP(while b do c, Q) =

$$(b \Rightarrow \mathrm{WP}(\mathsf{c}, \mathrm{WP}(\text{while b do c}, Q))) \wedge (\neg b \Rightarrow Q)$$

Équation récursive pas toujours possible à calculer

On approxime WP par un prédicat plus simple à calculer et utilisable automatiquement :
la condition de vérification

Condition de vérification

On part d'un programme annoté :

- invariant de boucle
- spécification des fonctions

On note VC(c, Q) la condition de vérification de c pour l'assertion Q
La différence avec WP est qu'on utilise les annotations pour calculer VC

Calcul de VC

- $\mathsf{VC}(\mathsf{skip}, Q) = Q$
- $\mathsf{VC}(\mathtt{x} := \mathtt{a}, Q) = \{a/x\}Q$
- $\mathsf{VC}((\mathtt{c1}; \mathtt{c2}), Q) = \mathsf{VC}(\mathtt{c1}, \mathsf{VC}(\mathtt{c2}, Q))$
- $\mathsf{VC}(\text{if } \mathtt{b} \text{ then } \mathtt{c1} \text{ else } \mathtt{c2}, Q) =$
  $(b \Rightarrow \mathsf{VC}(\mathtt{c1}, Q)) \wedge (\neg b \Rightarrow \mathsf{VC}(\mathtt{c2}, Q))$
- $\mathsf{VC}(\text{while } \mathtt{b} \text{ do } \mathtt{c} \texttt{ //invariant } I, Q) =$
  I (l'invariant est vérifié en début de boucle)
  $\wedge(\forall x_1, \ldots, x_m.\ I \wedge b \Rightarrow \mathsf{VC}(\mathtt{c}, I))$
  (l'invariant est préservé par une itération)
  $\wedge(\neg b \wedge I \Rightarrow Q)$
  (la postcondition est satisfaite lorsque la boucle se termine)

  $x_1 \ldots x_n$ sont les variables modifiées dans c

Exemple

On reprend la factorielle :

$\{x = n \wedge n > 0\}$`y := 1;`
`while not (x = 1) do (y := y * x; x := x - 1)`
$\{y = n! \wedge n > 0\}$

On prend comme invariant
$Inv = (y * x! = n! \wedge n > 0 \wedge x > 0)$

On a VC(`fact`, $y = n! \wedge n > 0$) =
$1 * x! = n! \wedge x > 0 \wedge n > 0$

$\wedge \forall xy.\ Inv \wedge x \neq 1 \Rightarrow y * x * (x-1)! = n! \wedge x - 1 > 0 \wedge n > 0$

$\wedge x = 1 \wedge x! = n! \wedge x > 0 \wedge n > 0 \Rightarrow (1 = n! \wedge n > 0)$

On peut “facilement” montrer que
$x = n \wedge n > 0 \Rightarrow$ VC(`fact`, $y = n! \wedge n > 0$)

Correction des conditions de vérification

Théorème

$\{\mathsf{VC}(\mathsf{c}, Q)\}\mathsf{c}\{Q\}$ *est valide.*

Preuve : par induction sur c

Théorème

Si $P \Rightarrow \mathsf{VC}(\mathsf{c}, Q)$ *alors* $\{P\}\mathsf{c}\{Q\}$ *est valide.*

Preuve : par induction sur c en montrant qu'il existe une dérivation dont les conditions d'applicabilité sont exactement $P \Rightarrow \mathsf{VC}(\mathsf{c}, Q)$.

Génération de conditions de vérification

Grâce aux programmes annotés et à VC, on obtient une méthode réaliste pour faire de la preuve de programmes impératifs :

- Écrire un programme contenant les invariants des boucles et éventuellement des assertions à certains point du programme difficiles pour la preuve ;
- Transmettre ce code annoté à un outil qui engendre les conditions de vérification (exemple : plugin WP de Frama-C, cf. TP) ;
- Prouver ces conditions de vérification, de préférence avec un prouveur automatique (alt-ergo, Z3, Vampire, ...) ou interactif (Coq, Isabelle, ...)