UNIVERSITÉ PARIS SUD

L3 informatique et MIAGE

Année 2016-2017

Génie logiciel avancé

Preuve de programmes (suite)

Delphine Longuet

delphine.longuet@lri.fr

http://www.lri.fr/~longuet/Enseignements/16-17/L3-GLA

Exemples d'invariants

Soient les programmes suivants calculant X^N pour N positif ou nul

```
P := 0 ;
S := 1 ;
WHILE P < N DO
   S := S * X ;
   P := P + 1
```

```
P := N ;
S := 1 ;
WHILE P ≥ 1 DO
   S := S * X ;
   P := P - 1
```

Entrées : `N`, `X` entiers

Sortie : `S` entier

Donner les invariants de boucle de ces programmes

$Inv : S = X^P \wedge P \leq N$

$Inv : S = X^{N-P} \wedge P \geq 0$

Exemples d'invariants

Soit le programme suivant multipliant par 2 tous les éléments d'un tableau

```
i := 0 ;
WHILE i < n DO
   tab[i] := 2 * tab[i] ;
   i := i + 1
```

Entrées : `tab` tableau d'entiers, `n` entier

Donner l'invariant de boucle de ce programme

pre : $n = \text{taille}(tab)$

post : $\forall k,\ 0 \leq k < n \Rightarrow tab[k] \bmod 2 = 0$

Inv : $\forall k,\ 0 \leq k < i \Rightarrow tab[i] \bmod 2 = 0 \wedge i \leq n$

Exemples d'invariants

Soit le programme suivant calculant le pgcd de deux entiers u et v positifs avec u non nul

```
m := u ;
n := v ;
WHILE n != 0 DO
   IF m > n
   THEN m := m - n
   ELSE n := n - m
```

Entrées : u, v entiers

Sortie : m entier

Donner l'invariant de boucle de ce programme

pre : $u > 0$ et $v \geq 0$ **post** : $m = \text{pgcd}(u,v)$

Inv : $\text{pgcd}(m,n) = \text{pgcd}(u,v)$

Calcul de Hoare : règles dérivées

Axiome false

$$\frac{}{\{false\}\ \mathit{ins}\ \{false\}}\ \text{false}$$

Dérivé des autres règles par induction sur *ins*

Intuition : *ins* ne peut pas être exécuté (pré-condition $false$) donc tout ce qui le suit ne peut pas non plus être exécuté (post-condition $false$: pré-condition de la suite du programme)

Calcul de Hoare : règles dérivées

Axiome falseE

$$\frac{}{\{false\}\ \mathit{ins}\ \{Q\}}\ \text{falseE}$$

Dérivé de l'axiome false par la règle de conséquence

Intuition : *ins* ne peut pas être exécuté, donc le triplet est valide pour toute post-condition Q

$$\frac{\dfrac{}{\{false\}\ \mathit{ins}\ \{false\}}\ \text{false} \qquad false \Rightarrow Q}{\{false\}\ \mathit{ins}\ \{Q\}}\ \text{cons}$$

Calcul de Hoare : règles dérivées

Axiome falseE

$$\frac{}{\{false\}\ ins\ \{Q\}}\ \text{falseE}$$

Dérivé de l'axiome false par la règle de conséquence

Intuition : *ins* ne peut pas être exécuté, donc le triplet est valide pour toute post-condition Q

Exemple :

$$\frac{}{\{false\}\ \mathtt{x}\ :=\ 25\ \{x = 0\}}\ \text{falseE}$$

Calcul de Hoare : règles dérivées

Axiome falseE

$$\frac{}{\{false\}\ ins\ \{Q\}}\ \text{falseE}$$

Dérivé de l'axiome false par la règle de conséquence

Intuition : *ins* ne peut pas être exécuté, donc le triplet est valide pour toute post-condition Q

Exemple (condition de boucle non satisfaite) :

$$\frac{\dfrac{}{\{x < 0 \wedge x > 0\}\ \texttt{x := x-1}\ \{x < 0\}}\ \text{falseE}}{\{x < 0\}\ \texttt{WHILE x > 0 DO x := x-1}\ \{x < 0\}}\ \text{while}$$

Méthode générale pour faire une preuve

But : prouver le triplet $\{P\}$ *prog* $\{Q\}$

1. Instruction à la racine de *prog* ?
   $\rightarrow$ Règle *rule* à appliquer pour faire cette étape de preuve
2. Conditions d'application de la règle *rule* vérifiées ?
   - Oui, application de la règle *rule*. SUITE
   - Non. Conditions d'application de la règle de conséquence vérifiées, pour pouvoir appliquer *rule* ensuite ?
     - Oui, application de la règle de conséquence puis *rule*. SUITE
     - Non, impossible d'appliquer la règle de conséquence de manière à pouvoir appliquer *rule*. Triplet invalide. FIN

SUITE : suite de la preuve sur les triplets obtenus en prémisses

Remarque sur la non terminaison

Preuve d'un programme qui ne termine pas

$$\{true\}\ \texttt{WHILE true DO SKIP}\ \{x = 42\}$$

$$\dfrac{\dfrac{}{\{true \wedge true\}\ \texttt{SKIP}\ \{true\}}\ \text{skip}}{\{true\}\ \texttt{WHILE true DO SKIP}\ \{false\}}\ \text{while} \qquad false \Rightarrow x = 42}{\{true\}\ \texttt{WHILE true DO SKIP}\ \{x = 42\}}\ \text{cons}$$

(Invariant de boucle : $true$)

Preuve en logique de Hoare $=$ preuve de correction partielle

Toute post-condition valide si le programme ne termine pas

Problème de la terminaison

Problème non décidable

Méthode classique : associer un compteur aux boucles ou aux appels récursifs (en général appelé *variant*)

- Montrer que la valeur de ce compteur est bornée inférieurement
- Montrer que la valeur initiale (à l'entrée de la boucle) est supérieure à cette borne
- Montrer que chaque passage dans la boucle ou chaque appel récursif fait strictement décroître la valeur du compteur

En pratique : variant = expression entière restant positive ou nulle et décroissant strictement à chaque tour de boucle

Correction totale = correction partielle + terminaison

Variant

Soient les programmes suivants calculant X^N pour N positif ou nul

```
P := 0 ;
S := 1 ;
WHILE P < N DO
   S := S * X ;
   P := P + 1
```

```
P := N ;
S := 1 ;
WHILE P ≥ 1 DO
   S := S * X ;
   P := P - 1
```

Pour chacun de ces programmes, donner un variant pour la boucle, c'est-à-dire une expression (entière)

- toujours positive ou nulle
- positive strictement au premier passage dans la boucle
- décroissant strictement à chaque passage dans la boucle

Variant : $N - P$

Variant : P

Calcul de Hoare en pratique

Problèmes de l'automatisation de la recherche de preuve :

1. Trouver la formule intermédiaire appropriée
   a. pour la règle de séquence
   b. pour la règle du WHILE
2. Savoir quand appliquer la règle de conséquence
3. Démontrer la validité des formules logiques dans la règle de conséquence ($P \Rightarrow P'$ et $Q' \Rightarrow Q$)

Calcul de Hoare en pratique

Éléments de réponse :

1. b. Demander à l'utilisateur de fournir un invariant
2. Appliquer la règle de conséquence à certains endroits spécifiques de la preuve (par ex. avant la règle du WHILE)
3. Appeler des démonstrateurs automatiques ou interactifs (Alt-Ergo, Z3, Coq...)

De manière générale, mener la preuve à partir des post-conditions, en cherchant les pré-conditions les plus faibles permettant de parvenir à ces post-conditions : calcul de *weakest preconditions* (wp)

$$\{P\}\ \mathit{prog}\ \{Q\} \text{ ssi } P \Rightarrow \mathsf{wp}(\mathit{prog}, Q)$$

Conclusion

Avantages de la preuve :

- Complétude
- Approche outillée, par exemple
  - Microsoft Visual-Studio + VCC + Boogie + Z3 (pour un sous-ensemble réaliste de C)
  - FramaC + Why3 + Alt-Ergo/Z3/CVC4... (pour C)

Inconvénients :

- Difficulté pour fournir les invariants
- Effort à fournir bien supérieur à un développement standard

Remarque : Effort raisonnable pour un système critique et presque comparable à du test minutieux