Polytech Paris-Sud
Formation initiale 4e année
Spécialité informatique
Année 2016-2017

Vérification et validation

Cours 6
Preuve de programmes

Delphine Longuet
delphine.longuet@lri.fr

http://www.lri.fr/~longuet/Enseignements/16-17/Et4-VV

Avantages et limitations du test

Avantages :

- Exécution du système
- Environnement et conditions réels d'utilisation
- Facilité de mise en œuvre
- Écriture possible des tests avant le développement
- Validation permanente des changements au sein du code

Principal inconvénient : incomplétude

Impossible d'exécuter des tests pour toutes les données d'entrée possibles d'un programme. Pas de certitude sur l'absence de fautes, seulement une certaine confiance dans la qualité

Vérification de programmes

But : démontrer mathématiquement l'absence de fautes dans un programme

Méthodes :

- Analyse statique (model-checking, interprétation abstraite) : démonstration de l'absence de blocage, de débordement d'entier, de débordement de tableau...
- Raffinement : code construit par raffinements successifs d'un modèle formel, code correct par construction
- Vérification déductive : à l'aide d'un modèle formel du langage de programmation, démonstration que le programme satisfait sa spécification

Utilisation dans l'industrie

Ligne 14 du métro à Paris : logiciel critique embarqué développé selon la méthode B (1998, approche par rafinement)

Logiciel de contrôle de vol de l'A380 : vérification de l'absence d'erreur à l'exécution avec Astree (2005, interprétation abstraite)

Hyperviseur de Microsoft : correction prouvée avec VCC et le prouveur automatique Z3 (2008, vérification déductive)
Plus récemment : vérification de PikeOS

Compilateur C certifié : développé avec l'assistant de preuve Coq (2009, code correct par construction généré par un assistant de preuve)

Vérification déductive

Ingrédients nécessaires :

- Spécification formelle du programme : pré et post-conditions
- Modèle formel du langage de programmation : signification mathématique de chacune des expressions et instructions de ce langage

Résultat obtenu : démonstration mathématique que le programme satisfait sa spécification, c'est-à-dire

Pour toutes les entrées du programme vérifiant la pré-condition,
si le programme termine, la post-condition est vérifiée

Rappels : système d'inférence

Système d'inférence : ensemble de règles permettant de déduire des faits depuis des axiomes et des hypothèses

Axiome : règle sans prémisse

$$\text{règle}\ \frac{A_1 \quad A_2 \quad \dots \quad A_n}{A}$$

prémisses

nom de la règle

conclusion

$$\text{ax}\ \frac{}{A}$$

axiome

Si A_1, A_2,…, A_n sont démontrées, alors la règle permet de démontrer A

Exemple : système d'inférence pour l'égalité

$$\text{ref}\ \frac{}{x = x} \qquad \text{sym}\ \frac{x = y}{y = x} \qquad \text{trans}\ \frac{x = y \quad y = z}{x = z} \qquad \text{subs}\ \frac{x = y \quad P(x)}{P(y)}$$

Rappels : arbre de preuve

Arbre de preuve : séquence d'applications des règles d'un système inférence permettant de démontrer un fait

hypothèses

$$\dfrac{\dfrac{\dfrac{f(a,b) = a}{a = f(a,b)}\,\text{sym} \qquad \dfrac{f(a,b) = a \qquad f(f(a,b),b) = c}{f(a,b) = c}\,\text{subs}}{a = c}\,\text{trans} \qquad \dfrac{}{g(a) = g(a)}\,\text{ref}}{g(a) = g(c)}\,\text{subs}$$

Conclusion dérivable à partir des hypothèses, noté :

$$\{f(a,b) = a,\ f(f(a,b),b) = c\} \vdash g(a) = g(c)$$

Rappels : théorème

Théorème : fait dérivable à partir d'hypothèses par l'application des règles d'un système d'inférence

Conclusion d'un arbre de preuve dont les feuilles sont des axiomes et des hypothèses

$$\{\mathrm{Hyp}_1, \ldots, \mathrm{Hyp}_n\} \vdash \mathrm{Th}$$

$$\dfrac{\mathrm{Hyp}_1 \qquad \dfrac{\dfrac{\mathrm{Hyp}_2 \quad \dfrac{}{\ldots}\,\mathrm{ax}}{\ldots} \qquad \dfrac{\dfrac{}{\ldots}\,\mathrm{ax} \quad \mathrm{Hyp}_n}{\ldots}}{\ldots}}{\mathrm{Th}}$$

Langage impératif simple

Types : booléens et entiers

Instructions :

`SKIP`	aucun effet
`x := exp`	affectation
ins_1 ; ins_2	séquence d'instructions
`IF cond THEN` ins_1 `ELSE` ins_2	branchement conditionnel
`WHILE cond DO` *ins*	boucle

avec :

`x` : variable entière ou booléenne

`exp` : expression arithmétique ou booléenne

`cond` : expression booléenne

ins, ins_1, ins_2: instructions

Spécification d'un programme

Programme *prog* : séquence d'instructions

Pré-condition P : propriété à vérifier pour exécuter le programme

Post-condition Q : propriété à vérifier après l'exécution du programme

Triplet de Hoare : $\{P\}$ *prog* $\{Q\}$

Signification : si le programme *prog* est exécuté sur des données d'entrée satisfaisant la pré-condition P, alors si l'exécution termine, la post-condition Q est vérifiée

Correction partielle : aucune information si le programme ne termine pas

Triplet de Hoare

Exemples de triplets de Hoare valides :

$\{x = 1\}$ `x := x+2` $\{x = 3\}$

$\{y = 2\}$ `x := y+5` $\{x = 7\}$

$\{y = 1\}$ `x := y+1 ; z := x-1` $\{z = 1\}$

$\{-4 \leq x \leq 4\}$ `IF x ≥ 0 THEN x:=x-2 ELSE x:=x+2` $\{-2 \leq x \leq 2\}$

$\{x \geq 0\}$ `WHILE x > 0 DO x := x-1` $\{x = 0\}$

Triplet de Hoare

Triplets de Hoare particuliers :

$\{true\}$ *prog* $\{Q\}$: le programme peut toujours être exécuté, aucune condition sur les entrées

$$\{true\}\ \texttt{x := 2*x}\ \{x \bmod 2 = 0\}$$

$\{false\}$ *prog* $\{Q\}$: le programme ne peut jamais être exécuté, aucune valeur d'entrée n'est possible. Le triplet est valide pour tout Q

$$\{false\}\ \texttt{x := 25}\ \{x = 0\}$$

$\{P\}$ *prog* $\{false\}$: le programme ne termine pas, aucune valeur de sortie n'est possible. Le triplet est valide pour tout P

$$\{b = true\}\ \texttt{WHILE b DO SKIP}\ \{false\}$$

Calcul de Hoare

Système d'inférence pour le langage impératif simple présenté

Une règle pour chaque type d'instruction

- axiome pour SKIP
- axiome pour l'affectation
- règle pour la séquence
- règle pour le branchement conditionnel
- règle pour la boucle

$+$ règle de conséquence : règle logique sur les pré et post-conditions

Calcul de Hoare : `SKIP`

Axiome pour `SKIP`

$$\frac{}{\{P\}\ \texttt{SKIP}\ \{P\}}\ \textsf{skip}$$

`SKIP` n'a aucun effet, donc toute propriété vraie avant est vraie après

Calcul de Hoare : affectation

Axiome pour l'affectation

$$\dfrac{}{\{P[x \mapsto exp]\}\ \texttt{x := exp}\ \{P\}}\,\text{aff}$$

où $P[x \mapsto exp]$ est la formule P dans laquelle toutes les occurrences de x ont été remplacées (syntaxiquement) par exp (substitution)

Pour que P soit vraie après l'affectation, il suffit que $P[x \mapsto exp]$ soit vraie avant

À appliquer de droite à gauche : $\cancel{\{x = y\}\ \texttt{x := 5}\ \{\ ??\ \}}$

Calcul de Hoare : affectation

Axiome pour l'affectation

$$\overline{\{P[x \mapsto exp]\}\ \texttt{x := exp}\ \{P\}}^{\text{aff}}$$

Exemple :

$$\overline{\{y = 2\}\ \texttt{x := y+5}\ \{x = 7\}}^{\text{aff}}$$

car $x = 7[x \mapsto y{+}5] \Leftrightarrow y + 5 = 7 \Leftrightarrow y = 2$

Calcul de Hoare : affectation

Axiome pour l'affectation

$$\dfrac{}{\{P[x \mapsto exp]\}\ \texttt{x := exp}\ \{P\}}\ \textsf{aff}$$

Exemple :

$$\dfrac{}{\{m = 2n\}\ \texttt{m := m-n}\ \{m = n\}}\ \textsf{aff}$$

car $m = n[m \mapsto m - n] \Leftrightarrow m - n = n \Leftrightarrow m = 2n$

Calcul de Hoare : affectation

Axiome pour l'affectation

$$\overline{\{P[x \mapsto exp]\}\ \texttt{x := exp}\ \{P\}}^{\ \text{aff}}$$

Exemple :

$$\overline{\{true\}\ \texttt{z := 3}\ \{z = 3\}}^{\ \text{aff}}$$

car $z = 3[z \mapsto 3] \Leftrightarrow 3 = 3 \Leftrightarrow true$

Calcul du Hoare : séquence

Règle pour la séquence :

$$\frac{\{P\}\ \textit{ins}_1\ \{Q\} \qquad\qquad \{Q\}\ \textit{ins}_2\ \{R\}}{\{P\}\ \textit{ins}_1\ ;\ \textit{ins}_2\ \{R\}}\ \text{seq}$$

Apparition d'une propriété intermédiaire Q qui soit vraie après $\textit{ins}_1$ et telle que R puisse être prouvée à partir de Q après $\textit{ins}_1$

Calcul du Hoare : séquence

Règle pour la séquence :

$$\frac{\{P\}\ \mathit{ins}_1\ \{Q\} \qquad \{Q\}\ \mathit{ins}_2\ \{R\}}{\{P\}\ \mathit{ins}_1\ ;\ \mathit{ins}_2\ \{R\}}\ \mathsf{seq}$$

Exemple :

$$\frac{\{y = 1\}\ \texttt{x:=y+1}\ \{\ ?\ \} \qquad \{\ ?\ \}\ \texttt{z:=x-1}\ \{z = 1\}}{\{y = 1\}\ \texttt{x:=y+1}\ ;\ \texttt{z:=x-1}\ \{z = 1\}}\ \mathsf{seq}$$

Calcul du Hoare : séquence

Règle pour la séquence :

$$\frac{\{P\}\ \mathit{ins}_1\ \{Q\} \qquad \{Q\}\ \mathit{ins}_2\ \{R\}}{\{P\}\ \mathit{ins}_1\ ;\ \mathit{ins}_2\ \{R\}}\,\mathsf{seq}$$

Exemple :

$$\frac{\dfrac{}{\{y=1\}\ \texttt{x:=y+1}\ \{x=2\}}\,\mathsf{aff} \qquad \dfrac{}{\{x=2\}\ \texttt{z:=x-1}\ \{z=1\}}\,\mathsf{aff}}{\{y=1\}\ \texttt{x:=y+1}\ ;\ \texttt{z:=x-1}\ \{z=1\}}\,\mathsf{seq}$$

Calcul du Hoare : séquence

Règle pour la séquence :

$$\frac{\{P\}\ \mathit{ins}_1\ \{Q\} \qquad \{Q\}\ \mathit{ins}_2\ \{R\}}{\{P\}\ \mathit{ins}_1\ ;\ \mathit{ins}_2\ \{R\}}\ \mathsf{seq}$$

Exemple :

$$\frac{\{S = X^{N\text{-}P}\}\ \texttt{S:=S*X}\ \{\ ?\ \} \qquad \{\ ?\ \}\ \texttt{P:=P-1}\ \{S = X^{N\text{-}P}\}}{\{S = X^{N\text{-}P}\}\ \texttt{S:=S*X}\ ;\ \texttt{P:=P-1}\ \{S = X^{N\text{-}P}\}}\ \mathsf{seq}$$

Calcul du Hoare : séquence

Règle pour la séquence :

$$\frac{\{P\}\ ins_1\ \{Q\} \qquad \{Q\}\ ins_2\ \{R\}}{\{P\}\ ins_1\ ;\ ins_2\ \{R\}}\ \text{seq}$$

Exemple :

$$\frac{\dfrac{}{\{S = X^{N\text{-}P}\}\ \texttt{S:=S*X}\ \{S = X^{N\text{-}P+1}\}}\ \text{aff} \qquad \dfrac{}{\{S = X^{N\text{-}P+1}\}\ \texttt{P:=P-1}\ \{S = X^{N\text{-}P}\}}\ \text{aff}}{\{S = X^{N\text{-}P}\}\ \texttt{S:=S*X}\ ;\ \texttt{P:=P-1}\ \{S = X^{N\text{-}P}\}}\ \text{seq}$$

Calcul de Hoare : conséquence

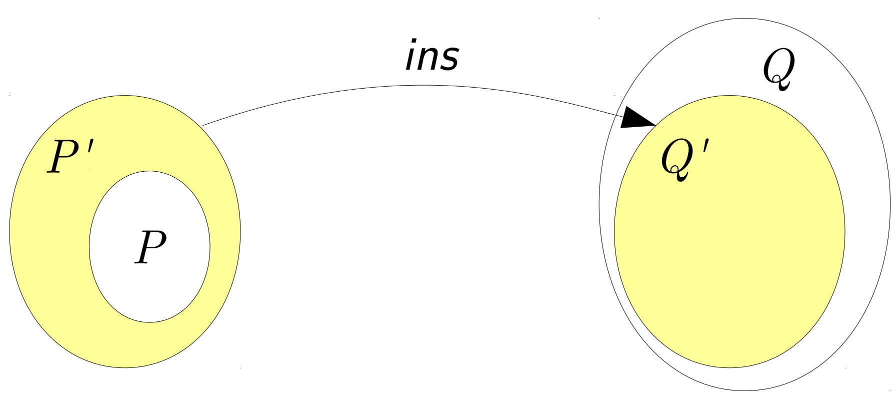
Règle de conséquence :

$$\frac{P \Rightarrow P' \qquad \{P'\}\ \mathit{ins}\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ \mathit{ins}\ \{Q\}}\ \text{cons}$$

Possibilité d'affaiblir P ou de renforcer Q pour prouver $\{P\}$ *ins* $\{Q\}$

Si $\{P'\}$ *ins* $\{Q'\}$ est valide, alors il est valide en particulier :

- pour un sous-ensemble des entrées satisfaisant P', défini par P
- pour un sur-ensemble des sorties satisfaisant Q', défini par Q



Calcul de Hoare : conséquence

Règle de conséquence :

$$\frac{P \Rightarrow P' \qquad \{P'\}\ \textit{ins}\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ \textit{ins}\ \{Q\}}\ \text{cons}$$

Possibilité d'affaiblir P ou de renforcer Q pour prouver $\{P\}$ *ins* $\{Q\}$

Exemple :

$$\frac{x \geq 10 \Rightarrow x \geq 0 \qquad \{x \geq 0\}\ \textit{prog}\ \{y = 1\} \qquad y = 1 \Rightarrow y \geq 0}{\{x \geq 10\}\ \textit{prog}\ \{y \geq 0\}}\ \text{cons}$$

Calcul de Hoare : conséquence

Règle de conséquence :

$$\frac{P \Rightarrow P' \qquad \{P'\}\ \textit{ins}\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ \textit{ins}\ \{Q\}}\ \text{cons}$$

Possibilité d'affaiblir P ou de renforcer Q pour prouver $\{P\}$ *ins* $\{Q\}$

Applications particulières :

$$\frac{P \Rightarrow P' \qquad \{P'\}\ \textit{ins}\ \{Q\}}{\{P\}\ \textit{ins}\ \{Q\}}\ \text{cons} \qquad \frac{\{P\}\ \textit{ins}\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ \textit{ins}\ \{Q\}}\ \text{cons}$$

Calcul de Hoare : `IF THEN ELSE`

Règle pour le branchement conditionnel

$$\frac{\{P \wedge cond\}\ \textit{ins}_1\ \{Q\} \qquad \{P \wedge \neg cond\}\ \textit{ins}_2\ \{Q\}}{\{P\}\ \texttt{IF cond THEN}\ \textit{ins}_1\ \texttt{ELSE}\ \textit{ins}_2\ \{Q\}}\ \text{if}$$

Pour que Q soit vraie après le branchement, il faut qu'elle soit vraie quelle que soit la branche exécutée : preuve divisée en deux sous-cas

Calcul de Hoare : `IF THEN ELSE`

Règle pour le branchement conditionnel

$$\frac{\{P \land cond\}\ \textit{ins}_1\ \{Q\} \qquad \{P \land \neg cond\}\ \textit{ins}_2\ \{Q\}}{\{P\}\ \texttt{IF cond THEN}\ \textit{ins}_1\ \texttt{ELSE}\ \textit{ins}_2\ \{Q\}}\ \textsf{if}$$

Exemple :

$$\dfrac{\dfrac{}{\{\text{-}4 \leq x \leq 4 \land x \geq 0\}\ \texttt{x:=x-2}\ \{\text{-}2 \leq x \leq 2\}}\ \textsf{aff} \qquad \dfrac{\dots}{\{\text{-}4 \leq x \leq 4 \land x < 0\}\ \texttt{x:=x+2}\ \{\text{-}2 \leq x \leq 2\}}}{\{\text{-}4 \leq x \leq 4\}\ \texttt{IF x} \geq \texttt{0 THEN x:=x-2 ELSE x:=x+2}\ \{\text{-}2 \leq x \leq 2\}}\ \textsf{if}$$

Calcul de Hoare : `IF THEN ELSE`

Règle pour le branchement conditionnel

$$\frac{\{P \wedge cond\}\ \textit{ins}_1\ \{Q\} \qquad \{P \wedge \neg cond\}\ \textit{ins}_2\ \{Q\}}{\{P\}\ \texttt{IF cond THEN}\ \textit{ins}_1\ \texttt{ELSE}\ \textit{ins}_2\ \{Q\}}\ \text{if}$$

Exemple :

$$\frac{\dfrac{\dots}{\{x < 0\}\ \texttt{x := -x}\ \{x \geq 0\}} \qquad \dfrac{}{\{x \geq 0\}\ \texttt{SKIP}\ \{x \geq 0\}}\ \text{skip}}{\{true\}\ \texttt{IF x < 0 THEN x := -x ELSE SKIP}\ \{x \geq 0\}}\ \text{if}$$

Preuve d'un programme

Soit le programme suivant :

```
IF x > y
THEN max := x
ELSE max := y
```

Entrées : `x`, `y`

Sortie : `max`

Donner la spécification de ce programme et démontrer la validité du triplet de Hoare correspondant

Calcul de Hoare : `WHILE`

Règle pour la boucle

$$\frac{\{P \wedge cond\}\ \textit{ins}\ \{P\}}{\{P\}\ \texttt{WHILE cond DO}\ \textit{ins}\ \{P \wedge \neg cond\}}\ \text{while}$$

La propriété P doit être vraie à l'initialisation et à la sortie de la boucle, et doit être préservée à chaque tour de boucle

P : invariant de boucle

Difficulté majeure de la preuve de programme : trouver cet invariant

Calcul de Hoare : WHILE

Règle pour la boucle

$$\frac{\{P \land cond\}\ \textsf{\textit{ins}}\ \{P\}}{\{P\}\ \texttt{WHILE cond DO}\ \textsf{\textit{ins}}\ \{P \land \neg cond\}}\ \text{while}$$

L'invariant de boucle P doit être vrai à l'initialisation et à la sortie de la boucle, et doit être préservé à chaque tour de boucle

Exemple : $P \Leftrightarrow x \geq 0$

$$\frac{\dfrac{}{\{x > 0\}\ \texttt{x := x-1}\ \{x \geq 0\}}\ \text{aff}}{\{x \geq 0\}\ \texttt{WHILE x > 0 DO x := x-1}\ \{x = 0\}}\ \text{while}$$

Calcul de Hoare : `WHILE`

Règle pour la boucle

$$\frac{\{P \wedge cond\}\ \textsf{\textit{ins}}\ \{P\}}{\{P\}\ \texttt{WHILE cond DO}\ \textsf{\textit{ins}}\ \{P \wedge \neg cond\}}\ \textsf{while}$$

L'invariant de boucle P doit être vrai à l'initialisation et à la sortie de la boucle, et doit être préservé à chaque tour de boucle

En général, règle de conséquence nécessaire pour introduire l'invariant

$$\frac{P \Rightarrow Inv \qquad \dfrac{\{Inv \wedge cond\}\ \textsf{\textit{ins}}\ \{Inv\}}{\{Inv\}\ \texttt{WHILE cond DO}\ \textsf{\textit{ins}}\ \{Inv \wedge \neg cond\}}\ \textsf{while} \qquad Inv \wedge \neg cond \Rightarrow Q}{\{P\}\ \texttt{WHILE cond DO}\ \textsf{\textit{ins}}\ \{Q\}}\ \textsf{cons}$$

Preuve d'un programme

Soit le programme suivant

```
X := 1 ;
S := 1 ;
WHILE X ≤ N DO
   S := S * X ;
   X := X + 1
```

Entrée : `N`

Sortie : `S`

Trouver Inv tel que :

- Inv soit vrai à l'entrée de la boucle
- Inv soit vrai à la sortie de la boucle
- Inv soit préservé à chaque tour de boucle

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** : $m = \mathrm{pgcd}(u,v)$

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

Calcul de Hoare : règles dérivées

Axiome false

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

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

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

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

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

Calcul de Hoare : règles dérivées

Axiome falseE

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

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 :

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

Calcul de Hoare : règles dérivées

Axiome falseE

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

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) :

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

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

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

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

(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)

$$\{P\}\ \textit{prog}\ \{Q\} \text{ ssi } P \Rightarrow \mathrm{wp}(\textit{prog}, Q)$$

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

Calcul de Hoare : règles dérivées

Axiome false

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

Dérivé des autres règles par induction sur *ins*

- Cas de base :

$$\dfrac{}{\{false\}\ \texttt{SKIP}\ \{false\}}\ \text{skip} \qquad \dfrac{}{\{false\}\ \texttt{x := exp}\ \{false\}}\ \text{aff}$$

- Induction : on suppose $\{false\}\ \mathit{ins}_1\ \{false\}$ et $\{false\}\ \mathit{ins}_2\ \{false\}$

$$\dfrac{\{false \wedge cond\}\ \mathit{ins}_1\ \{false\} \qquad \{false \wedge \neg cond\}\ \mathit{ins}_2\ \{false\}}{\{false\}\ \texttt{IF}\ cond\ \texttt{THEN}\ \mathit{ins}_1\ \texttt{ELSE}\ \mathit{ins}_2\ \{false\}}\ \text{if}$$

$$\dfrac{\{false\}\ \mathit{ins}_1\ \{false\} \qquad \{false\}\ \mathit{ins}_2\ \{false\}}{\{false\}\ \mathit{ins}_1\ ;\ \mathit{ins}_2\ \{false\}}\ \text{seq} \qquad \dfrac{\{false \wedge cond\}\ \mathit{ins}_1\ \{false\}}{\{false\}\ \texttt{WHILE}\ cond\ \texttt{DO}\ \mathit{ins}_1\ \{false\}}\ \text{while}$$