Logique de Hoare et Générateur de plus faible pré-condition

Benjamin Grégoire
INRIA Sophia-Antipolis (Projet EVEREST)

Plan

- Sémantique de While
- Logique de Hoare (correction)
- VCgen pour While (correction)
- Preuve d'équivalence entre Hoare et VCgen
- VCgen pour le bytecode (début)

Sémantique de While

expressions	e	$::=$	$n \mid x \mid e + e \mid e * e$
instructions	i	$::=$	$x := e$
		$\mid$	$?(e)\ S : S$
		$\mid$	while $(e)\ S$
séquence	S	$::=$	$\emptyset \mid i; S$
mémoire locale	l	$:$	$x \mapsto n$

Sémantique opérationnelle (Big Step)

expressions $e ::= n \mid x \mid e + e \mid e * e$

Sémantique des expressions : $\langle e \mid l \rangle \to n$

$$\frac{}{\langle n \mid l \rangle \to n} \qquad \frac{}{\langle x \mid l \rangle \to l[x]}$$

$$\frac{\langle e_1 \mid l \rangle \to n_1 \quad \langle e_2 \mid l \rangle \to n_2}{\langle e_1 + e_2 \mid l \rangle \to n_1 + n_2} \qquad \frac{\langle e_1 \mid l \rangle \to n_1 \quad \langle e_2 \mid l \rangle \to n_2}{\langle e_1 * e_2 \mid l \rangle \to n_1 * n_2}$$

Sémantique opérationnelle (Big Step)

instructions $i ::= x := e \mid ?(e)\ i : i \mid \text{while } (i)$
séquence $S ::= \emptyset \mid i; S$

Sémantique des séquences : $\langle S \mid l \rangle \to l'$

$$\frac{\langle i \mid l \rangle \to l_1 \quad \langle S \mid l_1 \rangle \to l_2}{\langle i;\ S \mid l \rangle \to l_2}$$

Sémantique des instructions : $\langle i \mid l \rangle \to l'$

$$\frac{\langle e \mid l \rangle \to n}{\langle x := e \mid l \rangle \to l[x \leftarrow n]}$$

Sémantique du if (Big Step)

$$\frac{\langle e \mid l\rangle \to 0 \quad \langle S_f \mid l\rangle \to l'}{\langle ?(e)\ S_t : S_f \mid l\rangle \to l'}$$

$$\frac{\langle e \mid l\rangle \to n \neq 0 \quad \langle S_f \mid l\rangle \to l'}{\langle ?(e)\ S_t : S_f \mid l\rangle \to l'}$$

Sémantique du while (Big Step)

$$\frac{\langle e \mid l \rangle \to 0}{\langle \text{while } (e)\ S \mid l \rangle \to l}$$

$$\frac{\langle e \mid l \rangle \to n \neq 0 \quad \langle S \mid l \rangle \to l_1 \quad \langle \text{while } (e)\ S \mid l_1 \rangle \to l_2}{\langle \text{while } (e)\ S \mid l \rangle \to l_2}$$

Remarque : On ne peut parler que des programmes qui terminent

Sémantique opérationnelle (Small Step)

L'exécution des programmes se fait pas à pas :

- Un pas d'exécution : $\langle i; S \mid l \rangle \hookrightarrow \langle S' \mid l' \rangle$
- Plusieurs pas d'exécution : $\langle S \mid l \rangle \overset{\star}{\hookrightarrow} \langle S' \mid l' \rangle$
- Le programme termine si : $\langle S \mid l \rangle \overset{\star}{\hookrightarrow} \langle \emptyset \mid l' \rangle$

$$\langle S_1 \mid l_1 \rangle \hookrightarrow \langle S_2 \mid l_2 \rangle \hookrightarrow \ldots \hookrightarrow \langle \emptyset \mid l_n \rangle$$

- Les programme qui ne termine pas correspondent aux séquences infinies :

$$\langle S_1 \mid l_1 \rangle \hookrightarrow \langle S_2 \mid l_2 \rangle \hookrightarrow \ldots \hookrightarrow \langle S_n \mid l_n \rangle \hookrightarrow \ldots$$

Sémantique opérationnelle (Small Step)

Exécution d'une affectation :

$$\frac{\langle e \mid l \rangle \rightarrow n}{\langle x := e;\ \ S \mid l \rangle \hookrightarrow \langle S \mid l[x \leftarrow n] \rangle}$$

Sémantique opérationnelle (Small Step)

Exécution d'une conditionnelle :

- cas true :

$$\frac{\langle e \mid l \rangle \rightarrow n \neq 0}{\langle ?(e)\ S_t\ :\ S_f;\ \ S \mid l \rangle \rightarrow \langle S_t; S \mid l \rangle}$$

- cas false :

$$\frac{\langle e \mid l \rangle \rightarrow 0}{\langle ?(e)\ S_t\ :\ S_f;\ \ S \mid l \rangle \rightarrow \langle S_f; S \mid l \rangle}$$

Sémantique opérationnelle (Small Step)

Exécution d'une boucle :

- entrée :

$$\frac{\langle e \mid l \rangle \rightarrow n \neq 0}{\langle \text{while } (e)\ S_b;\ S \mid l \rangle \rightarrow \langle S_b;\ \text{while } (e)\ S_b;\ S \mid l \rangle}$$

- sortie :

$$\frac{\langle e \mid l \rangle \rightarrow 0}{\langle \text{while } (e)\ S_b;\ S \mid l \rangle \rightarrow \langle S \mid l \rangle}$$

Logique de Hoare

$$\text{Propositions} \quad A \ ::= \ e = e \mid \neg A \mid A \wedge A \mid A \mid A \Rightarrow A$$

Dérivation logique : $\vdash A$

- Les dérivations de Hoare sont des triplets $\{P\}i\{Q\}$
- P est une proposition : la pré-condition
- Q est une proposition : la post-condition

Intuition : Si P est valide avant l'exécution de i alors Q est valide après l'exécution de i.

Règle de la logique de Hoare

$$\frac{}{\{P(e)\}x := e\{P(x)\}} \qquad \frac{\{P\}i\{R\} \quad \{R\}S\{Q\}}{\{P\}i; S\{Q\}}$$

$$\frac{\{P \wedge e \neq 0\}S_t\{Q\} \quad \{P \wedge e = 0\}S_f\{Q\}}{\{P\}?(e)\ S_t\ :\ S_f\{Q\}}$$

$$\frac{\vdash P \Rightarrow I \quad \{I \wedge e \neq 0\}S\{I\} \quad \vdash I \wedge e = 0 \Rightarrow Q}{\{P\}\text{while } (e)\ S\{Q\}}$$

Difficulté principale : trouver les bons invariants de boucle

Sémantique des propositions de la logique de Hoare

- Les variables d'un programme itératif ont des valeurs modifiables
- Une assertion (pré ou post) de la logique de Hoare est une proposition logique, décrivant une propriété sur la valeur des variables dans un état mémoire donné

Interprétation des assertions dans un état mémoire : $[\![A]\!]_l$

$$
\begin{array}{lcl}
[\![n]\!]_l & = & n \\
[\![x]\!]_l & = & l[x] \\
[\![e_1 + e_2]\!]_l & = & [\![e_1]\!]_l + [\![e_2]\!]_l \\
[\![e_1 * e_2]\!]_l & = & [\![e_1]\!]_l * [\![e_2]\!]_l \\
 & & \\
[\![e_1 = e_2]\!]_l & = & [\![e_1]\!]_l = [\![e_2]\!]_l \\
[\![\neg A]\!]_l & = & \neg [\![A]\!]_l \\
[\![A \wedge B]\!]_l & = & [\![A]\!]_l \wedge [\![B]\!]_l \\
[\![A \vee B]\!]_l & = & [\![A]\!]_l \vee [\![B]\!]_l \\
[\![A \Rightarrow B]\!]_l & = & [\![A]\!]_l \Rightarrow [\![B]\!]_l
\end{array}
$$

Correction partielle

On veut montrer :

$$\left.\begin{array}{l}\{P\}S\{Q\}\\ \vdash [\![P]\!]_l\\ \langle S \mid l\rangle \rightarrow l'\end{array}\right\} \Rightarrow \vdash [\![P]\!]_{l'}$$

Preuve par induction(récurrence) sur $\langle S \mid l\rangle \rightarrow l'$

En réalité par induction mutuelle sur $\langle S \mid l\rangle \rightarrow l'$ et $\langle i \mid l\rangle \rightarrow l'$

Cas de l'affectation

$$\left.\begin{array}{l} \dfrac{\langle e \mid l\rangle \to n}{\langle x := e \mid l\rangle \to l[x \leftarrow n]} \\[2ex] \dfrac{}{\{P(e)\}x := e\{P(x)\}} \\[2ex] \vdash [\![P(e)]\!]_l \end{array}\right\} \Rightarrow [\![P(x)]\!]_{l[x\leftarrow n]}$$

On montre que :

- Si $\langle e \mid l\rangle \to n$ alors $[\![e]\!]_l = n$ (par induction sur e)
- Si $[\![e]\!]_l = n$ alors $[\![P(e)]\!]_l = [\![P(x)]\!]_{l[x\leftarrow n]}$ (par induction sur P)

Séquence

$$\left.\begin{array}{l} \dfrac{\langle i \mid l\rangle \rightarrow l_1 \quad \langle S \mid l_1\rangle \rightarrow l_2}{\langle i; S \mid l\rangle \rightarrow l_2} \\[2ex] \dfrac{\{P\}i\{R\} \quad \{R\}S\{Q\}}{\{P\}i; S\{Q\}} \\[2ex] \vdash [\![P]\!]_l \end{array}\right\} \Rightarrow [\![Q]\!]_{l_2}$$

Par hypothèses d'induction on a :

- $\{P\}i\{R\} \wedge \vdash [\![P]\!]_l \wedge \langle i \mid l\rangle \rightarrow l_1 \Rightarrow \vdash [\![R]\!]_{l_1}$
- $\{R\}S\{Q\} \wedge \vdash [\![R]\!]_{l_1} \wedge \langle S \mid l_1\rangle \rightarrow l_2 \Rightarrow \vdash [\![Q]\!]_{l_2}$

If (cas false)

$$\left.\begin{array}{c} \dfrac{\langle e \mid l\rangle \to 0 \quad \langle S_f \mid l\rangle \to l'}{\langle ?(e)\ S_t : S_f \mid l\rangle \to l'} \\ \\ \dfrac{\{P \wedge e \neq 0\} S_t \{Q\} \quad \{P \wedge e = 0\} S_f \{Q\}}{\{P\} ?(e)\ S_t : S_f \{Q\}} \\ \\ \vdash [\![P]\!]_l \end{array}\right\} \Rightarrow [\![Q]\!]_{l'}$$

- Si $\langle e \mid l\rangle \to n$ alors $\vdash [\![e]\!]_l = n$
- On a donc $\vdash [\![e = 0]\!]_l$, d'où $\vdash [\![P \wedge e = 0]\!]_l$
- Par hypothèse de récurrence on obtient $\vdash [\![Q]\!]_{l'}$

Cas true : on doit montrer que si $\langle e \mid l\rangle \to n \neq 0$ alors $\vdash [\![e \neq n]\!]_l$

Sortie de boucle

$$\left.\begin{array}{c} \dfrac{\langle e \mid l\rangle \to 0}{\langle \text{while } (e)\ S \mid l\rangle \to l} \\[2ex] \dfrac{\vdash P \Rightarrow I \quad \{I \wedge e \neq 0\} S \{I\} \quad \vdash I \wedge e = 0 \Rightarrow Q}{\{P\}\text{while } (e)\ S\{Q\}} \\[2ex] \vdash [\![P]\!]_l \end{array}\right\} \Rightarrow \vdash [\![Q]\!]_l$$

Remarques :

- Dans $\vdash A$ les variables du programme sont implicitement quantifiées
- Donc si $\vdash A$ alors pour tout état mémoire l on a $\vdash [\![A]\!]_l$

Entrée de boucle

$$\left.\begin{array}{c} \dfrac{\langle e \mid l\rangle \to n \neq 0 \quad \langle S \mid l\rangle \Rightarrow l_1 \quad \langle \text{while } (e)\ S \mid l_1\rangle \to l_2}{\langle \text{while } (e)\ S \mid l\rangle \to l_2} \\[2ex] \dfrac{\vdash P \Rightarrow I \quad \{I \wedge e \neq 0\} S \{I\} \quad \vdash I \wedge e = 0 \Rightarrow Q}{\{P\}\text{while } (e)\ S\{Q\}} \\[2ex] \vdash [\![P]\!]_l \end{array}\right\} \Rightarrow \vdash [\![Q]\!]_{l_2}$$

Correction partielle - correction totale

- $\{P\}S\{Q\}$ ne donne aucune indication sur la terminaison de S, Q n'est vrai que si le programme termine

- On dit qu'on a une correction partielle

Correction totale :

$$\left.\begin{array}{l}\{P\}S\{Q\}\\ \\ \vdash [\![P]\!]_l\end{array}\right\} \Rightarrow \exists n,\ l',\ \left\{\begin{array}{l}\langle S \mid l\rangle \rightarrow l'\\ \\ \vdash [\![Q]\!]_{l'}\end{array}\right.$$

Pour cela il faut ajouter des variants dans les boucles

Problème de la logique de Hoare

- Il faut implémenter un prouver pour la logique de Hoare
- Dans le cadre du PCC, les preuves sont grosses (dérivation de Hoare + dérivation logique)
- Comment inférer la pré-condition ?

Le calcul de plus faible pré-condition

Idée : étant donné un programme S et une post-condition, générer un ensemble de propositions et la plus faible pré-condition tel que :

- Si les propositions sont valides
- Si La pré-condition est valide dans un état initial l
- Si $\langle S \mid l \rangle \overset{\star}{\hookrightarrow} \langle \emptyset \mid l' \rangle$

Alors la post-condition est valide dans l'état l'.

Le générateur de conditions

Etant donné un programme S, une post-condition θ, écrire un programme $\mathsf{VCgen}(S, \theta)$ rendant une pré-condition (φ) et un ensemble de propositions (P) tel que si :

- Pour tout $\psi \in P$, $\vdash \psi$
- $\vdash [\![\varphi]\!]_l$
- $\langle i \mid l \rangle \overset{\star}{\hookrightarrow} \langle \emptyset \mid l' \rangle$

alors $\vdash [\![\theta]\!]_{l'}$

Un programme est correct si $\forall \psi \in P, \vdash \psi$

Problème :

En général, il est impossible d'inférer les invariants de boucle (exemple du pgcd)

Solution :

- demander à l'utilisateur de les rajouter manuellement !!!

$$\text{while } \{I\}(e)\ S$$

Le générateur de conditions

$$
\begin{aligned}
\mathsf{VCgen}(\emptyset,\ \theta) &= \theta,\ \emptyset \\
\mathsf{VCgen}(i; S,\ \theta) &= \varphi_1,\ P_1 \cup P_2 \\
&\quad (\varphi_2,\ P_2) := \mathsf{VCgen}(S,\ \theta) \\
&\quad (\varphi_1,\ P_1) := \mathsf{VCgen}(i,\ \varphi_2) \\
\mathsf{VCgen}(x := e,\ \theta) &= \theta[x \leftarrow e],\ \emptyset \\
\mathsf{VCgen}(?(e)\ S_t : S_f,\ \theta) &= e \neq 0 \Rightarrow \varphi_t \wedge e = 0 \Rightarrow \varphi_f,\ P_t \cup P_f \\
&\quad (\varphi_t,\ P_t) := \mathsf{VCgen}(S_t,\ \theta) \\
&\quad (\varphi_f,\ P_f) := \mathsf{VCgen}(S_f,\ \theta) \\
\mathsf{VCgen}(\mathsf{while}\ \{I\}(e)\ S,\ \theta) &= (I,\ \{C_S, C_\theta\} \cup P_S) \\
&\quad (\varphi_S,\ P_S) := \mathsf{VCgen}(S,\ I) \\
&\quad C_S := I \wedge e \neq 0 \Rightarrow \varphi_S \\
&\quad C_\theta := I \wedge e = 0 \Rightarrow \theta
\end{aligned}
$$

Preuve de correction

On veut prouver que :

$$\left.\begin{array}{l}(\varphi, P) = \mathsf{VCgen}(S, \theta) \\ \forall \psi \in P, \vdash \psi \\ \vdash [\![\varphi]\!]_l \\ \langle S \mid l \rangle \overset{\star}{\hookrightarrow} \langle \emptyset \mid l' \rangle\end{array}\right\} \Rightarrow \vdash [\![\theta]\!]_{l'}$$

On va prouver une version plus générale :

$$\left.\begin{array}{l}(\varphi, P) = \mathsf{VCgen}(S, \theta) \\ \forall \psi \in P, \vdash \psi \\ \vdash [\![\varphi]\!]_l \\ \langle S \mid l \rangle \hookrightarrow \langle S' \mid l' \rangle\end{array}\right\} \Rightarrow \forall \varphi' \ P', \left\{\begin{array}{l}(\varphi', P') = \mathsf{VCgen}(S', \theta) \\ \forall \psi \in P', \vdash \psi \\ \vdash [\![\varphi']\!]_{l'}\end{array}\right.$$

Par cas sur $\langle S \mid l \rangle \hookrightarrow \langle S' \mid l' \rangle$

Premier lemme intermédiaire

Si :

- $(\varphi, P) = \mathsf{VCgen}(S, \theta)$ et $\forall \psi \in P, \vdash \psi$
- $\vdash \theta \Rightarrow \theta'$
- $(\varphi', P') = \mathsf{VCgen}(S; \theta')$

alors $\vdash \varphi \Rightarrow \varphi'$ et $\forall \psi \in P', \vdash \psi$

Preuve par induction

Deuxième lemme intermédiaire

Si :

- $(\varphi_1, P_1) = \mathsf{VCgen}(S_1, \theta_1)$ et $\forall \psi \in P_1, \vdash \psi$
- $(\varphi_2, P_2) = \mathsf{VCgen}(S_2, \theta_2)$ et $\forall \psi \in P_2, \vdash \psi$
- $\vdash \theta_1 \Rightarrow \varphi_2$
- $(\varphi, P) = \mathsf{VCgen}(S_1;\ S_2; \theta_2)$

alors $\vdash \varphi_1 \Rightarrow \varphi$ et $\forall \psi \in P, \vdash \psi$

Preuve par induction sur S_1

Conclusion sur le WP

- Comme pour la logique de Hoare le correction est partielle : $(\varphi, P) = \mathsf{VCgen}(S, \theta)$, $\vdash [\![\varphi]\!]_l$ et $\forall \psi in P$, $\vdash \psi$ n'implique pas que l'évaluation de S termine

- On sait que si l'évaluation de S termine dans un état l' alors l' valide θ

- De plus, les invariants sont valides à chaque entrée et sortie de boucle

$$\langle S \mid l \rangle \overset{\star}{\hookrightarrow} \langle \mathsf{while}\ (I)\ e S_1;\ S_2 \mid l' \rangle \Rightarrow \vdash [\![I]\!]'_l$$

- De même, si on rajoute des assertions

Equivalence entre logique de Hoare et WP

Tout programme prouvable avec la logique de Hoare est prouvable avec le WP.

- Si $\{\varphi_1\}S\{\theta\}$ et $(\varphi_2, P) = \mathsf{VCgen}(S, \theta)$ alors $\vdash \varphi_1 \Rightarrow \varphi_2$ et $\forall \psi \in P, \vdash \psi$
- Si $(\varphi, P) = \mathsf{VCgen}(S, \theta)$ et $\forall \psi \in P, \vdash \psi$ alors $\{\varphi\}S\{\theta\}$

Exercice : le démontrer

Un calcul de WP pour le bytecode

valeurs	v	$::=$	$n \in \mathbb{N}$
instructions	i	$::=$	$\mathbf{push}\ n \mid \mathbf{add} \mid \mathbf{store}\ x \mid \mathbf{load}\ x \mid \mathbf{if}\ \mathsf{pc} \mid \mathbf{goto}\ \mathsf{pc}$
code	c	$:$	$\mathsf{pc} \mapsto i$
pile	s	$::=$	$[] \mid v : s$
mémoire	l	$:$	$x \mapsto v$
état	E	$::=$	$\langle c \mid \mathsf{pc} \mid s \mid l \rangle$

Sémantique

$$
\begin{array}{lcrcl}
\textbf{push } n & : & \langle c \mid \mathsf{pc} \mid s \mid l \rangle & \rightarrow & \langle c \mid \mathsf{pc} + 1 \mid n : s \mid l \rangle \\
\textbf{add} & : & \langle c \mid \mathsf{pc} \mid n_1 : n_2 : s \mid l \rangle & \rightarrow & \langle c \mid \mathsf{pc} + 1 \mid n_1 + n_2 : s \mid l \rangle \\
 & \vdots & & & \\
\textbf{store } x & : & \langle c \mid \mathsf{pc} \mid n : s \mid l \rangle & \rightarrow & \langle c \mid \mathsf{pc} + 1 \mid s \mid l[x \leftarrow n] \rangle \\
\textbf{load } x & : & \langle c \mid \mathsf{pc} \mid s \mid l \rangle & \rightarrow & \langle c \mid \mathsf{pc} + 1 \mid l[x] : s \mid l \rangle \\
 & \vdots & & & \\
\textbf{if } \mathsf{pc}_f & : & \langle c \mid \mathsf{pc} \mid 0 : s \mid l \rangle & \rightarrow & \langle c \mid \mathsf{pc}_f \mid s \mid l \rangle \\
\textbf{if } \mathsf{pc}_f & : & \langle c \mid \mathsf{pc} \mid n_{\neq 0} : s \mid l \rangle & \rightarrow & \langle c \mid \mathsf{pc} + 1 \mid s \mid l \rangle \\
 & \vdots & & & \\
\textbf{goto } \mathsf{pc}' & : & \langle c \mid \mathsf{pc} \mid s \mid l \rangle & \rightarrow & \langle c \mid \mathsf{pc}' \mid s \mid l \rangle
\end{array}
$$

Difficultés

- le langage n'est plus structuré
- que deviennent les invariants de boucle ?
- le langage de propositions doit permettre de parler des valeurs contenues dans la pile

Les propositions de la logique

Le langage de propositions doit permettre de parler des valeurs contenues dans la pile

$$\begin{array}{lrcl} \text{expressions} & e & ::= & n \mid x \mid S[n] \mid e + e \\ \text{propositions} & A & ::= & e = e \mid \neg A \mid A \wedge A \mid A \Rightarrow A \end{array}$$

Interprétation des propositions : $[\![A]\!]_l^s$

Comme pour while avec en plus

- $[\![S[0]]\!]_l^{n:s} = n$
- $[\![S[m+1]]\!]_l^{n:s} = [\![S[m]]\!]_l^s$
- $[\![S[m+1]]\!]_l^{[]}$ n'est pas défini

Suite le dernier cours

Exercice :

- Définir un WP pour le bytecode (sans les instructions de branchement)
- Définir un WP pour le bytecode (avec les instructions de branchement)