Sémantique de Hoare, Weakest Preconditions de Dijkstra

Jean Goubault-Larrecq

LSV/CNRS UMR 8643 & ENS Cachan

La sémantique de Hoare (langages impératifs, style C)

Une sémantique orientée vers la *preuve de programmes*.

Programmes:

$$P ::= x = e | P; P | \texttt{if}\ (e)\ \texttt{then}\ P\ \texttt{else}\ P | \texttt{while}\ (e)\ P$$

On va maintenant exprimer des propriétés du programme par des *formules* de la forme $\{F\}P\{G\}$, où F et G sont des formules de votre logique préférée (par ex., logique du premier ordre).

Intuition: $\{F\}P\{G\}$ = "si F est vraie avant d'exécuter P, et si P termine, alors G est vraie après".
On utilise x pour la valeur de x avant, x' pour après (dans G uniquement).

Règles de la sémantique de Hoare

$$\frac{}{\{\text{true}\}x = e\{x' = e \wedge y' = y \wedge z' = z \wedge \ldots\}}$$

$$\frac{\{F\}P\{G\} \quad \{G'\}Q\{H\}}{\{F\}P;Q\{\forall \vec{x}_1 \cdot G[\vec{x}' := \vec{x}_1] \wedge G'[\vec{x} := \vec{x}_1] \Rightarrow H[\vec{x} := \vec{x}_1]\}}$$

$$\frac{F_1 \Rightarrow F \quad G \Rightarrow G_1 \quad \{F\}P\{G\}}{\{F_1\}P\{G_1\}}$$

$$\frac{\{F \wedge e \neq 0\}P\{G\} \quad \{F \wedge e = 0\}Q\{G\}}{\{F\}\texttt{if}\ (e)\ \texttt{then}\ P\ \texttt{else}\ Q\{G\}}$$

$$\frac{\{I \wedge e \neq 0\}P\{I[\vec{x} := \vec{x}'] \wedge G\}}{\{F \wedge I\}\texttt{while}\ (e)\ P\{I[\vec{x} := \vec{x}'] \wedge G \wedge e = 0\}}$$

I est l'invariant de boucle.

Avantage: la règle classique de sémantique naturelle pour le `while` doit être appliquée autant de fois qu'on fait de tours dans la boucle; ici, une seule application de la règle permet de conclure.

En particulier, on peut conclure même quand la boucle ne termine pas.

Inconvénient: il faut inventer la formule $I\ldots$

Un petit exemple

```
// calcul de factorielle de i, resultat dans n
  n = 1; j=i;
  while (j>1) {
    n = n*j; j=j-1; }
```

Invariant possible: $\mathtt{n} \times \mathtt{j}! = \mathtt{i}!$.

Exercice: cet invariant ne permet pas de conclure le résultat souhaité. Proposer un renforcement.

Exercice: et si on n'avait pas recopié `i` dans `j` au début, qu'aurait-on pu prouver? Proposer une extension de la logique remédiant à ce problème.

Discussion

- Règles de preuves simples, gros avantage par rapport à la sémantique naturelle dans les boucles `while`.
- Permet effectivement de prouver de (petits) programmes.
- Notion importante d'invariant.
- Mais ne parle que de correction partielle; autrement dit, de ce qui arrive quand le programme termine. On ne peut pas prouver qu'un programme termine.

  Exercice: proposer une modification de la règle `while` qui règle ce problème.

Les “weakest preconditions” de Dijkstra

Idée approximative: étant donnés P et G, trouver la formule $F = wp(P, G)$ la plus faible telle que $\{F\}P\{G\}$.
Autrement dit, $\{F\}P\{G\}$ et pour toute formule F' telle que $\{F'\}P\{G\}$, alors $F' \Rightarrow F$.

(Note: on n’a plus de variables x' cependant.)

Une idée qui a eu du succès: correspond au calcul de Pre^* dans les systèmes de transition.

Permet de prouver la correction totale, en particulier de prouver la terminaison de programmes.

Curiosité: si P ne termine pas, alors $wp(P, G)$ est la formule `true`.
Autrement dit, un programme qui ne termine pas vérifie toutes les propriétés G.

Calcul de wp

$$
\begin{array}{rcl}
wp(x = e, G) & = & G[x := e] \\
wp(P; Q, G) & = & wp(P, wp(Q, G)) \\
wp(\texttt{if}\ (e)\ \texttt{then}\ P\ \texttt{else}\ Q, G) & = & (e = 0 \wedge wp(Q, G)) \vee (e \neq 0 \wedge wp(P, G)) \\
wp(\texttt{while}\ (e)\ P, G) & = & lfp(F \mapsto (e = 0 \wedge G) \vee (e \neq 0 \wedge wp(P, F)))
\end{array}
$$

Note: l'affectation c'est la substitution!

Note: lfp est l'opérateur "plus petit point fixe".

Plus petit point fixe

Intuitivement, si $\mathcal{F} : D \to D$, et $(D, \leq)$ a un élément minimal $\leq$, alors $lfp(\mathcal{F}) : D = \bigvee_{k\in\mathbb{N}} \mathcal{F}^k(\perp)$. (Ici, D est l'espace des formules et $\leq$ est l'implication.)

Exercice: vérifier qu'avec cette définition de lfp, la sémantique de `while` (e) P signifie qu'une boucle `while` est équivalente à la disjonction des exécutions de k tours de boucle, $k \in \mathbb{N}$.

Corollaire: en déduire que wp est une sémantique de correction *totale*; comment peut-on exprimer que P termine?

Thm: si $\mathcal{F}$ est continue, alors $lfp(\mathcal{F})$ est le *plus petit point fixe* de $\mathcal{F}$. I.e., $\mathcal{F}(lfp(\mathcal{F})) = lfp(\mathcal{F})$ et pour tout x tel que $\mathcal{F}(x) = x$ alors $x \geq lfp(\mathcal{F})$.
Preuve: exercice.

Exercice: montrer que $wp(P, \_)$ est continue pour tout programme P.

Un autre moyen de retrouver le wp des `while`

Vous avez fait le calcul, et $\bigvee_{k \in \mathbb{N}} \mathcal{F}^k(\bot)$ pour $\mathcal{F} = F \mapsto (e = 0 \wedge G) \vee (e \neq 0 \wedge wp(P, F))$ ne ressemble pas beaucoup à ce à quoi vous vous attendiez pour la sémantique de `while` (e) P, à savoir:

$$
\begin{array}{lll}
(e = 0 \wedge G) & \vee & (e \neq 0 \wedge wp(P, e = 0 \wedge G)) \\
 & \vee & (e \neq 0 \wedge wp(P, e \neq 0 \wedge wp(P, e = 0 \wedge G))) \\
 & \vee & \ldots
\end{array}
$$

(Mais vous pouvez montrer que c'est équivalent!)

Le wp des `while` en tant que point fixe, directement

Faisons autrement: normalement $\texttt{while}\ (e)\ P$ devrait avoir la même sémantique que $\texttt{if}\ (e)\ \texttt{then}\ (P; \texttt{while}\ (e)\ P)\ \texttt{else}\ \texttt{skip}$, où $wp(\texttt{skip}, G) = G$. Posons $F = wp(\texttt{while}\ (e)\ P, G)$, donc

$$\begin{aligned} F &= (e = 0 \wedge G) \vee (e \neq 0 \wedge wp(P; \texttt{while}\ (e)\ P, G)) \\ &= (e = 0 \wedge G) \vee (e \neq 0 \wedge wp(P, F)) \end{aligned}$$

Donc $wp(\texttt{while}\ (e)\ P, G)$ *est* un point fixe de $\mathcal{F}$.
Le fait que c'est le plus petit exprime la terminaison...

Plus petit point fixe, cas général

La notion de plus petit point fixe est vue comme plus fondamentale en informatique que la notion d'itération.

On définit donc $lfp(\mathcal{F})$ en général comme le plus petit point fixe (s'il existe), pas par $lfp(\mathcal{F}) : D = \bigvee_{k \in \mathbb{N}} \mathcal{F}^k(\perp)$.

Intérêt: on peut par exemple étendre l'ensemble des programmes P pour inclure des constructions monotones mais *non continues*, c'est-à-dire non calculables. Permet de laisser des pans de programmes sous forme de spécifications. C'est par exemple l'approche de langages de spécification comme B [Abrial].

Une note sur la logique nécessaire pour exprimer wp

On a besoin de la logique du premier ordre + d'une construction exprimant les points fixes.

On peut utiliser par ex: FO+lfp, la logique du second ordre (cf. théorème de Tarski, transparent suivant), la logique d'ordre supérieur, etc.

Dijkstra s'embêtait beaucoup moins: il ne fixe pas de syntaxe, mais prend tous les prédicats possibles portant sur les variables $x_1, \ldots, x_k$. Si ces variables prennent leurs valeurs dans un ensemble X, l'ensemble des prédicats possibles est juste $\mathbb{P}(X^k)$.

Note: $\mathbb{P}(X^k)$ est un *treillis complet* (cf. transparent suivant.)

Théorème de Tarski

Defn: $(D, \leq)$ est un *treillis complet* si et seulement si tout sous-ensemble X de D a une borne supérieure $\bigvee X$ et une borne inférieure $\bigwedge X$.

Thm: Soit $(D, \leq)$ un treillis complet, et $\mathcal{F} : D \to D$ monotone. Alors $\mathcal{F}$ a un plus petit point fixe.

(En général, l'ensemble des points fixes de $\mathcal{F}$ est un sous-treillis complet de D.)

Preuve : Soit $Post(\mathcal{F}) = \{x \mid \mathcal{F}(x) \leq x\}$ l'ensemble des post-points fixes de $\mathcal{F}$.
Posons $v = \bigwedge Post(\mathcal{F})$.
Pour tout $x \in Post(\mathcal{F})$, $v \leq x$ (déf. de v). Donc $\mathcal{F}(v) \leq \mathcal{F}(x)$ ($\mathcal{F}$ monotone).
Comme $\mathcal{F}(x) \leq x$ (x post-point fixe de $\mathcal{F}$), $\mathcal{F}(v) \leq x$. Comme x est arbitraire dans $Post(\mathcal{F})$, $\mathcal{F}(v) \leq \bigwedge Post(\mathcal{F}) = v$ (a).
Par (a), et $\mathcal{F}$ monotone, $\mathcal{F}(\mathcal{F}(v)) \leq \mathcal{F}(v)$. Donc $\mathcal{F}(v) \in Post(\mathcal{F})$. Comme v minore $Post(\mathcal{F})$, $v \leq \mathcal{F}(v)$ (b).
Par (a), (b) et $\leq$ antisymétrique, $v = \mathcal{F}(v)$: v est un point fixe de $\mathcal{F}$.

Si $\mathcal{F}(x) = x$, $x \in Post(\mathcal{F})$, donc $v \leq x$: v est le plus petit point fixe de $\mathcal{F}$.

Exercices

Exercice: (pour ceux qui suivent le cours de logique) en déduire une sémantique pour FO+lfp, la logique du premier ordre avec opérateur $lfp(X \mapsto F)$.

Exercice: déduire de la construction de la preuve du théorème de Tarski que les points fixes de fonctions monotones sont définissables en logique du second ordre. Autrement dit, soit $F(X)$ une formule du premier ordre portant sur une variable X du second ordre (variable de prédicat). Montrer qu'il existe une formule $lfp(F)$ de la logique du second ordre telle que $F(lfp(F)) \Leftrightarrow lfp(F)$ et pour toute formule G telle que $F(G) \Leftrightarrow G$, alors $lfp(F) \Rightarrow G$.

Exercice: (pour les matheux) montrer le théorème de Schröder-Bernstein: si A et B sont deux ensembles, $f : A \to B$ et $g : B \to A$ sont injectives, alors A et B sont en bijection. (Indication: utiliser $\mathbb{P}(A) \times \mathbb{P}(B)$ avec ordre $\subseteq \times \supseteq$ comme treillis complet.)

Exercice: (pour les chercheurs) comment peut-on étendre la sémantique des wp aux programmes mini-Caml?