Logique de séparation

6 janvier 2010

Le but du projet est de modéliser dans Coq la logique de séparation qui permet de raisonner sur l'organisation de la mémoire. On prendra comme base l'article de O'Hearn, Reynolds et Yang [ORY01]. Des formalisations de la logique de séparation ont été réalisées en Coq on pourra en particulier se référer à [AB07, MAY06], même s'il n'est pas nécessaire de comprendre ces travaux pour réaliser le projet.

1 Langage

Le langage étudié est un langage de bas niveau dans lequel on ne manipule que des entiers qui seront interprétés indifféremment comme des valeurs directes ou des adresses.

Les variables de ce langage sont représentées par le type `var` que l'on pourra définir comme le type des entiers.

La mémoire est représentée par un environnement S qui associe des valeurs à des variables et un tas H qui associe des valeurs à des adresses.

1.1 Raisonnement sur les environnements et les tas

On se donne un module `Map` dans le fichier `fmap.v` qui permet de représenter des ensembles d'associations finis avec les opérations `empty`, `dom`, `find`, `add`, `new`, `equiv`, `remove` ...

Le fichier `ListSet` des théories de Coq(répertoire `Lists`) permet de représenter des ensembles finis comme des listes. Il pourra être utilisé pour représenter des ensembles de variables et contient les opérations : `empty_set`, `set_union`, `set_add`, `set_In` ...

Question Définir la relation $h \sharp h'$ qui exprime que les domaines de deux ensembles d'associations h et h' sont disjoints et $h * h'$ qui construit l'ensemble d'associations union de h et h' (supposés disjoints).

1.2 Expressions

Le langage d'expressions est donné par la grammaire suivante :

$$E ::= \texttt{var} \mid \mathbb{Z} \mid E + E \mid E - E \mid E \times E$$

Question Définir le type Coq `expr` correspondant aux expressions et programmer les fonctions `fvars_expr` qui calcule les variables libres d'une expression et `subst_expr` qui substitue une expression pour une variable.

1.3 Assertions

Le langage d'assertions est donné par la grammaire suivante :

$$\begin{aligned} A ::= \; & \texttt{true} \mid E = E \mid E < E \mid \neg A \mid A \Rightarrow A \mid A \wedge A \mid \forall \texttt{var}.A \mid \exists \texttt{var}.A \\ & \mid E \mapsto E \mid \mathbf{emp} \mid A * A \mid A \mathrel{-\!\!*} A \end{aligned}$$

Question

1. Définir le type Coq `assert` des assertions et les fonctions `fvars_assert` qui calcule les variables libres d'une assertion et `subst_assert` qui substitue une expression pour une variable dans une assertion.
2. Introduire les abbréviations $e \mapsto \_$ pour $\exists x, e \mapsto x$ et $e \mapsto e_0, \ldots, e_n$ pour $e \mapsto e_0 * \ldots e + n \;\mapsto\; e_n$.

1.4 Programmes

Le langage de base comporte cinq instructions atomiques qui permettent de mettre à jour une variable, d'accéder en lecture ou en écriture à un élément du tas identifié par une expression simple et d'allouer ou de désallouer un espace mémoire :

$$C \ ::= \ \texttt{var} := E \mid \texttt{var} := [E] \mid \texttt{var} := \mathbf{alloc}(E, \ldots, E) \mid \mathbf{dispose}(E) \mid [E] := E$$

Question Définir le type Coq `command` des programmes et les fonctions `fvars_command` qui calcule les variables libres d'une commande, `mod_command` qui calcule les variables modifiées dans une commande (ie qui apparaissent dans une commande `x:=...`) et `subst_command` qui substitue une expression e pour une variable x qui n'est pas modifiée et `subst_var_command` qui substitue une variable x (quelconque) par une variable y dans une commande.

2 Sémantique

2.1 Sémantique des expressions

Question Définir une fonction Coq `eval_expr` qui étant donnés une expression et un environnement renvoie la valeur de l'expression dans cet environnement (on remarque que le tas n'intervient pas dans cette sémantique car les expressions ne permettent pas de "suivre" les pointeurs dans le tas sans introduire des variables intermédiaires). On note $[\![e]\!]s$ la valeur entière de l'expression e dans l'environnement s. On pourra donner une valeur par défaut aux variables de l'expression e absentes de l'environnement.

2.2 Sémantique des assertions

Les assertions sont évaluées dans une mémoire composée d'un environnement et d'un tas. La définition de cette sémantique est donnée par les règles suivantes :

$$\begin{array}{lrcl}
[\![\texttt{true}]\!] & (s,h) & = & \texttt{True} \\
[\![e_1 = e_2]\!] & (s,h) & = & [\![e_1]\!]s = [\![e_2]\!]s \\
[\![e_1 < e_2]\!] & (s,h) & = & [\![e_1]\!]s < [\![e_2]\!]s \\
[\![\neg A]\!] & (s,h) & = & \neg [\![A]\!](s,h) \\
[\![A_1 \Rightarrow A_2]\!] & (s,h) & = & [\![A_1]\!](s,h) \Rightarrow [\![A_2]\!](s,h) \\
[\![A_1 \wedge A_2]\!] & (s,h) & = & [\![A_1]\!](s,h) \wedge [\![A_2]\!](s,h) \\
[\![\forall x.A]\!] & (s,h) & = & \forall v, [\![A]\!](s[v/x],h) \\
[\![\exists x.A]\!] & (s,h) & = & \exists v, [\![A]\!](s[v/x],h) \\
[\![e_1 \mapsto e_2]\!] & (s,h) & = & \texttt{dom}\, h = \{[\![e_1]\!]s\} \wedge h\,([\![e_1]\!]s) = [\![e_2]\!]s \\
[\![\mathbf{emp}]\!] & (s,h) & = & \texttt{dom}\, h = \emptyset \\
[\![a_1 * a_2]\!] & (s,h) & = & \exists h_1\ h_2, h_1 \sharp h_2 \wedge h = h_1 * h_2 \wedge [\![a_1]\!](s,h_1) \wedge [\![a_2]\!](s,h_2) \\
[\![a_1 \mathrel{-\!\!*} a_2]\!] & (s,h) & = & \forall h', (h' \sharp h \wedge [\![a_1]\!]s,h') \Rightarrow [\![a_2]\!](s,h * h')
\end{array}$$

Question

1. Définir par récurrence sur la structure de la formule a le prédicat $[\![a]\!]$(s,h).
2. Montrer que l'assertion $x \mapsto v$ pour une variable x et une valeur v n'est vraie dans un environnement s et un tas h que si le domaine de h est réduit à l'adresse $s\,x$ et qu'à cette adresse on trouve la valeur v alors que la formule $x \mapsto v * \texttt{true}$ est vraie pour tout tas h dont le domaine contient l'adresse $s\,x$ associée à la valeur v.

2.3 Sémantique des programmes

2.3.1 Sémantique opérationnelle

Chaque commande définit une transformation de la mémoire donnée par les règles suivantes :

$$\frac{s' = s[[\![e]\!]s/x]}{(s,h) \vdash x := e \rightsquigarrow (s',h)} \qquad \frac{h([\![e]\!]s) = m \quad s' = s[m/x]}{(s,h) \vdash x := [e] \rightsquigarrow (s',h)} \qquad \frac{h([\![e_1]\!]s) = m \quad h' = h[[\![e_2]\!]s/[\![e_1]\!]s]}{(s,h) \vdash [e_1] := e_2 \rightsquigarrow (s,h')}$$

$$\frac{[\![e]\!]s = l \quad h' = h \backslash l}{(s,h) \vdash \mathbf{dispose}(e) \rightsquigarrow (s,h')} \qquad \frac{l = \mathtt{new}\, h\, n \quad s' = s[l/x] \quad h' = h[[\![e_i]\!]s/l+i]_{i=0\ldots n}}{(s,h) \vdash x := \mathbf{alloc}(e_0,\ldots,e_n) \rightsquigarrow (s',h')}$$

Questions

1. Définir en Coq une relation décrivant cette sémantique.
2. Montrer les règles suivantes : (une assertion A sera interprétée comme la formule logique $\forall s\ h,\ [\![A]\!](s,h)$)

$$\frac{(P * R) \Rightarrow S}{P \Rightarrow (R \mathrel{-\!\!*} S)} \qquad \frac{P \Rightarrow (R \mathrel{-\!\!*} S) \quad Q \Rightarrow R}{(P * Q) \Rightarrow S}$$

2.3.2 Sémantique axiomatique

On définit une sémantique axiomatique sous la forme de triplets $\{P\}C\{Q\}$ avec P et Q des assertions comme la propriété :

$$\{P\}C\{Q\} \Leftrightarrow \forall s\, h\, s'\, h', [\![P]\!](s,h) \Rightarrow (s,h) \vdash C \rightsquigarrow (s',h') \Rightarrow [\![Q]\!](s',h')$$

Questions Énoncer en Coq et vérifier les règles sémantiques suivantes : (avec $m, n \in \mathbb{Z}$ des constantes)

$$\{x = m \wedge \mathbf{emp}\}x := e\{x = e[m/x] \wedge \mathbf{emp}\} \qquad \{x = m \wedge e \mapsto n\}x := [e]\{x = n \wedge e[m/x] \mapsto n\}$$

$$\{e_1 \mapsto \_\}[e_1] := e_2\{e_1 \mapsto e_2\} \qquad \{e \mapsto \_\}x := \mathbf{dispose}(e)\{\mathbf{emp}\}$$

$$\{x = m \wedge \mathbf{emp}\}x := \mathbf{alloc}(e_0,\ldots,e_n)\{x \mapsto e_0[m/x], \ldots e_n[m/x]\}$$

On pourra aussi vérifier les règles structurelles suivantes :

- Frame rule : $\dfrac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$ si R ne contient pas de variable libre qui sont modifiées dans C.
- Élimination de variable auxiliaire : $\dfrac{\{P\}C\{Q\}}{\{\exists x, P\}C\{\exists x, Q\}}$ si x n'est pas libre dans C.
- Substitution : $\dfrac{\{P\}C\{Q\}}{(\{P\}C\{Q\})[e/x]}$ si x n'est pas modifiée dans C ou bien si e est une variable.

Références

[AB07] Andrew W. Appel and Sandrine Blazy. Separation logic for small-step cminor. In Klaus Schneider and Jens Brandt, editors, *TPHOLs*, volume 4732 of *Lecture Notes in Computer Science*, pages 5–21. Springer, 2007.

[MAY06] Nicolas Marti, Reynald Affeldt, and Akinori Yonezawa. Formal verification of the heap manager of an operating system using separation logic. In Zhiming Liu and Jifeng He, editors, *ICFEM*, volume 4260 of *Lecture Notes in Computer Science*, pages 400–419. Springer, 2006.

[ORY01] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In Laurent Fribourg, editor, *CSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2001.