6. Separation-Logik

Ziel: Definition Separation-Logik als Erweiterung von Hoare-Logik um
- Regeln für Pointer
- die Frame-Rule
- unsere nächtigere Assertion-Sprache.

Definition:

Eine partielle Korrektheitsaussage oder Separation-Logik-Tripel hat die Form

$$\{A\}\, c\, \{B\},$$

wobei A und B Separation-Logik-Assertions sind und c ein Programm der Sprache W ist.

- Das Tripel ist gültig, geschrieben $\models \{A\}\, c\, \{B\}$, falls

$$\forall s, h \in \text{Stacks} \times \text{Heaps}.\ [\![A]\!]\, sh = 1 \Rightarrow \left[\begin{array}{l} (c, s, h) \not\longrightarrow^* \text{abort} \\ \wedge\ \forall s', h'.\ (c, s, h) \longrightarrow^* (s', h') \Rightarrow [\![B]\!]\, s'h' = 1. \end{array} \right]$$

Bemerkung:

Ein gültiges Separation-Logik-Tripel garantiert, dass das Programm nur aktive Adressen zugreift.
Hat man also einen Beweis vorliegen, braucht man keine Laufzeitchecks für Speicherfehler (sofern c nichts aufgerufen).

O'Hearn: "Well-specified programs don't go wrong."
Milner '78: "Well-typed programs cannot go wrong."; in Bezug auf ML.

6.1 Programmlogik

Ziel: Wie bei der Hoare-Logik geben wir ein Beweissystem an, dessen ableitbare Theoreme gültige SL-Tripel sind

Ansatz:
- Übernehmen alle Axiome und Regeln der Hoare-Logik.
- Füge Axiome für die neuen Heap-manipulierenden Befehle hinzu.
- Die Frame-Rule besprechen wir in 6.2.

6.5 Regeln für vorwärts und rückwärts Reasoning an.

Mutation $[a_1] := a_2$:

$$(\text{GMUTV}) \quad \frac{}{\{a_1 \mapsto \_ * B\}\ [a_1] := a_2\ \{a_1 \mapsto a_2 * B\}}$$

G = global, V = vorwärts

Wenn der durch a_1 beschriebene Referenz alloziert ist, kann ihr der von a_2 beschriebene Wert zugewiesen werden. und wir erhalten den zu erwartenden Heap als Postcondition. Weitere Zustandsteile B bleiben von der Mutation unangetastet.

$$(\text{GMUTR}) \quad \frac{}{\{a_1 \mapsto \_ * (a_1 \mapsto a_2 \mathrel{-\!\!*} B)\}\ [a_1] := a_2\ \{B\}}$$

Wie schon früher besprochen.
Die Separating-Implication filtert die Heaps heraus, die bei Zugabe von $a_1 \mapsto a_2$ B erfüllen.

Deallokation dispose a:

$$(\text{GDISP}) \quad \frac{}{\{a \mapsto \_ * B\}\ \text{dispose}\ a\ \{B\}}$$

Wenn a alloziiert ist, kann man es disposen
und erhält den resultierenden Heap.

Allokation $x := cons(a_1, \dots, a_n)$:

$$(GALLOCV)\quad \frac{}{\{B\}\ x := cons(a_1, \dots, a_n)\ \{\exists x'.\ x \mapsto a_1[x/x'], \dots, a_n[x/x'] * B[x/x']\}}$$

wobei $x' \neq x$ und weder in den a_i noch in B vorkommt.
Analog zur Bedingung der Stronget-Postcondition.

$$(GALLOCR)\quad \frac{}{\{\forall x'.\ x' \mapsto a_1, \dots, a_n \mathrel{-\!\!*} B[x/x']\}\ x := cons(a_1, \dots a_n)\ \{B\}}$$

wobei $x' \neq x$ und wieder frisch für die a_i und B.

Lookup $x := [a]$:

$$(GLOOKV)\quad \frac{}{\{\exists z.\ a \mapsto z * B(x, z)\}\ x := [a]\ \{\exists x'.\ a[x/x'] \mapsto x * B(x', x)\}}$$

mit z, x', x alle verschieden, $z, x' \notin fv(a)$, $z, x', x \notin fv(B)$.
Wir stellen uns $B(y_1, y_2)$ als eine Aussage mit zwei freien Variablen vor,
die geeignet ersetzt wird.
Wenn B für den alten Wert von x und den alten Wert, auf den a zeigt,
galt und wir machen das Lookup, dann gilt die Aussage:
Es gab einen Wert x' (nämlich den vorherigen Wert von x),
so dass $a[x/x']$ den aktuellen Wert von x hält.
Außerdem gilt für dieses x' und den aktuellen Wert von x B.

(GLOOKR) $\dfrac{}{\{\exists x'.\ a \mapsto x' * (a \mapsto x' \mathrel{-\!\!*} R[x'/x])\}\ x := [a]\ \{R\}}$

Viel einfacher.

Lemma (Ishtiaq & O'Hearn '01):

Die Rückwärts-Axiome liefern die Weakest-Precondition.

Beispiele:

(1) Konstruiere eine zyklische Struktur aus zwei Elementen mit relativer Adressierung.

(GALLOCV) $\{emp\}\ x := cons(u,u);\ \{x \mapsto u,u\}$

$y := cons(v,v);$ (GALLOCV)

$\{x \mapsto u,u * y \mapsto v,v\}$

(CONSEQUENCE) $\{x \mapsto u,\_ * y \mapsto v,\_\}$

(GMUTV) $[x+1] := y-x;$

$\{x \mapsto u, y-x * y \mapsto v,\_\}$

$[y+1] := x-y;$ (GMUTV)

$\{x \mapsto u, y-x * y \mapsto v, x-y\}$

(CONSEQUENCE) $\{\exists o.\ x \mapsto u, o * y \mapsto v, -o\}$

(2) Entferne den Kopf einer Liste:

$\{[lseg\ u\ \alpha]\ (i,h)\}$

$\{\exists j.\ i \mapsto u,j * [lseg\ \alpha]\ (j,h)\}$

$\{\underbrace{i \mapsto u}_{B'} * \exists j.\ \underbrace{\overset{a}{\overbrace{(i+1)}} \mapsto j}_{2} * [lseg\ \alpha]\ (j,h)\}$ (GLOOKV) j ist z, m ist x

$m := [i+1];$

(GDISP)
- $\{ i \mapsto u * i+1 \mapsto m * [lseg\ \alpha]\,(m, k) \}$
- dispose i;
- $\{ i+1 \mapsto m * [lseg\ \alpha]\,(m, k) \}$
- dispose $i+1$;

(GDISP)

(ASSIGN)
- $\{ [lseg\ \alpha]\,(m, k) \}$
- $i := m;$
- $\{ [lseg\ \alpha]\,(i, k) \}$