Amortised Resource Analysis and Functional Correctness with Separation Logic (Part I)

Summer School on Formal Reasoning and Representation of Complex Systems

Ewen Maclean

Dependable Systems Group
School of Mathematical And Computer Sciences
Heriot-Watt University

14th August 2010

Motivation

- Quick introduction to Hoare Triples
- The problem of Loop Invariant Discovery
- Quick introduction to Separation Logic
- Inductive Heap Structures
- Example of a functional correctness proof
- Exercise for discovering functional specification

Hoare Triples

- Reason about **imperative programs**
- Annotate with precondition P and postcondition Q
- Achieve partial correctness via Hoare **Triple** $\{P\}\texttt{C}\{Q\}$
  – precondition P holds before implementation $\texttt{C}$, after which postcondition Q holds
  – Total correctness if termination can also be proved
- P and Q expressed in a language, e.g. FOL or HOL

Dealing with Imperative Constructs

- Assignment axiom:

$$\overline{\{P[E/x]\}\ x{:=}E\ \{P\}}$$

$e.g.\{x = 5\}x := x \times x\{x = 25\}$

- Composition Rule:

$$\frac{\{P\}\ C\ \{Q\}\ ,\ \{Q\}\ D\{R\}}{\{P\}\ C;D\ \{R\}}$$

- Conditional Rule:

$$\frac{\{B \wedge P\}\ C\ \{Q\}\ ,\ \{\neg B \wedge P\}\ D\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \{\ C\ \}\textbf{else}\ \{D\}\{Q\}}$$

- Consequence Rule:

$$\frac{P' \rightarrow P\ ,\ \{P\}\ C\ \{Q\}\ ,\ Q \rightarrow Q'}{\{P'\ \}\ C\ \{Q'\}}$$

Weakest Precondition and Strongest Postcondition

Valid Hoare Triples:

$$\{x = 5\} \quad x := x \times x \quad \{x > 0\}$$
$$\{x = 5\} \quad x := x \times x \quad \{x > 0 \land x < 30\}$$
$$\{x = 5\} \quad x := x \times x \quad \{x = 25\}$$ **StrongestPostcondition**

Weakest Precondition: if $\{P\}C\{Q\}$ and for each P': $\{P'\}C\{Q\}$ and $P' \rightarrow P$

P is the weakest precondition

Loop Invariants

- While Rule:

$$\frac{\{P \wedge B\}\ C\ \{P\}}{\{P\}\ \textbf{while}\ B\ \{\ C\ \}\ \{\neg B \wedge P\}}$$

- Loop Invariant:
  Here P is a **loop invariant** – a statement which is provable before and during the loop, and is strong enough to prove any postcondition.
- Meta Variables:
  In what follows, Loop Invariants are represented as functions $\mathcal{F}$ – higher order existentially quantified variables, if not instantiated.

Separation Logic

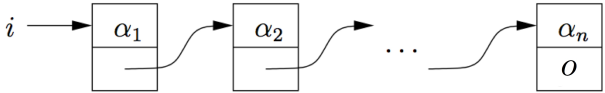
$P, Q ::=$	false	logical false
	$P \wedge Q$	classical conjunction
	$P \vee Q$	classical disjunction
	$P \rightarrow Q$	classical implication
	$P * Q$	separating conjunction
	$P \mathrel{-\!\!*} Q$	separating implication
	$E = E'$	expression equality
	$E \mapsto E'$	points to
	emp	the empty heap
	true	any heap
	$\exists x.P$	existential quantification

Semantics

Stack $S : Var \rightharpoonup Int$ and Heap $H : Loc \rightharpoonup Int$.

- $S, H \models P \wedge Q \;\longleftrightarrow\; S, H \models P \wedge S, H \models Q$
- $S, H \models E \mapsto E' \;\longleftrightarrow\; dom(H) = \{[E]_s\} \wedge H([E]_s) = [E']_s$
- $S, H \models P * Q \;\longleftrightarrow$
  $\exists H1, H2.(H1 \perp H2) \wedge (H1 \circ H2 = H) \wedge S, H_1 \models P \wedge S, H_2 \models Q$
- $S, H \models P \mathrel{-\!\!*} Q \;\longleftrightarrow$
  $\forall H'.(H \perp H') \wedge (S, H' \models P) \;\rightarrow\; S, H \circ H' \models Q$

Inductive Heap Structures



- Linked list structure:

$$
\begin{aligned}
i = o \rightarrow lseg(\alpha, i, o) \quad &\Rightarrow \quad \alpha = [] \\
i \neq o \rightarrow lseg(\alpha, i, o) \quad &\Rightarrow \quad \exists \alpha_h, \alpha_t, j . i \overset{data}{\mapsto} \alpha_h * i \overset{next}{\mapsto} j * \\
&\qquad lseg(\alpha_t, j, o) \wedge \alpha = \alpha_h :: \alpha_t
\end{aligned}
$$

- Field names:

$$
\begin{aligned}
i \rightarrow hd := E \quad &\Longleftrightarrow \quad i \overset{data}{\mapsto} E \\
i \rightarrow tl := E \quad &\Longleftrightarrow \quad i \overset{next}{\mapsto} E
\end{aligned}
$$

Extended Hoare Rules

- Lookup axiom (next field):

$$\overline{\{(\exists x'.\ (X \overset{next}{\mapsto} x') * (X \overset{next}{\mapsto} x' -\!\!* P))\}\ X := E \rightarrow tl\ \{P\}}$$

- Assignment axiom (next field):

$$\overline{\{(\exists x'.\ (X \overset{next}{\mapsto} x') * (X \overset{next}{\mapsto} E -\!\!* P))\}\ X \rightarrow tl := E\ \{P\}}$$

Program Example

```
{P}
j := nil
{R}
while !(i == nil) {
k := i.2;
i.2 := j;
j := i;
i := k;
}
{Q}
```

P : $lseg(\alpha_0, i, nil)$

Q : $lseg(rev(\alpha_0), j, nil)$

R : $\exists \alpha, \beta. lseg(\alpha, i, nil) * lseg(\beta, j, nil) \wedge P(\alpha_0, \alpha, \beta)$

Proof

$$\frac{(\exists x1.(\exists x2.(lseg(x1, l1, null) * (lseg(x2, l0, null) \wedge P(a, x1, x2)}{\begin{array}{l}((l1 = null \Rightarrow lseg(reverse(a), l0, null) \wedge (l1 \neq null \Rightarrow \\ (\exists x1.([l1 \overset{next}{\mapsto} x1] * ([l1 \overset{next}{\mapsto} x1] -\!\!* \\ (\exists x0.([l1 \overset{next}{\mapsto} x0] * ([l1 \overset{next}{\mapsto} l0] -\!\!* \\ (\exists x2.(\exists x3.(lseg(x2, x1, null) * (lseg(x3, l1, null) \wedge \\ \qquad P(a, x2, x3)))))\end{array}}$$

Central Proof Rule

$$A \mathrel{-\!\!*} (A * B) \Rightarrow B$$

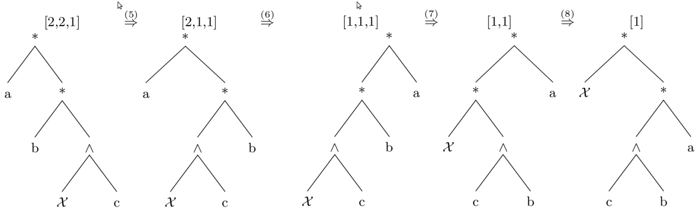
Mutation

$$\ldots(U \overset{X}{\mapsto} V) \quad \mathrel{-\!\!*} \quad ((\ldots) * (U' \overset{X}{\mapsto} V') * (\ldots))\ldots$$

$$\vdots \qquad \vdots$$

$$\ldots(U \overset{X}{\mapsto} V) \quad \mathrel{-\!\!*} \quad ((U' \overset{X}{\mapsto} V') * ((\ldots) * (\ldots)))\ldots$$

$$\ldots((\ldots) * (\ldots))\ldots$$



Invariant Branch

$$\frac{l1 \neq \mathit{null} \wedge (\mathit{lseg}(x1, l1, \mathit{null}) * (\mathit{lseg}(x2, l0, \mathit{null}) \wedge P(a, x1, x2)}{\begin{array}{l} \exists x1.([l1 \stackrel{\mathit{next}}{\mapsto} x1] * ([l1 \stackrel{\mathit{next}}{\mapsto} x1] \multimap * \\ (\exists x0.([l1 \stackrel{\mathit{next}}{\mapsto} x0] * ([l1 \stackrel{\mathit{next}}{\mapsto} l0] \\ \multimap * (\exists x2.(\exists x3.(\mathit{lseg}(x2, x1, \mathit{null}) * \\ ((\exists x4.(\exists x5.(\exists x6.((([l1 \stackrel{\mathit{data}}{\mapsto} x5] * ([l1 \stackrel{\mathit{next}}{\mapsto} x6] \\ * \mathit{lseg}(x4, x6, \mathit{null})) \wedge \mathit{cons}(x5, x4) = x6) \wedge P(a, x2, x3) \end{array}}$$

Final Entailment

$$\frac{l1 \neq null \wedge P(\mathcal{X}_a, \mathcal{X}_1, \mathcal{X}_2) \wedge cons(\mathcal{X}_4, \mathcal{X}_3) = \mathcal{X}_1}{cons(\mathcal{X}_4, \mathcal{X}_2) = \mathcal{F} \wedge P(\mathcal{X}_a, \mathcal{X}_2, \mathcal{F})}$$

Invariant Generation

Type constraint:

Predicate $P(a, b, c)$.

Variable constraint:

$\mathcal{X}_a$ always present in first position. Guess equality:

$$P \equiv \lambda x, y, z. x = F(y, z)$$

Theory constraint:

Post condition - rev, depends on append and cons.

Generate terms up to certain depth from:

$rev \quad append \quad [] \quad cons \quad \wedge \quad \vee$

Eliminate wrong using counter-example checker

e.g. $P \equiv \lambda x, y, z.\ x = append(z, y)$; $\mathcal{X}_4 = 0$; $\mathcal{X}_3 = [1, 2, 3]$; $\mathcal{X}_2 = [4, 5, 6]$:

$$[4, 5, 6, 0, 1, 2, 3] = [0, 4, 5, 6, 1, 2, 3]$$

Results

- Correct instantiaion is

$$P \equiv \lambda x, y, z.\ x = append(rev(z), y)$$

- Proof obligations:

$$append([], a) = a$$
$$append(rev(x), z :: y) = append(rev(z :: x), y)$$
$$rev(rev(x)) = x$$

Challenge Program

```
arguments: Pointers i,j and counter n
{P}
k := nil
l := nil
c := 0
{R}
while !(c<n) {
l := i;
k := i.2;
i.2 := j;
i := k;
j := l;
c:= c+1
}
{Q}
```

P: *lseg*(α, i, j) — What are *Q* and *R*?