COMP3610/6361
Principles of Programming Languages

Peter Höfner

Oct 17, 2023

Section 23

The Owicki-Gries Method

Motivation

- nondeterminism and concurrency required
- handle interleaving
- Floyd-Hoare logic only for sequential programs

- Owicki-Gries Logic/Method
  - a.k.a. interference freedom
  - Susan Owicki and PhD supervisor David Gries
  - add a construct to the programming language for threads
  - study the impact for Hoare triples

Floyd-Hoare Logic and Decorated Programs

Notation: *processes*: individual program
system: overall (concurrent) program will be

Floyd-Hoare logic

- each of the individual processes has an assertion
  - before its first statement (precondition)
  - between every pair of its statements (pre-/postcondition), and
  - after its last statement (postcondition)
- Hoare-triples can be checked (local correctness)
- Floyd-Hoare logic is compositional

Motivation

add pre- and postcondition for system, and a rule

$$\frac{\{P_1\}\ c_1\ \{Q_1\} \qquad \{P_2\}\ c_2\ \{Q_2\}}{\{P_1 \wedge P_2\}\ c_1 \parallel c_2\ \{Q_1 \wedge Q_2\}}$$

but this rule is incorrect

Note: we are considering an interleaving semantics

Australian National University

Simple Example

$$\{x == 0\}$$

$$\{x == 0 \lor x == 2\} \qquad\qquad \{x == 0 \lor x == 1\}$$

$$x := x + 1 \quad \Big\| \quad x := x + 2$$

$$\{x == 1 \lor x == 3\} \qquad\qquad \{x == 2 \lor x == 3\}$$

$$\{x == 3\}$$

What would we have to show?

The Rule of Owicki Gries

all rules of Floyd-Hoare logic remain valid

$$\frac{\{P_1\}\ c_1\ \{Q_1\} \ldots \{P_n\}\ c_n\ \{Q_n\} \qquad \textit{interference freedom}}{\{P_1 \wedge \cdots \wedge P_n\}\ c_1 \parallel \cdots \parallel c_n\ \{Q_1 \wedge \cdots \wedge Q_n\}}\ \text{(par)}$$

Interference Freedom

Interference freedom is a property of proofs of the $\{P_i\}\ c_i\ \{Q_i\}$

- suppose we have a proof for $\{P_i\}\ c_i\ \{Q_i\}$
- prove that the execution of any other statement c_j does not validate the reasoning for $\{P_i\}\ c_i\ \{Q_i\}$

it is a bit tricky

- interference freedom is a property of *proofs*, not Hoare triples
- identifying which parts of a proof need to be considered requires some effort

Formalising Interference Freedom

In a decorated program D and command c of the program, let

- pre(D, c) be the precondition (assumption/predicate) immediately before c, and
- post(D, c) the postcondition immediately after c
- remember $\{P\}\ c\ \{Q\}$ valid if there is a decorated program D with pre$(D, c) = P$ and post$(D, c) = Q$

Formalising Interference Freedom

$$\frac{\{P_1\}\ c_1\ \{Q_1\} \ldots \{P_n\}\ c_n\ \{Q_n\} \qquad \textit{interference freedom}}{\{P_1 \wedge \cdots \wedge P_n\}\ c_1 \parallel \cdots \parallel c_n\ \{Q_1 \wedge \cdots \wedge Q_n\}}\ (\text{par})$$

Suppose every c_i has a decorated program D_{c_i}.

Definition

D_{c_i} is *interference-free* with respect to D_{c_j} $(i \neq j)$ if for each statement c_i' in c_i and c_j' in c_j

- $\{\text{pre}(D_{c_i}, c_i') \wedge \text{pre}(D_{c_j}, c_j')\}\ c_j'\ \{\text{pre}(D_{c_i}, c_i')\}$
- $\{\text{post}(D_{c_i}, c_i') \wedge \text{pre}(D_{c_j}, c_j')\}\ c_j'\ \{\text{post}(D_{c_i}, c_i'))\}$

The D_{c_1}, D_{c_1}, $\ldots D_{c_n}$ are interference-free if they are pairwise interference-free with respect to one other.

Interference Freedom – Remark

- applying the Rule (par) requires the development of interference-free decorated programs for the c_i
- proving interference-freedom of D_{c_i} with respect to D_{c_j} focusses on
  - preconditions of each statement in c_i and postcondition of D_{c_i}

Simple Example

Why is interference freedom violated?

$$\{x == 0\}$$

$$\begin{array}{ccc} \{x == 0\} & & \{x == 0\} \\ x := x + 1 & \| & x := x + 2 \\ \{x == 1\} & & \{x == 1\} \end{array}$$

$$\{x == 1\}$$

Soundness

Theorem

If $\{P\}\ c\ \{Q\}$ *is derivable using the proof rules seen so far then* c *is valid*

Completeness

Can every correct Hoare triple be derived?

- completeness does not hold
- neither does relative completeness

Incompleteness

Lemma

The following valid Hoare triple cannot be derived using the rules so far.

$$\{\texttt{true}\} \quad x := x + 2 \parallel x := 0 \quad \{x == 0 \lor x == 2\}$$

Proof.

By contradiction. Suppose there were such a proof. Then there would be Q, R such that

$$\{\texttt{true}\}\ x := x + 2\ \{Q\}$$
$$\{\texttt{true}\}\ x := 0\ \{R\}$$
$$Q \land R \implies x == 0 \lor x == 2$$

By (assign) $\big(\{P[a/l]\}\ l := a\ \{P\}\big)$, $\texttt{true} \implies Q[x + 2/x]$ holds. Similarly, $R[0/x]$ holds.
By (par), $\{R \land \texttt{true}\}\ x := x + 2\ \{R\}$ holds, meaning $R \Rightarrow R[x + 2/x]$ is valid.
But then by induction, $\forall x.\ (x \geq 0 \land \text{even}(x)) \implies R$ is true. Since $Q \land R \implies x = 0 \lor x = 2$, it follows that

$$\forall x.\ (x \geq 0 \land \text{even}(x)) \implies (x == 0 \lor x == 2)\ ,$$

which is a contradiction. □

Fixing the Problem

We showed

- R must hold for all even, positive x
- R must hold after execution of $x := 0$
- R must also hold both before and after execution of $x := x + 2$

we need the capability in R to say that

until $x := x + 2$ is executed, $x = 0$ holds.

Auxiliary Variables

variables that are put into a program just to reason about progress in other processes

$$
\begin{array}{l}
\textsf{done} := 0\ ; \\
(\\
\qquad x, \textsf{done} := x + 2, 1 \\
\| \\
\qquad x := 0 \\
)
\end{array}
$$

- requires synchronous/atomic assignment
- proof is now possible

Australian National University

Decorated Programs with Auxiliary Variables

$\{\texttt{true}\}$
$\text{done} := 0$;
$\{\text{done} == 0\}$
$($
    $\{\text{done} == 0\}$
    $x, \text{done} := x + 2, 1$
    $\{\texttt{true}\}$
$\|$
    $\{\texttt{true}\}$
    $x := 0$
    $\{(x == 0 \vee x == 2) \wedge (\text{done} == 0 \Rightarrow x == 0)\}$
$)$
$\{c == 0 \vee x == 2\}$

Note: some implications skipped in the decorated program

Relative Completeness

- adding auxiliary variables enables proofs
- we do not want these variables to be in our code

$$\frac{\{P\}\ c\ \{Q\} \qquad x \text{ not free in } Q \qquad x \text{ auxiliary in } c}{\{P\}\ c'\ \{Q\}}\ (aux)$$

where c' is c with all references to x removed.

Theorem (Relative Completeness)

Adding Rules (par) and (aux) to the other rules of Floyd-Hoare logic yields a relatively complete proof system.

Problem

The Owicki-Griess Methods is *not compositional*.

Peterson's Algorithm for Mutual exclusion

the following 4 lines of (symmetric) code took 15 years to discover (mid 60's to early 80s)

let a, b be Booleans and $t : \{A, B\}$

$$\{\neg a \wedge \neg b\}$$

other code of A	other code of B
$a :=$ `true`	$b :=$ `true`
$t := A$	$t := B$
await $(\neg b \vee t == B)$	**await** $(\neg a \vee t == A)$
critical section A	critical section B
$a :=$ `false`	$b :=$ `false`

Notes on Peterson's Algorithm

- protects critical sections from mutual destructive interference
- guarantees fair treatment of A and B

- how do we show that A (or B) is never perpetually ignored in favour of B (A)?
  - requires *liveness* in this case
  - a topic for another course/research project
  - in fact there is one line that could potentially violate liveness (requires knowledge about hardware)

- 4 correct lines of code in 15 years is a coding rate of roughly

  1 LoC every 4 years

Yet Another Example

FindFirstPositive

$$i := 0 \; ; \; j := 1 \; ; \; x := |A| \; ; \; y := |A| \; ;$$

$$\begin{array}{lcl}
\textbf{while } i < \min(x,y) \textbf{ do} & & \textbf{while } j < \min(x,y) \textbf{ do} \\
\quad \textbf{if } A[i] > 0 \textbf{ then} & & \quad \textbf{if } A[j] > 0 \textbf{ then} \\
\quad\quad x := i & \Big\| & \quad\quad y := j \\
\quad \textbf{else} & & \quad \textbf{else} \\
\quad\quad i := i + 2 & & \quad\quad j := j + 2
\end{array}$$

$$r := \min(x, y)$$

$$i := 0\ ;\ j := 1\ ;\ x := |A|\ ;\ y := |A|\ ;$$
$$\{P_1 \wedge P_2\}$$

$\{P_1\}$
while $i < \min(x, y)$ **do**
  $\{P_1 \wedge i < x \wedge i < |A|\}$
  if $A[i] > 0$ **then**
    $\{P_1 \wedge i < x \wedge i < |A| \wedge A[i] > 0\}$
    $x := i$
    $\{P_1\}$
  else
    $\{P_1 \wedge i < x \wedge i < |A| \wedge A[i] \leq 0\}$
    $i := i + 2$
    $\{P_1\}$
  $\{P_1\}$
$\{P_1 \wedge i \geq \min(x, y)\}$

$\|$

$\{P_2\}$
while $j < \min(x, y)$ **do**
  $\{P_2 \wedge j < y \wedge j < |A|\}$
  if $A[j] > 0$ **then**
    $\{P_2 \wedge j < y \wedge j < |A| \wedge A[j] > 0\}$
    $y := j$
    $\{P_2\}$
  else
    $\{P_2 \wedge j < y \wedge j < |A| \wedge A[j] \leq 0\}$
    $j := j + 2$
    $\{P_2\}$
  $\{P_2\}$
$\{P_2 \wedge j \geq \min(x, y)\}$

$$\{P_1 \wedge P_2 \wedge i \geq \min(x, y) \wedge j \geq \min(x, y)\}$$
$$r := \min(x, y)$$
$$\{r \leq |A| \wedge (\forall k.\ 0 \leq k < r \Rightarrow A[k] \leq 0) \wedge (r < |A| \Rightarrow A[r] > 0)\}$$

$P_1 = x \leq |A| \wedge (\forall k.\ 0 \leq k < i \wedge k \text{ even} \Rightarrow A[k] \leq 0) \wedge i \text{ even} \wedge (x < |A| \Rightarrow A[x] > 0)$

$P_2 = y \leq |A| \wedge (\forall k.\ 0 \leq k < j \wedge k \text{ odd} \Rightarrow A[k] \leq 0) \wedge j \text{ odd} \wedge (y < |A| \Rightarrow A[y] > 0)$