Hoare Logic (II): Procedures and Concurrency

(Based on [Apt and Olderog 1991; Gries 1981; Lamport 1980; Owicki and Gries 1976; Slonneger and Kurtz 1995])

Yih-Kuen Tsay

(with help from Ming-Hsien Tsai)

Dept. of Information Management

National Taiwan University

Non-recursive Procedures

- We first consider procedures with *call-by-value* parameters (and *global* variables).
- Syntax:

$$\mathbf{proc}\ \mathrm{p}(\mathbf{in}\ x);\ S$$

where x may be a list of variables, S does not contain p, and S does not change x.

- Inference rule:

$$\frac{\{P\}\ S\ \{Q\}}{\{P[a/x] \wedge I\}\ \mathrm{p}(a)\ \{Q[a/x] \wedge I\}}$$

where a may not be a global variable changed by S and I does not refer to variables changed by S.

Non-recursive Procedures (cont.)

- We now consider procedures with *call-by-value*, *call-by-value-result*, and *call-by-result* parameters.
- Syntax:

$$\mathbf{proc}\ \mathrm{p}(\mathbf{in}\ x;\ \mathbf{in}\ \mathbf{out}\ y;\ \mathbf{out}\ z);\ S$$

  where x, y, z may be lists of variables, S does not contain p, and and S does not change x.
- Inference rule:

$$\frac{\{P\}\ S\ \{Q\}}{\{P[a,b/x,y] \wedge I)\}\ \mathrm{p}(a,b,c)\ \{Q[b,c/y,z] \wedge I\}}$$

  where b, c are (lists of) distinct variables, a, b, c may not be global variables changed by S, and I does not refer to variables changed by S.

Recursive Procedures

- A rule for recursive procedures without parameters:

$$\frac{\{P\}\ \mathrm{p}()\ \{Q\} \vdash \{P\}\ S\ \{Q\}}{\vdash \{P\}\ \mathrm{p}()\ \{Q\}}$$

where p is defined as "$\mathbf{proc}\ \mathrm{p}();\ S$".

- A rule for recursive procedures with parameters:

$$\frac{\forall v(\{P[v/x]\}\ \mathrm{p}(v)\ \{Q[v/x]\}) \vdash \{P\}\ S\ \{Q\}}{\vdash \{P[a/x]\}\ \mathrm{p}(a)\ \{Q[a/x]\}}$$

where p is defined as "$\mathbf{proc}\ \mathrm{p}(\mathbf{in}\ x);\ S$" and a may not be a global variable changed by S.

An Example

$$\begin{array}{l} \textbf{proc}\ \mathrm{nonzero}(); \\ \textbf{begin} \\ \quad \textbf{read}\ x; \\ \quad \textbf{if}\ x = 0\ \textbf{then}\ \mathrm{nonzero}()\ \textbf{fi}; \\ \textbf{end} \end{array}$$

- The semantics of "$\textbf{read}\ x$" is defined as follows:

$$\{IN = v \cdot L \wedge P[v/x]\}\ \textbf{read}\ x\ \{IN = L \wedge P\}$$

  where v is a single value and L is a stream of values.

- We wish to prove the following:

$\{IN = Z \cdot n \cdot L \wedge$ "Z contains only zeros" $\wedge\, n \neq 0\}$ // $\{P\}$
$\mathrm{nonzero}();$
$\{IN = L \wedge x = n \wedge n \neq 0\}$ // $\{Q\}$

An Example (cont.)

- It amounts to proving the following annotation:

$$
\begin{array}{l}
\textbf{proc}\ \mathrm{nonzero}();\\
\textbf{begin}\\
\quad \{IN = Z \cdot n \cdot L \wedge \text{“}Z \text{ contains only zeros”} \wedge n \neq 0\} \ \ /\!/\ \{P\}\\
\quad \textbf{read}\ x;\\
\quad \textbf{if}\ x = 0\ \textbf{then}\ \mathrm{nonzero}()\ \textbf{fi};\\
\quad \{IN = L \wedge x = n \wedge n \neq 0\} \ \ /\!/\ \{Q\}\\
\textbf{end}
\end{array}
$$

- The first step is to find a suitable assertion R between “$\textbf{read}\ x$” and the “$\textbf{if}$” statement.
- For this, we consider two cases: (1) Z is empty and (2) Z is not empty.

An Example (cont.)

- Case 1: Z is empty

  $\{IN = n \cdot L \wedge n \neq 0\}$

  $\textbf{read}\ x$

  $\{IN = L \wedge x = n \wedge n \neq 0\}$

- Case 2: Z is not empty

  $\{IN = 0 \cdot Z' \cdot n \cdot L \wedge \text{“}Z'\text{ contains only zeros”} \wedge n \neq 0\}$

  $\textbf{read}\ x$

  $\{IN = Z' \cdot n \cdot L \wedge \text{“}Z'\text{ contains only zeros”} \wedge n \neq 0 \wedge x = 0\}$

- Applying the Disjunction rule, we get a suitable R:

$$
\begin{aligned}
&(IN = L \wedge x = n \wedge n \neq 0) \vee \\
&(IN = Z' \cdot n \cdot L \wedge \text{“}Z'\text{ contains only zeros”} \wedge n \neq 0 \wedge x = 0)
\end{aligned}
$$

An Example (cont.)

- We now have to prove the following:

$$\{R\}\ \textbf{if}\ x = 0\ \textbf{then}\ \mathrm{nonzero}()\ \textbf{fi}\ \{IN = L \wedge x = n \wedge n \neq 0\}$$

- From the Conditional rule, this breaks down to
  - $\{R \wedge x = 0\}\ \mathrm{nonzero}()\ \{IN = L \wedge x = n \wedge n \neq 0\}$
  - $(R \wedge x \neq 0) \rightarrow (IN = L \wedge x = n \wedge n \neq 0)$ (obvious)
- The first case involving the recursive call simplifies to

$$\{IN = Z' \cdot n \cdot L \wedge \text{“}Z' \text{ contains only zeros”} \wedge n \neq 0 \wedge x = 0\}$$
$$\mathrm{nonzero}()$$
$$\{IN = L \wedge x = n \wedge n \neq 0\}$$

- The precondition is stronger than we need and $x = 0$ can be removed.

An Example (cont.)

- Finally, we are left with the following proof obligation:

$$\{IN = Z' \cdot n \cdot L \wedge \text{“}Z' \text{ contains only zeros”} \wedge n \neq 0 \wedge x = 0\}$$
$$\mathrm{nonzero}()$$
$$\{IN = L \wedge x = n \wedge n \neq 0\}$$

- The induction hypothesis gives us exactly the above.
- And, this completes the proof.

Termination of Recursive Procedures

- Consider the previous recursive procedure again.

  $\mathbf{proc}$ nonzero();
  $\mathbf{begin}$
  $\quad\mathbf{read}\ x$;
  $\quad\mathbf{if}\ x = 0\ \mathbf{then}$ nonzero() $\mathbf{fi}$;
  $\mathbf{end}$

- Given an input of the form $IN = L_1 \cdot n \cdot L_2$, where L_1 contains only zero values and $n \neq 0$, the command "nonzero()" will halt.
- We prove this *by induction* on the length of L_1.

Proving Termination by Induction

- Basis: $\mathrm{length}(L_1) = 0$
  - The input has the form $IN = n \cdot L_2$, where $n \neq 0$.
  - After "$\mathbf{read}\ x$", $x \neq 0$.
  - The boolean test $x = 0$ does not pass and the procedure call terminates.
- Induction step: $\mathrm{length}(L_1) = k > 0$
  - Hypothesis: $\mathrm{nonzero}()$ halts when $\mathrm{length}(L_1) = k - 1 \geq 0$.
  - Let $L_1 = 0 \cdot L_1'$.
  - The call $\mathrm{nonzero}()$ is invoked with $IN = 0 \cdot L_1' \cdot n \cdot L_2$, where L_1' contains only zero values and $n \neq 0$.

Proving Termination by Induction (cont.)

- Induction step (cont.)
  - After "$\textbf{read } x$", $x = 0$.
  - This boolean test $x = 0$ passes and a second call $\mathrm{nonzero}()$ is invoked inside the $\textbf{if}$ statement.
  - The second $\mathrm{nonzero}()$ is invoked with $L_1' \cdot n \cdot L_2$, where L_1' contains only zero values and $n \neq 0$
  - Since $\mathrm{length}(L_1') = k - 1$, termination is guaranteed by the hypothesis.

Proving Termination by Induction (cont.)

- A rule for proving termination of recursive procedures:

$$\frac{\{\exists u : W(u < T \wedge P(u))\}\ \mathrm{p}()\ \{Q\} \vdash \{P(T)\}\ S\ \{Q\}}{\vdash \{\exists t : W(P(t))\}\ \mathrm{p}()\ \{Q\}}$$

where

- $(W, <)$ is a well-founded set,
- p is defined as "$\textbf{proc}\ \mathrm{p}();\ S$", and
- T is a "rigid" variable that ranges over W and does not occur in P, Q, or S.

Sequential vs. Concurrent Programs

- Sequential programs (components) with the same input/output behavior may behave differently when executed in parallel with some other component.
- Consider two program components:

$$S_1 \triangleq x := x + 2 \;\text{ and }\; S_1' \triangleq x := x + 1; x := x + 1.$$

  Both increment x by 2.
- When executed in parallel with

$$S_2 \triangleq x := 0,$$

  S_1 and S_1' behave differently.

Sequential vs. Concurrent Programs (cont.)

Indeed,

$$\{true\}\ [S_1 \| S_2]\ \{x = 0 \vee x = 2\}$$

i.e.,

$$\{true\}\ [x := x + 2 \| x := 0]\ \{x = 0 \vee x = 2\}$$

but

$$\{true\}\ [S_1' \| S_2]\ \{x = 0 \vee x = 1 \vee x = 2\}$$

i.e.,

$$\{true\}\ [x := x + 1; x := x + 1 \| x := 0]\ \{x = 0 \vee x = 1 \vee x = 2\}.$$

Atomicity and Interleaving

- An action A (a statement or boolean expression) of a component is called *atomic* if during its execution no other components may change the variables of A.
- The computation of each component can be thought of as a sequence of executions of atomic actions.
- An atomic action is said to be *enabled* if its containing component is ready to execute it.
- Atomic actions enabled in different components are executed in an arbitrary sequential order; this is called the *interleaving* model.

Extending Hoare Logic

The best-known attempt at generalizing Hoare Logic to concurrent programs is:

> S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6:319-340, 1976.

- Proof outlines (for terminating programs)
- Interference freedom
- Auxiliary variables

Proof Outlines

Let S^* stand for a program S annotated with assertions. A proof outline (for partial correctness) is defined by the following formation rules.

$$\frac{}{\{P\}\ \textbf{skip}\ \{P\}} \qquad \text{(Skip)}$$

$$\frac{}{\{Q[E/x]\}\ x := E\ \{Q\}} \qquad \text{(Assignment)}$$

$$\frac{\{P\}\ S_1^*\ \{R\} \qquad \{R\}\ S_2^*\ \{Q\}}{\{P\}\ S_1^*; \{R\}\ S_2^*\ \{Q\}} \qquad \text{(Sequence)}$$

$$\frac{\{P \wedge B\}\ S_1^*\ \{Q\} \qquad \{P \wedge \neg B\}\ S_2^*\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ \{P \wedge B\}\ S_1^*\ \{Q\}\ \textbf{else}\ \{P \wedge \neg B\}\ S_2^*\ \{Q\}\ \textbf{fi}\ \{Q\}} \qquad \text{(Conditional)}$$

Proof Outlines (cont.)

$$\frac{\{P \wedge B\}\ S^*\ \{P\}}{\{\mathbf{inv} : P\}\ \mathbf{while}\ B\ \mathbf{do}\ \{P \wedge B\}\ S^*\ \{P\}\ \mathbf{od}\ \{P \wedge \neg B\}}\quad \textbf{(while)}$$

$$\frac{P \rightarrow P' \qquad \{P'\}\ S^*\ \{Q'\} \qquad Q' \rightarrow Q}{\{P\}\ \{P'\}\ S^*\ \{Q'\}\ \{Q\}}\quad \text{(Consequence)}$$

$$\frac{\{P\}\ S^*\ \{Q\}}{\{P\}\ S^{**}\ \{Q\}}\quad \text{(Omission)}$$

where S^{**} is obtained from S^* by omitting some of the intermediate assertions not labeled by **inv**.

A proof outline $\{P\}\ S^*\ \{Q\}$ is said to be *standard* if every subprogram T of S is preceded by exactly one assertion, called $pre(T)$, and there are no other assertions.

Atomic Regions

- We enclose multiple statements in a pair of "$\langle$" and "$\rangle$" to form *atomic regions* such as $\langle S_1; S_2 \rangle$, indicating that the enclosed statements are to be executed atomically.
- Proof rule:

$$\frac{\{P\}\ S\ \{Q\}}{\{P\}\ \langle S \rangle\ \{Q\}} \qquad \text{(Atomic Region)}$$

- Proof outline formation:

$$\frac{\{P\}\ S^*\ \{Q\}}{\{P\}\ \langle S^* \rangle\ \{Q\}} \qquad \text{(Atomic Region)}$$

- A proof outline with atomic regions is standard if every normal subprogram is preceded by exactly one assertion (and there are no other assertions).

Interference Freedom

- A standard proof outline $\{p_i\}\ S_i^*\ \{q_i\}$ *does not interfere* with another proof outline $\{p_j\}\ S_j^*\ \{q_j\}$ if the following holds:

  For every normal assignment or atomic region R in S_i and every assertion r in $\{p_j\}\ S_j^*\ \{q_j\}$,

$$\{r \wedge pre(R)\}\ R\ \{r\}.$$

- Given a parallel program $[S_1 \| \cdots \| S_n]$, the standard proof outlines $\{p_i\}\ S_i^*\ \{q_i\}$, $1 \leq i \leq n$, are said to be *interference free* if none of the proof outlines interferes with any other.

Interference Freedom (cont.)

- Proof rule:

$$\frac{\{p_i\}\ S_i^*\ \{q_i\},\ 1 \leq i \leq n, \text{ are standard and interference free}}{\{\bigwedge_{i=1}^{n} p_i\}\ [S_1 \| \cdots \| S_n]\ \{\bigwedge_{i=1}^{n} q_i\}}$$

An Example

$$\begin{array}{ll} \{x = 0\} & \{true\} \\ x := x + 2 & x := 0 \\ \{x = 2\} & \{x = 0\} \end{array}$$

are not interference free.

$$\begin{array}{ll} \{x = 0\} & \{true\} \\ x := x + 2 & x := 0 \\ \{x = 0 \lor x = 2\} & \{x = 0 \lor x = 2\} \end{array}$$

are interference free and yield

$$\{x = 0\}\ [x := x + 2 \| x := 0]\ \{x = 0 \lor x = 2\}.$$

An Example (cont.)

- Can we prove the following stronger claim?

$$\{true\}\ [x := x + 2 \| x := 0]\ \{x = 0 \vee x = 2\}$$

- This is not possible if we rely only on the proof rules introduced so far.
- It is easy to see that we must prove, for some q_1 and q_2,

$$\{true\}\ [x := x + 2]\ \{q_1\} \text{ and } \{true\}\ [x := 0]\ \{q_2\}.$$

  From $\{true\}\ [x := x + 2]\ \{q_1\}$, q_1 equals $true$ and hence q_2 along must imply $(x = 0 \vee x = 2)$.
  - From $\{true\}\ [x := 0]\ \{q_2\}$, $q_2[0/x]$ holds.
  - From $\{true \wedge q_2\}\ [x := x + 2]\ \{q_2\}$, $q_2 \to q_2[x + 2/x]$ holds.
  - By induction, q_2 holds for all even x's, a contradiction.

Auxiliary Variables

- A variable z in a program is called auxiliary if it only appears in assignments of the form $z := t$.
- Rule for auxiliary variables

$$\frac{\{p\}\ S\ \{q\}}{\{p\}\ S_0\ \{q\}} \qquad \text{(Auxiliary Variables)}$$

where S_0 is obtained from S by deleting some assignments with an auxiliary variable that does not occur free in q.

An Example (cont.)

$$\{\neg done\} \quad \langle x := x + 2; done := true\rangle \quad \{x = 0 \vee x = 2\}$$

$$\{true\} \quad x := 0 \quad \{(x = 0 \vee x = 2) \wedge (\neg done \rightarrow x = 0)\}.$$

are interference free and yield

$$\{\neg done\}$$
$$[\langle x := x + 2; done := true\rangle \| x := 0]$$
$$\{(x = 0 \vee x = 2) \wedge (\neg done \rightarrow x = 0)\}$$

The conjunct $(\neg done \rightarrow x = 0)$ can now be dropped (for our purpose).

An Example (cont.)

$$\{true\}$$
$$done := false;$$
$$\{\neg done\}$$
$$[\langle x := x + 2; done := true \rangle \| x := 0]$$
$$\{x = 0 \vee x = 2\}$$

from which we infer

$$\{true\}$$
$$[x := x + 2 \| x := 0]$$
$$\{x = 0 \vee x = 2\}.$$

The await Statement

- Syntax:

$$\textbf{await } B \textbf{ then } S \textbf{ end}$$

  The special case "$\textbf{await } B \textbf{ then } skip \textbf{ end}$" is simply written as "$\textbf{await } B$".

- Semantics:
  If B evaluates to $true$, S is executed as an atomic region and the component then proceeds to the next action. If B evaluates to $false$, the component is *blocked* and continues to be blocked unless B becomes $true$ later (because of the executions of other components).

The await Statement (cont.)

- Proof rule:

$$\frac{\{P \land B\}\ S\ \{Q\}}{\{P\}\ \textbf{await}\ B\ \textbf{then}\ S\ \textbf{end}\ \{Q\}} \qquad \textbf{(await)}$$

- Proof outline formation:

$$\frac{\{P \land B\}\ S^*\ \{Q\}}{\{P\}\ \textbf{await}\ B\ \textbf{then}\ \{P \land B\}\ S^*\ \{Q\}\ \textbf{end}\ \{Q\}} \qquad \textbf{(await)}$$

- For a proof outline to be standard, assertions within an **await** statement must be removed.

An Example with await

$\ldots$
$Q[0]$:= $true$;
$\mathbf{await}\ \neg Q[1]$;
/* critical section */
$Q[0]$:= $false$;
$\ldots$

$\ldots$
$Q[1]$:= $true$;
$\mathbf{await}\ \neg Q[0]$;
/* critical section */
$Q[1]$:= $false$;
$\ldots$

Note 1: This is the "first half" of Peterson's algorithm for two-process mutual exclusion.

Note 2: $Q[0]$ and $Q[1]$ are $false$ initially.

An Example with await (cont.)

$\{\neg Q[0]\}$	$\{\neg Q[1]\}$
$Q[0] := \mathit{true}$;	$Q[1] := \mathit{true}$;
$\{Q[0]\}$	$\{Q[1]\}$
$\textbf{await}\ \neg Q[1]$;	$\textbf{await}\ \neg Q[0]$;
$\{Q[0]\}$	$\{Q[1]\}$
$Q[0] := \mathit{false}$;	$Q[1] := \mathit{false}$;
$\{\neg Q[0]\}$	$\{\neg Q[1]\}$

Note: interference free, but not very useful . . . .
We should look for assertions at the two critical sections such that their conjunction results in a contradiction.

An Example with await (cont.)

$\{\neg Q[0]\}$
$Q[0] := true;$
$\{Q[0]\}$
$\mathbf{await}\ \neg Q[1];$
$\{Q[0] \wedge \neg Q[1]\}$
$Q[0] := false;$
$\{\neg Q[0]\}$

$\{\neg Q[1]\}$
$Q[1] := true;$
$\{Q[1]\}$
$\mathbf{await}\ \neg Q[0];$
$\{Q[1] \wedge \neg Q[0]\}$
$Q[1] := false;$
$\{\neg Q[1]\}$

Note: looks useful, but not interference free . . . .

An Example with await (cont.)

$\{\neg Q[0] \wedge \neg X[0]\}$
$\langle Q[0], X[0] := true, true; \rangle$
$\{Q[0] \wedge X[0]\}$
$\langle \textbf{await } \neg Q[1]; X[0] := false; \rangle$
$\{Q[0] \wedge \neg X[0] \wedge (\neg Q[1] \vee X[1])\}$
$Q[0] := false;$
$\{\neg Q[0] \wedge \neg X[0]\}$

$\{\neg Q[1] \wedge \neg X[1]\}$
$\langle Q[1], X[1] := true, true; \rangle$
$\{Q[1] \wedge X[1]\}$
$\langle \textbf{await } \neg Q[0]; X[1] := false; \rangle$
$\{Q[1] \wedge \neg X[1] \wedge (\neg Q[0] \vee X[0])\}$
$Q[1] := false;$
$\{\neg Q[1] \wedge \neg X[1]\}$

Note 1: "$\langle \textbf{await } \neg Q[0]; X[1] := false; \rangle$" is a shorter form for "$\textbf{await } \neg Q[0] \textbf{ then } X[1] := false \textbf{ end}$".

Note 2: conjoining the two assertions at the two critical sections gives the needed contradiction.

Lamport's 'Hoare Logic'

In this probably forgotten paper, Lamport proposed a new interpretation to pre and post-conditions:

L. Lamport. The 'Hoare Logic' of concurrent programs. *Acta Informatica*, 14:21-37, 1980.

- Notation: $\{P\}\ S\ \{Q\}$
  Meaning: If execution starts anywhere in S with P true, then executing S (1) will leave P true while control is in S and (2) if terminating, will make Q true.
- The usual Hoare triple would be expressed as $\{P\}\ \langle S\rangle\ \{Q\}$, where $\langle\cdot\rangle$ indicates atomic execution.

Lamport's 'Hoare Logic' (cont.)

- Rule of consequence (can't strengthen the pre-condition):

$$\frac{\{P\}\ S\ \{Q'\},\ Q' \to Q}{\{P\}\ S\ \{Q\}}$$

- Rules of Conjunction and Disjunction:

$$\frac{\{P\}\ S\ \{Q\},\ \{P'\}\ S\ \{Q'\}}{\{P \wedge P'\}\ S\ \{Q \wedge Q'\}} \qquad \frac{\{P\}\ S\ \{Q\},\ \{P'\}\ S\ \{Q'\}}{\{P \vee P'\}\ S\ \{Q \vee Q'\}}$$

Lamport's 'Hoare Logic' (cont.)

- Rule of Sequential Composition:

$$\frac{\{P\}\ S\ \{Q\},\ \{R\}\ T\ \{U\},\ Q \wedge at(T) \rightarrow R}{\{(in(S) \rightarrow P) \wedge (in(T) \rightarrow R)\}\ S;T\ \{U\}}$$

- Rule of Parallel Composition:

$$\frac{\{P\}\ S_i\ \{P\},\ 1 \leq i \leq n}{\{P\}\ \mathbf{cobegin}\ \overset{n}{\underset{i=1}{\|}}\ S_i\ \mathbf{coend}\ \{P\}}$$

References

- [Apt and Olderog 1991] *Verification of Sequential and Concurrent Programs*, K.R. Apt and E.-R. Olderog, Springer-Verlag, 1991.
- [Gries 1981] *The Science of Programming*, D. Gries, Springer-Verlag, 1981.
- [Lamport 1980] “The ‘Hoare Logic’ of concurrent programs”, L. Lamport, *Acta Informatica*, 14:21-37, 1980.
- [Owicki and Gries 1976] “An axiomatic proof technique for parallel programs I”, S. Owicki and D. Gries, *Acta Informatica*, 6:319–340, 1976
- [Slonneger and Kurtz 1995] *Formal Syntax and Semantics of Programming Languages*, K. Slonneger and B.L. Kurtz, Addison-Wesley, 1995.