

# Concurrency: Hoare Logic (III)

(Based on [Apt and Olderog 1997; Lamport 1980; Owicki and Gries 1976])

Yih-Kuen Tsay

Dept. of Information Management  
National Taiwan University

# 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:

$$S_1 \triangleq x := x + 2 \quad \text{and} \quad S'_1 \triangleq x := x + 1; x := x + 1.$$

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,

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

i.e.,

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

but

$$\{true\} [S'_1 \parallel S_2] \{x = 0 \vee x = 1 \vee x = 2\}$$

i.e.,

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




# 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.

$$\frac{}{\{P\} \text{ skip } \{P\}}$$

(Skip)

$$\frac{}{\{Q[E/x]\} x := E \{Q\}}$$

(Assignment)

$$\frac{\{P\} S_1^* \{R\} \quad \{R\} S_2^* \{Q\}}{\{P\} S_1^*; \{R\} S_2^* \{Q\}}$$

(Sequence)

$$\frac{\{P \wedge B\} S_1^* \{Q\} \quad \{P \wedge \neg B\} S_2^* \{Q\}}{\{P\} \text{ if } B \text{ then } \{P \wedge B\} S_1^* \{Q\} \text{ else } \{P \wedge \neg B\} S_2^* \{Q\} \text{ fi } \{Q\}}$$

(Conditional)

# Proof Outlines (cont.)

$$\{P \wedge B\} S^* \{P\}$$

(while)

---


$$\{\mathbf{inv} : P\} \mathbf{while} B \mathbf{do} \{P \wedge B\} S^* \{P\} \mathbf{od} \{P \wedge \neg B\}$$

$$P \rightarrow P' \quad \{P'\} S^* \{Q'\} \quad Q' \rightarrow Q$$

(Consequence)

---


$$\{P\} \{P'\} S^* \{Q'\} \{Q\}$$

$$\{P\} S^* \{Q\}$$

(Omission)

---


$$\{P\} S^{**} \{Q\}$$

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:

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

- Proof outline formation:

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

- 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  $\{p_j\} S_j^* \{q_j\}$ ,*

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

- Given a parallel program  $[S_1 \parallel \dots \parallel S_n]$ , 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.

🌐 Proof rule:

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

# An Example

$$\begin{array}{l} \{x = 0\} \\ x := x + 2 \\ \{x = 2\} \end{array}$$

$$\begin{array}{l} \{true\} \\ x := 0 \\ \{x = 0\} \end{array}$$

are not interference free.

$$\begin{array}{l} \{x = 0\} \\ x := x + 2 \\ \{x = 0 \vee x = 2\} \end{array}$$

$$\begin{array}{l} \{true\} \\ x := 0 \\ \{x = 0 \vee x = 2\} \end{array}$$

are interference free and yield

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

# An Example (cont.)

🌐 Can we prove the following stronger claim?

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

🌐 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  $q_2$ ,

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

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  $\{true \wedge q_2\} [x := x + 2] \{q_2\}$ ,  $q_2 \rightarrow q_2[x + 2/x]$  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

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

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.)

$$\begin{array}{ll} \{\neg done\} & \{true\} \\ \langle x := x + 2; done := true \rangle & x := 0 \\ \{true\} & \{(x = 0 \vee x = 2) \wedge (\neg done \rightarrow x = 0)\}. \end{array}$$

are interference free and yield

$$\begin{array}{l} \{\neg done\} \\ \langle x := x + 2; done := true \rangle \parallel x := 0 \\ \{(x = 0 \vee x = 2) \wedge (\neg done \rightarrow x = 0)\} \end{array}$$

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


## An Example (cont.)

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

from which we infer

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

# The await Statement

 Syntax:

**await  $B$  then  $S$  end**

The special case “**await  $B$  then skip end**” is simply written as “**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:

$$\frac{\{P \wedge B\} S \{Q\}}{\{P\} \text{await } B \text{ then } S \text{ end } \{Q\}} \quad (\text{await})$$

🌐 Proof outline formation:

$$\frac{\{P \wedge B\} S^* \{Q\}}{\{P\} \text{await } B \text{ then } \{P \wedge B\} S^* \{Q\} \text{ end } \{Q\}} \quad (\text{await})$$

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

# An Example with await

```
...  
Q[0] := true;  
await  $\neg Q[1]$ ;  
/* critical section */  
Q[0] := false;  
...
```

```
...  
Q[1] := true;  
await  $\neg Q[0]$ ;  
/* critical section */  
Q[1] := false;  
...
```

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.)

|                           |                           |
|---------------------------|---------------------------|
| $\{\neg Q[0]\}$           | $\{\neg Q[1]\}$           |
| $Q[0] := \text{true};$    | $Q[1] := \text{true};$    |
| $\{Q[0]\}$                | $\{Q[1]\}$                |
| <b>await</b> $\neg Q[1];$ | <b>await</b> $\neg Q[0];$ |
| $\{Q[0]\}$                | $\{Q[1]\}$                |
| $Q[0] := \text{false};$   | $Q[1] := \text{false};$   |
| $\{\neg Q[0]\}$           | $\{\neg Q[1]\}$           |

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.)

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

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

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

# An Example with await (cont.)

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

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


Note 1: “ $\langle \text{await } \neg Q[0]; X[1] := \text{false}; \rangle$ ” is a shorter form for “**await**  $\neg Q[0]$  **then**  $X[1] := \text{false}$  **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):

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

- 🌐 Rules of Conjunction and Disjunction:

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

# 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:

$$\frac{\{P\} S_i \{P\}, 1 \leq i \leq n}{\{P\} \text{cobegin} \parallel_{i=1}^n S_i \text{coend} \{P\}}$$



- 🌐 K.R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs*, Springer-Verlag, 1997.
- 🌐 L. Lamport. The 'Hoare logic' of concurrent programs, *Acta Informatica* 14, 21-37, Springer-Verlag, June 1980.
- 🌐 S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I, *Acta Informatica* 6, 319-340, Springer-Verlag, December 1976.