IN5170/IN9170 – Models of concurrency

Program Analysis II, lecture 6

Fall 2022

26.09.2022



Today

Analysis Part I: Reasoning about sequential programs.

Analysis Part II: How to reason about concurrent systems.

- Repetition of Rules
- Await statements
- Shared variables and interference
  - Concurrent execution
  - Interference freedom
  - Example: Producer / Consumer (await language)
- Monitors
  - invariants
  - Signal/wait
  - Example: reader/writer
  - Example: semaphores

Program Logic (PL)

- PL lets us *express* and *prove* properties about programs
- *Formulas* are on the form

"triple" (or "Hoare triple")

$$\{ P \} S \{ Q \}$$

- S: program statement(s)
- P and Q: assertions over program states
- P: Pre-condition
- Q: Post-condition

If we can use PL to prove some property of a program, then this property will hold for *all* executions of the program

PL rules from part I: structural inference rules

$$\frac{P' \Rightarrow P \qquad \{\, P \,\}\ S\ \{\, Q \,\} \qquad Q \Rightarrow Q'}{\{\, P' \,\}\ S\ \{\, Q' \,\}}\ \text{Consequence}$$

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

$$\frac{\{\, P \,\}\ S_1\ \{\, R \,\} \qquad \{\, R \,\}\ S_2\ \{\, Q \,\}}{\{\, P \,\}\ S_1; S_2\ \{\, Q \,\}}\ \text{Seq}$$

- Red: logical proof obligations

PL rules from part I: if/while/assignment

$$\frac{\{\ P \wedge B\ \}\ S\ \{\ Q\ \} \qquad P \wedge \neg B \Rightarrow Q}{\{\ P\ \}\ \text{if } B \text{ then } S \text{ fi}\ \{\ Q\ \}}\ \text{Cond1}$$

$$\frac{\{\ P \wedge B\ \}\ S\ \{\ Q\ \} \qquad \{\ P \wedge \neg B\ \}\ S'\ \{\ Q\ \}}{\{\ P\ \}\ \text{if } B \text{ then } S \text{ else } S'\ \{\ Q\ \}}\ \text{Cond2}$$

$$\{\ P[e/x]\ \}\ x := e\ \{\ P\ \}\quad \text{Assign}$$

$$\frac{\{\ I \wedge B\ \}\ S\ \{\ I\ \}}{\{\ I\ \}\ \text{while } B \text{ do } S\ \{\ I \wedge \neg B\ \}}\ \text{While}$$

Rule for await

$$\frac{\{\ P \wedge B\ \}\ S\ \{\ Q\ \}}{\{\ P\ \}\ \langle \text{await}\,(B)\ S \rangle\ \{\ Q\ \}}\ \text{Await}$$

Remember:
We are reasoning about safety properties/partial correctness

- termination is assumed/ignored
- the rule does not ensure waiting nor progress

Assume $\{ P_i \}\ S_i\ \{ Q_i \}$ for all $S_1, \ldots, S_n$

$$\frac{\{ P_i \}\ S_i\ \{ Q_i \} \quad \text{are } \textit{interference free}}{\{ P_1 \wedge \ldots \wedge P_n \}\ \text{co}\ S_1 \parallel \ldots \parallel S_n\ \text{oc}\ \{ Q_1 \wedge \ldots \wedge Q_n \}}\ \text{Co}$$

Interference freedom

A process interferes with (the specification of) another process, if its execution changes the assertions[a] of the other process.

[a] Only "critical assertions" considered, i.e. outside mutex-protected sections

- Assertions inside awaits: not endangered
- Critical assertions or critical conditions: assertions outside await statement bodies and mutex-protected sections.

Interference freedom

- S: statement in some process, with pre-condition $pre(S)$
- C: critical assertion in another process
- S does not interfere with C, if

$$\vdash \{\ C \wedge pre(S)\ \}\ S\ \{\ C\ \}$$

is derivable in PL (= theorem).

"C is *invariant* under the execution of the other process"

Avoiding interference: Global invariants

Global inductive invariants

- An assertion that only refers to global (shared) variables
- Holds initially.
- Preserved by all assignments/transitions ("inductive")

"Separation of concerns": We avoid interference if critical conditions are on the form $\{ I \wedge L \}$ where:

- I is a global invariant
- L only refers to local variables of the considered process

Example: Producer/ consumer synchronization

Let process `Producer` deliver data to a `Consumer` process

$PC : c \leq p \leq c + 1 \wedge (p = c + 1 \Rightarrow buf = a[p - 1])$

PC a *global inductive invariant* of the producer/consumer?

```
                int buf, p := 0; c := 0;

process Producer {                    process Consumer {
  int a[N];...                        int b[N];...
  while (p < N) {                     while (c < N) {
    < await (p = c) ; >                < await (p > c) ; >
     buf := a[p];                       b[c] := buf;
       p := p+1;                           c := c+1;
  }                                     }
}                                     }
```

Example: Producer

Loop invariant of Producer:
$I_P : PC \wedge p \leq n$

```
process Producer {
  int a[n];
  { I_P }                          //  entering loop
  while (p < n) {                  { I_P ∧ p < n }
    < await (p = c); >             { I_P ∧ p < n ∧ p = c }
                                   { I_P[p + 1/p][a[p]/buf] }
    buf := a[p];                   { I_P[p + 1/p] }
    p := p + 1;                    {I_P}
  } {I_P ∧ ¬(p < n)}               //  exit loop
     (⇔ {PC ∧ p = n})
}
```

Proof obligation:
$I_P \wedge p < n \wedge p = c \Rightarrow I_P[p + 1/p][a[p]/buf]$

Example: Consumer

Loop invariant of Consumer:
$I_C : PC \wedge c \leq n \wedge b[0 : c-1] = a[0 : c-1]$

```
process Consumer {
  int b[n];
  {I_C}                          // entering loop
  while (c < n) {                {I_C ∧ c < n}
    < await (p > c) ; >          {I_C ∧ c < n ∧ p > c}
                                 {I_C[c+1/c][buf/b[c]]}
    b[c] := buf;                 {I_C}[c+1/c]
    c := c + 1;                  {I_C}
  } {I_C ∧ ¬(c < n)}             // exit loop
     (⇔ {PC ∧ c = n ∧ b[0 : c-1] = a[0 : c-1]})
}
```

Proof Obligation:

$I_C \wedge c < n \wedge p > c \;\Rightarrow\; I_C[c+1/c][buf/b[c]]$

Example: Producer/Consumer

Assume that the processes are interference free (argumentation on next slides).

The final state of the program satisfies:

$$PC \wedge p = n \wedge c = n \wedge b[0 : c - 1] = a[0 : c - 1]$$

which ensures that all elements in a are received and occur in the same order in b

Example: Producer/Consumer

We need to show interference freedom.
Most assertions follow the global/local discipline and are therefore not critical. Nevertheless, there are some critical assertions.

Critical assertions of Producer:

Critical assertions of Consumer:

Example: Producer/Consumer

We need to show interference freedom.
Most assertions follow the global/local discipline and are therefore not critical. Nevertheless, there are some critical assertions.

Critical assertions of Producer:

$$C_P^1: \; I_p \wedge p < n \wedge p = c$$
$$C_P^2: \; I_p[p+1/p][a[p]/buf]$$
$$C_P^3: \; I_P[p+1/p]$$

Critical assertions of Consumer:

$$C_C^1: \; I_C \wedge c < n \wedge p > c$$
$$C_C^2: \; I_C[c+1/c][buf/b[c]]$$
$$C_C^3: \; I_C[c+1/c]$$

We need to check for interference freedom for all of them for all statements of the other process.

Example: Producer/Consumer

Example for *one* interference check:

- $C_P^1 : \ I_p \wedge p < n \wedge p = c$
- $S : \ c := c + 1$ with $pre(S) = I_C[c + 1/c]$

To test:

Example: Producer/Consumer

Example for *one* interference check:

- C_P^1 : $I_p \wedge p < n \wedge p = c$
- S : $c := c + 1$ with $pre(S) = I_C[c + 1/c]$

To test:

$$\overset{?}{\vdash} \{\ C_P^1 \wedge I_C[c + 1/c]\ \}\ c := c + 1\ \{\ C_P^1\ \}$$

Example: Producer/Consumer

Example for *one* interference check:

- C_P^1 : $I_p \wedge p < n \wedge p = c$
- S : $c := c + 1$ with $pre(S) = I_C[c + 1/c]$

To test:

$$\overset{?}{\vdash} \{ C_P^1 \wedge I_C[c + 1/c] \}\ c := c + 1\ \{ C_P^1 \}$$

Observation:

$$C_P^1 \wedge I_C[c + 1/c] \equiv (I_p \wedge p < n \wedge p = c) \wedge (c + 1 \leq p \leq c + 2 \wedge [...])$$
$$\Rightarrow \mathit{false}$$

Example: Producer/Consumer

Example for *one* interference check:

- C_P^1 : $I_p \wedge p < n \wedge p = c$
- S : $c := c + 1$ with $pre(S) = I_C[c + 1/c]$

To test:

$$\overset{?}{\vdash} \{\ C_P^1 \wedge I_C[c + 1/c]\ \}\ c := c + 1\ \{\ C_P^1\ \}$$

Observation:

$$C_P^1 \wedge I_C[c + 1/c] \equiv (I_p \wedge p < n \wedge p = c) \wedge (c + 1 \leq p \leq c + 2 \wedge [...])$$
$$\Rightarrow \mathit{false}$$

We can use Consequence rule:

$$\frac{C_P^1 \wedge I_C[c + 1] \Rightarrow \mathit{false} \qquad \{\ \mathit{false}\ \}\ c := c + 1\ \{\ C_P^1\ \}}{\{\ C_P^1 \wedge I_C[c + 1]\ \}\ c := c + 1\ \{\ C_P^1\ \}}\ \text{Consequence}$$

Similar for other interference checks $\Rightarrow$ processes do not interfere.

Example: Producer/Consumer

Summary:

- defined global invariant PC
- defined loop invariants
- verified invariants locally and calculated post-conditions
- showed interference freedom
- combining post-conditions leads to global post-condition

This showed, that the concurrent execution of producer and consumer transfers the elements from a to b in the same order.

Monitor invariant

```
monitor name {
   monitor variables
   initialization
   procedures
}
```

- A monitor invariant (I): describe the monitor's inner state
- Express relationship between monitor variables

Monitor invariant

```
monitor name {
   monitor variables
   initialization
   procedures
}
```

- A monitor invariant (I): describe the monitor's inner state
- Express relationship between monitor variables
- Maintained by execution of procedures:
  - Must hold after initialization
  - Must hold when a procedure terminates

Monitor invariant

```
monitor name {
   monitor variables
   initialization
   procedures
}
```

- A monitor invariant (I): describe the monitor's inner state
- Express relationship between monitor variables
- Maintained by execution of procedures:
  - Must hold after initialization
  - Must hold when a procedure terminates
  - Must hold when we suspend execution due to a call to `wait`
  - Can assume that the invariant holds *after* `wait` and when a procedure starts
- Should be as *strong* as possible!

Axioms for signal and continue (1)

Assume that the monitor invariant I and predicate P *do not* mention cv. Then we can set up the following axioms:

$$\{\ I\ \}\ wait(cv)\ \{\ I\ \}$$
$$\{\ P\ \}\ signal(cv)\ \{\ P\ \} \qquad \text{for arbitrary } P$$
$$\{\ P\ \}\ signal_all(cv)\ \{\ P\ \} \qquad \text{for arbitrary } P$$

Monitor solution to reader/writer problem

Verification of the invariant over `request_read`

$$I : (nr = 0 \vee nw = 0) \wedge nw \leq 1$$

```
procedure request_read() {

        while (nw > 0) {
                    wait(oktoread);
        }

        nr := nr + 1;

}
```

Monitor solution to reader/writer problem

Verification of the invariant over `request_read`

$$I : (nr = 0 \vee nw = 0) \wedge nw \leq 1$$

```
procedure request_read() {
        { I }
        while (nw > 0) {
                { I } wait(oktoread); { I }
        }

        nr := nr + 1;
        { I }
}
```

Monitor solution to reader/writer problem

Verification of the invariant over `request_read`

$$I : (nr = 0 \vee nw = 0) \wedge nw \leq 1$$

```
procedure request_read() {
        { I }
        while (nw > 0) {        { I ∧ nw > 0 }
              { I } wait(oktoread); { I }
        }     { I ∧ nw = 0 }

        nr := nr + 1;
        { I }
}
```

Monitor solution to reader/writer problem

Verification of the invariant over `request_read`

$$I : (nr = 0 \vee nw = 0) \wedge nw \leq 1$$

```
procedure request_read() {
        { I }
        while (nw > 0) {        { I ∧ nw > 0 }
              { I } wait(oktoread); { I }
        }    { I ∧ nw = 0 }
        { I[nr + 1/nr] }
        nr := nr + 1;
        { I }
}
```

Monitor solution to reader/writer problem

Verification of the invariant over `request_read`

$$I : (nr = 0 \vee nw = 0) \wedge nw \leq 1$$

```
procedure request_read() {
        { I }
        while (nw > 0) {        { I ∧ nw > 0 }
              { I } wait(oktoread); { I }
        }     { I ∧ nw = 0 }
        { I[nr + 1/nr] }
        nr := nr + 1;
        { I }
}
```

Verification obligation:

$(I \wedge nw > 0) \Rightarrow I$

$(I \wedge nw = 0) \Rightarrow I[nr + 1/nr]$

Axioms for Signal and Continue (2)

Assume that the invariant can mention the number of processes in the queue to a condition variable.

- Let $\#cv$ be the number of proc's waiting in the queue to cv.
- The test empty(cv) thus corresponds to $\#cv = 0$

Axioms for Signal and Continue (2)

Assume that the invariant can mention the number of processes in the queue to a condition variable.

- Let $\#cv$ be the number of proc's waiting in the queue to cv.
- The test empty(cv) thus corresponds to $\#cv = 0$

wait(cv) is *modelled* as an extension of the queue followed by processor release:

$$wait(cv): \qquad \#cv := \#cv + 1; \qquad \text{“}sleep\text{”}$$

Axioms for Signal and Continue (2)

Assume that the invariant can mention the number of processes in the queue to a condition variable.

- Let $\#cv$ be the number of proc's waiting in the queue to cv.
- The test empty(cv) thus corresponds to $\#cv = 0$

wait(cv) is *modelled* as an extension of the queue followed by processor release:

$$wait(cv): \qquad \#cv := \#cv + 1; \qquad \text{“}sleep\text{”}\{P\}$$

Axioms for Signal and Continue (2)

Assume that the invariant can mention the number of processes in the queue to a condition variable.

- Let $\#cv$ be the number of proc's waiting in the queue to cv.
- The test empty(cv) thus corresponds to $\#cv = 0$

wait(cv) is *modelled* as an extension of the queue followed by processor release:

$$wait(cv): \qquad \#cv := \#cv + 1; \{P\}\ \text{“}sleep''\{P\}$$

Axioms for Signal and Continue (2)

Assume that the invariant can mention the number of processes in the queue to a condition variable.

- Let $\#cv$ be the number of proc's waiting in the queue to cv.
- The test empty(cv) thus corresponds to $\#cv = 0$

wait(cv) is *modelled* as an extension of the queue followed by processor release:

$$wait(cv) : \{?\}\ \#cv := \#cv + 1; \{P\}\ \text{“}sleep\text{”}\{P\}$$

Axioms for Signal and Continue (2)

Assume that the invariant can mention the number of processes in the queue to a condition variable.

- Let $\#cv$ be the number of proc's waiting in the queue to cv.
- The test empty(cv) thus corresponds to $\#cv = 0$

wait(cv) is *modelled* as an extension of the queue followed by processor release:

$$wait(cv) : \{?\}\ \#cv := \#cv + 1; \{P\}\ \text{“}sleep''\{P\}$$

by assignment axiom:

$$wait(cv) : \{P[\#cv + 1/\#cv]; \#cv := \#cv + 1; \{\ P\ \}\ \text{“}sleep''\{\ P\ \}$$

Axioms for Signal and Continue (3)

signal(cv) can be modelled as a reduction of the queue, if the queue is not empty:

$$\text{signal}(cv): \qquad \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1$$

Axioms for Signal and Continue (3)

signal(cv) can be modelled as a reduction of the queue, if the queue is not empty:

$$\text{signal}(cv): \quad \textit{if } (\#cv \neq 0)\ \#cv := \#cv - 1\ \{\ P\ \}$$

Axioms for Signal and Continue (3)

signal(cv) can be modelled as a reduction of the queue, if the queue is not empty:

$$\text{signal}(cv) : \{\ ?\ \}\ \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1\ \{\ P\ \}$$

Axioms for Signal and Continue (3)

signal(cv) can be modelled as a reduction of the queue, if the queue is not empty:

$$\text{signal}(cv) : \{\ ?\ \}\ \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1\ \{\ P\ \}$$

$$\begin{aligned}\text{signal}(cv) : \quad & \{((\#cv = 0) \Rightarrow P) \wedge ((\#cv \neq 0) \Rightarrow P[\#cv - 1/\#cv])\} \\ & \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1 \\ & \{P\}\end{aligned}$$

Axioms for Signal and Continue (3)

signal(cv) can be modelled as a reduction of the queue, if the queue is not empty:

$$\text{signal}(cv) : \{\ ?\ \}\ \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1\ \{\ P\ \}$$

$$\begin{aligned}\text{signal}(cv) : \quad & \{((\#cv = 0) \Rightarrow P) \wedge ((\#cv \neq 0) \Rightarrow P[\#cv - 1/\#cv])\} \\ & \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1 \\ & \{P\}\end{aligned}$$

- *signal_all*(cv): $\#cv := 0$

Axioms for Signal and Continue (3)

signal(cv) can be modelled as a reduction of the queue, if the queue is not empty:

$$\text{signal}(cv) : \{\ ?\ \}\ \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1\ \{\ P\ \}$$

$$\begin{aligned}\text{signal}(cv) : \quad & \{((\#cv = 0) \Rightarrow P) \wedge ((\#cv \neq 0) \Rightarrow P[\#cv - 1/\#cv])\} \\ & \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1 \\ & \{P\}\end{aligned}$$

- *signal_all*(cv): $\#cv := 0\ \{P\}$

Axioms for Signal and Continue (3)

signal(cv) can be modelled as a reduction of the queue, if the queue is not empty:

$$\text{signal}(cv) : \{\ ?\ \}\ \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1\ \{\ P\ \}$$

$$\begin{aligned}\text{signal}(cv) : \quad & \{((\#cv = 0) \Rightarrow P) \wedge ((\#cv \neq 0) \Rightarrow P[\#cv - 1/\#cv])\} \\ & \mathit{if}\ (\#cv \neq 0)\ \#cv := \#cv - 1 \\ & \{P\}\end{aligned}$$

- *signal_all*(cv): $\{\ P[0/\#cv]\ \}\ \ \#cv := 0\ \{P\}$

Axioms for Signal and Continue (4)

Together this gives:

Axioms for monitor communication

$$\{\ P[\#cv+1/\#cv]\ \}\ \text{wait}(cv)\ \{\ P\ \} \quad \text{wait}$$

$$\{\ ((\#cv=0)\Rightarrow P)\wedge((\#cv\neq 0)\Rightarrow P[\#cv-1/\#cv])\ \}\ \text{signal}(cv)\ \{\ P\ \} \quad \text{signal}$$

$$\{\ P[0/\#cv]\ \}\ \text{signal_all}(cv)\ \{\ P\ \} \quad \text{signalAll}$$

If we know that $\#cv \neq 0$ whenever we signal, then the axiom for `signal(cv)` be simplified to:

$$\{\ P[\#cv-1/\#cv]\ \}\ \text{signal}(cv)\ \{\ P\ \}$$

Note! *$\#cv$ is not allowed in statements!* Only used for reasoning.

Example: FIFO semaphore verification (1)

```
monitor Semaphore          { # monitor invariant: s ≥ 0
  int s := 0;                # value of the semaphore
  cond pos;                  # wait condition

  procedure Psem() {
    if   (s=0)
         wait (pos);
    else
         s := s - 1
  }

  procedure Vsem() {
    if   empty(pos)
         s := s + 1
    else
         signal(pos);
  }
}
```

Example: FIFO semaphore verification (1)

```
monitor Semaphore          { # monitor invariant: s ≥ 0
  int s := 0;                # value of the semaphore
  cond pos;                  # wait condition

  procedure Psem() {
    if   (s=0)
         wait (pos);
    else
         s := s - 1
  }

  procedure Vsem() {
    if   empty(pos)
         s := s + 1
    else
         signal(pos);
  }
}
```

Consider the following monitor invariant:

$$s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

No process is waiting if the semaphore value is positive!

Example: FIFO semaphore verification: Psem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Psem() {

   if (s=0)
                              wait(pos);
   else
                     s := s-1;

}
```

Example: FIFO semaphore verification: Psem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Psem() {
{I}
   if (s=0)
                              wait(pos);
   else
                     s := s-1;

}
```

Example: FIFO semaphore verification: Psem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Psem() {
{I}
   if (s=0)
                              wait(pos);
   else
                     s := s-1;
{I}
}
```

Example: FIFO semaphore verification: Psem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Psem() {
{I}
   if (s=0) {I ∧ s = 0}
                              wait(pos);
   else {I ∧ s ≠ 0}
                        s := s-1;
{I}
}
```

Example: FIFO semaphore verification: Psem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Psem() {
{I}
   if (s=0) {I ∧ s = 0}
                              wait(pos); {I}
   else {I ∧ s ≠ 0}
                        s := s-1; {I}
{I}
}
```

Example: FIFO semaphore verification: Psem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Psem() {
```
$\{I\}$
```
   if (s=0)
```
$\{I \wedge s = 0\}$
```
      wait(pos);
```
$\{I[\#pos + 1/\#pos]\}$ $\{I\}$
```
   else
```
$\{I \wedge s \neq 0\}$
```
      s := s-1;
```
$\{I\}$
```
}
```

Example: FIFO semaphore verification: Psem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Psem() {
```
$\{I\}$
```
   if (s=0)
```
$\{I \wedge s = 0\}$
```
      wait(pos);
```
$\{I[\#pos + 1/\#pos]\}$ $\{I\}$
```
   else
```
$\{I \wedge s \neq 0\}$
```
      s := s-1;
```
$\{I[s - 1/s]\}$ $\{I\}$
```
}
```

Example: FIFO semaphore verification: Psem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Psem() {
```
$\{I\}$
```
   if (s=0)
```
$\{I \wedge s = 0\}$
$\{I[\#pos + 1/\#pos]\}$ `wait(pos);` $\{I\}$
```
   else
```
$\{I \wedge s \neq 0\}$
$\{I[s - 1/s]\}$ `s := s-1;` $\{I\}$
$\{I\}$
```
}
```

Two proof obligations:

- Then-branch: $(I \wedge s = 0) \Rightarrow I[\#pos + 1/\#pos]$
- Else-branch: $(I \wedge s \neq 0) \Rightarrow I[s - 1/s]$

Example: FIFO semaphore verification (3)

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

This gives two proof obligations:

Then-branch:

$$\begin{array}{lcl} (I \wedge s = 0) & \Rightarrow & I[\#pos + 1/\#pos] \\ s = 0 & \Rightarrow & s \geq 0 \wedge (s > 0 \Rightarrow \#pos + 1 = 0) \\ s = 0 & \Rightarrow & s \geq 0 \end{array}$$

Example: FIFO semaphore verification (3)

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

This gives two proof obligations:

Then-branch:

$$\begin{array}{lcl} (I \wedge s = 0) & \Rightarrow & I[\#pos + 1/\#pos] \\ s = 0 & \Rightarrow & s \geq 0 \wedge (s > 0 \Rightarrow \#pos + 1 = 0) \\ s = 0 & \Rightarrow & s \geq 0 \end{array}$$

Else-branch:

$$\begin{array}{lcl} (I \wedge s \neq 0) & \Rightarrow & I[s - 1/s] \\ (s > 0 \wedge \#pos = 0) & \Rightarrow & s - 1 \geq 0 \wedge (s - 1 > 0 \Rightarrow \#pos = 0) \\ (s > 0 \wedge \#pos = 0) & \Rightarrow & s > 0 \wedge \#pos = 0 \end{array}$$

Note: $(s - 1 > 0 \Rightarrow \#pos = 0)$ follows since we have $\#pos = 0$.

Example: FIFO semaphore verification: Vsem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Vsem() {

   if empty(pos)
                  s:=s+1;
   else
                        signal(pos);

}
```

Example: FIFO semaphore verification: Vsem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Vsem() {
{I}
   if empty(pos)
                  s:=s+1;
   else
                        signal(pos);
{I}
}
```

Example: FIFO semaphore verification: Vsem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Vsem() {
{I}
   if empty(pos) {I ∧ #pos = 0}
                 s:=s+1;
   else {I ∧ #pos ≠ 0}
                          signal(pos);
{I}
}
```

Example: FIFO semaphore verification: Vsem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Vsem() {
{I}
   if empty(pos) {I ∧ #pos = 0}
                 s:=s+1; {I}
   else {I ∧ #pos ≠ 0}
                         signal(pos); {I}
{I}
}
```

Example: FIFO semaphore verification: Vsem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Vsem() {
{I}
   if empty(pos) {I ∧ #pos = 0}
       {I[s + 1/s]}s:=s+1; {I}
   else {I ∧ #pos ≠ 0}
                          signal(pos); {I}
{I}
}
```

Example: FIFO semaphore verification: Vsem

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

```
procedure Vsem() {
```
$\{I\}$
```
   if empty(pos)
```
$\{I \wedge \#pos = 0\}$
$\{I[s+1/s]\}$`s:=s+1;` $\{I\}$
```
   else
```
$\{I \wedge \#pos \neq 0\}$
$\{I[\#pos - 1/\#pos]\}$ `signal(pos);` $\{I\}$
$\{I\}$
```
}
```

Example: FIFO semaphore verification (5)

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

As above, this gives two proof obligations:

Then-branch:

$$\begin{array}{lcl} (I \wedge \#pos = 0) & \Rightarrow & I[s+1/s] \\ (s \geq 0 \wedge \#pos = 0) & \Rightarrow & s+1 \geq 0 \wedge (s+1 > 0 \Rightarrow \#pos = 0) \\ (s \geq 0 \wedge \#pos = 0) & \Rightarrow & s+1 \geq 0 \wedge \#pos = 0 \end{array}$$

Example: FIFO semaphore verification (5)

$$I: \quad s \geq 0 \wedge (s > 0 \Rightarrow \#pos = 0)$$

As above, this gives two proof obligations:

Then-branch:

$$\begin{array}{lcl} (I \wedge \#pos = 0) & \Rightarrow & I[s+1/s] \\ (s \geq 0 \wedge \#pos = 0) & \Rightarrow & s+1 \geq 0 \wedge (s+1 > 0 \Rightarrow \#pos = 0) \\ (s \geq 0 \wedge \#pos = 0) & \Rightarrow & s+1 \geq 0 \wedge \#pos = 0 \end{array}$$

Else-branch:

$$\begin{array}{lcl} (I \wedge \#pos \neq 0) & \Rightarrow & I[\#pos - 1/\#pos] \\ (s = 0 \wedge \#pos \neq 0) & \Rightarrow & s \geq 0 \wedge (s > 0 \Rightarrow \#pos - 1 = 0) \\ s = 0 & \Rightarrow & s \geq 0 \end{array}$$

This concludes the proof that I always holds.

Summary

- Interference proofs tricky – easy to forget something
  - Critical regions help (but may reduce program efficiency)
  - Global invariants help
    - Avoid interference by splitting in global invariant and local condition
- Reasoning about global/local invariants in the *await* language
  - Sequential reasoning for each atomic statement/critical section
  - The conditions of *await* help in reasoning
  - Verified await programs can be transformed to other languages
- Reasoning about monitor invariants much easier
  - Like local, sequential reasoning
  - . . . but *signal/wait* need special care
  - Can reason about the length of the condition queues

References I

[Andrews, 2000] Andrews, G. R. (2000).
Foundations of Multithreaded, Parallel, and Distributed Programming.
Addison-Wesley.

[Owicki and Gries, 1976] Owicki, S. and Gries, D. (1976).
An axiomatic proof technique for parallel programs I.
Acta Informatica, 6(4):319–340.