G52CON:
Concepts of Concurrency

Lecture 16 Proving Correctness

Brian Logan

School of Computer Science & IT

bsl@cs.nott.ac.uk

Outline of this lecture

- correctness of concurrent programs
- proving correctness
- proving the correctness of Peterson’s algorithm
  - Mutual Exclusion
  - Absence of Livelock
  - Absence of Unnecessary Delay
  - Eventual Entry

Criteria for a solution

A mutual exclusion protocol should satisfy the following properties:

- **Mutual Exclusion:** at most one process at a time is executing its critical section.
- **Absence of Deadlock (Livelock):** if two or more processes are attempting to enter their critical sections, at least one will succeed.
- **Absence of Unnecessary Delay:** if a process is trying to enter its critical section and other processes are executingtheir noncritical sections (or have terminated), the first process is not prevented from entering its critical section.
- **Eventual Entry:** a process that is attempting to enter its critical section will eventually succeed.

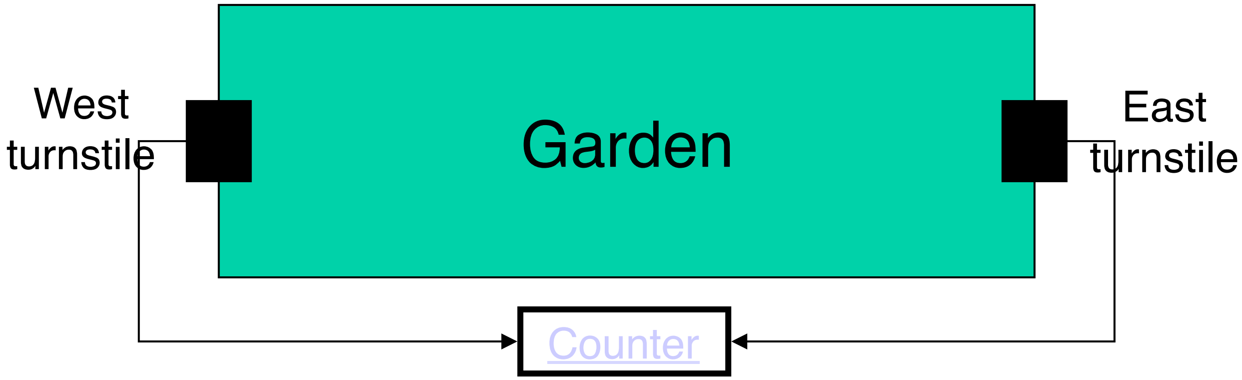
Finding bugs

How can we determine if an algorithm satisfies these properties?

- if an algorithm is broken, it is often relatively easy to find a trace which violates one or more of the properties
- however showing that there is *no* such trace is much harder
- (non-exhaustive) testing can only show the existence of bugs, not their absence

Ornamental Gardens problem

A large ornamental garden is open to members of the public who can enter through either of two turnstiles.



- the owner of the garden writes a computer program to count how many people are in the garden at any one time
- the program has two processes, each of which monitors a turnstile and increments a shared counter whenever someone enters via that processes' turnstile.

Ornamental Gardens program

```
// West turnstile

init1;
while(true) {
  // wait for turnstile
  count = count + 1;
  // other stuff ...

}
```

```
// East turnstile

init2;
while(true) {
  // wait for turnstile
  count = count + 1;
  // other stuff ...

}
```

`count == 0`

Loss of increment

```
// shared variable
integer count = 10;
```

West turnstile process

```
count = count + 1;
```

1. loads the value of `count` into a CPU register (`r == 10`)

4. increments the value in its register

(`r == 11`)

6. stores the value in its register in `count` (`count == 11`)

East turnstile process

```
count = count + 1;
```

2. loads the value of `count` into a CPU register (`r == 10`)

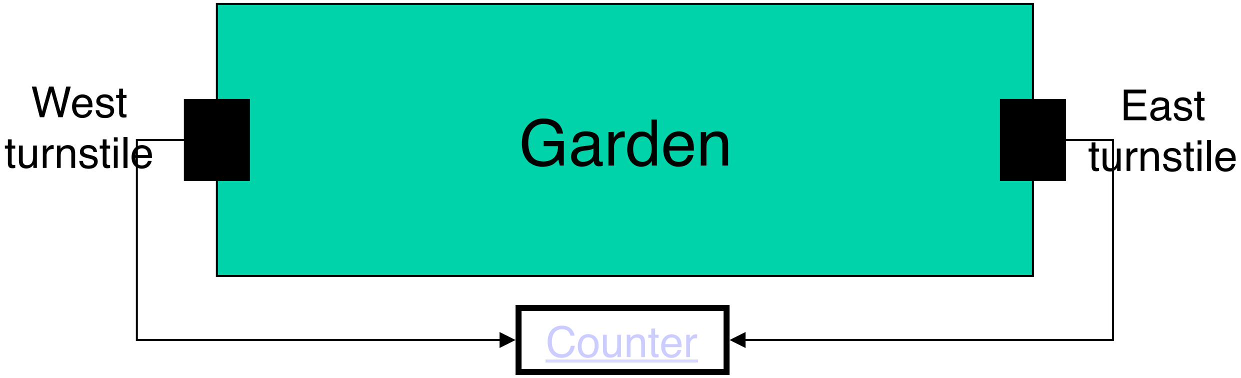
3. increments the value in its register

(`r == 11`)

5. stores the value in its register in `count` (`count == 11`)

Proof Garden

A small untidy garden is open to computer scientists who can enter through either of two turnstiles.



- a student writes a Java program to count how many people are in the garden at any one time
- the program has two processes, each of which monitors a turnstile and increments a shared counter whenever someone enters via that processes' turnstile.

Demonstrating correctness

- *Testing* can only consider a limited number of program executions
- some logically possible interleavings may not be generated by a particular implementation
- the only way to ensure that a concurrent program is correct is to *prove* that it is
- we do this by proving that certain properties are true of *all* executions of the program

Proving Correctness

There are two ways of proving correctness:

- **Assertional reasoning:** involves using assertions and invariants specified in predicate logic.
- **Model checking:** involves showing that a program represented as a finite state machine or a labelled transition system is a valid model of a formula expressing the desired property.

Peterson’s algorithm

```
// Process 1
init1;
while(true) {
    // entry protocol
    c1 = true;
    turn = 2;
    while (c2 && turn == 2) {};
    crit1;
    // exit protocol
    c1 = false;
    rem1;
}
```

```
// Process 2
init2;
while(true) {
    // entry protocol
    c2 = true;
    turn = 1;
    while (c1 && turn == 1) {};
    crit2;
    // exit protocol
    c2 = false;
    rem2;
}
```

```
// shared variables
bool c1 = c2 = false;
integer turn == 1;
```

Criteria for a Solution

A mutual exclusion protocol should satisfy the following properties:

- **Mutual Exclusion:** at most one process at a time is executing its critical section.
- **Absence of Deadlock (Livelock):** if two or more processes are attempting to enter their critical sections, at least one will succeed.
- **Absence of Unnecessary Delay:** if a process is trying to enter its critical section and other processes are executing their noncritical sections (or have terminated), the first process is not prevented from entering its critical section.
- **Eventual Entry:** a process that is attempting to enter its critical section will eventually succeed.

Proving mutual exclusion

We need to show that “never (Process in `crit1` and Process 2 in `crit2`)”:

- which is equivalent to showing “Process 1 in `crit1` implies Process 2 is not in `crit2`”

Proving mutual exclusion 1

1. When Process 1 enters `crit1`, `c2` is false or `turn` is 1 (or both).

—this follows from the test of `c2` and `turn` by Process 1 in the while loop of its entry protocol.

Proving mutual exclusion 2

1. When Process 1 enters `crit1`, `c2` is false or `turn` is 1 (or both).
2. If `c2` is false then Process 2 is not in `crit2` when Process 1 enters `crit1`.

— `crit2` is bracketed between assignments to `c2` which ensure this is always true.

Proving mutual exclusion 3

1. When Process 1 enters `crit1`, `c2` is false or `turn` is 1 (or both).
2. If `c2` is false then Process 2 is not in `crit2` when Process 1 enters `crit1`.
3. If `c2` is true when Process 1 enters `crit1`, then `turn` must be 1.

—this is a logical consequence of (1) and (2).

Proving mutual exclusion 4

1. When Process 1 enters `crit1`, `c2` is false or `turn` is 1 (or both).
2. If `c2` is false then Process 2 is not in `crit2` when Process 1 enters `crit1`.
3. If `c2` is true when Process 1 enters `crit1`, then `turn` must be 1.
4. If `c2` is true and `turn` is 1, then Process 2 must have set `turn` to 1 after Process 1 set it to 2.

—by inspection.

Proving mutual exclusion 5

1. When Process 1 enters `crit1`, `c2` is false or `turn` is 1 (or both).
2. If `c2` is false then Process 2 is not in `crit2` when Process 1 enters `crit1`.
3. If `c2` is true when Process 1 enters `crit1`, then `turn` must be 1.
4. If `c2` is true and `turn` is 1, then Process 2 must have set `turn` to 1 after Process 1 set it to 2.
5. Process 2 set `turn` to 1 after Process 1 set `c1` to true.

—from (4) and the the order of assignments in Process 1's entry protocol.

Proving mutual exclusion 6

1. When Process 1 enters `crit1`, `c2` is false or `turn` is 1 (or both).
2. If `c2` is false then Process 2 is not in `crit2` when Process 1 enters `crit1`.
3. If `c2` is true when Process 1 enters `crit1`, then `turn` must be 1.
4. If `c2` is true and `turn` is 1, then Process 2 must have set `turn` to 1 after Process 1 set it to 2.
5. Process 2 set `turn` to 1 after Process 1 set `c1` to true.
6. Had Process 2 evaluated the loop condition in its entry protocol when `c1` was true and `turn` was 1 then it would have spun

—the while condition in Process 2's entry protocol would have evaluated to true. Process 2 therefore can't have been in `crit2` when Process 1 enters `crit1`

Proving mutual exclusion summary

1. When Process 1 enters `crit1`, `c2` is false or `turn` is 1 (or both).
2. If `c2` is false then Process 2 is not in `crit2` when Process 1 enters `crit1`.
3. If `c2` is true when Process 1 enters `crit1`, then `turn` must be 1.
4. If `c2` is true and `turn` is 1, then Process 2 must have set `turn` to 1 after Process 1 set it to 2.
5. Process 2 set `turn` to 1 after Process 1 set `c1` to true.
6. Had Process 2 evaluated the loop condition in its entry protocol when `c1` was true and `turn` was 1 then it would have spun

Criteria for a Solution

A mutual exclusion protocol should satisfy the following properties:

- **Mutual Exclusion:** at most one process at a time is executing its critical section.
- **Absence of Deadlock (Livelock):** if two or more processes are attempting to enter their critical sections, at least one will succeed.
- **Absence of Unnecessary Delay:** if a process is trying to enter its critical section and other processes are executing their noncritical sections (or have terminated), the first process is not prevented from entering its critical section.
- **Eventual Entry:** a process that is attempting to enter its critical section will eventually succeed.

Peterson’s algorithm

```
// Process 1
init1;
while(true) {
     // entry protocol
    entry1;
    while ( ... ) {spin1};
    crit1;
    // exit protocol
    exit1;
    rem1;
}
```

```
// Process 2
init2;
while(true) {
     // entry protocol
    entry2;
    while ( ... ) {spin2};
    crit2;
    // exit protocol
    exit2;
    rem2;
}
```

```
// shared variables
bool c1 = c2 = false;
integer turn == 1;
```

Proving absence of livelock

We need to show that "always (`spin1` and `spin2`)" is false

- both processes spinning together is the only way to achieve livelock

Proving absence of livelock 1

1. For Process 1 to spin in its entry protocol, `c2` must always be true and `turn` must always be 2.

— if `c2` is ever false or `turn` is ever 1 when they are tested in the while condition of Process 1’s entry protocol, Process 1 will cease to spin.

Proving absence of livelock 2

1. For Process 1 to spin in its entry protocol, `c2` must always be true and `turn` must always be 2.
2. For Process 2 to spin in its entry protocol, `c1` must always be true and `turn` must always be 1.

—if `c1` is ever false or `turn` is ever 2 when they are tested in the while condition of Process 2’s entry protocol, Process 2 will cease to spin.

Proving absence of livelock 3

1. For Process 1 to spin in its entry protocol, `c2` must always be true and `turn` must always be 2.
2. For Process 2 to spin in its entry protocol, `c1` must always be true and `turn` must always be 1.
3. For Process 1 and Process 2 to both spin, `turn` must always be 2 and `turn` must always be 1.

—this is a logical consequence of (1) and (2).

Proving absence of livelock 4

1. For Process 1 to spin in its entry protocol, `c2` must always be true and `turn` must always be 2.
2. For Process 2 to spin in its entry protocol, `c1` must always be true and `turn` must always be 1.
3. For Process 1 and Process 2 to both spin, `turn` must always be 2 and `turn` must always be 1.
4. $\perp$

—the assumption that both processes always spin leads to a contradiction.

Criteria for a Solution

A mutual exclusion protocol should satisfy the following properties:

- **Mutual Exclusion:** at most one process at a time is executing its critical section.
- **Absence of Deadlock (Livelock):** if two or more processes are attempting to enter their critical sections, at least one will succeed.
- **Absence of Unnecessary Delay:** if a process is trying to enter its critical section and other processes are executing their noncritical sections (or have terminated), the first process is not prevented from entering its critical section.
- **Eventual Entry:** a process that is attempting to enter its critical section will eventually succeed.

Proving absence of unnecessary delay

We need to show that

- `entry1` and not (`entry2` or `crit2` or `exit2`) implies `crit1`
- i.e., that `entry1` and (`init2` or `rem2` or `terminated2`) implies `crit1`
- by symmetry, `entry2` and not (`entry1` or `crit1` or `exit1`) implies `crit2`and we have established absence of unnecessary delay

Proving absence of unnecessary delay 1

1. not (`entry2` or `crit2` or `exit2`) implies that `c2` is false.

— `c2` is only true in Process 2’s entry protocol, it’s critical section and immediately prior to the completion of its exit protocol.

Proving absence of unnecessary delay 2

1. not (`entry2` or `crit2` or `exit2`) implies `c2` is false.
2. `c2` is false implies not `spin1`.

— `c2` must be true for Process 1 to spin from the while condition in Process 1's entry protocol.

Proving absence of unnecessary delay 3

1. not (`entry2` or `crit2` or `exit2`) implies `c2` is false.
2. `c2` is false implies not `spin1`.
3. `entry1` and not `spin1` implies eventually `crit1`.

– if Process 1 completes its entry protocol but doesn't spin, then it must enter its critical section.

Criteria for a Solution

The protocols should satisfy the following properties:

- **Mutual Exclusion:** at most one process at a time is executing its critical section.
- **Absence of Deadlock (Livelock):** if two or more processes are attempting to enter their critical sections, at least one will succeed.
- **Absence of Unnecessary Delay:** if a process is trying to enter its critical section and other processes are executingtheir noncritical sections (or have terminated), the first process is not prevented from entering its critical section.
- **Eventual Entry:** a process that is attempting to enter its critical section will eventually succeed.

Proving eventual entry

We need to show that `spin1` implies eventually `crit1`

- we proceed by showing that the assumption that Process 1 spins forever (i.e., always `spin1`) leads to a contradiction, and hence that if Process 1 does spin it will eventually enter its critical section;

- by symmetry, `spin2` implies eventually `crit2` and we have established eventual entry

Proving eventual entry 1

1. Always `spin1` implies `c2` must always be true and `turn` must always be 2.

— if `c2` is ever false or `turn` is ever 1 when they are tested in the while condition of Process 1's entry protocol, Process 1 will cease to spin.

Proving eventual entry 2

1. Always `spin1` implies `c2` must always be true and `turn` must always be 2.
2. `turn` always 2 implies that Process 2 never executes `turn = 1`.

— Process 1 sets turn to 2 in its entry protocol before it starts to spin; for it to keep this value, the assignment statement in Process 2's entry protocol must never be executed.

Proving eventual entry 3

1. Always `spin1` implies `c2` must always be true and `turn` must always be 2.
2. `turn` always 2 implies that Process 2 never executes `turn = 1`.
3. Process 2 never executes `turn = 1` implies Process 2 never executes `c2 = true`.

—we assume that Process 2 does not terminate in its entry protocol and always eventually executes the next statement; if Process 2 ever set `c2` to true, it must eventually set `turn` to 1.

Proving eventual entry 4

1. Always `spin1` implies `c2` must always be true and `turn` must always be 2.
2. `turn` always 2 implies that Process 2 never executes `turn = 1`.
3. Process 2 never executes `turn = 1` implies Process 2 never executes `c2 = true`.
4. Process 2 never executes `c2 = true` implies that eventually `c2` will always be false.

—we assume that Process 2 does not terminate in its critical section or exit protocol, so if `c2` was true when Process 1 started spinning, it must eventually be set to false in Process 2's exit protocol and thereafter it will remain false.

Proving eventual entry 5

1. Always `spin1` implies `c2` must always be true and `turn` must always be 2.
2. `turn` always 2 implies that Process 2 never executes `turn = 1`.
3. Process 2 never executes `turn = 1` implies Process 2 never executes `c2 = true`.
4. Process 2 never executes `c2 = true` implies that eventually `c2` will always be false.
5. `turn` always 2 implies that eventually `c2` will always be false.

—this is a logical consequence of (2) and (4).

Proving eventual entry 6

1. Always `spin1` implies `c2` must always be true and `turn` must always be 2.
2. `turn` always 2 implies that Process 2 never executes `turn = 1`.
3. Process 2 never executes `turn = 1` implies Process 2 never executes `c2 = true`.
4. Process 2 never executes `c2 = true` implies that eventually `c2` will always be false.
5. `turn` always 2 implies that eventually `c2` will always be false.
6. ⊥

—assuming that Process 1 spins forever leads to a contradiction.

Proving eventual entry 7

1. Always `spin1` implies `c2` must always be true and `turn` must always be 2.
2. `turn` always 2 implies that Process 2 never executes `turn = 1`.
3. Process 2 never executes `turn = 1` implies Process 2 never executes `c2 = true`.
4. Process 2 never executes `c2 = true` implies that eventually `c2` will always be false.
5. `turn` always 2 implies that eventually `c2` will always be false.
6. ⊥
7. `spin1` implies eventually `crit1`.

The next lecture

Model Checking I

Suggested reading:

- Huth & Ryan (2000), chapter 3.