Australian National University

M02

Verification of Concurrent Programs

More Attempts at the Critical Section Problem

Temporal Logic

Specification of Correctness

Model Checking

The Critical Section Problem (recap)

- N processes execute (infinite) instruction sequences concurrently
- Each process is divided into two sub-sequences: *critical* section and *non-critical* section
- Correctness properties:
  - **Mutual exclusion**: Instructions from critical sections of two or more processes must never be interleaved
  - **Freedom from deadlock**: If *some* processes are trying to enter their critical sections, then *one* of them must eventually succeed
  - **Freedom from starvation**: If *any* process tries to enter its critical section, then it must eventually succeed

M. Ben-Ari *Principles of Concurrent and Distributed Programming*, Addison-Wesley, second edition, 2006

Second Attempt

```
Want_P, Want_Q : Boolean := False;
```



```
   task body P is
   begin
     loop
p1     -- non-critical section P
p2     loop
         exit when Want_Q = False;
       end loop;
p3     Want_P := True;
p4     -- critical section P
p5     Want_P := False;
     end loop;
   end P;
```

```
   task body Q is
   begin
     loop
q1     -- non-critical section Q
q2     loop
         exit when Want_P = False;
       end loop;
q3     Want_Q := True;
q4     -- critical section Q
q5     Want_Q := False;
     end loop;
   end Q;
```

State Diagram for Second Attempt (Table)

Process P	Process Q	Want_P	Want_Q
p1: -- non-critical section P	q1: -- non-critical section Q	False	False
p2: loop exit when Want_Q = False;	**q1: -- non-critical section Q**	False	False
p2: loop exit when Want_Q = False;	q2: loop exit when Want_P = False;	False	False
p3: Want_P := True;	**q2: loop exit when Want_P = False;**	False	False
p3: Want_P := True;	q3: Want_Q := True;	False	False
p4: -- critical section P	**q3: Want_Q := True;**	True	False
p4: -- critical section P	q4: -- critical section Q	True	True

~~Mutual exclusion~~

Third Attempt

```
Want_P, Want_Q : Boolean := False;
```



```
   task body P is
   begin
     loop
p1     -- non-critical section P
p2     Want_P := True;
p3     loop
         exit when Want_Q = False;
       end loop;
p4     -- critical section P
p5     Want_P := False;
     end loop;
   end P;
```

```
   task body Q is
   begin
     loop
q1     -- non-critical section Q
q2     Want_Q := True;
q3     loop
         exit when Want_P = False;
       end loop;
q4     -- critical section Q
q5     Want_Q := False;
     end loop;
   end Q;
```

State Diagram for Third Attempt (Table)

Process P	Process Q	Want_P	Want_Q
p1: -- non-critical section P	q1: -- non-critical section Q	False	False
p2: Want_P := True;	**q1: -- non-critical section Q**	False	False
p2: Want_P := True;	q2: Want_Q := True;	False	False
p3: loop exit when Want_Q = False;	**q2: Want_Q := True;**	True	False
p3: loop exit when Want_Q = False;	q3: loop exit when Want_P = False;	True	True
p4: -- critical section P	**q3: loop exit when Want_P = False;**	True	True

~~Freedom from deadlock~~

Temporal Logic

Symbol	Meaning	read as
$\neg$	negation	not
$\wedge$	conjunction	and
$\vee$	disjunction	or
$\rightarrow$	implication	implies
$\leftrightarrow$	biconditional	is equivalent to
$\Box$	global	always
$\Diamond$	final	eventually
$\leadsto$	$(x \leadsto y) \leftrightarrow (\Box(x \rightarrow \Diamond y))$	leads to

Logical Specification of Correctness

- A formula is either true or false for a given state of the system e.g.

  $p1 \land q1 \land \neg wantp \land \neg wantq$

  is true in the initial state of the system

- Mutual exclusion property:

  $\neg(p4 \land q4)$

  must be true for *all possible states* of *all possible computations*

Inductive Proofs of Invariants

- To prove invariance of logical formula A:
  - prove that A is true in the initial state; and then
  - assuming that A is true in all states up to the current state: prove that A true in the next state.
- Easy for implication, e.g. $p4 \rightarrow wantp$ either:
  - *wantp* is true, therefore can only falsify if next step changes value of *wantp*; or
  - *p*4 is false, therefore can only change if next step changes program counter (pc[P]) to ($\neg p4$) without also setting *wantp* = true

Inductive Proofs of Invariants

```
   loop
p1    -- non-critical section P
p2    Want_P := True;
p3    loop
        exit when Want_Q = False;
      end loop;
p4    -- critical section P
p5    Want_P := False;
   end loop;
```

```
   loop
q1    -- non-critical section Q
q2    Want_Q := True;
q3    loop
        exit when Want_P = False;
      end loop;
q4    -- critical section Q
q5    Want_Q := False;
   end loop;
```


- Prove invariant A = ($p3 \vee p4 \vee p5) \rightarrow wantp$
- Base case: trivially true because *p*1 is true, so (*p*3..5) is false
- Only need to consider statements p2 (changes pc[P] to *p*3) and p5 (sets *wantp* = false)

Inductive Proofs of Invariants

```
   loop
p1    -- non-critical section P
p2    Want_P := True;
p3    loop
        exit when Want_Q = False;
      end loop;
p4    -- critical section P
p5    Want_P := False;
    end loop;
```

```
   loop
q1    -- non-critical section Q
q2    Want_Q := True;
q3    loop
        exit when Want_P = False;
      end loop;
q4    -- critical section Q
q5    Want_Q := False;
    end loop;
```


- p2 : pc[P] ← p3; wantp ← true
  *p*3..5, *wantp*
- p5 : pc[P] ← p1; wantp ← false
  ¬*p*3..5, ¬*wantp*

Proved:
A = (*p*3 ∨ *p*4 ∨ *p*5) → *wantp*

Inductive Proofs of Invariants

```
    loop
p1      -- non-critical section P
p2      Want_P := True;
p3      loop
          exit when Want_Q = False;
        end loop;
p4      -- critical section P
p5      Want_P := False;
      end loop;
```

```
    loop
q1      -- non-critical section Q
q2      Want_Q := True;
q3      loop
          exit when Want_P = False;
        end loop;
q4      -- critical section Q
q5      Want_Q := False;
      end loop;
```


- Prove invariant B = $wantp \rightarrow (p3 \vee p4 \vee p5)$
- Base case: trivially true because *wantp* is false
- Again, only need to consider statements p2 (changes pc[P] to *p*3) and p5 (sets *wantp* = false)

Inductive Proofs of Invariants

```
   loop
p1    -- non-critical section P
p2    Want_P := True;
p3    loop
        exit when Want_Q = False;
      end loop;
p4    -- critical section P
p5    Want_P := False;
   end loop;
```

```
   loop
q1    -- non-critical section Q
q2    Want_Q := True;
q3    loop
        exit when Want_P = False;
      end loop;
q4    -- critical section Q
q5    Want_Q := False;
   end loop;
```


- p2 : pc[P] ← p3; wantp ← true
  *p*3..5, *wantp*
- p5 : pc[P] ← p1; wantp ← false
  ¬*p*3..5, ¬*wantp*

Proved:
B = *wantp* → (*p*3..5)

With A, symmetry for process Q:
(*p*3..5) ↔ *wantp*
(*q*3..5) ↔ *wantq*

Inductive Proofs of Invariants: Mutual Exclusion

```
   loop
p1    -- non-critical section P
p2    Want_P := True;
p3    loop
        exit when Want_Q = False;
      end loop;
p4    -- critical section P
p5    Want_P := False;
   end loop;
```

```
   loop
q1    -- non-critical section Q
q2    Want_Q := True;
q3    loop
        exit when Want_P = False;
      end loop;
q4    -- critical section Q
q5    Want_Q := False;
   end loop;
```


- Prove invariant M = $\neg(p4 \wedge q4)$
- Base case: trivially true because *p*4 and *q*4 are both false
- Only need to consider statements p3 (changes pc[P] to *p*4) and q3 (sets pc[Q] to *q*4)

Inductive Proofs of Invariants

```
   loop
p1     -- non-critical section P
p2     Want_P := True;
p3     loop
         exit when Want_Q = False;
       end loop;
p4     -- critical section P
p5     Want_P := False;
     end loop;
```

```
   loop
q1     -- non-critical section Q
q2     Want_Q := True;
q3     loop
         exit when Want_P = False;
       end loop;
q4     -- critical section Q
q5     Want_Q := False;
     end loop;
```


- p3 : can progress only when $\neg wantq$
  previously: $(q3..5) \leftrightarrow wantq$
- q3: symmetric argument

Model Checking

- Specify states, transitions, and correctness properties formally in temporal logic
- Check each possible state for violations of correctness properties
  - (Deadlock = no allowed transitions)

TLA+

- TLA+: language for specifying properties of concurrent systems
- PlusCal: language for specifying concurrent processes
- TLC: model checker for TLA+
- TLA+ Toolbox

Lamport (1994). *The Temporal Logic of Actions*. ACM TOPLAS

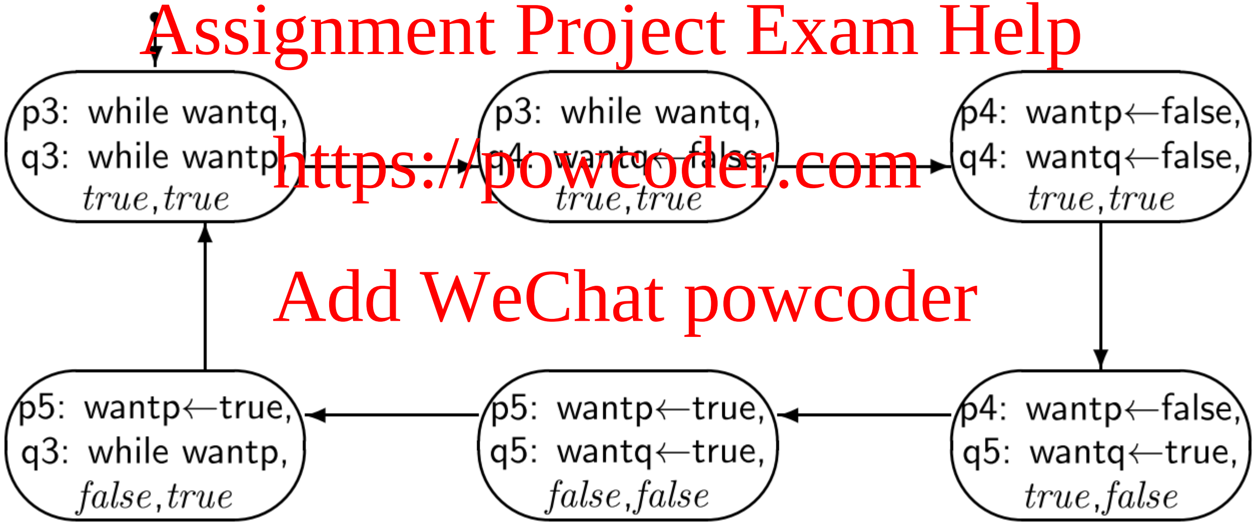
Fourth Attempt

```
Want_P, Want_Q : Boolean := False;
```

```
   task body P is
   begin
     loop
p1     -- non-critical section P
p2     Want_P := True;
p3     loop
         exit when Want_Q = False;
p4       Want_P := False;
p5       Want_P := True;
       end loop;
p6     -- critical section P
p7     Want_P := False;
     end loop;
   end P;
```

```
   task body Q is
   begin
     loop
q1     -- non-critical section Q
q2     Want_Q := True;
q3     loop
         exit when Want_P = False;
q4       Want_Q := False;
q5       Want_Q := True;
       end loop;
q6     -- critical section Q
q7     Want_Q := False;
     end loop;
   end Q;
```

Fourth Attempt



Livelock!

M. Ben-Ari , 2006