CS 361: Concurrent Correctness

Mark Boady

Department of Computer Science
Drexel University

September 9, 2022

This lecture is adapted from material in Chapters 2-4 of
Principles of Concurrent and Distributed Programming
M. Ben-Ari
Second Edition
Available online from library:
`https://drexel.primo.exlibrisgroup.com/permalink/01DRXU_INST/7nf6pv/cdi_askewsholts_vlebooks_9781292122588`

Correctness

- Debugging Concurrent Programs is drastically harder then sequential programs
- Each execution may have a different interleaving
- We need to reason about concurrent programs correctness!

Correctness

- The Correctness of a concurrent program is defined in terms of properties of computations.
- **Safety Properties**: the property must *always* be true
- **Liveness Properties**: the property must *eventually* become true

Relation to State Diagrams

- How does these properties related to state diagrams and scenarios?
- **Safety Properties**: must be true in *every* state for *every* scenario
- **Liveness Properties**: must be true in *some* state for *every* scenario

Temporal Logic

- **Linear Temporal Logic**: expresses properties that must be true at a state in a *single* arbitrary scenario
- **Branching Temporal Logic**: expresses properties that must be true in *some or all* scenarios starting from a selected state
- In short: do we look at one path or all paths

Fairness

- We generally try to avoid assumptions about what the OS/hardware will do
- **Weakly Fair**: for any state in the scenario, a statement that is continually enabled eventually gets executed
- In short: eventually a context switch always happens
- It is not useful to worry about situations in which one thread **never** executes commands

Example

- Does this code halt?

Stop Loop A	
Setup: n=0, flag=false	
Process P	**Process Q**
p1: while flag == false p2: n=1-n	q1: flag=true

Only If Fair

- If Thread Q never executes, then it will run forever
- Nothing in Thread P can ever cause the loop to halt
- In **fair** scenarios, Thread Q will *eventually* get to execute to halt the loop

Induction

- **Invariant**: an invariant is something we want to be true at *every* state of computation
- We can verify these using **induction**!
- **Base Case**: verify that the property holds for the initial state
- **Inductive Hypothesis**: assume property holds in all states leading up to and including current state
- **Inductive Case**: prove that the property continues to hold in the all successor states

Mutual Exclusion

- We can apply *informal* induction to a mutual exclusion problem
- **Goal**: Verify that two programs never enter the critical section at the same time

Mutual Exclusion

- Is it impossible for both threads to be in the critical section?

Mutual Exclusion Attempt	
Setup: wantp=false, wantq=false	
Process P	**Process Q**
loop as needed:	loop as needed:
p1: noncriticalCode()	q1: noncriticalCode()
p2: wantp = true	q2: wantq = true
p3: await wantq==false	q3: await wantp == false
p4: criticalCode()	q4: criticalCode()
p5: wantp=false	q5: wantq=false

Property 1

- $(p3 \vee p4 \vee p5) \implies wantp$
- If the program is *about* to execute lines p3, p4, or p5 then wantp is true
- Reminder: Truth Table for Implies

A	B	$A \implies B$
T	T	T
T	F	F
F	T	T
F	F	T

Thread Q

- Thread Q never changes the value of wantp
- We can ignore execution of the Q Thread for this step
- Thread Q has **read-only** usage of wantp

```
loop as needed:
q1: noncriticalCode()
q2: wantq = true
q3: await wantp == false
q4: criticalCode()
q5: wantq=false
```

Line P1 About to Execute

- Goal: $(p3 \vee p4 \vee p5) \implies wantp$
- Step 1: Is the goal true before p1 executes?
- $(p3 \vee p4 \vee p5)$ =False
- Implies is true $((F \implies F) \iff T)$

Line (bold - next to execute)	p3	p4	p5	wantp
p1: noncriticalCode()	F	F	F	F
p2: wantp = true				
p3: await wantq==false				
p4: criticalCode()				
p5: wantp=false				
p1: noncriticalCode()				

Line P2 About to Execute

- Goal: $(p3 \vee p4 \vee p5) \implies wantp$
- Step 2: Is the goal true before p2 executes?
- $(p3 \vee p4 \vee p5) =$False
- Implies is true $((F \implies F) \iff T)$

Line (bold - next to execute)	p3	p4	p5	wantp
p1: noncriticalCode()	F	F	F	F
p2: wantp = true	F	F	F	F
p3: await wantq==false				
p4: criticalCode()				
p5: wantp=false				
p1: noncriticalCode()				

Line P3 About to Execute

- Goal: $(p3 \vee p4 \vee p5) \implies wantp$
- Step 3: Is the goal true before p3 executes?
- $(p3 \vee p4 \vee p5) =$True because p3 is True
- wantp=true because line p2 must have executed
- Implies is true $((T \implies T) \iff T)$

Line (bold - next to execute)	p3	p4	p5	wantp
p1: noncriticalCode()	F	F	F	F
p2: wantp = true	F	F	F	F
p3: await wantq==false	T	F	F	T
p4: criticalCode()				
p5: wantp=false				
p1: noncriticalCode()				

Line P4 About to Execute

- Goal: $(p3 \vee p4 \vee p5) \implies wantp$
- Step 4: Is the goal true before p4 executes?
- $(p3 \vee p4 \vee p5) =$True because p4 is True
- wantp=true because line p2 must have executed
- p3 did not change the value of wantp
- Implies is true $((T \implies T) \iff T)$

Line (bold - next to execute)	p3	p4	p5	wantp
p1: noncriticalCode()	F	F	F	F
p2: wantp = true	F	F	F	F
p3: await wantq==false	T	F	F	T
p4: criticalCode()	F	T	F	T
p5: wantp=false				
p1: noncriticalCode()				

Line P5 About to Execute

- Goal: $(p3 \lor p4 \lor p5) \implies wantp$
- Step 5: Is the goal true before p5 executes?
- $(p3 \lor p4 \lor p5) =$True because p5 is True
- wantp=true because line p2 must have executed
- No other lines changed wantp
- Implies is true $((T \implies T) \iff T)$

Line (bold - next to execute)	p3	p4	p5	wantp
p1: noncriticalCode()	F	F	F	F
p2: wantp = true	F	F	F	F
p3: await wantq==false	T	F	F	T
p4: criticalCode()	F	T	F	T
p5: wantp=false	F	F	T	T
p1: noncriticalCode()				

Line P1 About to Execute

- Goal: $(p3 \lor p4 \lor p5) \implies wantp$
- Step 6: Is the goal true before looping back to p1?
- $(p3 \lor p4 \lor p5)$ =False because p1 is True
- wantp=false because line p5 must have executed
- Implies is true $((F \implies F) \iff T)$

Line (bold - next to execute)	p3	p4	p5	wantp
p1: noncriticalCode()	F	F	F	F
p2: wantp = true	F	F	F	F
p3: await wantq==false	T	F	F	T
p4: criticalCode()	F	T	F	T
p5: wantp=false	F	F	T	T
p1: noncriticalCode()	F	F	F	F

Invariant Properties

- $(p3 \vee p4 \vee p5) \implies$ *wantp* is **always** true
- $(q3 \vee q4 \vee q5) \implies$ *wantq* is **always** true
- The two threads are mirrors, the proof of the second statement is trivially similar
- What about $wantp \implies (p3 \vee p4 \vee p5)$?
- We already make the table, we can reuse it

Implies Direction 2

- Goal: $wantp \implies (p3 \vee p4 \vee p5)$

Line (bold - next to execute)	p3	p4	p5	wantp	Implies
p1: noncriticalCode()	F	F	F	F	T
p2: wantp = true	F	F	F	F	T
p3: await wantq==false	T	F	F	T	T
p4: criticalCode()	F	T	F	T	T
p5: wantp=false	F	F	T	T	T
p1: noncriticalCode()	F	F	F	F	T

Invariant Properties

- $(p3 \vee p4 \vee p5) \implies wantp$ is **always** true
- $(q3 \vee q4 \vee q5) \implies wantq$ is **always** true
- $wantp \implies (p3 \vee p4 \vee p5)$ is **always** true
- $wantq \implies (q3 \vee q4 \vee q5)$ is **always** true
- The Q Thread is a mirror of the Thread P justification
- Two directions of a implies proves **if and only if**

Biconditional Review

- $(A \implies B) \land (B \implies A)$ is a bi-conditional
- A is true **if and only if** B is true
- The two values are equal

A	B	$A \iff B$
T	T	T
T	F	F
F	T	F
F	F	T

Invariant Properties

- $(p3 \lor p4 \lor p5) \iff wantp$
- $(q3 \lor q4 \lor q5) \iff wantq$
- Is it possible for two threads to be in the critical section at the same time?
- Invariant: $\neg(p4 \land q4)$
- Assume: $p4 \land q4$ and show a contradiction

Mutual Exclusion

- Initially $p4 \wedge q4$ is false since no code has executed
- Which lines could make it true?

Mutual Exclusion Attempt	
Setup: wantp=false, wantq=false	
Process P	**Process Q**
loop as needed:	loop as needed:
p1: noncriticalCode()	q1: noncriticalCode()
p2: wantp = true	q2: wantq = true
p3: await wantq==false	q3: await wantp == false
p4: criticalCode()	q4: criticalCode()
p5: wantp=false	q5: wantq=false

- Goal: Contradict $p4 \wedge q4$
- These lines only become true if p3 and/or q4 executes

Mutual Exclusion Attempt	
Setup: wantp=false, wantq=false	
Process P	**Process Q**
loop as needed:	loop as needed:
p1: noncriticalCode()	q1: noncriticalCode()
p2: wantp = true	q2: wantq = true
p3: await wantq==false	**q3: await wantp == false**
p4: criticalCode()	q4: criticalCode()
p5: wantp=false	q5: wantq=false

Mutual Exclusion

- Goal: Contradict $p4 \land q4$
- If p3 executes, wantq must be false
- $wantq \iff q3 \lor q4 \lor q5$
- If p3 executes Q cannot be about to execute q3-q5

Mutual Exclusion Attempt	
Setup: wantp=false, wantq=false	
Process P	**Process Q**
loop as needed:	loop as needed:
p1: noncriticalCode()	q1: noncriticalCode()
p2: wantp = true	q2: wantq = true
p3: await wantq==false	q3: await wantp == false
p4: criticalCode()	q4: criticalCode()
p5: wantp=false	q5: wantq=false

Mutual Exclusion

- Goal: Contradict $p4 \wedge q4$
- If q3 executes, wantp must be false
- $wantp \iff p3 \vee p4 \vee p5$
- If q3 executes P cannot be about to execute p3-p5

Mutual Exclusion Attempt	
Setup: wantp=false, wantq=false	
Process P	**Process Q**
loop as needed:	loop as needed:
p1: noncriticalCode()	q1: noncriticalCode()
p2: wantp = true	q2: wantq = true
p3: await wantq==false	**q3: await wantp == false**
p4: criticalCode()	q4: criticalCode()
p5: wantp=false	q5: wantq=false

Invariant Properties

- Invariant: $(p3 \vee p4 \vee p5) \iff wantp$
- Invariant: $(q3 \vee q4 \vee q5) \iff wantq$
- We showed it was impossible for $p4 \wedge q4$ to be true
- Invariant: $\neg(p4 \wedge q4)$
- It is impossible for both threads to be about to enter critical sections