CSCI 322
Principles of Concurrent Programming

Filip Jagodzinski
Announcements

Homework #3

• Due tomorrow, Tuesday, 11:59pm
• 2 “book” questions only
• Solutions will be made available during Wednesday’s lecture

Midterm

• 12 February
• Closed Book, Closed Notes
• Sample exam has been posted to the course website
• Solutions to sample exam will be provided orally during Wednesday’s class
From last time

**Atomic action**: Makes an *indivisible* state transformation. Any intermediate state that might exist in the implementation of the action must NOT be visible to other processes.

```c
int y = 0, z = 0;
co x = y+z; // y = 1; z = 2; oc;
```

When executed concurrently, however, the assignment statements might be implemented by a sequence of fine-grained machine instructions, and depending on the execution history one of several (potentially many) different results may result.
From last time

- A critical reference in an expression is a reference to a variable that is change(able) by another process.
- An assignment $x = e$ is at-most-once if either
  - $e$ contains at MOST one critical reference and $x$ is not read by another process.
  - $e$ contains no critical references.

```c
int x = 0, y = 0;
co x = y+1; // y = y+1; oc;
```

Q: Which of these are at-most once?

```c
int x = 0, y = 0;
co x = y+1; // y = x+1; oc;
```
From last time

- `< await (B) S; >`
- B specifies a Delay condition
- S is a sequence of statements Guaranteed to terminate (a sequence of assignment operations)
- `<>` specify atomic action
- Therefore B is guaranteed to be true when execution of S begins
- No internal state of S is visible to other processes

```plaintext
< await (s > 0) s = s - 1 >
```

await specifies mutual exclusion AND conditional synchronization
Assume the following two code statements

S1 : x = x + y;
S2 : y = x * y;

And that the variables x and y have initial values x=3 and y=4. Then, what is the final state (values for x and y) for the following program P3 (assume S1 and S2 are executed atomically):

P3 : co await (y < x) S1; // S2; oc

In class exercise
Today

Formal Logic
Logic Axioms
Conditional Proofs
Indirect Proofs
Synchronization

Recall that when we discussed finding patterns in a file ...

```c
string buffer;  // contains one line of input
bool done = false;  // used to signal termination

// # process 1: find patterns
co
    string line1;
    while (true) {
        wait for buffer to be full or done to be true;
        if (done) break;
        line1 = buffer;
        signal that buffer is empty;
        look for pattern in line1;
        if (pattern is in line1)
            write line1;
    }

// # process 2: read new lines
string line2;
while (true) {
    read next line of input into line2;
    if (EOF) {done = true; break; }
    wait for buffer to be empty;
    buffer = line2;
    signal that buffer is full;
}
```
Synchronization

Recall that when we discussed finding patterns in a file ...

These two processes, one a “consumer” and another a “producer” communicate via a shared buffer.

Q: How is access to buffer synchronized?
Synchronization

- The producer and consumer have to coordinate access to the buffer.
- There is a back-and-forth of placing information into the buffer and retrieving from it.

Q: How might you implement a protocol so that neither producer nor consumer perform successive reads or writes?
Synchronization

- The producer and consumer have to coordinate access to the buffer.
- There is a back-and-forth of placing information into the buffer and retrieving from it.

Q: How might you implement a protocol so that neither producer nor consumer perform successive reads or writes?

Rely on shared variables $p$ and $c$ to keep track of the number of items read from/ written to by producer and consumer.

Q: Does using these variables enforce “fair” back-and-forth access to buffer?
Synchronization

- The producer and consumer have to coordinate access to the buffer.
- There is a back-and-forth of placing information into the buffer and retrieving from it.

Q: How might you implement a protocol so that neither producer nor consumer perform successive reads or writes?

Rely on shared variables $p$ and $c$ to keep track of the number of items read from/ written to by producer and consumer.

Q: Does using these variables enforce “fair” back-and-forth access to buffer?

No. Thus, we want to impose such behavior, and we can do that by enforcing synchronization requirements:

\[ c \leq p \leq c+1 \]
The producer and consumer have to coordinate access to the buffer.
There is a back-and-forth of placing information into the buffer and retrieving from it.

Q: How might you implement a protocol so that neither producer nor consumer perform successive reads or writes?

Rely on shared variables $p$ and $c$ to keep track of the number of items read from/ written to by producer and consumer.

Q: What do the following “states” specify?

- $c == p$
- $c < p$
- $p < c$

Synchronization requirements:

$c <= p <= c+1$
Synchronization

- The producer and consumer have to coordinate access to the buffer.
- There is a back-and-forth of placing information into the buffer and retrieving from it.

Q: How might you implement a protocol so that neither producer nor consumer perform successive reads or writes?

Rely on shared variables $p$ and $c$ to keep track of the number of items read from/ written to by producer and consumer.

Q: Which of the following constitute “bad” states?

- $c=10; p=11$
- $p=3; c=7$
- $c=0; p=0$
- $c=1; p=0$
- $c=0; p=1$

Synchronization requirements:

$$c \leq p \leq c+1$$
Synchronization

- The producer and consumer have to coordinate access to the buffer.
- There is a back-and-forth of placing information into the buffer and retrieving from it.

Q: How might you implement a protocol so that neither producer nor consumer perform successive reads or writes?

Rely on shared variables \( p \) and \( c \) to keep track of the number of items read from/ written to by producer and consumer.

Q: Which of the following constitute “bad” states?

- \( c=10; p=11; \)
- \( p=3; c=7; \)
- \( c=0; p=0; \)
- \( c=1; p=0; \)
- \( c=0; p=1; \)

Synchronization requirements:

\[ c \leq p \leq c+1 \]

Goal: We want to “prove” that a concurrently program NEVER enters a bad state.
Assertional reasoning allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states
**Formal Logic**

**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A ⇒ B</th>
<th>A ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$\neg A$</th>
<th>$A \land B$</th>
<th>$A \lor B$</th>
<th>$A \implies B$</th>
<th>$A \equiv B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

**Q:** What do $A$ and $B$ represent?

**Q:** What do the symbols represent?

**Task:** Be able to explain the difference between logical implication and logical biconditional.
**Formal Logic**

**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from truth tables.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Q: What are the values for the second row?
Assertional reasoning allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from truth tables.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(\neg A)</th>
<th>(A \land B)</th>
<th>(A \lor B)</th>
<th>(A \Rightarrow B)</th>
<th>(A \equiv B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Q: What are the values for the third and fourth rows?
Assertional reasoning allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from truth tables.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A ⇒ B</th>
<th>A ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
**Formal Logic**

**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$\neg A$</td>
<td>$A \land B$</td>
<td>$A \lor B$</td>
<td>$A \implies B$</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Task: Be able to provide a concrete example of logical implication.

Q: What are 2 statements $A$ and $B$ such that $A \rightarrow B$?
Assertional reasoning allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from truth tables.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A ⇒ B</th>
<th>A ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

A: Clouds are present

B: You cannot see the sun

Q: How do we interpret this column?
**Formal Logic**

**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(\neg A)</th>
<th>(A \land B)</th>
<th>(A \lor B)</th>
<th>(A \implies B)</th>
<th>(A \equiv B)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
</tr>
<tr>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td><strong>false</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td><strong>false</strong></td>
</tr>
<tr>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
</tr>
<tr>
<td><strong>false</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td><strong>false</strong></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
</tr>
</tbody>
</table>

A: Clouds are present  
B: You cannot see the sun  
Q: If there ARE clouds, and you CANNOT see the sun, does A imply B?
**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A ⇒ B</th>
<th>A ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

A: Clouds are present
B: You cannot see the sun

It is overcast, regardless whether day or night
It is possible ... We say this is valid
**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A ⇒ B</th>
<th>A ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

A: Clouds are present  
B: You cannot see the sun  
Q: If there ARE clouds, and you CAN see the sun, does A imply B?
**Formal Logic**

**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( \neg A )</th>
<th>( A \land B )</th>
<th>( A \lor B )</th>
<th>( A \Rightarrow B )</th>
<th>( A \equiv B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>( \text{false} )</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

A: Clouds are present
B: You cannot see the sun

**NOT possible ... in ALL cases!**
We say this is invalid
Assertional reasoning allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from truth tables.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$\neg A$</th>
<th>$A \land B$</th>
<th>$A \lor B$</th>
<th>$A \implies B$</th>
<th>$A \equiv B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

A: Clouds are present
B: You cannot see the sun
Q: If there are NO clouds, and you CANNOT see the sun, does $A$ imply $B$?
**Formal Logic**

**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A ⇒ B</th>
<th>A ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

A: Clouds are present
B: You cannot see the sun

It is night time, or an eclipse during the day

It is possible ... We say this is valid
**Formal Logic**

**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$\neg A$</th>
<th>$A \land B$</th>
<th>$A \lor B$</th>
<th>$A \implies B$</th>
<th>$A \equiv B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

A: Clouds are present
B: You cannot see the sun

Q: If there are NO clouds, and you CAN see the sun, does $A$ imply $B$?
**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A ⇒ B</th>
<th>A ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

**A:** Clouds are present
**B:** You cannot see the sun

**It is daytime**

It is possible ... We say this is valid.
Assertional reasoning allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from truth tables.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$\neg A$</th>
<th>$A \land B$</th>
<th>$A \lor B$</th>
<th>$A \Rightarrow B$</th>
<th>$A \equiv B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

How else is this often referred to in medical studies, psychology, etc.?
Assertional reasoning allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from truth tables.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>\neg A</th>
<th>A \land B</th>
<th>A \lor B</th>
<th>A \Rightarrow B</th>
<th>A \equiv B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

How else is this often referred to in medical studies, psychology, etc.?

False positive. If you conclude A (true), but the “truth” is B (false), saying that A implies B is very bad!
**Formal Logic**

**Assertional reasoning** allows us to better understand concurrent programs, AND it can help us to develop correct concurrent programs by logically reasoning about the possibilities of “bad” states.

The “simplest” kind of logic is built up from **truth tables**.

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A ⇒ B</th>
<th>A ≡ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

**Task:** Be able to explain the difference between logical implication and logical biconditional.
Logical axioms are logical statements that can be proven using such a truth table approach. Some of the more common axioms are shown on the right.

\[
\begin{align*}
A \land (B \lor C) & \equiv (A \land B) \lor (A \land C) \\
A \lor (B \land C) & \equiv (A \lor B) \land (A \lor C) \\
\neg \neg A & \equiv A \\
A \lor A & \equiv A \\
A \lor \neg A & \equiv \text{true} \\
A \lor \text{true} & \equiv \text{true} \\
A \lor \text{false} & \equiv A \\
A \land \text{true} & \equiv A \\
A \land \text{false} & \equiv \text{false} \\
A \land \neg A & \equiv \text{false} \\
A \Rightarrow \text{true} & \equiv \text{true} \\
A \Rightarrow \text{false} & \equiv \neg A \\
\text{true} \Rightarrow A & \equiv A \\
\text{false} \Rightarrow A & \equiv \text{true} \\
A \Rightarrow B & \equiv \neg A \lor B \\
A \land \neg (A \Rightarrow B) & \equiv A \land \neg B \\
A \land (A \lor B) & \equiv A \\
A \lor (A \land B) & \equiv A \\
A \land (\neg A \lor B) & \equiv A \land B \\
A \lor (\neg A \land B) & \equiv A \lor B \\
\neg (A \land B) & \equiv \neg A \lor \neg B \\
\neg (A \lor B) & \equiv \neg A \land \neg B
\end{align*}
\]

These axioms are the foundation from which more complex statements can be constructed.
Formal Logic

Logical axioms are logical statements that can be proven using such a truth table approach. Some of the more common axioms are shown on the right.

\[ A \land (B \lor C) \equiv (A \land B) \lor (A \land C) \]
\[ A \lor (B \land C) \equiv (A \lor B) \land (A \lor C) \]

\[ \neg \neg A \equiv A \]
\[ A \lor A \equiv A \]
\[ A \lor \neg A \equiv true \]
\[ A \lor true \equiv true \]
\[ A \lor false \equiv A \]
\[ A \land true \equiv A \]
\[ A \land false \equiv false \]
\[ A \land \neg A \equiv false \]
\[ A \Rightarrow true \equiv true \]
\[ A \Rightarrow false \equiv \neg A \]
\[ true \Rightarrow A \equiv A \]
\[ false \Rightarrow A \equiv true \]
\[ A \Rightarrow B \equiv \neg A \lor B \]
\[ A \Rightarrow B \equiv \neg B \Rightarrow \neg A \]
\[ \neg (A \Rightarrow B) \equiv A \land \neg B \]
\[ A \land (A \lor B) \equiv A \]
\[ A \lor (A \land B) \equiv A \]
\[ A \land (\neg A \lor B) \equiv A \land B \]
\[ A \lor (\neg A \land B) \equiv A \lor B \]
\[ \neg (A \land B) \equiv \neg A \lor \neg B \]
\[ \neg (A \lor B) \equiv \neg A \land \neg B \]
Formal Logic

Logical axioms are logical statements that can be proven using such a truth table approach. Some of the more common axioms are shown on the right.

\[
\begin{align*}
A \land (B \lor C) &\equiv (A \land B) \lor (A \land C) \\
A \lor (B \land C) &\equiv (A \lor B) \land (A \lor C) \\
\neg A &\equiv A \\
A \lor A &\equiv A \\
A \lor \neg A &\equiv \text{true} \\
A \lor \text{true} &\equiv \text{true} \\
A \lor \text{false} &\equiv A \\
A \land \text{true} &\equiv A \\
A \land \text{false} &\equiv \text{false} \\
A \land A &\equiv A \\
A \land \neg A &\equiv \text{false} \\
A \Rightarrow \text{true} &\equiv \text{true} \\
A \Rightarrow \text{false} &\equiv \neg A \\
\text{true} &\equiv A \\
\text{false} &\equiv A \equiv \text{true} \\
A &\Rightarrow A \equiv \text{true} \\
A &\Rightarrow B \equiv \neg A \lor B \\
A &\Rightarrow B \equiv \neg B \Rightarrow \neg A \\
\neg(A \Rightarrow B) &\equiv A \land \neg B \\
A \land (A \lor B) &\equiv A \\
A \lor (A \land B) &\equiv A \\
A \land (\neg A \lor B) &\equiv A \land B \\
A \lor (\neg A \land B) &\equiv A \lor B \\
\neg(A \land B) &\equiv \neg A \lor \neg B \\
\neg(A \lor B) &\equiv \neg A \land \neg B
\end{align*}
\]

Some are “simple” … others may be not as intuitive
Formal Logic

Logical axioms are logical statements that can be proven using such a truth table approach. Some of the more common axioms are shown on the right.

\[ A \land (B \lor C) \equiv (A \land B) \lor (A \land C) \]
\[ A \lor (B \land C) \equiv (A \lor B) \land (A \lor C) \]
\[ \neg A \equiv A \]
\[ A \lor A \equiv A \]
\[ A \lor \neg A \equiv \text{true} \]
\[ A \lor \text{true} \equiv \text{true} \]
\[ A \lor \text{false} \equiv A \]
\[ A \land \text{true} \equiv A \]
\[ A \land \text{false} \equiv \text{false} \]
\[ A \land \neg A \equiv \text{false} \]
\[ A \Rightarrow \text{true} \equiv \text{true} \]
\[ A \Rightarrow \text{false} \equiv \neg A \]
\[ \text{true} \Rightarrow A \equiv A \]
\[ \text{false} \Rightarrow A \equiv \text{true} \]
\[ A \Rightarrow B \equiv \neg A \lor B \]
\[ A \lor B \equiv \neg B \Rightarrow \neg A \]
\[ \neg (A \Rightarrow B) \equiv A \land \neg B \]
\[ A \land (A \lor B) \equiv A \]
\[ A \lor (A \land B) \equiv A \]
\[ A \land (\neg A \lor B) \equiv A \land B \]
\[ A \lor (\neg A \land B) \equiv A \lor B \]
\[ \neg (A \land B) \equiv \neg A \lor \neg B \]
\[ \neg (A \lor B) \equiv \neg A \land \neg B \]

Some are “simple” … others may be not as intuitive … and others still less “visible” to intuition can be “shown” via a truth table approach.

In-class exercise
Formal Logic Systems

To reason logically about statements, we need formal rules to prevent confusion. We want to make statements that are precise and unambiguous.
Formal Logic Systems

To reason logically about statements, we need formal rules to prevent confusion. We want to make statements that are precise and unambiguous.

Symbols: a set of “characters”
Formulas: Constructed from symbols; well-formed
Axioms: special formula that are a priori assumed to be True
Inference Rules: “rules” for how to derive additional formulas that are True
Formal Logic Systems

To reason logically about statements, we need formal rules to prevent confusion. We want to make statements that are precise and unambiguous.

Symbols: a set of “characters”
Formulas: Constructed from symbols; well-formed
Axioms: special formula that are assumed to be True
Inference Rules: “rules” for how to derive additional formulas that are True

Q: What does a priori mean?
Q: What is the “opposite” of a priori?
Formal Logic Systems

**Inference Rules**: “rules” for how to derive additional formulas that are True

- **Modus Ponens**
  \[
  A \implies B, A \\
  \hline
  B
  \]

- **Modus Tollens**
  \[
  A \implies B, \neg B \\
  \hline
  \neg A
  \]

- **Conjunction**
  \[
  A, B \\
  \hline
  A \land B
  \]

- **Simplification**
  \[
  A \land B \\
  \hline
  A
  \]

- **Addition**
  \[
  A \\
  \hline
  A \lor B
  \]

- **Disjunctive syllogism**
  \[
  A \lor B, \neg A \\
  \hline
  B
  \]

- **Hypothetical syllogism**
  \[
  A \implies B, B \implies C \\
  \hline
  A \implies C
  \]

- **Constructive dilemma**
  \[
  A \lor B, A \implies C, B \implies D \\
  \hline
  C \lor D
  \]

- **Destructive dilemma**
  \[
  \neg C \lor \neg D, A \implies C, B \implies D \\
  \hline
  \neg A \lor \neg B
  \]
Formal Logic Systems

**Inference Rules**: “rules” for how to derive additional formulas that are True

- **Modus Ponens**
  \[ A \Rightarrow B, A \]
  \[ B \]

- **Modus Tollens**
  \[ A \Rightarrow B, \neg B \]
  \[ \neg A \]

- **Conjunction**
  \[ A, B \]
  \[ A \land B \]

- **Simplification**
  \[ A \land B \]
  \[ A \]

- **Addition**
  \[ A \]
  \[ A \lor B \]

Inference rules have a **hypothesis**, and a **conclusion**

If the hypothesis is true, then the conclusion is true.
Formal Logic Systems

**Inference Rules**: “rules” for how to derive additional formulas that are True

- **Modus Ponens**
  \[
  A \implies B, \ A \\
  \hline
  B
  \]

- **Modus Tollens**
  \[
  A \implies B, \ \neg B \\
  \hline
  \neg A
  \]

- **Conjunction**
  \[
  A, \ B \\
  \hline
  A \land B
  \]

- **Simplification**
  \[
  A \land B \\
  \hline
  A
  \]

- **Addition**
  \[
  A \\
  \hline
  A \lor B
  \]

**Q**: How do you “read” modus ponens?

**Q**: How is Modus Ponens related to Implication?
Formal Logic Systems

Inference Rules: “rules” for how to derive additional formulas that are True

- Modus Ponens
  \[ \frac{A \Rightarrow B, A}{B} \]

- Modus Tollens
  \[ \frac{A \Rightarrow B, \neg B}{\neg A} \]

- Conjunction
  \[ \frac{A, B}{A \land B} \]

- Simplification
  \[ \frac{A \land B}{A} \]

- Addition
  \[ \frac{A}{A \lor B} \]

Q: How do you “read” modus ponens?
Q: How is Modus Ponens related to Implication?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>\neg A</th>
<th>A \land B</th>
<th>A \lor B</th>
<th>A \Rightarrow B</th>
<th>A \equiv B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Inference Rules: “rules” for how to derive additional formulas that are True

- **Modus Ponens**
  \[
  \frac{A \Rightarrow B, A}{B}
  \]

- **Modus Tollens**
  \[
  \frac{A \Rightarrow B, \neg B}{\neg A}
  \]

- **Conjunction**
  \[
  \frac{A, B}{A \land B}
  \]

- **Simplification**
  \[
  \frac{A \land B}{A}
  \]

- **Addition**
  \[
  \frac{A}{A \lor B}
  \]

Q: How do you “read” modus ponens?
Q: How is Modus Ponens related to Implication?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(\neg A)</th>
<th>(A \land B)</th>
<th>(A \lor B)</th>
<th>(A \Rightarrow B)</th>
<th>(A \equiv B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Inference Rules: “rules” for how to derive additional formulas that are True

- **Modus Ponens**
  \[
  \frac{A \Rightarrow B, A}{B}
  \]

- **Modus Tollens**
  \[
  \frac{A \Rightarrow B, \neg B}{\neg A}
  \]

- **Conjunction**
  \[
  \frac{A, B}{A \land B}
  \]

- **Simplification**
  \[
  \frac{A \land B}{A}
  \]

- **Addition**
  \[
  \frac{A}{A \lor B}
  \]

Q: How do you “read” modus ponens?
Q: How is Modus Ponens related to Implication?

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>(\neg A)</td>
<td>A \land B</td>
<td>A \lor B</td>
<td>A \Rightarrow B</td>
<td>A \equiv B</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>
Formal Logic Systems

Inference Rules: “rules” for how to derive additional formulas that are True

- **Modus Ponens**
  \[
  A \Rightarrow B, \quad A \\
  \quad B
  \]

- **Modus Tollens**
  \[
  A \Rightarrow B, \neg B \\
  \neg A
  \]

- **Conjunction**
  \[
  A, B \\
  A \land B
  \]

- **Simplification**
  \[
  A \land B \\
  A
  \]

- **Addition**
  \[
  A \\
  A \lor B
  \]

Q: How do you “read” modus ponens?
Q: How is Modus Ponens related to Implication?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>\neg A</th>
<th>A \land B</th>
<th>A \lor B</th>
<th>A \Rightarrow B</th>
<th>A \equiv B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Q: How are these axioms and inference rules “used”?

Conditional Proof (prove that \( X \) is a logical implication to \( Y \))

Indirect Proof (proof by contradiction; to prove \( X \))
Q: How are these axioms and inference rules “used”?

Conditional Proof (prove that \(X\) is a logical implication to \(Y\))

- Assume \(X\) is true
- Show that \(Y\) follows

Indirect Proof (proof by contradiction; to prove \(X\))

- Assume \(\text{NOT } X\)
- Show a contradiction (\(Z\) AND \(\text{NOT } Z\) for example)
- Therefore \(X\) is True
Proof Strategies

Prove: $P \Rightarrow (Q \Rightarrow (P \land Q))$
Proof Strategies

Prove: \( P \Rightarrow (Q \Rightarrow (P \land Q)) \)

1. \( P \)
2. \( \ldots \)
3. \( \ldots \)
4. \( Q \Rightarrow (P \land Q) \)
5. \( P \Rightarrow (Q \Rightarrow (P \land Q)) \)
Proof Strategies

This is the goal

Prove: $P \Rightarrow (Q \Rightarrow (P \land Q))$

1. $P$
2. $\ldots$
3. $\ldots$
4. $Q \Rightarrow (P \land Q)$

5. $P \Rightarrow (Q \Rightarrow (P \land Q))$
Proof Strategies

A box is used to specify a logical argument or sequence of using axioms and logic rules.

Prove: $P \Rightarrow (Q \Rightarrow (P \land Q))$

1. $P$
2. ...
3. ...
4. $Q \Rightarrow (P \land Q)$
5. $P \Rightarrow (Q \Rightarrow (P \land Q))$
Proof Strategies

The first line in this case is an assumption ... hence we are constructing a conditional proof.

Q: What “rules” or axioms should we apply for steps 2 and 3?
Proof Strategies

Prove: $P \Rightarrow (Q \Rightarrow (P \land Q))$

We use a nested proof:

1. $P$
2. $Q$
3. ...
4. ...
5. $P \land Q$
6. $Q \Rightarrow (P \land Q)$
7. $P \Rightarrow (Q \Rightarrow (P \land Q))$

Q: How does this help us? Or does it?
Proof Strategies

Prove: \( P \Rightarrow (Q \Rightarrow (P \land Q)) \)
We use a nested proof:

1. \( P \)
2. \( Q \)
3. \( \ldots \)
4. \( \ldots \)
5. \( P \land Q \)
6. \( Q \Rightarrow (P \land Q) \)
7. \( P \Rightarrow (Q \Rightarrow (P \land Q)) \)

Q: How does this help us? Or does it?

Line 5, follows from lines 1 and 2 via conjunction, so you can eliminate lines 3 and 4
Q: How does this help us? Or does it?

Line 5, follows from lines 1 and 2 via conjunction, so you can eliminate lines 3 and 4.
Proof Strategies

Prove: $A \Rightarrow (B \Rightarrow A)$

Q: What is proof by contradiction?

In-class exercise
Proof Strategies

Prove: \( A \Rightarrow (B \Rightarrow A) \)

1. \( \neg(A \Rightarrow (B \Rightarrow A)) \)
2. ...
3. \( Z \)
4. ...
5. \( \neg Z \)
6. \( A \Rightarrow (B \Rightarrow A) \)

The first line in this case is a negation of what you are trying to prove ... hence we are constructing a proof by contradiction.
Proof Strategies

The first line in this case is a negation of what you are trying to prove ... hence we are constructing a proof by contradiction.

\( \neg(A \Rightarrow (B \Rightarrow A)) \)

\( Z \) can be ANYTHING, but because each statement (line) in a proof must be valid and not contradict another statement, you cannot have BOTH \( Z \) and \( \neg Z \). If you do, that’s a contradiction, hence you invalidate the hypothesis, hence the original statement is True.
Proof Strategies

Prove: \( A \Rightarrow (B \Rightarrow A) \)

1. \( \neg( A \Rightarrow (B \Rightarrow A)) \)
2. ...
3. \( Z \)
4. ...
5. \( \neg Z \)
6. \( A \Rightarrow (B \Rightarrow A) \)

Statements can be proven via either of the 2 methods. It’s up to you. Using one may be “easier” than the other.

Hint: use the axioms to write equivalence statements. Use ...

\[ \neg( A \Rightarrow B ) \equiv A \land \neg B \]

Q: Does that help?

Take home exercise ... we’ll start tomorrow with the completed proof
Midterm Exam

Everything up to this point in lecture is fair game for the exam

(don’t worry too much about lengthy proofs, but be able to perform a “proof” by proof table and be able to “prove” any of the axioms)
Returning Homework #2

- **Question 1**: Brute force and math (on paper) calculation were both feasible
Returning Homework #2

- **Question 1**: Brute force and math (on paper) calculation were both feasible

- **Question 2**: Why do semaphores not have a strict getter method?
Returning Homework #2

- **Question 1**: Brute force and math (on paper) calculation were both feasible

- **Question 2**: Why do semaphores not have a strict getter method?

- **Q**: Why do canoes NOT have holes in their hull?
Returning Homework #2

- **Question 1**: Brute force and math (on paper) calculation were both feasible

- **Question 2**: Why do semaphores not have a strict getter method?
  
  - **Q: Why do canoes NOT have holes in their hull?**
    - Because the manufacturer makes canoes w/o holes
    - Because canoes don’t need holes
    - Because a canoe glides on the water
    - Because a canoe with a hole in its hull cannot perform its intended function ... it sinks!
Returning Homework #2

• **Question 1**: Brute force and math (on paper) calculation were both feasible

• **Question 2**: Why do semaphores not have a strict getter method?

• **Q**: Why do canoes NOT have holes in their hull?
  - Because the manufacturer makes canoes w/o holes
  - Because canoes don’t need holes
  - Because a canoe slides on the water
  - Because a canoe with a hole in its hull cannot perform its intended function ... it sinks!
Returning Homework #2

• **Question 1**: Brute force and math (on paper) calculation were both feasible

• **Question 2**: Why do semaphores not have a strict getter method?
  • Because semaphores don’t have strict getter methods
  • Because semaphores stall threads, so a strict getter method is not needed
  • Because a semaphore’s wait and signal methods increment or decrement the semaphore’s value, so a strict getter method is not needed
  • Because a strict getter method could provide a value that is obsolete by the time it is received by the caller

• **Q: Why do canoes NOT have holes in their hull?**
  • Because the manufacturer makes canoes w/o holes
  • Because canoes don’t need holes
  • Because a canoe glides on the water
  • Because a canoe with a hole in its hull cannot perform its intended function ... it sinks!
Returning Homework #2

- **Question 1**: Brute force and math (on paper) calculation were both feasible

- **Question 2**: Why do semaphores not have a strict getter method?
  - Because semaphores don’t have strict getter methods
  - Because semaphores stall threads, so a strict getter method is not needed
  - Because a semaphore’s wait and signal methods increment or decrement the semaphore’s value, so a strict getter method is not needed
  - Because a strict getter method could provide a value that is obsolete by the time it is received by the caller

- **Q**: Why do canoes NOT have holes in their hull?
  - Because the manufacturer makes canoes w/o holes
  - Because canoes don’t need holes
  - Because a canoe glides on the water
  - Because a canoe with a hole in its hull cannot perform its intended function ... it sinks!
Returning Homework #2

• **Question 1**: Brute force and math (on paper) calculation were both feasible

• **Question 2**: Why do semaphores not have a strict getter method?
  - Because semaphores don’t have strict getter methods
  - Because semaphores stall threads, so a strict getter method is not needed
  - Because a semaphore’s wait and signal methods increment or decrement the semaphore’s value, so a strict getter method is not needed
  - Because a strict getter method could provide a value that is obsolete by the time it is received by the caller

• **Question 3**: A solution with a single semaphore is NOT feasible

• **Q**: Why do canoes NOT have holes in their hull?
  - Because the manufacturer makes canoes w/o holes
  - Because canoes don’t need holes
  - Because a canoe glides on the water
  - Because a canoe with a hole in its hull cannot perform its intended function ... it sinks!
Returning Homework #2

• **Question 1**: Brute force and math (on paper) calculation were both feasible

• **Question 2**: Why do semaphores not have a strict getter method?
  • Because semaphores don’t have strict getter methods
  • Because semaphores stall threads, so a strict getter method is not needed
  • Because a semaphore’s wait and signal methods increment or decrement the semaphore’s value, so a strict getter method is not needed
  • Because a strict getter method could provide a value that is obsolete by the time it is received by the caller

• **Question 3**: A solution with a single semaphore is NOT feasible

• **Q**: Why do canoes NOT have holes in their hull?
  • Because the manufacturer makes canoes w/o holes
  • Because canoes don’t need holes
  • Because a canoe glides on the water
  • Because a canoe with a hole in its hull cannot perform its intended function ... it sinks!

• **Question 4**: Gain != speed
Proofs of Concurrency Programming