COMP3151/9154

Foundations of Concurrency

Shared Variable Proof Methods, Hardware-Assisted Critical Sections

Johannes Åman Pohjola
CSE, UNSW
Term 2 2022

Where we are at

In the last lecture we introduced the critical section problem, the four properties of critical section solutions, and some solutions for two processes.

Where we are at

In the last lecture we introduced the critical section problem, the four properties of critical section solutions, and some solutions for two processes.

We also introduced the SPIN model checking tool for rigorous analysis of candidate solutions.

Where we are at

In the last lecture we introduced the critical section problem, the four properties of critical section solutions, and some solutions for two processes.

We also introduced the SPIN model checking tool for rigorous analysis of candidate solutions.

In this lecture, we will introduce a formal proof method for verifying safety properties, and apply it to a new kind of critical section solution that relies on hardware support.

Transition Diagrams

Definition

Transition Diagrams

Definition

A *transition diagram* is a tuple (L, T, s, t) where:

- L is a set of *locations* (program counter values).
- $s \in L$ is a *entry location*.
- $t \in L$ is a *exit location*.
- T is a set of *transitions*.

Transition Diagrams

Definition

A *transition diagram* is a tuple (L, T, s, t) where:

- L is a set of *locations* (program counter values).
- $s \in L$ is a *entry location*.
- $t \in L$ is a *exit location*.
- T is a set of *transitions*.

A *transition* is written as $\ell_i \xrightarrow{g;f} \ell_j$ where:

- ℓ_i and ℓ_j are locations.
- g is a *guard* $\Sigma \to \mathbb{B}$
- f is a *state update* $\Sigma \to \Sigma$.

Transition Diagrams

Definition

A *transition diagram* is a tuple (L, T, s, t) where:

- L is a set of *locations* (program counter values).
- $s \in L$ is a *entry location*.
- $t \in L$ is a *exit location*.
- T is a set of *transitions*.

A *transition* is written as $\ell_i \xrightarrow{g;f} \ell_j$ where:

- ℓ_i and ℓ_j are locations.
- g is a *guard* $\Sigma \to \mathbb{B}$
- f is a *state update* $\Sigma \to \Sigma$.

$i \leftarrow 0;$
$s \leftarrow 0;$
while $i \neq N$ **do**
  $s \leftarrow s + i;$
  $i \leftarrow i + 1$
od

Transition Diagrams

Definition

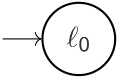
A *transition diagram* is a tuple (L, T, s, t) where:

- L is a set of *locations* (program counter values).
- $s \in L$ is a *entry location*.
- $t \in L$ is a *exit location*.
- T is a set of *transitions*.

A *transition* is written as $\ell_i \xrightarrow{g;f} \ell_j$ where:

- ℓ_i and ℓ_j are locations.
- g is a *guard* $\Sigma \to \mathbb{B}$
- f is a *state update* $\Sigma \to \Sigma$.

$i \leftarrow 0;$
$s \leftarrow 0;$
while $i \neq N$ **do**
  $s \leftarrow s + i;$
  $i \leftarrow i + 1$
od



Transition Diagrams

Definition

A *transition diagram* is a tuple (L, T, s, t) where:

- L is a set of *locations* (program counter values).
- $s \in L$ is a *entry location*.
- $t \in L$ is a *exit location*.
- T is a set of *transitions*.

A *transition* is written as $\ell_i \xrightarrow{g;f} \ell_j$ where:

- ℓ_i and ℓ_j are locations.
- g is a *guard* $\Sigma \to \mathbb{B}$
- f is a *state update* $\Sigma \to \Sigma$.

$i \leftarrow 0;$
$s \leftarrow 0;$
while $i \neq N$ **do**
  $s \leftarrow s + i;$
  $i \leftarrow i + 1$
od

$$\to \ell_0 \xrightarrow{\top;\, i \leftarrow 0} \ell_1$$

Transition Diagrams

Definition

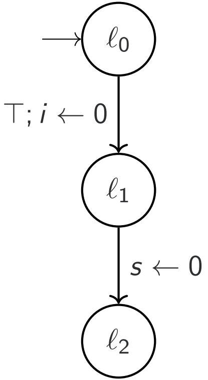
A *transition diagram* is a tuple (L, T, s, t) where:

- L is a set of *locations* (program counter values).
- $s \in L$ is a *entry location*.
- $t \in L$ is a *exit location*.
- T is a set of *transitions*.

A *transition* is written as $\ell_i \xrightarrow{g;f} \ell_j$ where:

- ℓ_i and ℓ_j are locations.
- g is a *guard* $\Sigma \to \mathbb{B}$
- f is a *state update* $\Sigma \to \Sigma$.

$i \leftarrow 0;$
$s \leftarrow 0;$
while $i \neq N$ **do**
  $s \leftarrow s + i;$
  $i \leftarrow i + 1$
od



Transition Diagrams

Definition

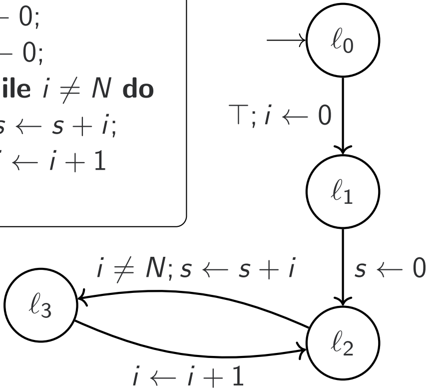
A *transition diagram* is a tuple (L, T, s, t) where:

- L is a set of *locations* (program counter values).
- $s \in L$ is a *entry location*.
- $t \in L$ is a *exit location*.
- T is a set of *transitions*.

A *transition* is written as $\ell_i \xrightarrow{g;f} \ell_j$ where:

- ℓ_i and ℓ_j are locations.
- g is a *guard* $\Sigma \to \mathbb{B}$
- f is a *state update* $\Sigma \to \Sigma$.

$i \leftarrow 0$;
$s \leftarrow 0$;
while $i \neq N$ **do**
  $s \leftarrow s + i$;
  $i \leftarrow i + 1$
od



Transition Diagrams

Definition

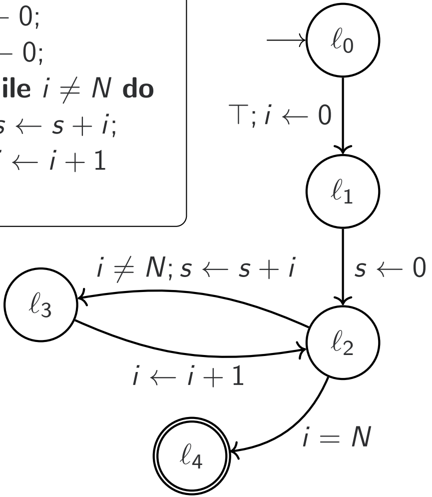
A *transition diagram* is a tuple (L, T, s, t) where:

- L is a set of *locations* (program counter values).
- $s \in L$ is a *entry location*.
- $t \in L$ is a *exit location*.
- T is a set of *transitions*.

A *transition* is written as $\ell_i \xrightarrow{g;f} \ell_j$ where:

- ℓ_i and ℓ_j are locations.
- g is a *guard* $\Sigma \rightarrow \mathbb{B}$
- f is a *state update* $\Sigma \rightarrow \Sigma$.

$i \leftarrow 0;$
$s \leftarrow 0;$
while $i \neq N$ **do**
    $s \leftarrow s + i;$
    $i \leftarrow i + 1$
od



Floyd Verification

Recall the definition of a Hoare triple for *partial correctness*:

$$\{\varphi\}\ P\ \{\psi\}$$

Floyd Verification

Recall the definition of a Hoare triple for *partial correctness*:

$$\{\varphi\}\ P\ \{\psi\}$$

This states that if the program P successfully executes from a starting state satisfying φ, the result state will satisfy ψ.

Floyd Verification

Recall the definition of a Hoare triple for *partial correctness*:

$$\{\varphi\}\ P\ \{\psi\}$$

This states that if the program P successfully executes from a starting state satisfying φ, the result state will satisfy ψ. Observe that this is a safety property.

Verifying Partial Correctness

Given a transition diagram (L, T, s, t):

Floyd Verification

Recall the definition of a Hoare triple for *partial correctness*:

$$\{\varphi\}\ P\ \{\psi\}$$

This states that if the program P successfully executes from a starting state satisfying φ, the result state will satisfy ψ. Observe that this is a safety property.

Verifying Partial Correctness

Given a transition diagram (L, T, s, t):

1. Associate with each location $\ell \in L$ an assertion $\mathcal{Q}(\ell) : \Sigma \to \mathbb{B}$.

Floyd Verification

Recall the definition of a Hoare triple for *partial correctness*:

$$\{\varphi\}\ P\ \{\psi\}$$

This states that if the program P successfully executes from a starting state satisfying φ, the result state will satisfy ψ. Observe that this is a safety property.

Verifying Partial Correctness

Given a transition diagram (L, T, s, t):

1. Associate with each location $\ell \in L$ an assertion $\mathcal{Q}(\ell) : \Sigma \to \mathbb{B}$.
2. Prove that this assertion network is *inductive*, that is: For each transition in T $\ell_i \xrightarrow{g;f} \ell_j$ show that:

$$\mathcal{Q}(\ell_i) \wedge g \Rightarrow \mathcal{Q}(\ell_j) \circ f$$

Floyd Verification

Recall the definition of a Hoare triple for *partial correctness*:

$$\{\varphi\}\ P\ \{\psi\}$$

This states that if the program P successfully executes from a starting state satisfying φ, the result state will satisfy ψ. Observe that this is a safety property.

Verifying Partial Correctness

Given a transition diagram (L, T, s, t):

1. Associate with each location $\ell \in L$ an assertion $\mathcal{Q}(\ell) : \Sigma \to \mathbb{B}$.
2. Prove that this assertion network is *inductive*, that is: For each transition in T $\ell_i \xrightarrow{g;f} \ell_j$ show that:

$$\mathcal{Q}(\ell_i) \wedge g \Rightarrow \mathcal{Q}(\ell_j) \circ f$$

3. Show that $\varphi \Rightarrow \mathcal{Q}(s)$ and $\mathcal{Q}(t) \Rightarrow \psi$.

Floyd Verification

Recall the definition of a Hoare triple for *partial correctness*:

$$\{\varphi\}\ P\ \{\psi\}$$

This states that if the program P successfully executes from a starting state satisfying φ, the result state will satisfy ψ. Observe that this is a safety property.

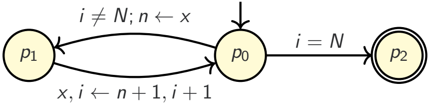
Verifying Partial Correctness

Given a transition diagram (L, T, s, t):

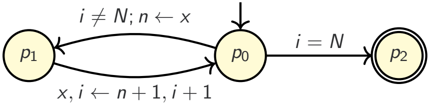
1. Associate with each location $\ell \in L$ an assertion $\mathcal{Q}(\ell) : \Sigma \to \mathbb{B}$.
2. Prove that this assertion network is *inductive*, that is: For each transition in T $\ell_i \xrightarrow{g;f} \ell_j$ show that:
$$\mathcal{Q}(\ell_i) \wedge g \Rightarrow \mathcal{Q}(\ell_j) \circ f$$
3. Show that $\varphi \Rightarrow \mathcal{Q}(s)$ and $\mathcal{Q}(t) \Rightarrow \psi$.

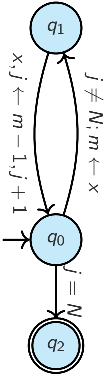
Johannes will now demonstrate on the previous example

Adding Concurrency

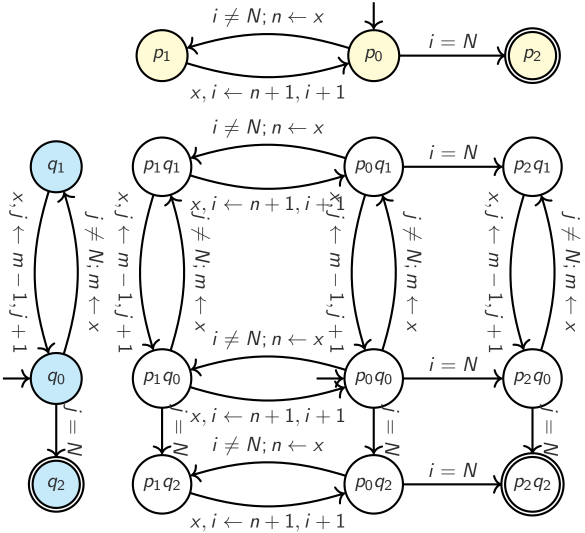


Adding Concurrency

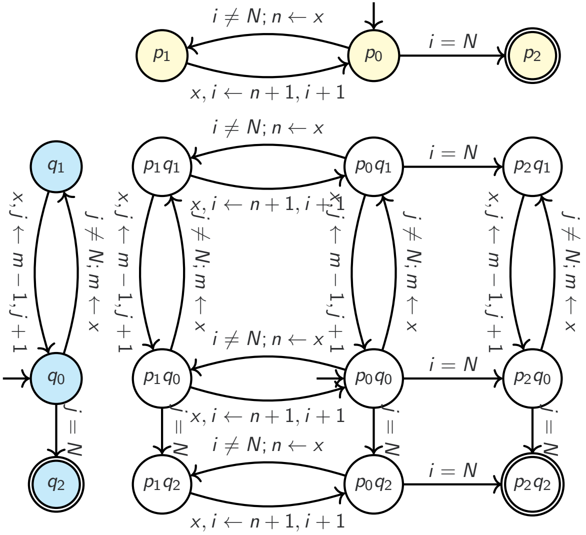




Adding Concurrency



Adding Concurrency



Parallel Composition

Given two processes P and Q with transition diagrams (L_P, T_P, s_P, t_P) and (L_Q, T_Q, s_Q, t_Q), the *parallel composition* of P and Q, written $P \parallel Q$ is defined as (L, T, s, t) where:

- $L = L_P \times L_Q$
- $s = s_P s_Q$
- $t = t_P t_Q$
- $p_i q_i \xrightarrow{g;f} p_j q_i \in T$ if $p_i \xrightarrow{g;f} p_j \in T_P$
- $p_i q_i \xrightarrow{g;f} p_i q_j \in T$ if $q_i \xrightarrow{g;f} q_j \in T_Q$

State Space Explosion

If we were SPIN, we would immediately begin exhaustively analysing this large diagram. But human brains don't have that much storage space.

State Space Explosion

If we were SPIN, we would immediately begin exhaustively analysing this large diagram. But human brains don't have that much storage space.

Problem

The number of locations and transitions grows exponentially as the number of processes increases.

State Space Explosion

If we were SPIN, we would immediately begin exhaustively analysing this large diagram. But human brains don't have that much storage space.

Problem

The number of locations and transitions grows exponentially as the number of processes increases.

We can only use Floyd's method directly on the parallel composition (product) diagram in the most basic examples.

State Space Explosion

If we were SPIN, we would immediately begin exhaustively analysing this large diagram. But human brains don't have that much storage space.

Problem

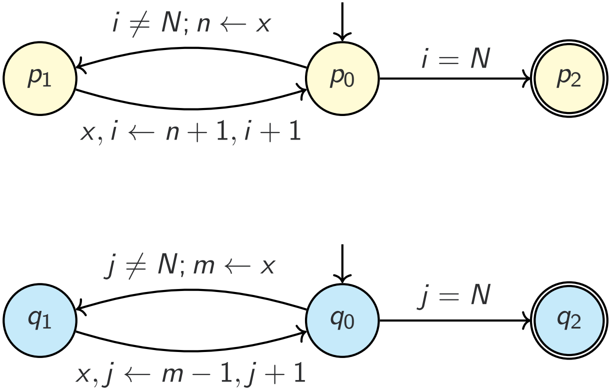
The number of locations and transitions grows exponentially as the number of processes increases.

We can only use Floyd's method directly on the parallel composition (product) diagram in the most basic examples.

Susan Owicki's solution

Define inductive assertion networks for P and Q separately. By proving some non-interference properties derive an inductive network for $P \parallel Q$ automatically. This means we won't have to draw that large product diagram!

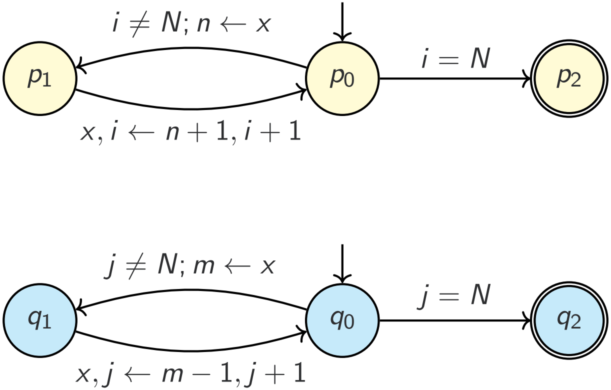
Owicki-Gries Method



Steps

To show $\{\varphi\}\ P \parallel Q\ \{\psi\}$:

Owicki-Gries Method

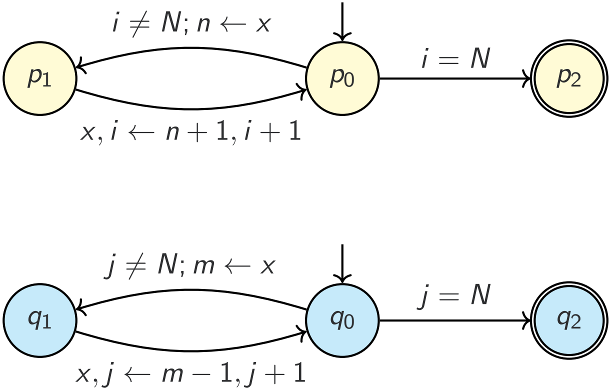


Steps

To show $\{\varphi\}\ P \parallel Q\ \{\psi\}$:

1. Define local assertion networks $\mathcal{P}$ and $\mathcal{Q}$ for both processes. Show that they're inductive.

Owicki-Gries Method



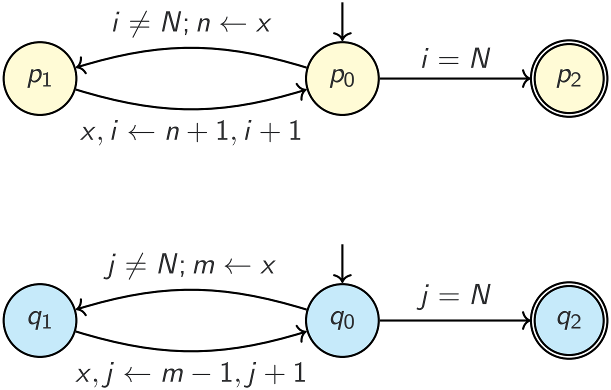
Steps

To show $\{\varphi\}\ P \parallel Q\ \{\psi\}$:

1. Define local assertion networks $\mathcal{P}$ and $\mathcal{Q}$ for both processes. Show that they're inductive.
2. For each location $p \in L_P$, show that $\mathcal{P}(p)$ is not falsified by any transition of Q. That is, for each $q \xrightarrow{g;f} q' \in T_Q$:

$$\mathcal{P}(p) \wedge \mathcal{Q}(q) \wedge g \Rightarrow \mathcal{P}(p) \circ f$$

Owicki-Gries Method

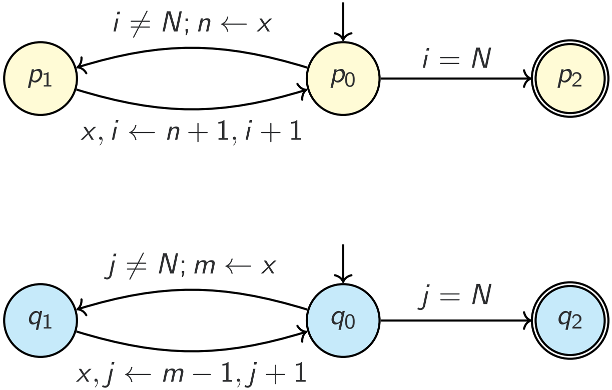


Steps

To show $\{\varphi\}\ P \parallel Q\ \{\psi\}$:

1. Define local assertion networks $\mathcal{P}$ and $\mathcal{Q}$ for both processes. Show that they're inductive.
2. For each location $p \in L_P$, show that $\mathcal{P}(p)$ is not falsified by any transition of Q. That is, for each $q \xrightarrow{g;f} q' \in T_Q$:
   $$\mathcal{P}(p) \wedge \mathcal{Q}(q) \wedge g \Rightarrow \mathcal{P}(p) \circ f$$
3. Vice versa for $\mathcal{Q}$.

Owicki-Gries Method



Steps

To show $\{\varphi\}\ P \parallel Q\ \{\psi\}$:

1. Define local assertion networks $\mathcal{P}$ and $\mathcal{Q}$ for both processes. Show that they're inductive.
2. For each location $p \in L_P$, show that $\mathcal{P}(p)$ is not falsified by any transition of Q. That is, for each $q \xrightarrow{g;f} q' \in T_Q$:

   $$\mathcal{P}(p) \wedge \mathcal{Q}(q) \wedge g \Rightarrow \mathcal{P}(p) \circ f$$
3. Vice versa for $\mathcal{Q}$.
4. Show that $\varphi \Rightarrow \mathcal{P}(s_P) \wedge \mathcal{Q}(s_Q)$ and $\mathcal{P}(t_P) \wedge \mathcal{Q}(t_Q) \Rightarrow \psi$.

How does it help?

The Owicki-Gries method generalises to n processes, by requiring more interference freedom obligations.

How does it help?

The Owicki-Gries method generalises to n processes, by requiring more interference freedom obligations.

Derived Assertion Network

The automatic assertion network we get for the parallel composition from the Owicki-Gries method is the conjunction of the local assertions at each of the component states.

How does it help?

The Owicki-Gries method generalises to n processes, by requiring more interference freedom obligations.

Derived Assertion Network

The automatic assertion network we get for the parallel composition from the Owicki-Gries method is the conjunction of the local assertions at each of the component states.

Assume k transitions and m locations per process. For m processes, Floyd's method spawns $2 + n \cdot k \cdot m^{n-1}$ proof obligations!

How does it help?

The Owicki-Gries method generalises to n processes, by requiring more interference freedom obligations.

Derived Assertion Network

The automatic assertion network we get for the parallel composition from the Owicki-Gries method is the conjunction of the local assertions at each of the component states.

Assume k transitions and m locations per process. For m processes, Floyd's method spawns $2 + n \cdot k \cdot m^{n-1}$ proof obligations!

Owicki-Gries reduces that to $2 + n \cdot k \cdot (1 + (n-1) \cdot m)$ — merely quadratic in n.

Proving Mutual Exclusion

The Owicki-Gries method can be used to prove properties like Mutual Exclusion.

Proving Mutual Exclusion

The Owicki-Gries method can be used to prove properties like Mutual Exclusion.

How? Don't have a postcondition. Instead make the assertions at the critical sections contradictory – so that they cannot be true simultaneously.

Proving Mutual Exclusion

The Owicki-Gries method can be used to prove properties like Mutual Exclusion.

How? Don't have a postcondition. Instead make the assertions at the critical sections contradictory – so that they cannot be true simultaneously.

Caution: Ensure that each transition does not violate the limited critical reference rule!

Proving Mutual Exclusion

The Owicki-Gries method can be used to prove properties like Mutual Exclusion.

How? Don't have a postcondition. Instead make the assertions at the critical sections contradictory – so that they cannot be true simultaneously.

Caution: Ensure that each transition does not violate the limited critical reference rule!

Manna-Pnueli Algorithm			
integer wantp, wantq $\leftarrow 0, 0$			
	forever do		**forever do**
p_1	*non-critical section*	q_1	*non-critical section*
p_2	**if** wantq $= -1$ **then** wantp $\leftarrow -1$ **else** wantp $\leftarrow 1$	q_2	**if** wantp $= -1$ **then** wantq $\leftarrow 1$ **else** wantq $\leftarrow -1$
p_3	**await** wantq $\neq$ wantp	q_3	**await** wantq $\neq -$wantp
p_4	*critical section*	q_4	*critical section*
p_5	wantp $\leftarrow 0$	q_5	wantq $\leftarrow 0$

Note: The p_2 and q_2 steps are one atomic step!

Machine Instructions

What about if we had a single machine instruction to swap two values atomically, `XC`?

Machine Instructions

What about if we had a single machine instruction to swap two values atomically, `XC`?

bit common ← 1	
bit tp ← 0	bit tq ← 0
forever do	**forever do**
p_1 *non-critical section*	q_1 *non-critical section*
repeat	**repeat**
p_2 `XC`(tp, common)	q_2 `XC`(tq, common);
p_3 **until** tp = 1	q_3 **until** tq = 1
p_4 **critical section**	q_4 **critical section**
p_5 `XC`(tp, common)	q_5 `XC`(tq, common)

One Big Invariant

Imagine assertion network(s) where every assertion is the same: An invariant.

One Big Invariant

Imagine assertion network(s) where every assertion is the same: An invariant.

Benefit: We don't need to prove interference freedom — the local verification conditions already show that the invariant is preserved.

One Big Invariant

Imagine assertion network(s) where every assertion is the same: An invariant.

Benefit: We don't need to prove interference freedom — the local verification conditions already show that the invariant is preserved.

Example (Exchange-based Critical Section Solution)

Using assertions about the program counters, we can craft an invariant for the `XC` example!

One Big Invariant

Imagine assertion network(s) where every assertion is the same: An invariant.

Benefit: We don't need to prove interference freedom — the local verification conditions already show that the invariant is preserved.

Example (Exchange-based Critical Section Solution)

Using assertions about the program counters, we can craft an invariant for the `XC` example!

$$\begin{array}{rcl}\mathcal{I} & \equiv & (\mathsf{common} \oplus \mathsf{tp} \oplus \mathsf{tq}) = 1 \land (P@p_4 \Rightarrow \mathsf{tp} = 1) \land \\ & & (Q@q_4 \Rightarrow \mathsf{tq} = 1) \land \neg(\mathsf{common} = \mathsf{tp} \land \mathsf{common} = \mathsf{tq})\end{array}$$

Where $\oplus$ is `xor`. Note: $\mathcal{I}$ is false at $p_4 q_4$. So if this invariant is preserved we have mutex.

One Big Invariant

Imagine assertion network(s) where every assertion is the same: An invariant.

Benefit: We don't need to prove interference freedom — the local verification conditions already show that the invariant is preserved.

Example (Exchange-based Critical Section Solution)

Using assertions about the program counters, we can craft an invariant for the `XC` example!

$$\begin{aligned}\mathcal{I} \quad \equiv \quad & (\mathsf{common} \oplus \mathsf{tp} \oplus \mathsf{tq}) = 1 \land (P@p_4 \Rightarrow \mathsf{tp} = 1) \land \\ & (Q@q_4 \Rightarrow \mathsf{tq} = 1) \land \neg(\mathsf{common} = \mathsf{tp} \land \mathsf{common} = \mathsf{tq})\end{aligned}$$

Where $\oplus$ is `xor`. Note: $\mathcal{I}$ is false at $p_4 q_4$. So if this invariant is preserved we have mutex.

Lets prove mutual exclusion for `XC`!

What now?

- You now have all you need to complete Assignment 0 (warm-up), due Monday the 20th.
- Next week: We will examine some more sophisticated critical section solutions for n processes.
- We may also learn about *semaphores*, time permitting!.