

Foundational and Compositional Verification of Layered Concurrent Objects

Yicheng Ni Yuting Wang

Shanghai Jiao Tong University

ECOOP, July 2026, Brussels

Motivation

Problem

Verification of Layered and Concurrent Systems

Background: Layered concurrent systems in the real world

- 1 OS kernels
- 2 File systems
- 3 I/O frameworks

Challenges:

- 1 Complexity of the system itself
- 2 Concurrent executions of multiple threads

Approaches:

- 1 Verification as a whole
- 2 Compositional Verification – Divide, Conquer, and Compose

File descriptor

Pathname

Directory

Inode

Logging

Buffer cache

Disk

Running Example

```
class BoundedQueue
```

```
    Array queue;
```

```
    Counter head;
```

```
    Counter tail;
```

```
    ENQ(val);
```

```
    DEQ();
```

```
class Counter
```

```
    Register reg;
```

```
    INC();
```

```
    GET();
```

```
class Array
```

```
    int ary[N];
```

```
    CAS(i,old,new);
```

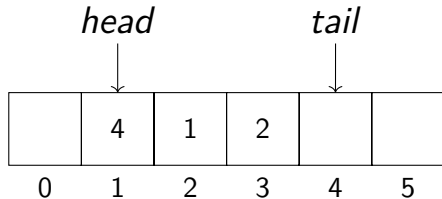
```
    READ(i);
```

```
class Register
```

```
    int value;
```

```
    CAS(old,new);
```

```
    READ();
```



Running Example

```
class BoundedQueue
```

```
    Array queue;
```

```
    Counter head;
```

```
    Counter tail;
```

```
    ENQ(val);
```

```
    DEQ();
```

Threads 1, 2, 3, ...

```
class Counter
```

```
    Register reg;
```

```
    INC();
```

```
    GET();
```

```
class Array
```

```
    int ary[N];
```

```
    CAS(i,old,new);
```

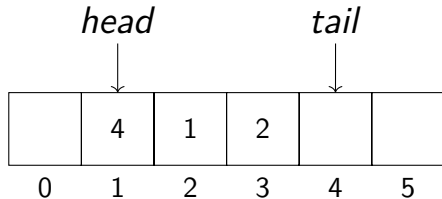
```
    READ(i);
```

```
class Register
```

```
    int value;
```

```
    CAS(old,new);
```

```
    READ();
```



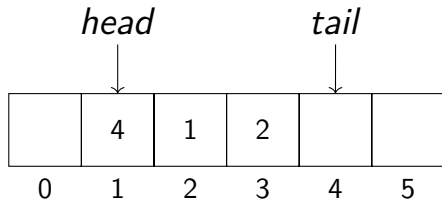
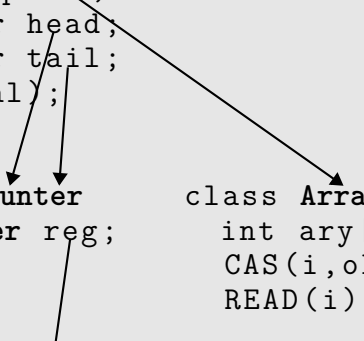
Running Example

```
class BoundedQueue
  Array queue;
  Counter head;
  Counter tail;
  ENQ(val);
  DEQ();

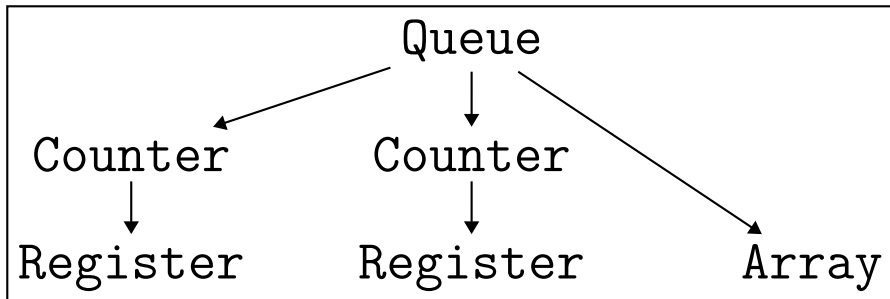
class Counter
  Register reg;
  INC();
  GET();

class Register
  int value;
  CAS(old,new);
  READ();

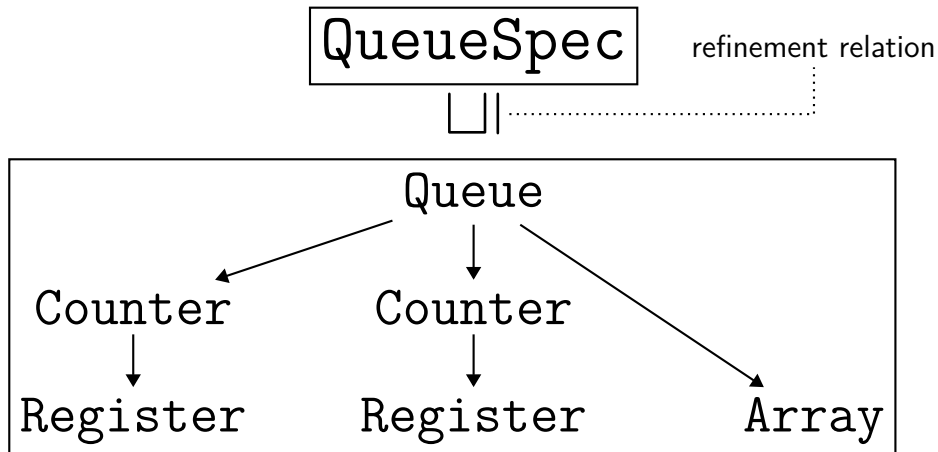
class Array
  int ary[N];
  CAS(i,old,new);
  READ(i);
```



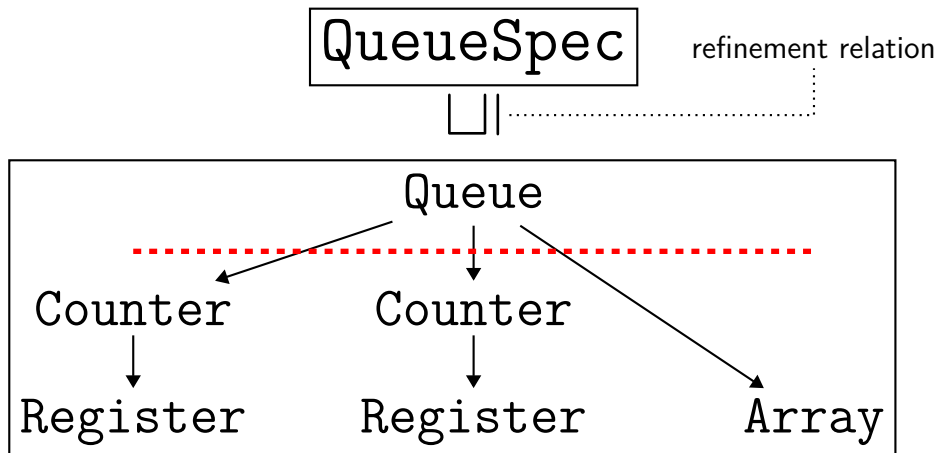
Compositional Verification



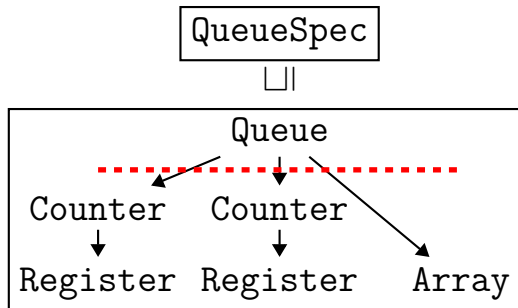
Compositional Verification



Compositional Verification: Vertically



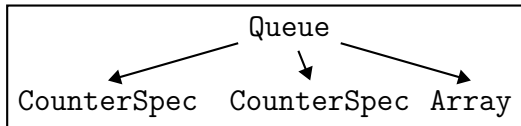
Compositional Verification: Vertically



Compositional Verification: Vertically

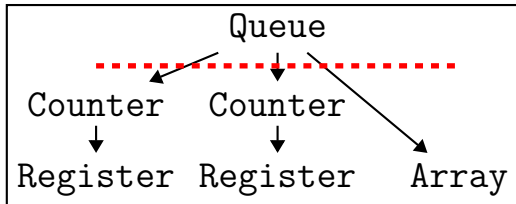
QueueSpec

□|



QueueSpec

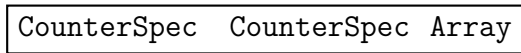
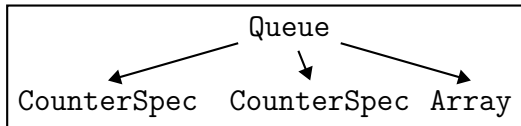
□|



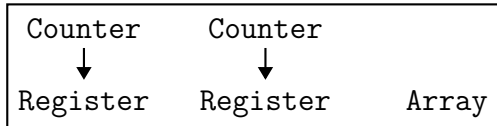
Compositional Verification: Vertically

QueueSpec

□|

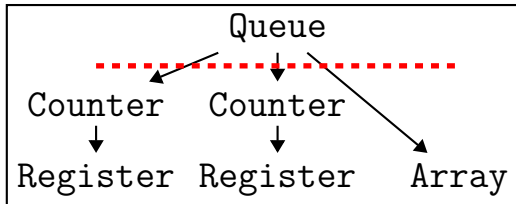


□|

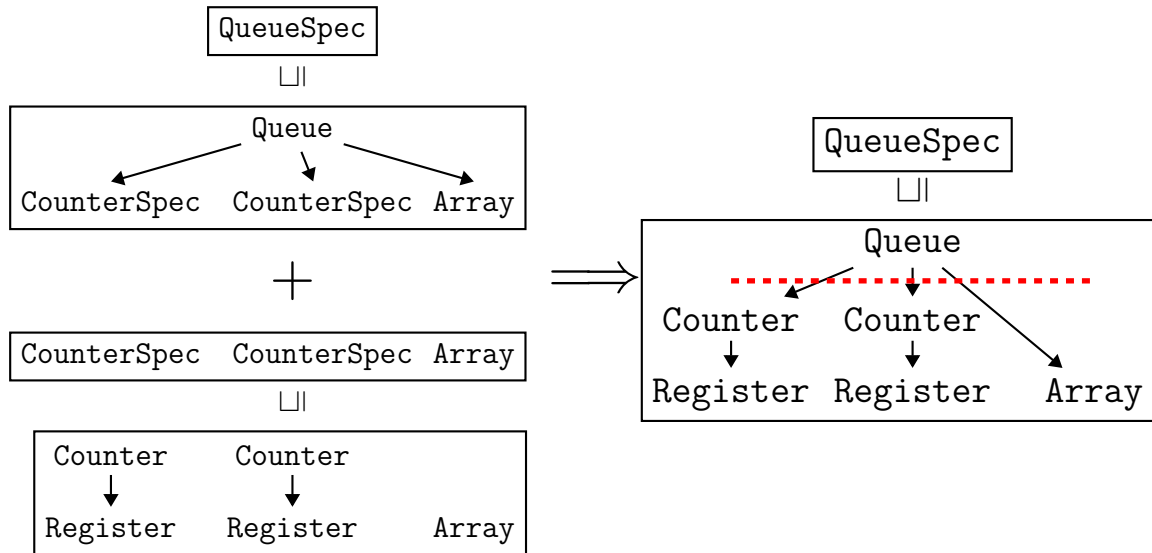


QueueSpec

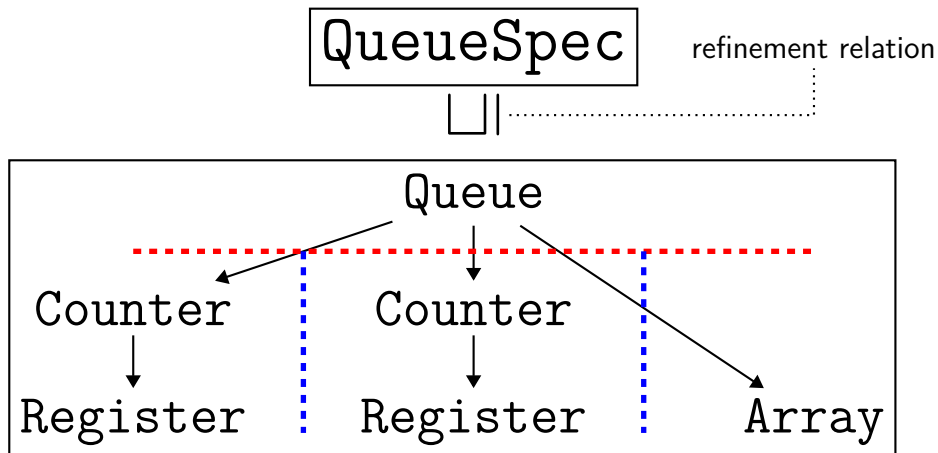
□|



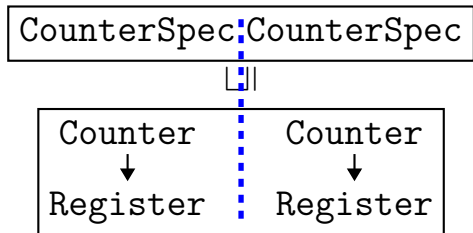
Compositional Verification: Vertically



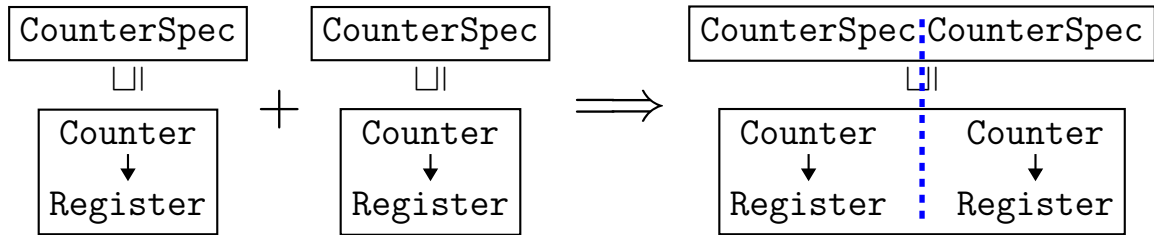
Compositional Verification: Horizontally



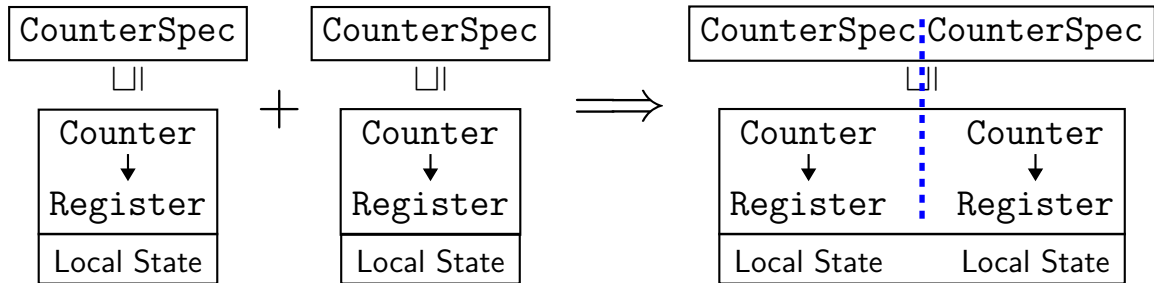
Compositional Verification: Horizontally



Compositional Verification: Horizontally



Compositional Verification: Horizontally



State-of-the-art

Reasoning with	Framework	Concurrency	Vertical Compositionality	H. Comp

State-of-the-art

Reasoning with	Framework	Concurrency	Vertical Compositionality	H. Comp
Global states	CertiKOS (OS kernel)	✓	✓	✗

State-of-the-art

Reasoning with	Framework	Concurrency	Vertical Compositionality	H. Comp
Global states	CertiKOS (OS kernel)	✓	✓	✗
Local states	ConFrm (File system)	✗	✓	✓

State-of-the-art

Reasoning with	Framework	Concurrency	Vertical Compositionality	H. Comp
Global states	CertiKOS (OS kernel)	✓	✓	✗
Local states	ConFrm (File system)	✗	✓	✓
Local states	Compositional Linearizability	✓	✓	✓

State-of-the-art

Reasoning with	Framework	Concurrency	Vertical Compositionality	H. Comp	Foundational
Global states	CertiKOS (OS kernel)	✓	✓	✗	✓
Local states	ConFrm (File system)	✗	✓	✓	✓
Local states	Compositional Linearizability	✓	✓	✓	✗

Our Contributions

Reasoning with	Framework	Concurrency	V. Comp	H. Comp	Foundational
Global states	CertiKOS	✓	✓	✗	✓
Local states	ConFrm	✗	✓	✓	✓
Local states	CompLin	✓	✓	✓	✗

Compositionality ← A unifying and compositional semantic model

Foundational proofs ← A model based on operational semantics

Our Contributions

Reasoning with	Framework	Concurrency	V. Comp	H. Comp	Foundational
Global states	CertiKOS	✓	✓	✗	✓
Local states	ConFrm	✗	✓	✓	✓
Local states	CompLin	✓	✓	✓	✗
Local states	Our Work	✓	✓	✓	✓

Our Contributions

Reasoning with	Framework	Concurrency	V. Comp	H. Comp	Foundational
Global states	CertiKOS	✓	✓	✗	✓
Local states	ConFrm	✗	✓	✓	✓
Local states	CompLin	✓	✓	✓	✗
Local states	Our Work	✓	✓	✓	✓

A **uniform**, **operational** and **compositional** framework for **concurrent** verification

Our Contributions

Reasoning with	Framework	Concurrency	V. Comp	H. Comp	Foundational
Global states	CertiKOS	✓	✓	✗	✓
Local states	ConFrm	✗	✓	✓	✓
Local states	CompLin	✓	✓	✓	✗
Local states	Our Work	✓	✓	✓	✓

A **uniform**, **operational** and **compositional** framework for **concurrent** verification

- 1 **Uniform** and Compositional Semantics for Concurrent Objects
- 2 **Uniform** and Compositional Correctness Criterion

Uniform and Compositional Semantics for Concurrent Objects

Case Study

```
class Counter
```

```
    Register rg;
```

```
    INC();
```

```
    GET();
```

```
void INC ()
```

```
1. int a := rg.READ()
```

```
2. int b := rg.CAS(a,a+1)
```

```
3. if !b then goto 1
```

```
4. return
```

```
int GET ()
```

```
1. int w := rg.READ()
```

```
2. return w
```

```
class Register
```

```
    int reg;
```

```
    // Compare-and-swap
```

```
    CAS(old, new);
```

```
    READ();
```

```
bool CAS (old, new)
```

```
1. if reg == old then
```

```
    reg := new
```

```
2. return true
```

```
3. return false
```

```
int READ ()
```

```
1. int v := reg
```

```
2. return v
```

$$M : A \rightarrow B$$

M is a concurrent object using a set of operations A to implement B

$$M : A \rightarrow B$$

M is a concurrent object using a set of operations A to implement B

Based on this idea, in our framework

$$M : A \rightarrow B$$

M is a concurrent object using a set of operations A to implement B

Based on this idea, in our framework

M is a concurrent transition system with layered interfaces A and B

An interface $A = (Q, R)$ specifies the set of query and reply actions

$$M_{cnt} : l_{reg} \rightarrow l_{cnt}$$

$$\begin{aligned} l_{cnt} &= (\{\text{INC}, \text{GET}\}, \{\text{INCok}, \text{GETok } n\}) \\ l_{reg} &= (\{\text{CAS } o \ n, \text{READ}\}, \{\text{CASok } b, \text{READok } n\}) \end{aligned}$$

Concurrent LTS with Layered Interfaces

```
void INC ()  $\leftarrow t_1$   
1.int a := rg.READ()  
2.int b := rg.CAS(a,a+1)  
3.if !b then goto 1  
4.return
```

```
int GET ()  
1.int w := rg.READ()  
2.return w
```

Concurrent LTS with Layered Interfaces

```
void INC ()
```

```
1.int a := rg.READ()  $\leftarrow t_1$ 
```

```
2.int b := rg.CAS(a,a+1)
```

```
3.if !b then goto 1
```

```
4.return
```

```
int GET ()
```

```
1.int w := rg.READ()
```

```
2.return w
```

Concurrent LTS with Layered Interfaces

```
void INC ()
```

```
1.int a := rg.READ()  $\leftarrow t_1$ 
```

```
2.int b := rg.CAS(a,a+1)
```

```
3.if !b then goto 1
```

```
4.return
```

```
int GET ()  $\leftarrow t_2$ 
```

```
1.int w := rg.READ()
```

```
2.return w
```

Concurrent LTS with Layered Interfaces

```
void INC ()
```

```
1.int a := rg.READ()  $\leftarrow t_1$ 
```

```
2.int b := rg.CAS(a,a+1)
```

```
3.if !b then goto 1
```

```
4.return
```

```
int GET ()
```

```
1.int w := rg.READ()  $\leftarrow t_2$ 
```

```
2.return w
```

Concurrent LTS with Layered Interfaces

```
void INC ()
```

```
1.int a := rg.READ()  $\rightarrow t_1$ 
```

```
2.int b := rg.CAS(a,a+1)
```

```
3.if !b then goto 1
```

```
4.return
```

```
int GET ()
```

```
1.int w := rg.READ()  $\leftarrow t_2$ 
```

```
2.return w
```

Concurrent LTS with Layered Interfaces

```
void INC ()
```

```
1.int a := rg.READ()  $\rightarrow t_1$ 
```

```
2.int b := rg.CAS(a,a+1)
```

```
3.if !b then goto 1
```

```
4.return
```

```
int GET ()
```

```
1.int w := rg.READ()  $\rightarrow t_2$ 
```

```
2.return w
```

Concurrent LTS with Layered Interfaces

```
void INC ()
```

```
1.int a := rg.READ()
```

```
2.int b := rg.CAS(a,a+1) ←  $t_1$ 
```

```
3.if !b then goto 1
```

```
4.return
```

```
int GET ()
```

```
1.int w := rg.READ() →  $t_2$ 
```

```
2.return w
```

Concurrent LTS with Layered Interfaces

```
void INC ()
```

```
1.int a := rg.READ()
```

```
2.int b := rg.CAS(a,a+1)  $\rightarrow t_1$ 
```

```
3.if !b then goto 1
```

```
4.return
```

```
int GET ()
```

```
1.int w := rg.READ()  $\rightarrow t_2$ 
```

```
2.return w
```

Concurrent LTS with Layered Interfaces

<pre>void INC () 1.int a := rg.READ() 2.int b := rg.CAS(a,a+1) → t_1 3.if !b then goto 1 4.return</pre>	<pre>int GET () 1.int w := rg.READ() → t_2 2.return w</pre>
--	--

State transition of $M_{cnt} : I_{reg} \rightarrow I_{cnt}$:



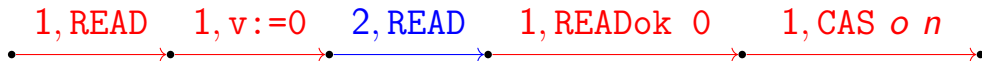
Concurrent LTS with Layered Interfaces

Concurrent LTS with Layered Interfaces

```
bool CAS (old, new)
1. if reg == old then
    reg := new
2. return true
3. return false
```

```
int READ ()
1. int v := reg
2. return v
```

State transition of $M_{reg} : \emptyset \rightarrow I_{reg}$:



Vertical and Horizontal Composition

Vertical Composition

Vertical and Horizontal Composition

Vertical Composition

$$M_{reg} : \emptyset \rightarrow I_{reg}$$

 $\implies$

$$M_{reg} \triangleleft M_{cnt} : \emptyset \rightarrow I_{cnt}$$

$$M_{cnt} : I_{reg} \rightarrow I_{cnt}$$

Vertical and Horizontal Composition

Vertical Composition

$$M_{reg} : \emptyset \rightarrow I_{reg}$$

 $\implies$

$$M_{reg} \triangleleft M_{cnt} : \emptyset \rightarrow I_{cnt}$$

$$M_{cnt} : I_{reg} \rightarrow I_{cnt}$$

Given a state transition of M_{cnt}



Vertical and Horizontal Composition

Vertical Composition

$$M_{reg} : \emptyset \rightarrow I_{reg}$$

 $\implies$

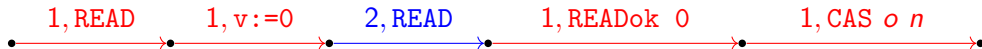
$$M_{reg} \triangleleft M_{cnt} : \emptyset \rightarrow I_{cnt}$$

$$M_{cnt} : I_{reg} \rightarrow I_{cnt}$$

Given a state transition of M_{cnt}



and a state transition of M_{reg}



Vertical and Horizontal Composition

Vertical Composition

$$M_{reg} : \emptyset \rightarrow I_{reg}$$

 $\implies$

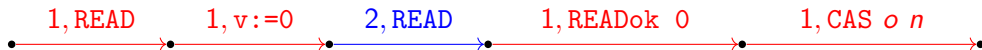
$$M_{reg} \triangleleft M_{cnt} : \emptyset \rightarrow I_{cnt}$$

$$M_{cnt} : I_{reg} \rightarrow I_{cnt}$$

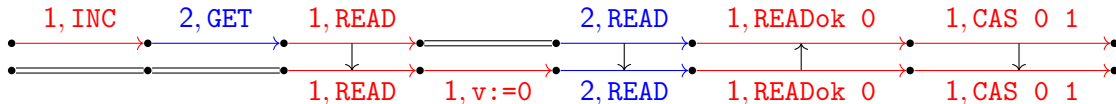
Given a state transition of M_{cnt}



and a state transition of M_{reg}



Add stuttering steps and match the actions on I_{reg} :



Vertical and Horizontal Composition

Horizontal Composition ($M_{cnt} \otimes M_{cnt} : I_{reg} \otimes I_{reg} \rightarrow I_{cnt} \otimes I_{cnt}$)

Vertical and Horizontal Composition

Horizontal Composition ($M_{cnt} \otimes M_{cnt} : I_{reg} \otimes I_{reg} \rightarrow I_{cnt} \otimes I_{cnt}$)

Given a state transition of M_{cnt}



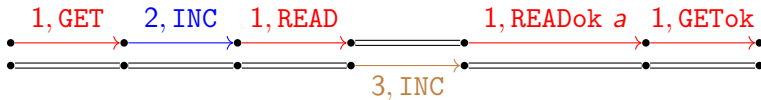
Vertical and Horizontal Composition

Horizontal Composition ($M_{cnt} \otimes M_{cnt} : I_{reg} \otimes I_{reg} \rightarrow I_{cnt} \otimes I_{cnt}$)

Given a state transition of M_{cnt}



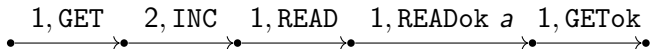
Interleave it with a state transition of another M_{cnt} :



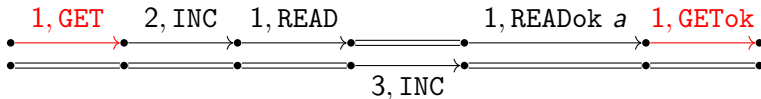
Vertical and Horizontal Composition

Horizontal Composition ($M_{cnt} \otimes M_{cnt} : I_{reg} \otimes I_{reg} \rightarrow I_{cnt} \otimes I_{cnt}$)

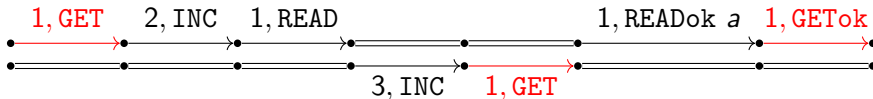
Given a state transition of M_{cnt}



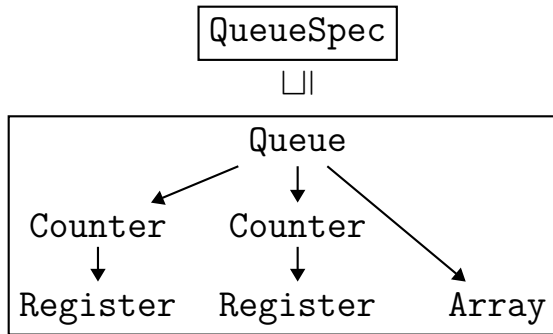
Interleave it with a state transition of another M_{cnt} :



Enforce Sequential Consistency



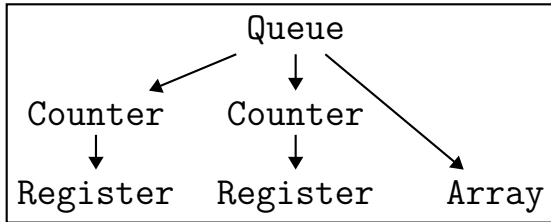
Unifying and Compositional Construction



Unifying and Compositional Construction

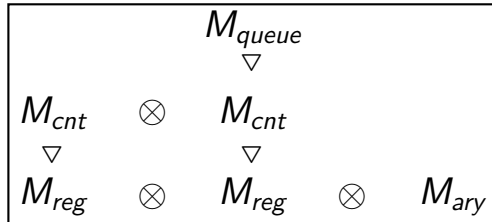
QueueSpec

$\sqcup$



$M_{queuespec}$

$\sqcup$



Uniform and Compositional Correctness Criterion

Unifying Correctness Criterion: Trace Refinement

Given a state transition



it produces a trace

$[(1, \text{INC}), (2, \text{GET}), (1, \text{READ}), (2, \text{READ}), (1, \text{READOk } a), (1, \text{CAS } a (a + 1))]$

Unifying Correctness Criterion: Trace Refinement

Given a state transition



it produces a trace

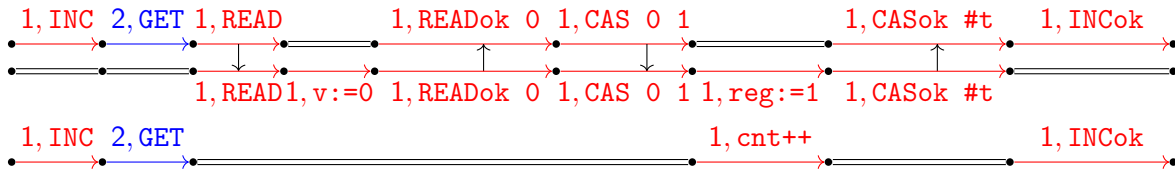
$[(1, \text{INC}), (2, \text{GET}), (1, \text{READ}), (2, \text{READ}), (1, \text{READOk } a), (1, \text{CAS } a (a + 1))]$

Trace Refinement $M_1 \sqsubseteq M_2$:

Given $M_1, M_2 : A \rightarrow B$, M_1 trace refines M_2
if for any trace t of M_1 , t is also a trace of M_2 .

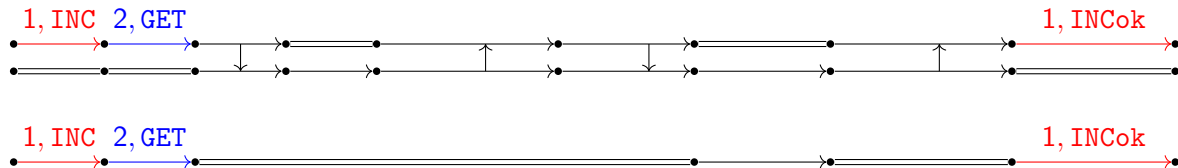
Unifying Correctness Criterion: Trace Refinement

Forward simulation between Counter ($M_{reg} \triangleleft M_{cnt}$) and CounterSpec ($M_{cntspec}$),
with state invariant f : $reg = cnt$



Unifying Correctness Criterion: Trace Refinement

Forward simulation between Counter ($M_{reg} \triangleleft M_{cnt}$) and CounterSpec ($M_{cntspec}$),
with state invariant f : $reg = cnt$



$$M_{reg} \triangleleft M_{cnt} \sim_f M_{cntspec} \implies M_{reg} \triangleleft M_{cnt} \sqsubseteq M_{cntspec}$$

Vertical Compositionality

$$\frac{M_1 \sqsubseteq M'_1}{M_1 \triangleleft M_2 \sqsubseteq M'_1 \triangleleft M_2}$$

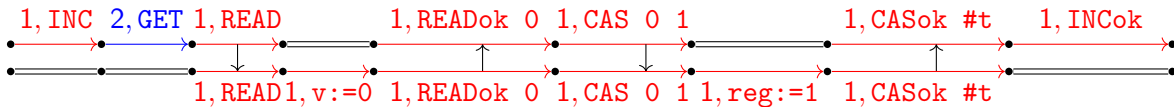
Horizontal Compositionality

$$\frac{M_1 \sqsubseteq M'_1 \quad M_2 \sqsubseteq M'_2}{M_1 \otimes M_2 \sqsubseteq M'_1 \otimes M'_2}$$

Vertical and Horizontal Compositionality

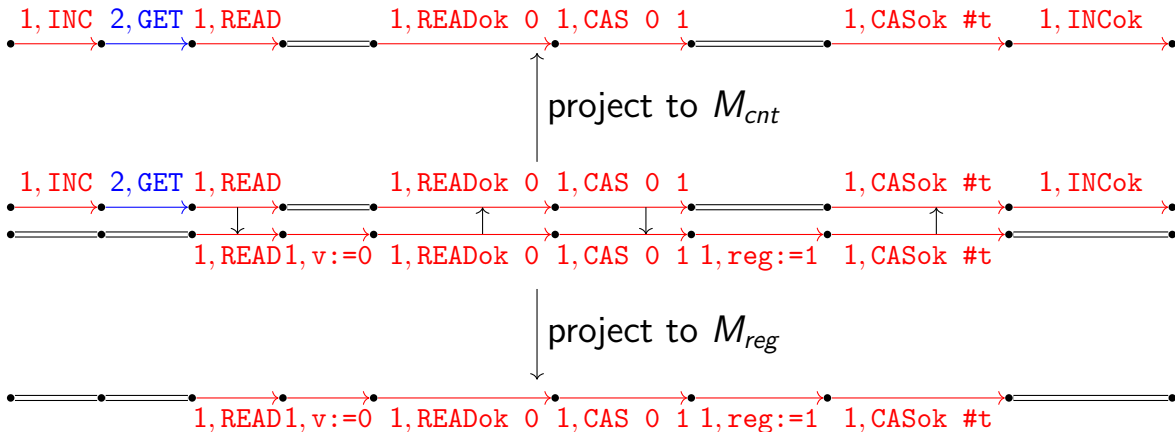
Vertical and Horizontal Compositionality

Projection of the vertically composed system $M_{reg} \triangleleft M_{cnt}$



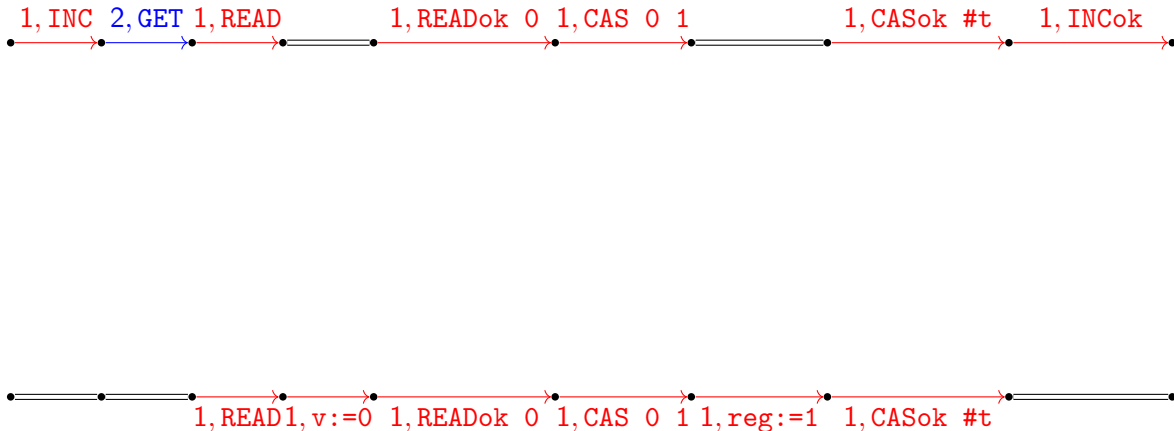
Vertical and Horizontal Compositionality

Projection of the vertically composed system $M_{reg} \triangleleft M_{cnt}$



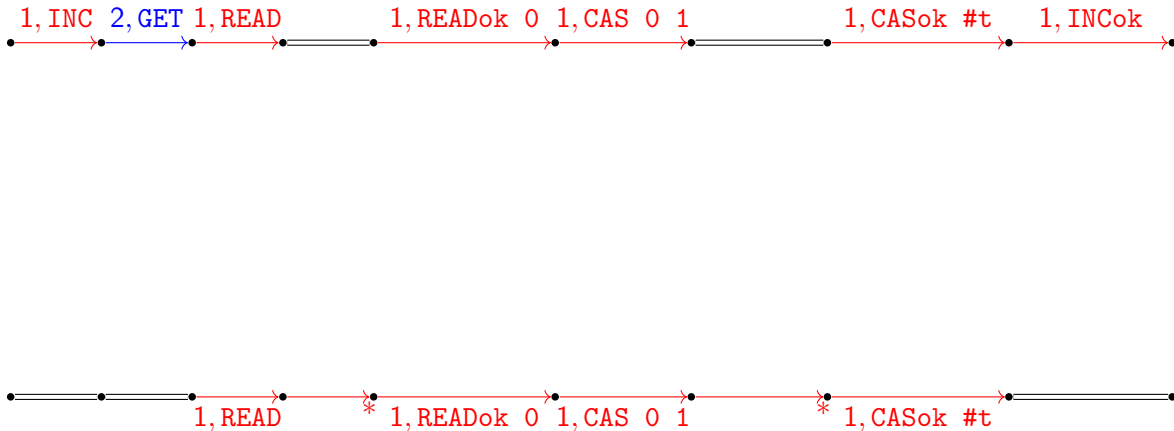
Vertical and Horizontal Compositionality

Given $M_{reg} \sqsubseteq M_{reg'}$,



Vertical and Horizontal Compositionality

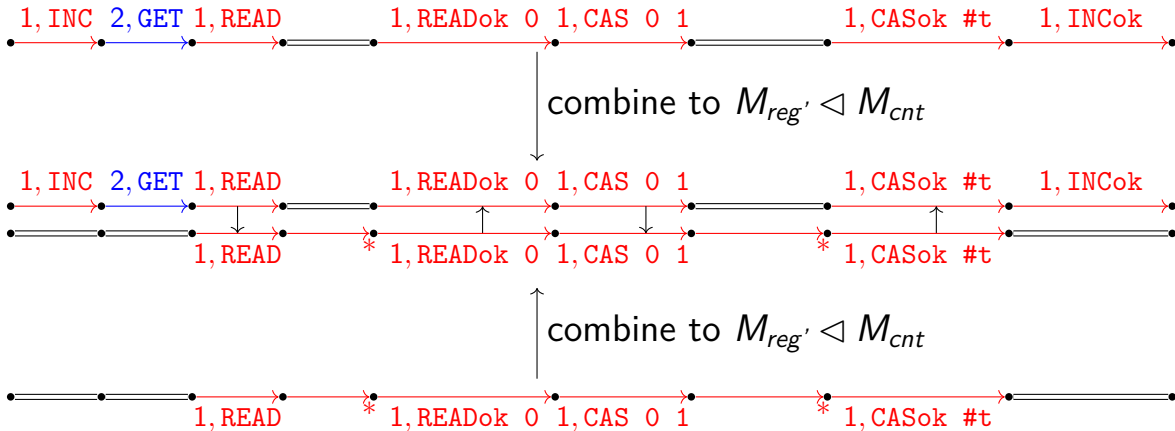
Given $M_{reg} \sqsubseteq M_{reg'}$,



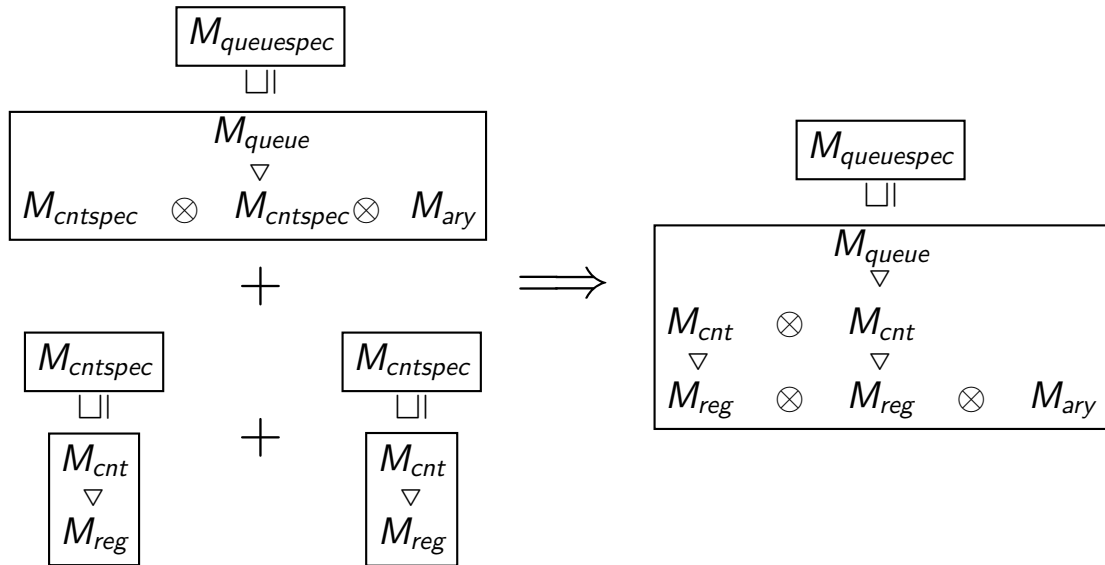
Vertical and Horizontal Compositionality

Given $M_{reg} \sqsubseteq M_{reg'}$,

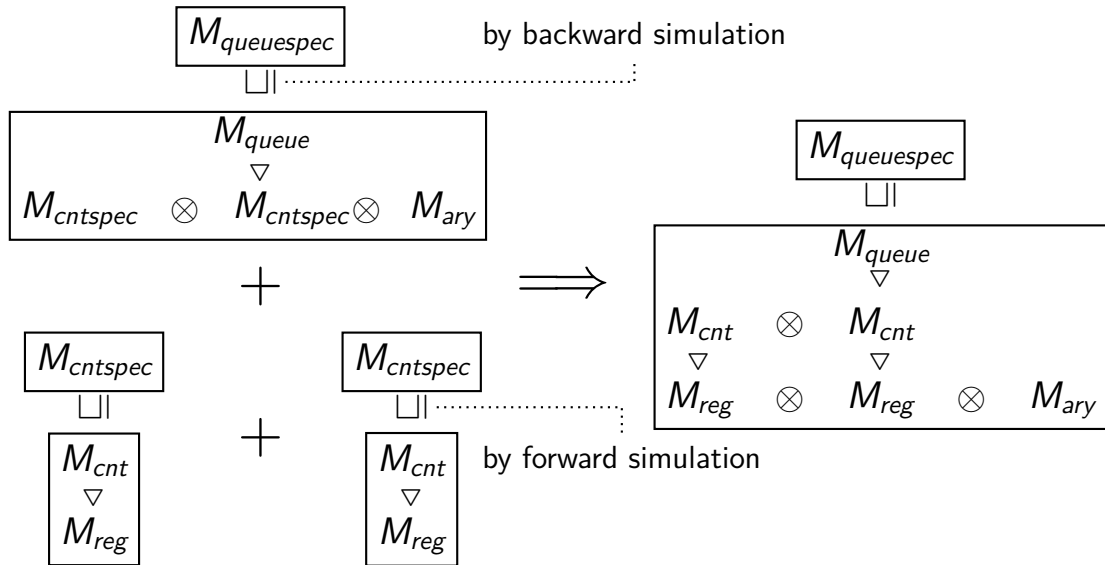
combination of $M_{reg'}$ and M_{cnt} forms $M_{reg'} \triangleleft M_{cnt}$



Vertical and Horizontal Compositionality



Vertical and Horizontal Compositionality



Evaluation and Conclusion

- ① A uniform, operational and compositional framework for concurrent verification
- ② Foundational proofs that adopts existing verification techniques

Statistics of Our Development

Components	LOC
Basic data types	1.4k
Compositional Verification Framework	14.3k
Register-Counter-Timer	5.4k
Bounded Queue	53.9k

Future Work:

- ① Verify real-world concurrent systems
- ② Support reduction techniques
- ③ Explore automation techniques



[https://doi.org/10.5281/
zenodo.19876027](https://doi.org/10.5281/zenodo.19876027)