

# Concurrent specifications beyond linearizability

Éric Goubault   **Jérémy Ledent**   Samuel Mimram

École Polytechnique, France

OPODIS 2018, Hong Kong

December 19, 2018

# Objects

Processes communicate through shared **objects**. For example:

# Objects

Processes communicate through shared **objects**. For example:

- ▶ Hardware: Read/Write registers, test&set, CAS, ...

# Objects

Processes communicate through shared **objects**. For example:

- ▶ Hardware: Read/Write registers, test&set, CAS, ...
- ▶ Data structures: lists, queues, hashmaps, ...

# Objects

Processes communicate through shared **objects**. For example:

- ▶ Hardware: Read/Write registers, test&set, CAS, ...
- ▶ Data structures: lists, queues, hashmaps, ...
- ▶ Message passing interfaces

# Objects

Processes communicate through shared **objects**. For example:

- ▶ Hardware: Read/Write registers, test&set, CAS, ...
- ▶ Data structures: lists, queues, hashmaps, ...
- ▶ Message passing interfaces
- ▶ Immediate-snapshot, consensus, set-agreement, ...

# Objects

Processes communicate through shared **objects**. For example:

- ▶ Hardware: Read/Write registers, test&set, CAS, ...
- ▶ Data structures: lists, queues, hashmaps, ...
- ▶ Message passing interfaces
- ▶ Immediate-snapshot, consensus, set-agreement, ...

**Goal:** can we implement object  $B$  using objects  $A_1, \dots, A_k$ ?

# Objects

Processes communicate through shared **objects**. For example:

- ▶ Hardware: Read/Write registers, test&set, CAS, ...
- ▶ Data structures: lists, queues, hashmaps, ...
- ▶ Message passing interfaces
- ▶ Immediate-snapshot, consensus, set-agreement, ...

**Goal:** can we implement object  $B$  using objects  $A_1, \dots, A_k$ ?

→ We need to **specify** the behavior of the objects.

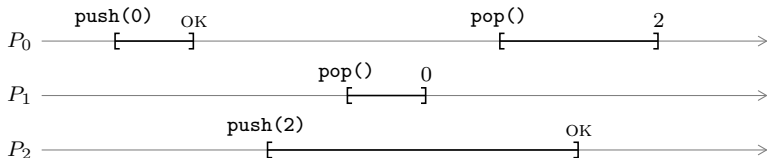


## Concurrent specifications

**Idea:** the specification of an object is the set of all the correct execution traces (Lamport, 1986).

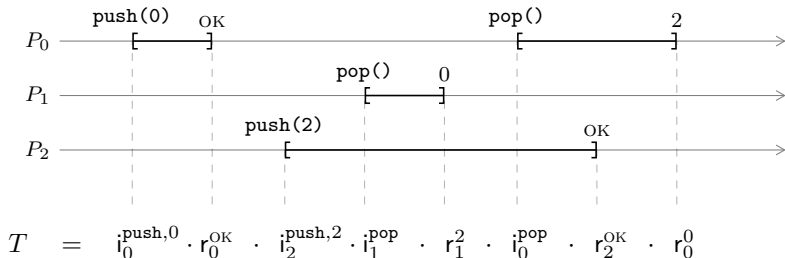
## Concurrent specifications

**Idea:** the specification of an object is the set of all the correct execution traces (Lamport, 1986).



## Concurrent specifications

**Idea:** the specification of an object is the set of all the correct execution traces (Lamport, 1986).

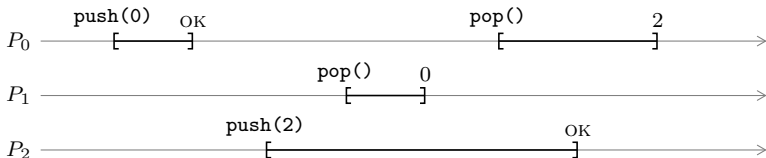


### Trace formalism:

- ▶ Time is abstracted away.
- ▶ Alternation of invocations and responses on each process.

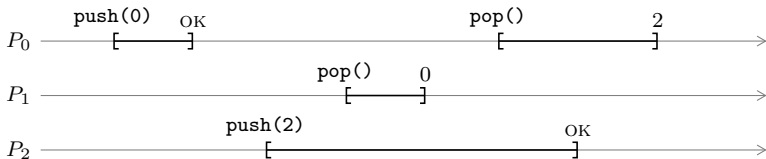
## Concurrent specifications

**Idea:** the specification of an object is the set of all the correct execution traces (Lamport, 1986).



## Concurrent specifications

**Idea:** the specification of an object is the set of all the correct execution traces (Lamport, 1986).

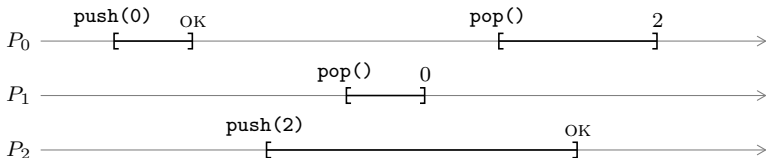


Write  $\mathcal{T}$  for the set of all execution traces.

- ▶ A *concurrent specification* is a subset  $\sigma \subseteq \mathcal{T}$ .

## Concurrent specifications

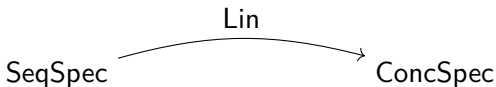
**Idea:** the specification of an object is the set of all the correct execution traces (Lamport, 1986).



Write  $\mathcal{T}$  for the set of all execution traces.

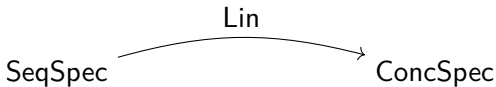
- ▶ A *concurrent specification* is a subset  $\sigma \subseteq \mathcal{T}$ .
- ▶ A program *implements* a specification  $\sigma$  if all the traces that it can produce belong to  $\sigma$ .

## Linearizability (Herlihy & Wing, 1990)

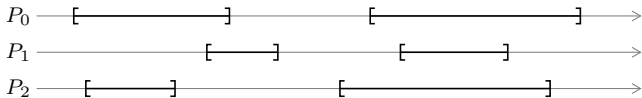


- ▶ **Input:** a sequential specification  $\sigma$  (e.g. list, queue, ...).
- ▶ **Output:** a concurrent specification  $\text{Lin}(\sigma)$ .

## Linearizability (Herlihy & Wing, 1990)

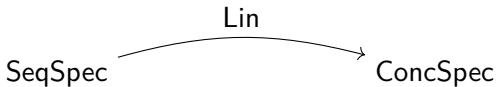


- ▶ **Input:** a sequential specification  $\sigma$  (e.g. list, queue, ...).
- ▶ **Output:** a concurrent specification  $\text{Lin}(\sigma)$ .

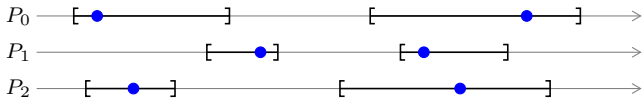




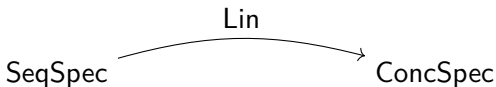
# Linearizability (Herlihy & Wing, 1990)



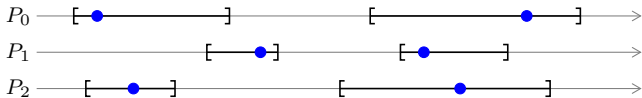
- ▶ **Input:** a sequential specification  $\sigma$  (e.g. list, queue, ...).
- ▶ **Output:** a concurrent specification  $\text{Lin}(\sigma)$ .



# Linearizability (Herlihy & Wing, 1990)

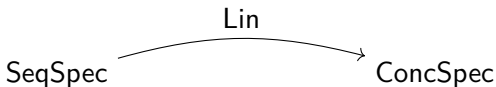


- ▶ **Input:** a sequential specification  $\sigma$  (e.g. list, queue, ...).
- ▶ **Output:** a concurrent specification  $\text{Lin}(\sigma)$ .

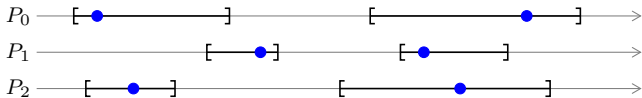


$$\text{Lin}(\sigma) = \{T \text{ concurrent trace} \mid T \text{ is linearizable w.r.t. } \sigma\}$$

# Linearizability (Herlihy & Wing, 1990)



- ▶ **Input:** a sequential specification  $\sigma$  (e.g. list, queue, ...).
- ▶ **Output:** a concurrent specification  $\text{Lin}(\sigma)$ .



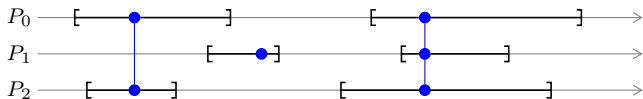
$$\text{Lin}(\sigma) = \{T \text{ concurrent trace} \mid T \text{ is linearizable w.r.t. } \sigma\}$$

Some objects are not linearizable!

Their specification cannot be expressed as  $\text{Lin}(\sigma)$ , for any  $\sigma$ .

## Concurrent variants of linearizability

### Set-linearizability (Neiger, 1994)



- ▶ Can specify: exchanger, immediate snapshot, set agreement.

## Concurrent variants of linearizability

### Set-linearizability (Neiger, 1994)



- ▶ Can specify: exchanger, immediate snapshot, set agreement.
- ▶ Cannot specify: validity, write-snapshot.

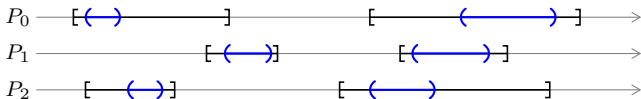
## Concurrent variants of linearizability

### Set-linearizability (Neiger, 1994)



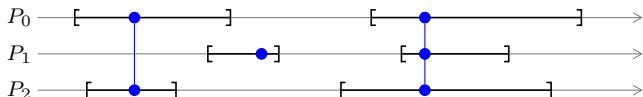
- ▶ Can specify: exchanger, immediate snapshot, set agreement.
- ▶ Cannot specify: validity, write-snapshot.

### Interval-linearizability (Castañeda, Rajsbaum, Raynal, 2015)



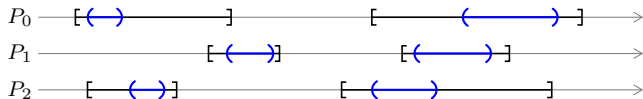
## Concurrent variants of linearizability

### Set-linearizability (Neiger, 1994)



- ▶ Can specify: exchanger, immediate snapshot, set agreement.
- ▶ Cannot specify: validity, write-snapshot.

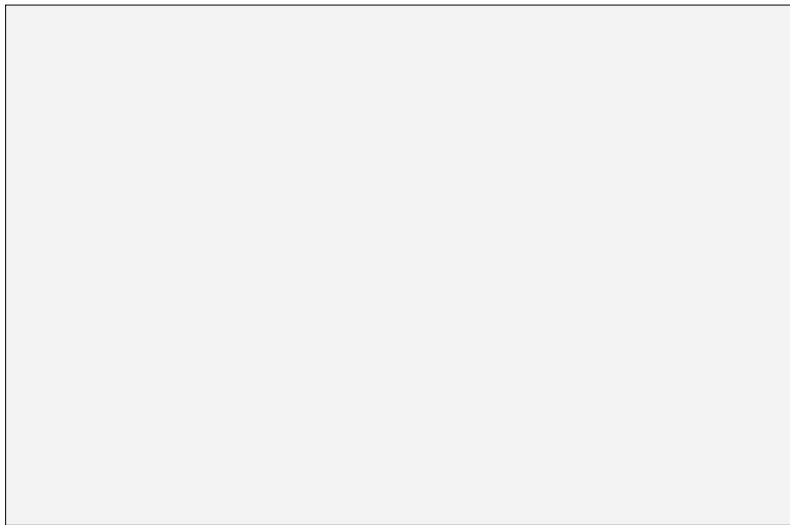
### Interval-linearizability (Castañeda, Rajsbaum, Raynal, 2015)



- ▶ Can specify every task!

# Overview

Concurrent specifications





# Overview

## Concurrent specifications

Linearizability

stack  
queue  
test&set

# Overview

## Concurrent specifications

Linearizability

stack  
queue  
test&set

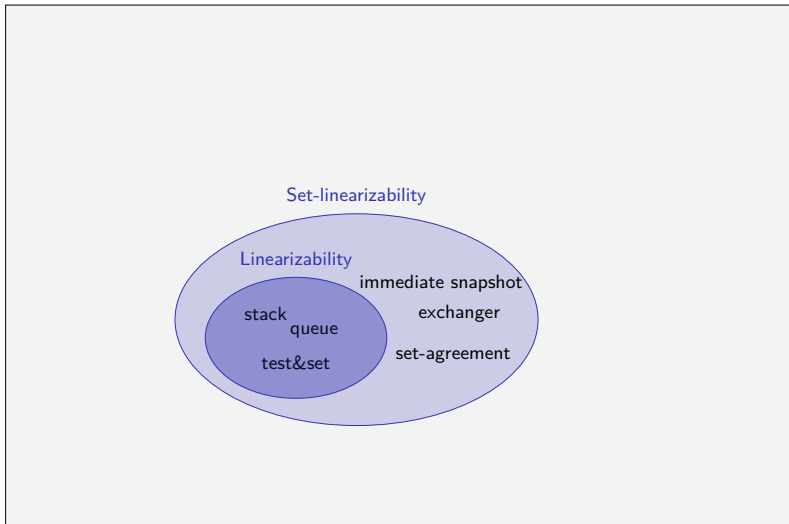
immediate snapshot

exchanger

set-agreement

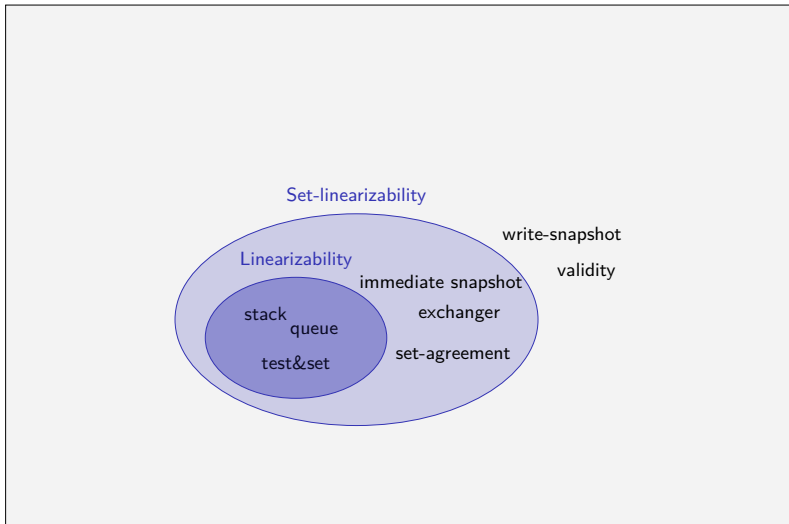
# Overview

## Concurrent specifications



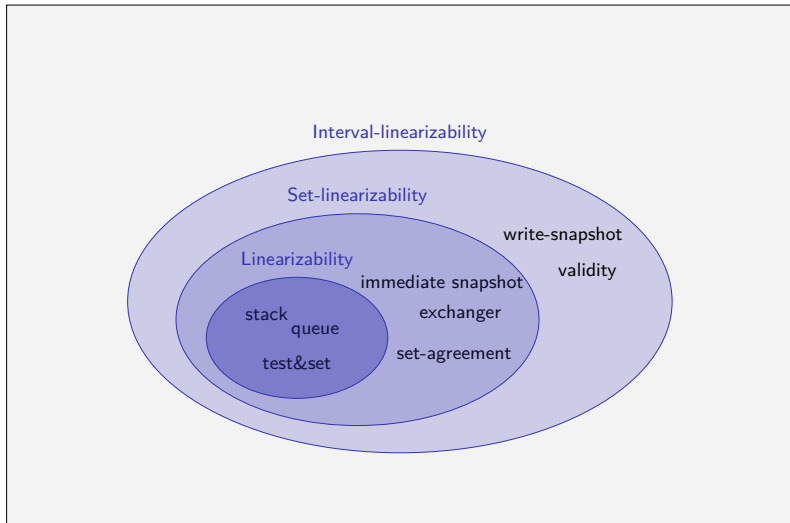
# Overview

## Concurrent specifications



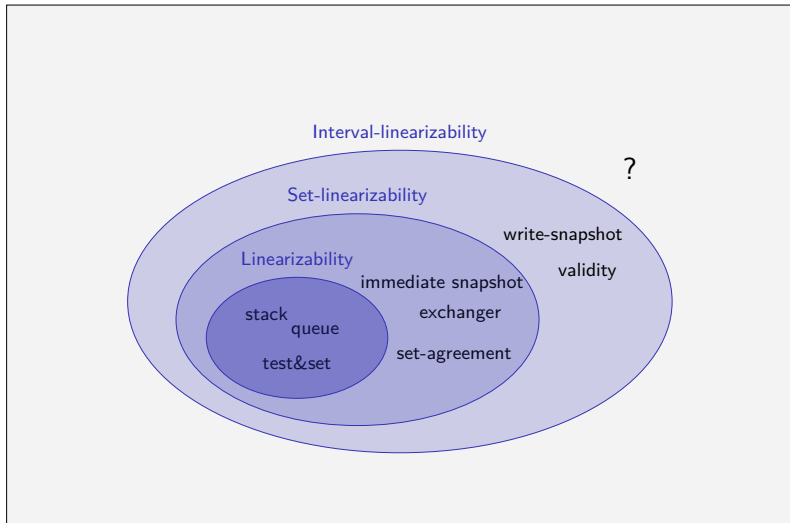
# Overview

## Concurrent specifications



# Overview

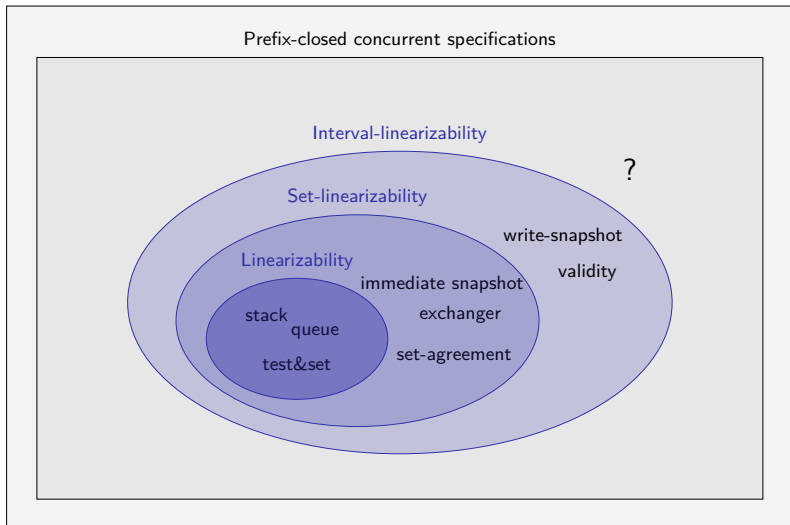
Concurrent specifications



# Overview

Concurrent specifications

Prefix-closed concurrent specifications

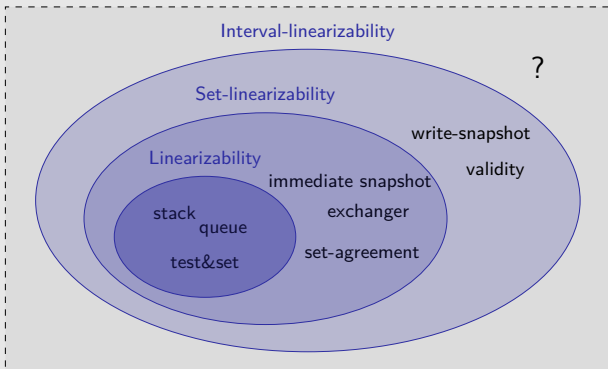


# Overview

Concurrent specifications

Prefix-closed concurrent specifications

**This talk:** add a few more "desirable" properties





# Overview

Concurrent specifications

Prefix-closed concurrent specifications

Interval-linearizability

Set-linearizability

Linearizability

stack  
queue  
test&set

immediate snapshot  
exchanger  
set-agreement

write-snapshot  
validity

## Relevant concurrent specifications

We write **ConcSpec** for the set of concurrent specifications  $\sigma \subseteq \mathcal{T}$  satisfying the following properties.

- (1) *prefix-closure*: if  $t \cdot t' \in \sigma$  then  $t \in \sigma$ ,
- (2) *non-emptiness*:  $\varepsilon \in \sigma$ ,
- (3) *receptivity*: if  $t \in \sigma$  and  $t$  has no pending invocation of process  $i$ , then  $t \cdot i_i^x \in \sigma$  for every input value  $x$ ,

## Relevant concurrent specifications

We write **ConcSpec** for the set of concurrent specifications  $\sigma \subseteq \mathcal{T}$  satisfying the following properties.

- (1) *prefix-closure*: if  $t \cdot t' \in \sigma$  then  $t \in \sigma$ ,
- (2) *non-emptiness*:  $\varepsilon \in \sigma$ ,
- (3) *receptivity*: if  $t \in \sigma$  and  $t$  has no pending invocation of process  $i$ , then  $t \cdot i_i^x \in \sigma$  for every input value  $x$ ,
- (4) *totality*: if  $t \in \sigma$  and  $t$  has a pending invocation of process  $i$ , then there exists an output  $x$  such that  $t \cdot r_i^x \in \sigma$ ,

## Relevant concurrent specifications

We write **ConcSpec** for the set of concurrent specifications  $\sigma \subseteq \mathcal{T}$  satisfying the following properties.

- (1) *prefix-closure*: if  $t \cdot t' \in \sigma$  then  $t \in \sigma$ ,
- (2) *non-emptiness*:  $\varepsilon \in \sigma$ ,
- (3) *receptivity*: if  $t \in \sigma$  and  $t$  has no pending invocation of process  $i$ , then  $t \cdot i_i^x \in \sigma$  for every input value  $x$ ,
- (4) *totality*: if  $t \in \sigma$  and  $t$  has a pending invocation of process  $i$ , then there exists an output  $x$  such that  $t \cdot r_i^x \in \sigma$ ,
- (5)  $\sigma$  has the *expansion* property.

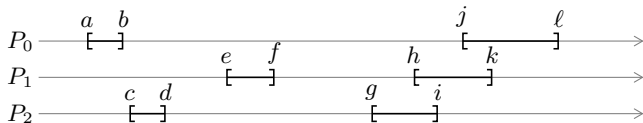
## Expansion of intervals

A concurrent specification satisfies the **expansion** property if:

## Expansion of intervals

A concurrent specification satisfies the **expansion** property if:

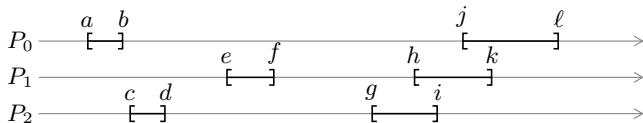
For any correct execution trace,



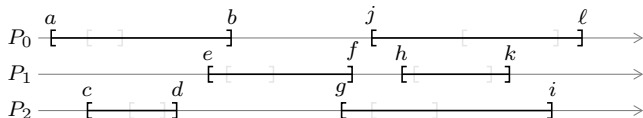
## Expansion of intervals

A concurrent specification satisfies the **expansion** property if:

For any correct execution trace,



if we *expand* the intervals,



then the resulting trace is still correct.

## Example: the Exchanger object

Similar to the one available in Java<sup>1</sup>: *“A synchronization point at which threads can pair and swap elements within pairs”*.

Here, we consider a wait-free variant.

---

<sup>1</sup>`java.util.concurrent.Exchanger<V>`

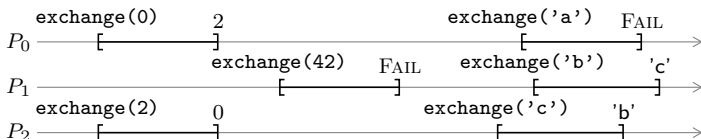


## Example: the Exchanger object

Similar to the one available in Java<sup>1</sup>: “A synchronization point at which threads can pair and swap elements within pairs”.

Here, we consider a wait-free variant.

A typical execution of the exchanger looks like this:

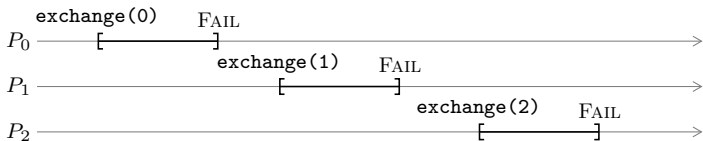


---

<sup>1</sup>`java.util.concurrent.Exchanger<V>`

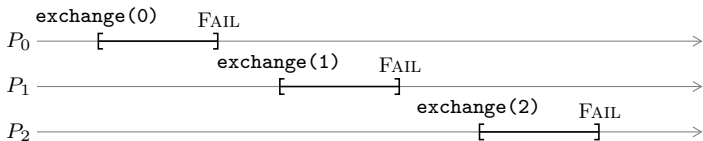
## Example: the Exchanger object (2)

The following execution is correct:

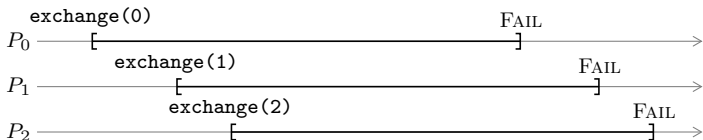


## Example: the Exchanger object (2)

The following execution is correct:



Hence, according to the expansion property,



should be considered correct too!

## Expansion is a desirable property

We fix a set  $\{A_1, \dots, A_k\}$  of shared objects, along with their concurrent specifications.

## Expansion is a desirable property

We fix a set  $\{A_1, \dots, A_k\}$  of shared objects, along with their concurrent specifications.

A **program**  $P$  using these objects can:

- ▶ call the objects,
- ▶ do local computations,
- ▶ use branching, loops.

## Expansion is a desirable property

We fix a set  $\{A_1, \dots, A_k\}$  of shared objects, along with their concurrent specifications.

A **program**  $P$  using these objects can:

- ▶ call the objects,
- ▶ do local computations,
- ▶ use branching, loops.

Given a program  $P$ , we can define its **semantics**  $\llbracket P \rrbracket$ , which is the set of execution traces that  $P$  can produce.

## Expansion is a desirable property

We fix a set  $\{A_1, \dots, A_k\}$  of shared objects, along with their concurrent specifications.

A **program**  $P$  using these objects can:

- ▶ call the objects,
- ▶ do local computations,
- ▶ use branching, loops.

Given a program  $P$ , we can define its **semantics**  $\llbracket P \rrbracket$ , which is the set of execution traces that  $P$  can produce.

### Theorem

*The semantics  $\llbracket P \rrbracket$  of any program  $P$  has the expansion property. Moreover, if  $P$  is wait-free, then  $\llbracket P \rrbracket \in \text{ConcSpec}$ .*

## Linearizability gives expansion for free

Linearizability-based techniques always produce specifications which satisfy the expansion property.

### Theorem

*For every sequential specification  $\sigma$ ,  $\text{Lin}(\sigma) \in \text{ConcSpec}$ .*



## Linearizability gives expansion for free

Linearizability-based techniques always produce specifications which satisfy the expansion property.

### Theorem

*For every sequential specification  $\sigma$ ,  $\text{Lin}(\sigma) \in \text{ConcSpec}$ .*

*Proof.*

If some execution trace is linearizable,



## Linearizability gives expansion for free

Linearizability-based techniques always produce specifications which satisfy the expansion property.

### Theorem

For every sequential specification  $\sigma$ ,  $\text{Lin}(\sigma) \in \text{ConcSpec}$ .

*Proof.*

If some execution trace is linearizable,



## Linearizability gives expansion for free

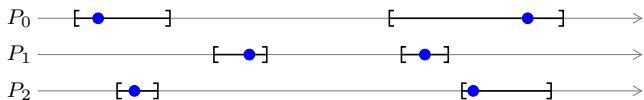
Linearizability-based techniques always produce specifications which satisfy the expansion property.

### Theorem

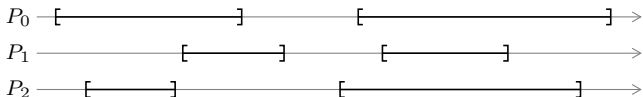
For every sequential specification  $\sigma$ ,  $\text{Lin}(\sigma) \in \text{ConcSpec}$ .

*Proof.*

If some execution trace is linearizable,



Then any trace obtained by expanding it is still linearizable.



## Linearizability gives expansion for free

Linearizability-based techniques always produce specifications which satisfy the expansion property.

### Theorem

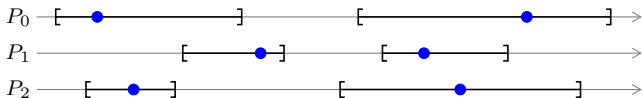
For every sequential specification  $\sigma$ ,  $\text{Lin}(\sigma) \in \text{ConcSpec}$ .

*Proof.*

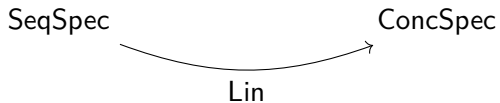
If some execution trace is linearizable,



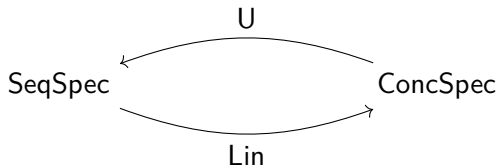
Then any trace obtained by expanding it is still linearizable.



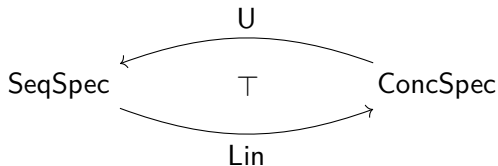
## A Galois connection



## A Galois connection



## A Galois connection



### Theorem

The maps  $\text{Lin}$  and  $U$  form a Galois connection: for every  $\sigma \in \text{SeqSpec}$  and  $\tau \in \text{ConcSpec}$ ,

$$\text{Lin}(\sigma) \subseteq \tau \quad \iff \quad \sigma \subseteq U(\tau)$$

# Applications

- ▶ By the properties of Galois connections,

$$\text{Lin}(\text{U}(\text{Lin}(\sigma))) = \text{Lin}(\sigma)$$

This yields a simple criterion to check whether a given specification  $\tau$  is linearizable: check whether  $\text{Lin}(\text{U}(\tau)) = \tau$ .



## Applications

- ▶ By the properties of Galois connections,

$$\text{Lin}(\text{U}(\text{Lin}(\sigma))) = \text{Lin}(\sigma)$$

This yields a simple criterion to check whether a given specification  $\tau$  is linearizable: check whether  $\text{Lin}(\text{U}(\tau)) = \tau$ .

- ▶ The Galois connection for interval linearizability has the following corollary:

### Theorem

*ConcSpec is the set of interval-linearizable specifications.*

Thanks!