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 $A_1, \ldots, A_k$?

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 $A_1, \ldots, A_k$?

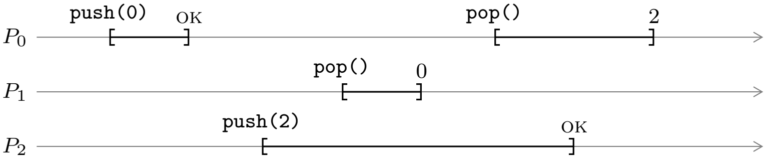
$\rightarrow$ We need to specify the behavior of the objects.

Concurrent specifications

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

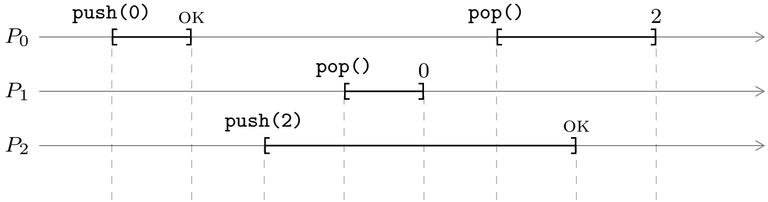
Concurrent specifications

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).



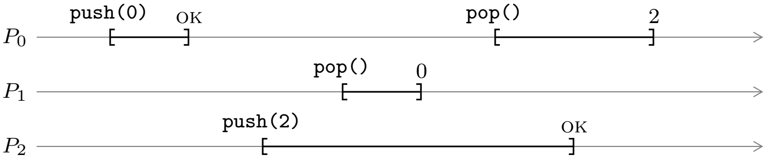
$$T = \mathsf{i}_0^{\mathtt{push},0} \cdot \mathsf{r}_0^{\mathrm{OK}} \cdot \mathsf{i}_2^{\mathtt{push},2} \cdot \mathsf{i}_1^{\mathtt{pop}} \cdot \mathsf{r}_1^{2} \cdot \mathsf{i}_0^{\mathtt{pop}} \cdot \mathsf{r}_2^{\mathrm{OK}} \cdot \mathsf{r}_0^{0}$$

Trace formalism:

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

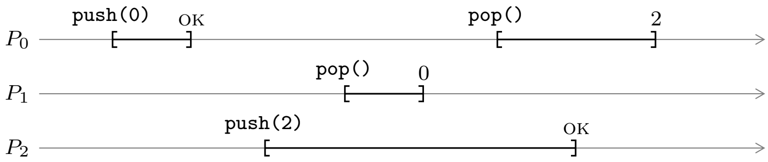
Concurrent specifications

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).

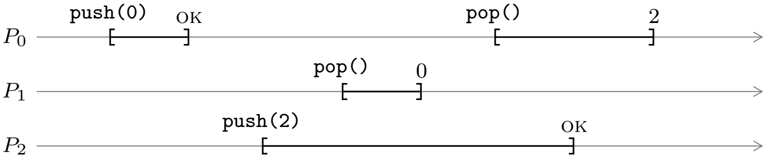


Write $\mathcal{T}$ for the set of all execution traces.

- 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 σ if all the traces that it can produce belong to σ.

Linearizability (Herlihy & Wing, 1990)

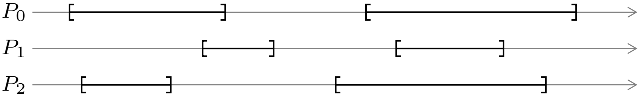
$$\text{SeqSpec} \xrightarrow{\text{Lin}} \text{ConcSpec}$$

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

Linearizability (Herlihy & Wing, 1990)

$$\text{SeqSpec} \xrightarrow{\text{Lin}} \text{ConcSpec}$$

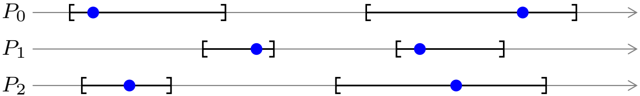
- **Input:** a sequential specification σ (e.g. list, queue, ...).
- **Output:** a concurrent specification Lin(σ).



Linearizability (Herlihy & Wing, 1990)

$$\text{SeqSpec} \xrightarrow{\text{Lin}} \text{ConcSpec}$$

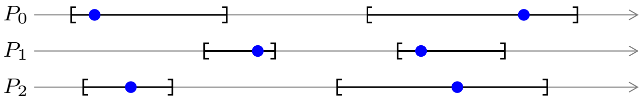
- **Input:** a sequential specification σ (e.g. list, queue, ...).
- **Output:** a concurrent specification $\text{Lin}(\sigma)$.



Linearizability (Herlihy & Wing, 1990)

$$\text{SeqSpec} \xrightarrow{\text{Lin}} \text{ConcSpec}$$

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

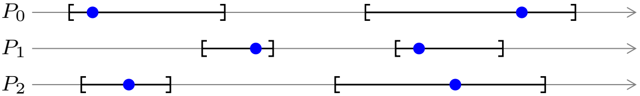


$$\text{Lin}(\sigma) = \{T \text{ concurrent trace} \mid T \text{ is linearizable w.r.t. } \sigma\}$$

Linearizability (Herlihy & Wing, 1990)

$$\text{SeqSpec} \xrightarrow{\text{Lin}} \text{ConcSpec}$$

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



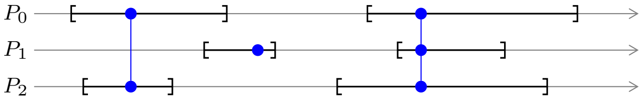
$$\mathsf{Lin}(\sigma) = \{T \text{ concurrent trace} \mid T \text{ is linearizable w.r.t. } \sigma\}$$

Some objects are not linearizable!

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

Concurrent variants of linearizability

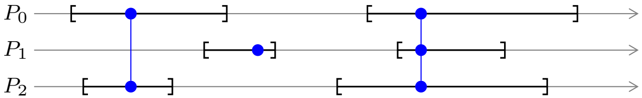
Set-linearizability (Neiger, 1994)



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

Concurrent variants of linearizability

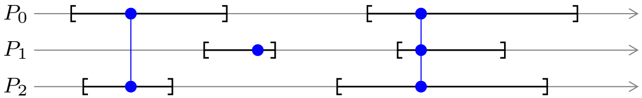
Set-linearizability (Neiger, 1994)



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

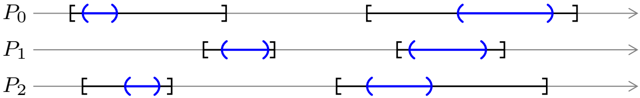
Concurrent variants of linearizability

Set-linearizability (Neiger, 1994)



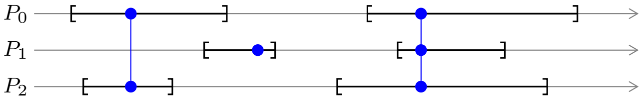
- Can specify: exchanger, immediate snapshot, set agreement.
- Cannot specify: validity, write-snapshot.

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



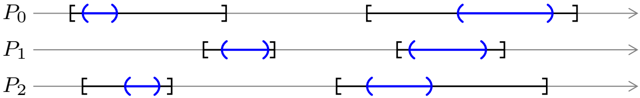
Concurrent variants of linearizability

Set-linearizability (Neiger, 1994)



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

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



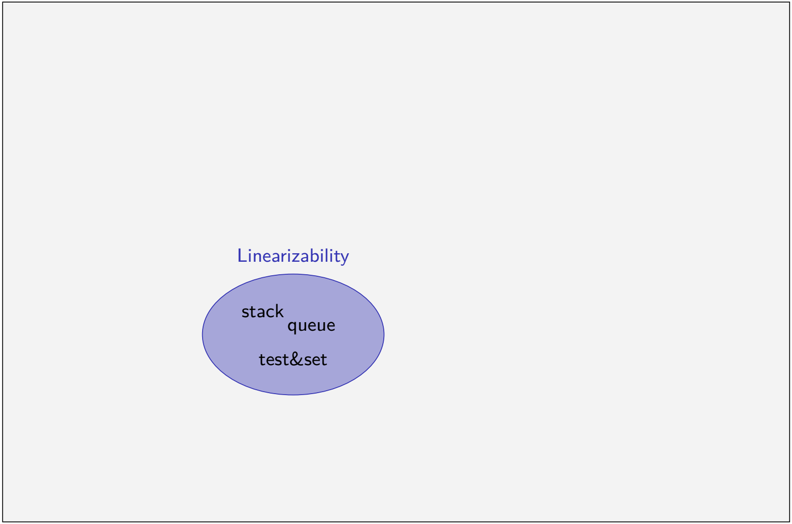
- Can specify every task!

Overview

Concurrent specifications

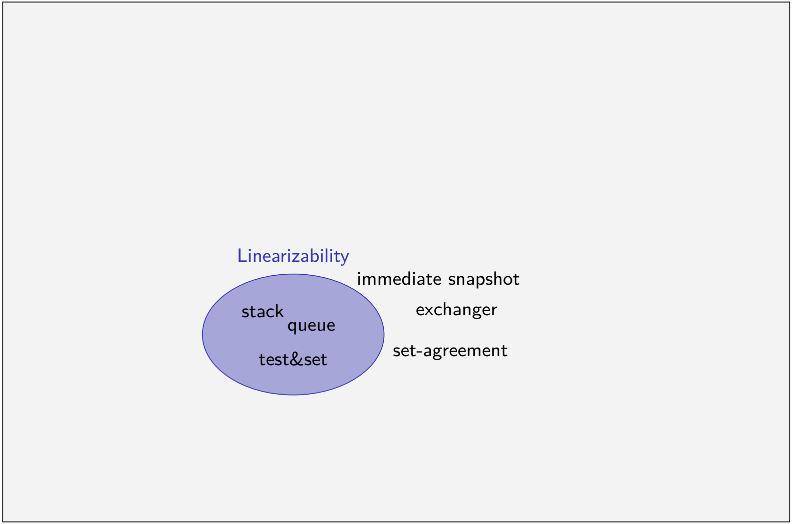
Overview

Concurrent specifications



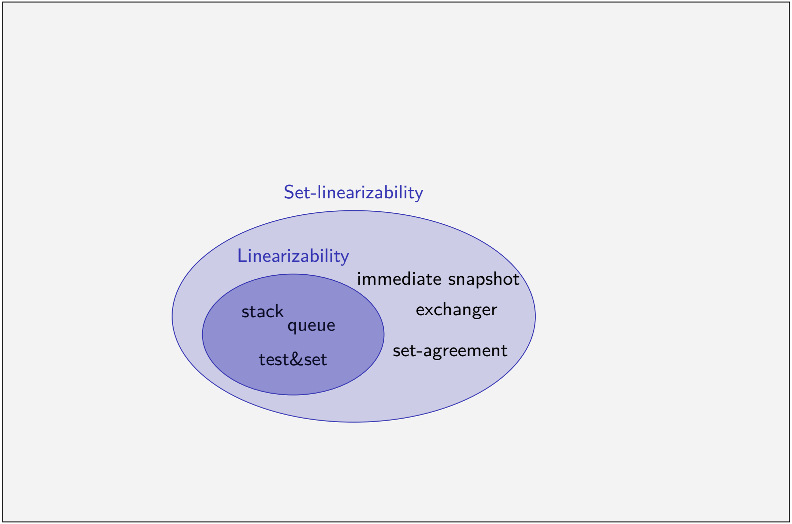
Overview

Concurrent specifications



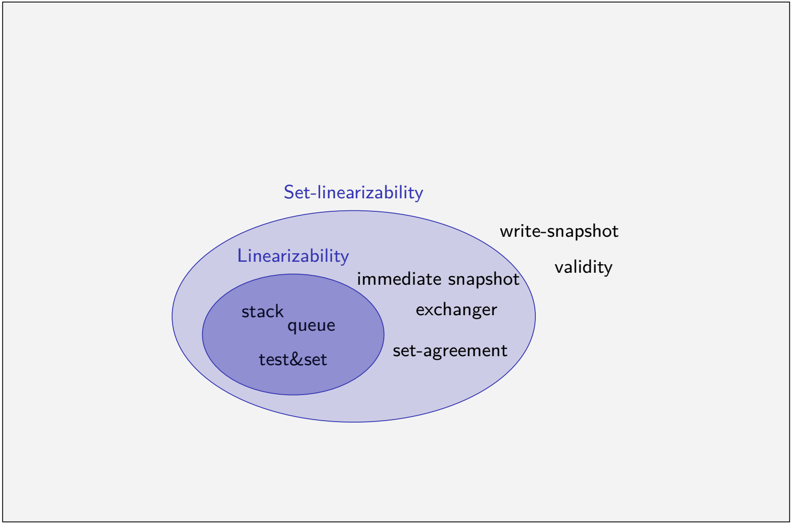
Overview

Concurrent specifications



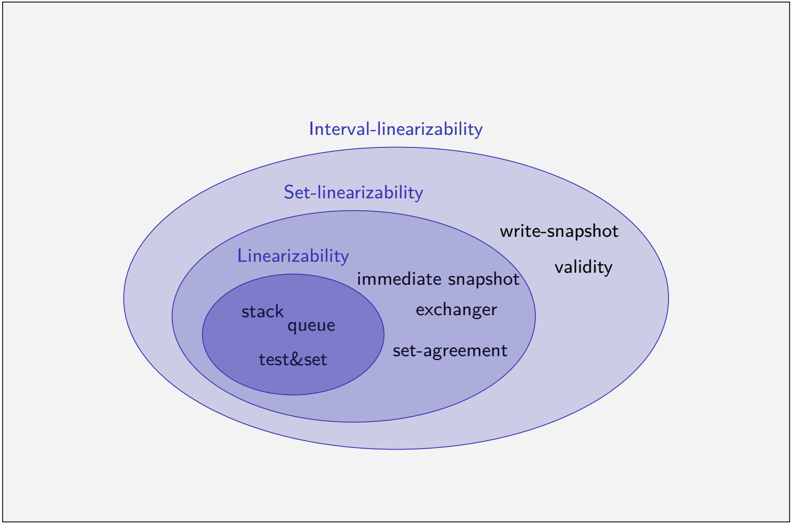
Overview

Concurrent specifications



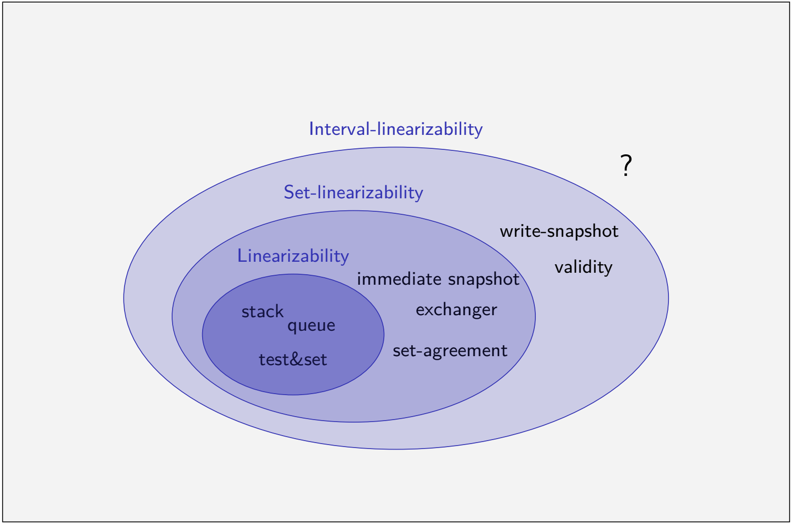
Overview

Concurrent specifications

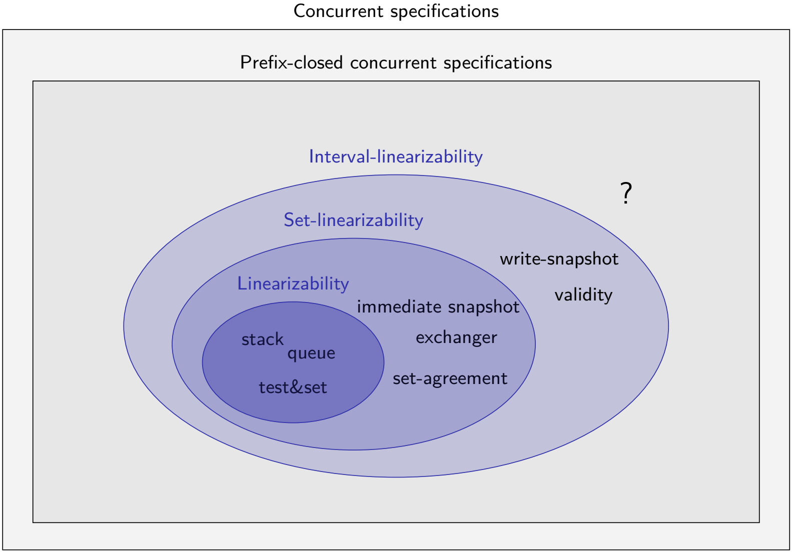


Overview

Concurrent specifications

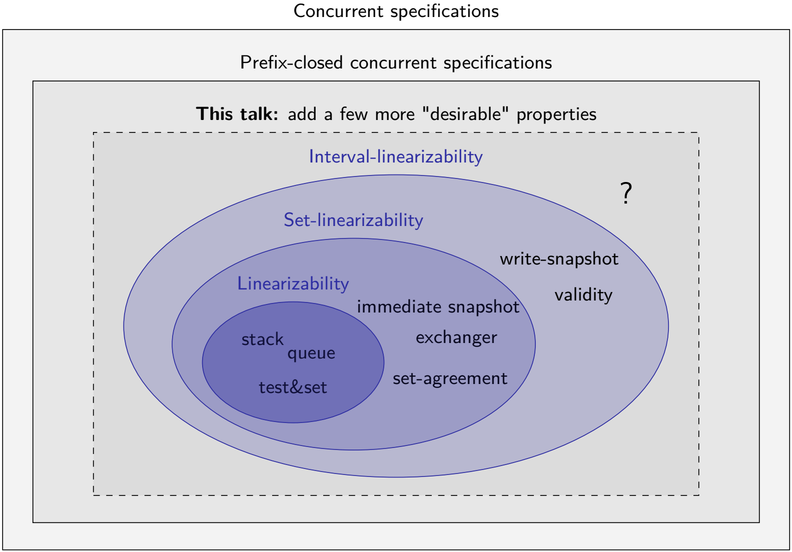


Overview

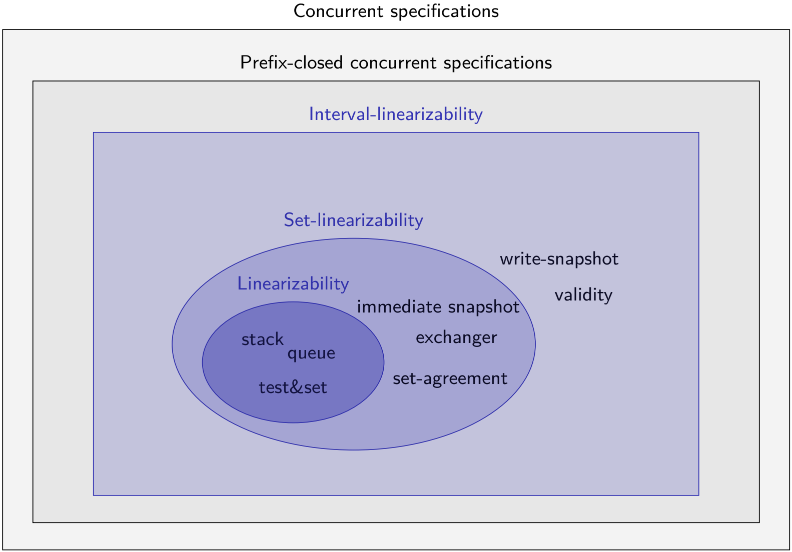


Concurrent specifications
Prefix-closed concurrent specifications
Interval-linearizability
?
Set-linearizability
write-snapshot
validity
Linearizability
immediate snapshot
exchanger
set-agreement
stack
queue
test&set

Overview


Concurrent specifications
Prefix-closed concurrent specifications
This talk: add a few more "desirable" properties
Interval-linearizability
?
Set-linearizability
write-snapshot
validity
Linearizability
immediate snapshot
exchanger
set-agreement
stack
queue
test&set

Overview


Concurrent specifications
Prefix-closed concurrent specifications
Interval-linearizability
Set-linearizability
write-snapshot
validity
Linearizability
immediate snapshot
exchanger
set-agreement
stack
queue
test&set

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 $t \cdot \mathsf{i}_i^x \in \sigma$ 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 $t \cdot \mathsf{i}_i^x \in \sigma$ 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 $t \cdot \mathsf{r}_i^x \in \sigma$,

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 $t \cdot \mathsf{i}_i^x \in \sigma$ 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 $t \cdot \mathsf{r}_i^x \in \sigma$,

(5) σ has the *expansion* property.

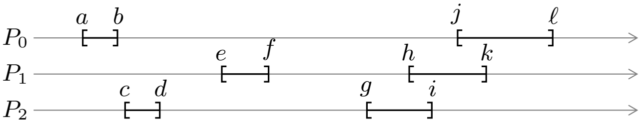
Expansion of intervals

A concurrent specification satisfies the expansion property if:

Expansion of intervals

A concurrent specification satisfies the expansion property if:

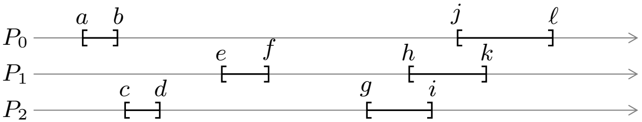
For any correct execution trace,



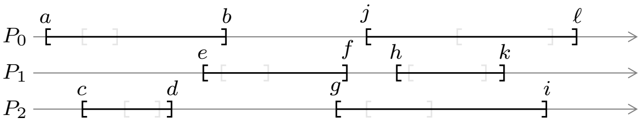
Expansion of intervals

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

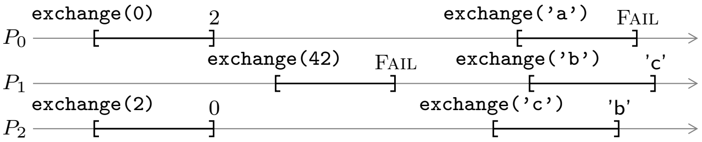
Similar to the one available in Java[1]: *"A synchronization point at which threads can pair and swap elements within pairs".*
Here, we consider a wait-free variant.

[1] `java.util.concurrent.Exchanger<V>`

Example: the Exchanger object

Similar to the one available in Java[1]: *"A synchronization point at which threads can pair and swap elements within pairs".*
Here, we consider a wait-free variant.

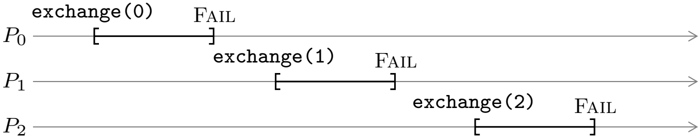
A typical execution of the exchanger looks like this:



[1] java.util.concurrent.Exchanger<V>

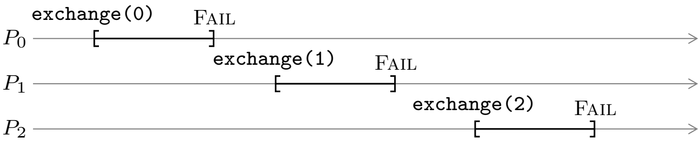
Example: the Exchanger object (2)

The following execution is correct:

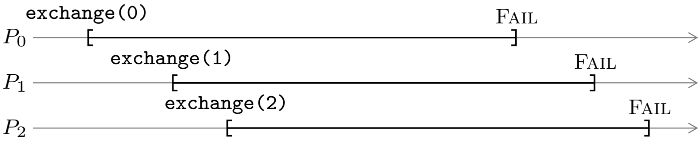


Example: the Exchanger object (2)

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 $\{A_1, \ldots, A_k\}$ of shared objects, along with their concurrent specifications.

Expansion is a desirable property

We fix a set $\{A_1, \ldots, A_k\}$ 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 $\{A_1, \ldots, A_k\}$ 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 $[\![P]\!]$, which is the set of execution traces that P can produce.

Expansion is a desirable property

We fix a set $\{A_1, \ldots, A_k\}$ 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 $[\![P]\!]$, which is the set of execution traces that P can produce.

Theorem

The semantics $[\![P]\!]$ of any program P has the expansion property. Moreover, if P is wait-free, then $[\![P]\!] \in \mathrm{ConcSpec}$.

Linearizability gives expansion for free

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

Theorem

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

Linearizability gives expansion for free

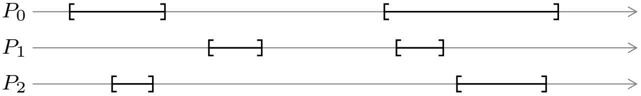
Linearizability-based techniques always produce specifications which satisfy the expansion property.

Theorem

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

Proof.

If some execution trace is linearizable,



Linearizability gives expansion for free

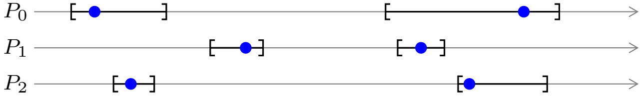
Linearizability-based techniques always produce specifications which satisfy the expansion property.

Theorem

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

Proof.

If some execution trace is linearizable,



Linearizability gives expansion for free

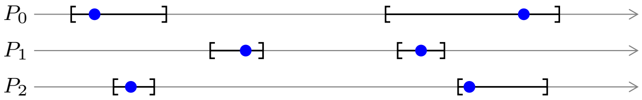
Linearizability-based techniques always produce specifications which satisfy the expansion property.

Theorem

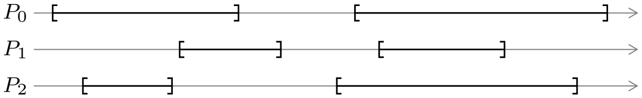
For every sequential specification σ, $\mathsf{Lin}(\sigma) \in \mathsf{ConcSpec}$.

Proof.

If some execution trace is linearizable,



Then any trace obtained by expanding it is still linearizable.



□

Linearizability gives expansion for free

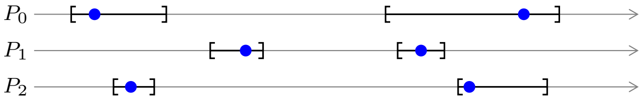
Linearizability-based techniques always produce specifications which satisfy the expansion property.

Theorem

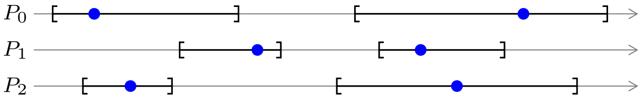
For every sequential specification σ, $\mathrm{Lin}(\sigma) \in \mathrm{ConcSpec}$.

Proof.

If some execution trace is linearizable,



Then any trace obtained by expanding it is still linearizable.



□

A Galois connection

$$\begin{tikzcd} \mathsf{SeqSpec} \arrow[rr, bend right, "\mathsf{Lin}"'] & & \mathsf{ConcSpec} \end{tikzcd}$$

A Galois connection

$$\begin{tikzcd}
\mathsf{SeqSpec} \arrow[rr, bend right, "\mathsf{Lin}"'] & & \mathsf{ConcSpec} \arrow[ll, bend right, "\mathsf{U}"']
\end{tikzcd}$$

A Galois connection

$$\begin{tikzcd} \mathsf{SeqSpec} \arrow[rr, bend right, "\mathsf{Lin}"'] \arrow[rr, phantom, "\top"] & & \mathsf{ConcSpec} \arrow[ll, bend right, "\mathsf{U}"'] \end{tikzcd}$$

Theorem

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

$$\mathsf{Lin}(\sigma) \subseteq \tau \qquad \iff \qquad \sigma \subseteq \mathsf{U}(\tau)$$

Applications

- By the properties of Galois connections,

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

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

Applications

- By the properties of Galois connections,

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

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

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

Theorem

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

Thanks!