Concurrent specifications beyond linearizability

Éric Goubault  Jérémy Ledent  Samuel Mimram

École Polytechnique, France

OPODIS 2018, Hong Kong
December 19, 2018
Processes communicate through shared objects. For example:
Processes communicate through shared objects. For example:

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

Goal: can we implement object $B$ using objects $A_1, ..., A_k$?

$\Rightarrow$ We need to specify the behavior of the objects.
Objects

Processes communicate through shared objects. For example:

- Hardware: Read/Write registers, test&set, CAS, 
- Data structures: lists, queues, hashmaps,
Processes communicate through shared objects. For example:

- Hardware: Read/Write registers, test&set, CAS, ...
- Data structures: lists, queues, hashmaps, ...
- Message passing interfaces
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, ...
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 $A_1, \ldots, A_k$?
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 $A_1, \ldots, A_k$?

→ 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).

\[ P_0 \begin{array}{c} \text{push}(0) \quad \text{OK} \\ \text{pop}() \quad 2 \end{array} \]

\[ P_1 \begin{array}{c} \text{pop}() \quad 0 \end{array} \]

\[ P_2 \begin{array}{c} \text{push}(2) \quad \text{OK} \end{array} \]
Concurrent specifications

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

\[
T = i_0^{\text{push},0} \cdot r_0^{\text{OK}} \cdot i_2^{\text{push},2} \cdot i_1^{\text{pop}} \cdot r_1^2 \cdot i_0^{\text{pop}} \cdot r_2^{\text{OK}} \cdot r_0^0
\]

**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).

\[ P_0 \quad \text{push}(0) \quad \text{OK} \quad \text{pop()} \quad 2 \]

\[ P_1 \quad \text{pop()} \quad 0 \]

\[ P_2 \quad \text{push}(2) \quad \text{OK} \]
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).

![Diagram]

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 $\text{Lin}(\sigma)$.

Some objects are not linearizable! Their specification cannot be expressed as $\text{Lin}(\sigma)$, for any $\sigma$. 
Linearizability (Herlihy & Wing, 1990)

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

Some objects are not linearizable! Their specification cannot be expressed as $\text{Lin}(\sigma)$, for any $\sigma$. 

\begin{align*}
P_0 & \quad [\quad] \quad [\quad] \quad \rightarrow \\
P_1 & \quad [\quad] \quad [\quad] \quad \rightarrow \\
P_2 & \quad [\quad] \quad [\quad] \quad \rightarrow \\
\end{align*}
Linearizability (Herlihy & Wing, 1990)

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

Some objects are not linearizable! Their specification cannot be expressed as \( \text{Lin}(\sigma) \), for any \( \sigma \).
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.
- 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.
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
Overview

Concurrent specifications

Set-linearizability

Linearizability

- immediate snapshot
- exchanger
- set-agreement

- stack
- queue
- test&set
Overview

Concurrent specifications

Set-linearizability

Linearizability

stack
queue
test&set

write-snapshot
validity
immediate snapshot
exchanger
set-agreement
Overview

Concurrent specifications

Interval-linearizability

Set-linearizability

Linearizability

Immediate snapshot

Write-snapshot validity

Set-agreement

Exchanger

Stack

Queue

Test&set
Overview

Concurrent specifications

Interval-linearizability

Set-linearizability

Linearizability

stack
queue
test&set

immediate snapshot
exchanger
set-agreement

write-snapshot
validity
Overview

Concurrent specifications

Prefix-closed concurrent specifications

Interval-linearizability

Set-linearizability

Linearizability

write-snapshot
validity

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
- Linearizability

Stack, queue, test&set

Write-snapshot, validity, immediate snapshot, exchanger, set-agreement
Relevant concurrent specifications

We write $\text{ConcSpec}$ 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 $t \cdot i^x_i \in \sigma$ 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) \textit{prefix-closure}: if $t \cdot t' \in \sigma$ then $t \in \sigma$, 

(2) \textit{non-emptiness}: $\varepsilon \in \sigma$, 

(3) \textit{receptivity}: if $t \in \sigma$ and $t$ has no pending invocation of process $i$, then $t \cdot i^x_i \in \sigma$ for every input value $x$, 

(4) \textit{totality}: if $t \in \sigma$ and $t$ has a pending invocation of process $i$, then there exists an output $x$ such that $t \cdot r^x_i \in \sigma$,
Relevant concurrent specifications

We write ConcSpec for the set of concurrent specifications $\sigma \subseteq T$ 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 $t \cdot i^x_i \in \sigma$ 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 $t \cdot r^x_i \in \sigma$,

(5) $\sigma$ has the *expansion* property.
Expansion of intervals

A concurrent specification satisfies the expansion property if:

For any correct execution trace,

then the resulting trace is still correct.
Expansion of intervals

A concurrent specification satisfies the expansion property if:

For any correct execution trace,

\[
\begin{array}{c}
P_0 \quad \boxed{a \ b} \quad e \ f \quad \boxed{j \ l} \\
P_1 \quad \boxed{c \ d} \quad g \ i \\
P_2 \quad \boxed{\ } \quad \boxed{\ }
\end{array}
\]

if we expand the intervals, then the resulting trace is still correct.
Expansion of intervals

A concurrent specification satisfies the expansion property if:

For any correct execution trace,

\[ P_0 \quad a \quad b \quad j \quad \ell \]
\[ P_1 \quad e \quad f \quad h \quad k \]
\[ P_2 \quad c \quad d \quad g \quad i \]

if we expand the intervals,

\[ P_0 \quad a \quad b \quad j \quad \ell \]
\[ P_1 \quad e \quad f \quad h \quad k \]
\[ P_2 \quad c \quad d \quad g \quad i \]

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\textsuperscript{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:

\begin{center}
\begin{tikzpicture}
  \node at (0,0) (P0) {$P_0$};
  \node at (5,0) (P1) {$P_1$};
  \node at (10,0) (P2) {$P_2$};
  \node at (15,0) (P3) {};\node at (15,-0.5) {$\rightarrow$};

  \node at (0,-1) {$\text{exchange}(0)$};
  \node at (5,-1) {$2$};
  \node at (10,-1) {$\text{exchange}(\text{\textquoteleft}a\text{\textquoteleft})$};
  \node at (15,-1) {$\text{FAIL}$};

  \node at (0,-2) {$\text{exchange}(42)$};
  \node at (5,-2) {$\text{FAIL}$};
  \node at (10,-2) {$\text{exchange}(\text{\textquoteleft}b\text{\textquoteleft})$};
  \node at (15,-2) {'c'};

  \node at (0,-3) {$\text{exchange}(2)$};
  \node at (5,-3) {$0$};
  \node at (10,-3) {$\text{exchange}(\text{\textquoteleft}c\text{\textquoteleft})$};
  \node at (15,-3) {'b'};

  \node at (0,-1.5) {$\vdash$};
\end{tikzpicture}
\end{center}

\textsuperscript{1}java.util.concurrent.Exchanger\textless{}V\textgreater{}
Example: the Exchanger object (2)

The following execution is correct:

\[ P_0 \xrightarrow{\text{exchange}(0)} \text{FAIL} \quad P_1 \xrightarrow{\text{exchange}(1)} \text{FAIL} \quad P_2 \xrightarrow{\text{exchange}(2)} \text{FAIL} \]
Example: the Exchanger object (2)

The following execution is correct:

\[ P_0 \xrightarrow{\text{exchange(0)}} \text{FAIL} \]
\[ P_1 \xrightarrow{\text{exchange(1)}} \text{FAIL} \]
\[ P_2 \xrightarrow{\text{exchange(2)}} \text{FAIL} \]

Hence, according to the expansion property,

\[ P_0 \xrightarrow{\text{exchange(0)}} \text{FAIL} \]
\[ P_1 \xrightarrow{\text{exchange(1)}} \text{FAIL} \]
\[ P_2 \xrightarrow{\text{exchange(2)}} \text{FAIL} \]

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.

Theorem

The semantics \( J_P \) of any program \( P \) has the expansion property. Moreover, if \( P \) is wait-free, then \( J_P \in \text{ConcSpec} \).
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 \( \lbrack P \rbrack \), 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 \([P]\) \( \in \) 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,

\[
\begin{align*}
P_0 \quad [ & ] \quad [ & ] \quad [ & ] \quad \rightarrow \\
P_1 \quad [ & ] \quad [ & ] \quad \rightarrow \\
P_2 \quad [ & ] \quad [ & ] \quad \rightarrow
\end{align*}
\]
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,

\[
\begin{align*}
P_0 & \quad [\bullet] \quad [\bullet] \\
P_1 & \quad [\bullet] \quad [\bullet] \\
P_2 & \quad [\bullet] \quad [\bullet]
\end{align*}
\]
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,

\[ P_0 \rightarrow [\bullet] \rightarrow [\bullet] \rightarrow \]
\[ P_1 \rightarrow [\bullet] \rightarrow [\bullet] \rightarrow \]
\[ P_2 \rightarrow [\bullet] \rightarrow [\bullet] \rightarrow \]

Then any trace obtained by expanding it is still linearizable.

\[ P_0 \rightarrow \rightarrow \rightarrow \]
\[ P_1 \rightarrow \rightarrow \rightarrow \]
\[ P_2 \rightarrow \rightarrow \rightarrow \]
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,

\[
P_0 \quad [ \bullet \quad ] \quad [ \bullet \bullet \ ] \quad [ \bullet \bullet \ ] \quad \rightarrow
\]

\[
P_1 \quad [ \bullet \bullet \ ] \quad [ \bullet \bullet \ ] \quad \rightarrow
\]

\[
P_2 \quad [ \bullet \bullet \ ] \quad [ \bullet \bullet \ ] \quad \rightarrow
\]

Then any trace obtained by expanding it is still linearizable.

\[
P_0 \quad [ \bullet \bullet \bullet \bullet ] \quad [ \bullet \bullet \bullet \bullet ] \quad \rightarrow
\]

\[
P_1 \quad [ \bullet \bullet \bullet \bullet ] \quad [ \bullet \bullet \bullet \bullet ] \quad \rightarrow
\]

\[
P_2 \quad [ \bullet \bullet \bullet \bullet ] \quad [ \bullet \bullet \bullet \bullet ] \quad \rightarrow
\]
A Galois connection

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

\[
\text{Lin}(\sigma) \subseteq \tau \iff \sigma \subseteq U(\tau)
\]
The maps $\text{Lin}$ and $\text{U}$ form a Galois connection: for every $\sigma \in \text{SeqSpec}$ and $\tau \in \text{ConcSpec}$,

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

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

- By the properties of Galois connections,

\[ \text{Lin}(U(\text{Lin}(\sigma))) = \text{Lin}(\sigma) \]

This yields a simple criterion to check whether a given specification \( \tau \) is linearizable: check whether \( \text{Lin}(U(\tau)) = \tau \).
By the properties of Galois connections,

\[ \text{Lin}(U(\text{Lin}(\sigma))) = \text{Lin}(\sigma) \]

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

The Galois connection for interval linearizability has the following corollary:

**Theorem**

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