Brief announcement: On the impossibility of detecting concurrency

Éric Goubault  Jérémie Ledent  Samuel Mimram

École Polytechnique, Paris

DISC 2018, New Orleans
October 16, 2018
Concurrent specifications

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

\[ \text{push}(0) \]
\[ \text{ok} \]
\[ \text{push}(2) \]
\[ \text{ok} \]
\[ \text{pop()} \]
\[ 0 \]
\[ \text{pop()} \]
\[ 2 \]

\[ P_0 \]
\[ P_1 \]
\[ P_2 \]

\[ \text{Write} \]
\[ T \]

▶ A concurrent specification is a subset \( \sigma \subseteq T \).

▶ A program implements a specification \( \sigma \) if all the traces that it can produce belong to \( \sigma \).
Concurrent specifications

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

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

Write \( T \) for the set of all execution traces. ▶ A concurrent specification is a subset \( \sigma \subseteq T \). ▶ A program implements a specification \( \sigma \) if all the traces that it can produce belong to \( \sigma \).
**Concurrent specifications**

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

- $P_0$: `push(0)` OK $\rightarrow$ pop() $\rightarrow$ 2
- $P_1$: $\rightarrow$ pop() 0 $\rightarrow$ push(2) $\rightarrow$ OK
- $P_2$: $\rightarrow$
**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 $\sigma \subseteq \mathcal{T}$. 

[Diagram showing execution traces with push(0), pop(), push(2), and pop() operations.]
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 $\text{Lin}(\sigma)$.

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

<table>
<thead>
<tr>
<th>$P_0$</th>
<th>[ ]</th>
<th>[ ]</th>
<th>$\rightarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>[ ]</td>
<td>[ ]</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>[ ]</td>
<td>[ ]</td>
<td>$\rightarrow$</td>
</tr>
</tbody>
</table>
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 $\sigma$. 

![Diagram showing the transformation from SeqSpec to ConcSpec](image-url)
Input: a sequential specification $\sigma$ (e.g. list, queue, ...).

Output: a concurrent specification $\text{Lin}(\sigma)$.

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

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

Interval-linearizability (Rajsbaum, Castañeda, Raynal)

- Can specify every task!
Concurrent variants of linearizability

Set-linearizability (Neiger)

- Can specify: exchanger, immediate snapshot, set agreement.
Concurrent variants of linearizability

**Set-linearizability** (Neiger)

- Can specify: exchanger, immediate snapshot, set agreement.
- Cannot specify: validity, write-snapshot.
Concurrent variants of linearizability

**Set-linearizability** (Neiger)

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

**Interval-linearizability** (Rajsbaum, Castañeda, Raynal)
Concurrent variants of linearizability

**Set-linearizability** (Neiger)

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

**Interval-linearizability** (Rajsbaum, Castañoeda, Raynal)

- 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

Set-linearizability

Linearizability

- immediate snapshot
- exchanger
- set-agreement

- stack
- queue
- test&set
Overview

Concurrent specifications

- Linearizability
  - stack
  - queue
  - test&set
- Set-linearizability
  - immediate snapshot
  - exchanger
  - set-agreement
  - write-snapshot
  - validity
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

stack
queue
test&set

immediate snapshot
exchanger
set-agreement

write-snapshot
validity
Overview

Concurrent specifications

Prefix-closed concurrent specifications

Some more “obvious” properties + expansion property

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

- stack
- queue
- test&set

Immediate snapshot
- exchanger
- set-agreement

Write-snapshot
- validity
Expansion of intervals

A concurrent specification satisfies the expansion property if:

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,

\[
\begin{align*}
    P_0 & \rightarrow [a, b] \rightarrow [e, f] \rightarrow [j, \ell] \\
    P_1 & \rightarrow [c, d] \rightarrow [h, k] \\
    P_2 & \rightarrow [g, i] \\
\end{align*}
\]
Expansion of intervals

A concurrent specification satisfies the expansion property if:

For any correct execution trace,

if we \textit{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 
\textit{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{equation}
\begin{aligned}
P_0 & \quad \text{exchange}(0) \quad 2 \quad \text{exchange(‘a’) \quad FAIL} \\
\quad & \quad \text{exchange(42) \quad FAIL} \quad \text{exchange(‘b’) \quad ‘c’} \\
P_1 & \quad \text{exchange(2) \quad 0} \quad \text{exchange(‘c’) \quad ‘b’} \\
P_2 & \quad \end{aligned}
\end{equation}

\(^1\)\texttt{java.util.concurrent.Exchanger\langle V\rangle}
Example: the Exchanger object

The following execution is correct:

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

Hence, according to the expansion property, the execution should be considered correct too!
Example: the Exchanger object

The following execution is correct:

\[\text{exchange}(0) \quad \text{FAIL} \]
\[\begin{array}{c}
P_0 \\
\hline
\end{array} \quad \begin{array}{c}
\text{exchange}(1) \quad \text{FAIL} \\
P_1 \\
\hline
\end{array} \quad \begin{array}{c}
\text{exchange}(2) \quad \text{FAIL} \\
P_2 \\
\hline
\end{array}\]

Hence, according to the expansion property,

\[\text{exchange}(0) \quad \text{FAIL} \]
\[\begin{array}{c}
P_0 \\
\hline
\end{array} \quad \begin{array}{c}
\text{exchange}(1) \quad \text{FAIL} \\
P_1 \\
\hline
\end{array} \quad \begin{array}{c}
\text{exchange}(2) \quad \text{FAIL} \\
P_2 \\
\hline
\end{array}\]

should be considered correct too!
Results

- In a reasonable computational model:

**Theorem**

The semantics $[P]$ of any program $P$ has the expansion property.
Results

- In a reasonable computational model:

**Theorem**

The semantics $[P]$ of any program $P$ has the expansion property.

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

**Theorem**

For every sequential specification $\sigma$, $\text{Lin}(\sigma)$ has the expansion property.
Results

▶ In a reasonable computational model:

**Theorem**

The semantics $[P]$ of any program $P$ has the expansion property.

▶ Linearizability-based techniques can only produce specifications which satisfy the expansion property.

**Theorem**

For every sequential specification $\sigma$, Lin$(\sigma)$ has the expansion property.

We write ConcSpec for the set of concurrent specifications satisfying the expansion property (and prefix-closure, etc).
The maps $\text{Lin}$ and $U$ form a Galois connection: for every $\sigma \in \text{SeqSpec}$ and $\tau \in \text{ConcSpec}$,

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

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 \( \text{Lin}(U(\tau)) = \tau \).
Results

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!