A Compositional Theory of Linearizability

Arthur Oliveira Vale    Zhong Shao    Yixuan Chen

Yale University, USA
Linearizability: A Correctness Condition for Concurrent Objects

MAURICE P. HERLIHY and JEANNETTE M. WING
Carnegie Mellon University

A concurrent object is a data object shared by concurrent processes. Linearizability is a correctness condition for concurrent objects that exploits the semantics of abstract data types. It permits a high degree of concurrency, yet it permits programmers to specify and reason about concurrent objects using known techniques from the sequential domain. Linearizability provides the illusion that each operation applied by concurrent processes takes effect instantaneously at some point between its invocation and its response, implying that the meaning of a concurrent object's operations can be given by pre- and post-conditions. This paper defines linearizability, compares it to other correctness conditions, presents and demonstrates a method for proving the correctness of implementations, and shows how to reason about concurrent objects, given they are linearizable.

Categories and Subject Descriptors: D.1.3 [Programming Techniques]: Concurrent Programming; D.2.1 [Software Engineering]: Requirements/Specifications; D.3.3 [Programming Languages]: Language Constructs—abstract data types, concurrent programming structures, data types and structures; F.1.2 [Computation by Abstract Devices]: Modes of Computation—parallelism; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs—pre- and post-conditions, specification techniques

General Terms: Theory, Verification

Additional Key Words and Phrases: Concurrency, correctness, Larch, linearizability, multiprocessing, serializability, shared memory, specification

What is a concurrent object?

Queue := \{enq : \mathbb{N} \rightarrow \{\text{ok}\}, \text{deq} : \mathbb{N} + \{\bot\}\}
What is a concurrent object?

Queue := \{\text{enq} : \mathbb{N} \rightarrow \{\text{ok}\}, \text{deq} : \mathbb{N} + \{\perp\}\}

- **Sequential:**
  - deq

- **Concurrent:**

---

**Linearizability: A Correctness Condition for Concurrent Objects**

MAURICE P. HERLIHY and JEANNETTE M. WING
Carnegie Mellon University

A concurrent object is a data object shared by concurrent processes. Linearizability is a correctness condition for concurrent objects that exploits the semantics of abstract data types. It permits a high degree of concurrency, yet it permits programmers to specify and reason about concurrent objects using known techniques from the sequential domain. Linearizability provides the illusion that each operation applied by concurrent processes takes effect instantaneously at some point between its invocation and its response, implying that the meaning of a concurrent object's operations can be given by pre- and post-conditions. This paper defines linearizability, compares it to other correctness conditions, presents and demonstrates a method for proving the correctness of implementations, and shows how to reason about concurrent objects, given they are linearizable.

Categories and Subject Descriptors: D.1.3 [Programming Techniques]: Concurrent Programming; D.2.1 [Software Engineering]: Requirements/Specifications; D.3.3 [Programming Languages]: Language Constructs—abstract data types, concurrent programming structures, data types and structures; F.1.2 [Computation by Abstract Devices]: Modes of Computation—parallelism; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs—pre- and post-conditions, specification techniques

General Terms: Theory, Verification

Additional Key Words and Phrases: Concurrency, correctness, Larch, linearizability, multiprocessing, serializability, shared memory, specification
Linearizability: A Correctness Condition for Concurrent Objects

MAURICE P. HERLIHY and JEANNETTE M. WING
Carnegie Mellon University

A concurrent object is a data object shared by concurrent processes. Linearizability is a correctness condition for concurrent objects that exploits the semantics of abstract data types. It permits a high degree of concurrency, yet it permits programmers to specify and reason about concurrent objects using known techniques from the sequential domain. Linearizability provides the illusion that each operation applied by concurrent processes takes effect instantaneously at some point between its invocation and its response, implying that the meaning of a concurrent object’s operations can be given by pre- and post-conditions. This paper defines linearizability, compares it to other correctness conditions, presents and demonstrates a method for proving the correctness of implementations, and shows how to reason about concurrent objects, given they are linearizable.

Categories and Subject Descriptors: D.1.3 [Programming Techniques]: Concurrent Programming; D.2.1 [Software Engineering]: Requirements/Specifications; D.3.3 [Programming Languages]: Language Constructs—abstract data types, concurrent programming structures, data types and structures; F.1.2 [Computation by Abstract Devices]: Modes of Computation—parallelism; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs—pre- and post-conditions, specification techniques

General Terms: Theory, Verification

Additional Key Words and Phrases: Concurrency, correctness, Larch, linearizability, multiprocessing, serializability, shared memory, specification

What is a concurrent object?

Queue := {enq : \mathbb{N} → \{ok\}, deq : \mathbb{N} + \{⊥\} }

► Sequential:
  deq ←→ ⊥

► Concurrent:
What is a concurrent object?

Queue := \{\text{enq} : \mathbb{N} \rightarrow \{\text{ok}\}, \text{deq} : \mathbb{N} + \{\bot\}\}

- Sequential:
  \text{deq} \rightarrow \bot

- Concurrent:
What is a concurrent object?

Queue := \{\text{enq} : \mathbb{N} \to \{\text{ok}\}, \text{deq} : \mathbb{N} + \{\bot\}\}

- **Sequential:**
  \[
  \alpha_0 : \text{deq} \quad \rightarrow \quad \alpha_0 : \bot
  \]

- **Concurrent:**
What is a concurrent object?

Queue := \{enq : \mathbb{N} \rightarrow \{ok\}, \text{deq} : \mathbb{N} + \{\bot\}\}

- **Sequential:**
  \[
  \text{deq} \quad \rightarrow \quad \bot
  \]

- **Concurrent:**

\[\alpha_0 : \text{deq} \quad \alpha_1 : \text{enq}(1) \quad \alpha_2 : \text{enq}(2) \quad \alpha_1 : \text{ok} \quad \alpha_2 : \text{ok} \quad \alpha_0 : \bot/1/2\]
Classical Linearizability: Example

\[ \nu'_\text{queue} : \quad \alpha_0:\text{deq} \xrightarrow{\alpha_1:\text{enq}(1)} \alpha_0:1 \xrightarrow{\alpha_2:\text{enq}(2)} \alpha_1:ok \xrightarrow{\alpha_0:\text{deq}} \alpha_2:ok \]

\[ \nu_{\text{queue}} : \quad \alpha_1:\text{enq}(1) \xrightarrow{} \alpha_1:ok \xrightarrow{} \alpha_2:ok \]

\[ \alpha_0:\text{deq} \xrightarrow{} \alpha_0:1 \xrightarrow{} \alpha_0:\text{deq} \]

\[ \alpha_2:\text{enq}(2) \xrightarrow{} \alpha_2:ok \]

\[ \alpha_0:\text{deq} \xrightarrow{} \alpha_0:1 \xrightarrow{} \alpha_0:\text{deq} \]

\[ \alpha_1:\text{enq}(1) \xrightarrow{} \alpha_1:ok \]

\[ \alpha_2:\text{enq}(2) \xrightarrow{} \alpha_2:ok \]
Classical Linearizability: Example

\[ \nu_{\text{queue}} : \begin{array}{c}
\alpha_0:\text{deq} \\
\alpha_1:\text{enq}(1) \\
\alpha_0:\text{l} \\
\alpha_2:\text{enq}(2) \\
\alpha_1:\text{ok} \\
\alpha_0:\text{deq} \\
\alpha_2:\text{ok}
\end{array} \]
Classical Linearizability: Example

\[ \nu_s^{\prime} \text{queue} : \]

\[ \alpha_0: \text{deq} \xrightarrow{\alpha_1: \text{enq(1)}} \alpha_0: \text{l} \xrightarrow{\alpha_2: \text{enq(2)}} \alpha_1: \text{ok} \xrightarrow{\alpha_0: \text{deq}} \alpha_2: \text{ok} \]

\[ \alpha_0: \text{deq} \rightarrow \alpha_0: \text{l} \rightarrow \alpha_0: \text{deq} \]

\[ \alpha_1: \text{enq(1)} \rightarrow \alpha_1: \text{ok} \]

\[ \alpha_2: \text{enq(2)} \rightarrow \alpha_2: \text{ok} \]

\[ \nu_s \text{queue} : \]

\[ \alpha_1: \text{enq(1)} \rightarrow \alpha_1: \text{ok} \]

\[ \alpha_0: \text{deq} \rightarrow \alpha_0: \text{l} \rightarrow \alpha_0: \text{deq} \]

\[ \alpha_2: \text{enq(2)} \rightarrow \alpha_2: \text{ok} \]
Classical Linearizability: Example

\[\nu_{\text{queue}} : \]

\[\alpha_0: \text{deq} \rightarrow \alpha_1: \text{enq}(1) \rightarrow \alpha_0: \text{ok} \rightarrow \alpha_0: \text{deq} \rightarrow \alpha_2: \text{ok} \]

\[\alpha_1: \text{enq}(1) \rightarrow \alpha_0: \text{ok} \rightarrow \alpha_1: \text{ok} \rightarrow \alpha_2: \text{ok} \]

\[\alpha_2: \text{enq}(2) \rightarrow \alpha_2: \text{ok} \]

\[\nu_{\text{queue}} : \]

\[\alpha_1: \text{enq}(1) \rightarrow \alpha_1: \text{ok} \rightarrow \alpha_2: \text{enq}(2) \rightarrow \alpha_2: \text{ok} \]
Classical Linearizability: Example

\[ \nu'_{\text{queue}} : \]

\[ \alpha_0: \text{deq} \rightarrow \alpha_1: \text{enq}(1) \rightarrow \alpha_0: \text{l} \rightarrow \alpha_2: \text{enq}(2) \rightarrow \alpha_1: \text{ok} \rightarrow \alpha_0: \text{deq} \rightarrow \alpha_2: \text{ok} \]
PROPOSITION

$H$ is linearizable if and only if, for each object $x$, $H | x$ is linearizable.
Equivalence with Contextual Refinement [Filipović et Al. 2010]

\[ \text{Obj}_{\text{Conc}} \text{ observationally refines } (\sqsubseteq) \text{ Obj}_{\text{Atom}} \text{ when} \]

\[ \forall \text{ programs } P. \forall \text{ states } s. [P](\text{Obj}_{\text{Conc}})(s) \subseteq [P](\text{Obj}_{\text{Atom}})(s) \]

**PROPOSITION**

\[ \text{Obj}_{\text{Conc}} \text{ linearizes to } \text{Obj}_{\text{Atom}} \iff \text{Obj}_{\text{Conc}} \text{ observationally refines } \text{Obj}_{\text{Atom}} \]
Why?

Where does linearizability come from and why does it work?
Key Contributions

- A new generalized definition of linearizability not tied to atomicity.
- The first model of linearizability that supports refinement, horizontal and vertical composition.
- A general (category-theoretic) methodology for deriving linearizability from a model of concurrent computation.
- New simpler proofs of the locality and refinement properties.
- A new program logic that is sound for our formulation of linearizability.
- Applications to compositional verification.
Outline

Introduction

Compositionality

Sequentially Consistent Computation

Linearizability

Properties

Applications
Typical Approach for Verifying Concurrent Objects

\[ \nu_A' \subseteq \nu_A \rightarrow P \subseteq P \]

\[ \nu_A' \subseteq \nu_A \]

\[ P \subseteq P \]

\[ \nu_B \subseteq \nu_A \]
Typical Approach for Verifying Concurrent Objects

\[ \nu_A' \subseteq \nu_A \Rightarrow \nu_A' \subseteq \nu_A \]

\[ P \subseteq \nu_B \]
Typical Approach for Verifying Concurrent Objects

\[ \nu_A' \subseteq \nu_A \implies P \subseteq P \]

\[ P \subseteq \nu_{A'} \]

\[ P \subseteq \nu_A \]

\[ P \subseteq \nu_B \]
Implementing a Shared Queue

\[ M_{\text{queue}}: \]
Import Q:Queue
Import L:Lock

\[
\begin{align*}
\text{enq}(k) \{ & \quad \text{deq()} \{ \\
& \quad \text{L.acq();} \quad \text{L.acq();} \\
& \quad r <- Q.enq(k); \quad r <- Q.deq(); \\
& \quad L.rel(); \quad L.rel(); \\
& \quad \text{ret } r \quad \text{ret } r \\
\} \quad \} \\
\end{align*}
\]

- No account of how locality interacts with refinement.
- Locality doesn’t apply! The queue has a race (not linearizable).
Implementing a Shared Queue (Continued)

\[ \nu'_{\text{queue}} \leftarrow \nu'_{\text{squeue}} \]

\[ M_{\text{squeue}} \]

\[ \nu'_{\text{lock}} \quad \nu'_{\text{queue}} \]

\[ \text{inline} \quad \Rightarrow \]

\[ \nu'_{\text{queue}} \leftarrow \nu'_{\text{squeue}} \]

\[ M'_{\text{squeue}} \]

\[ \nu'_{\text{lock}} \]

\[ \text{refine} \]

\[ \nu'_{\text{lock}} \quad \nu'_{\text{queue}} \]

\[ M'_{\text{squeue}} \]

\[ \nu'_{\text{lock}} \]
Inlining? Syntactic Linking?
Vertical Composition

\[ \nu'_\text{lock} \land \nu'_\text{queue} \Rightarrow \nu'_\text{lock} + \nu'_\text{queue} \]

Inlining ? Syntactic Linking?
<table>
<thead>
<tr>
<th>Linearizability</th>
<th>Refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Compositionality</td>
<td>⊆</td>
</tr>
<tr>
<td>(Vertical) Composition</td>
<td>; ;</td>
</tr>
</tbody>
</table>
Outline

Introduction

Compositionality

Sequentially Consistent Computation

Linearizability

Properties

Applications
Our Methodology

1. Base Model of Computation
   (A semicategory enriched with a notion of refinement)

2. Choose identity programs
   (Usually obvious)

3. Compute a Compositional Model out of (1) and (2)
   (The Karoubi Envelope)

4. Abstract Linearizability $\iff$ Concrete Linearizability

5. One Extra Axiom $\implies$ Refinement Property

6. Tensor Product + One Extra Axiom $\implies$ Locality
Types correspond to Games $A, B, C$.

Programs correspond to strategies $\sigma : A \rightarrow B$ of the game $A \leftarrow B$.

Object specifications correspond to strategies $\nu : 1 \rightarrow A$. 
We start by defining a sequential model of computation.

A set of agent names $\alpha \in \Upsilon$.

A concurrent game $A$ is specified by the sequential game $A$ that all agents play.

A move looks like $\alpha : m$ where $\alpha \in \Upsilon$ and $m$ is a move of $A$.

The set of plays of $A$ is the set of sequentially consistent interleavings of plays from $A$.

Example:

Counter = \{get : \mathbb{N}, \text{inc} : \text{ok}\}

\[\alpha_0 : \text{inc} \rightarrow \alpha_1 : \text{get} \rightarrow \alpha_0 : \text{ok} \rightarrow \alpha_0 : \text{get} \rightarrow \alpha_1 : n \rightarrow \alpha_2 : \text{get}\]
Vertical Composition

There is a composition operation defined per usual by

”Parallel composition + Hiding”

Denoted by

\[ \sigma : A \rightarrow B \quad \tau : B \rightarrow C \quad \sigma; \tau : A \rightarrow C \]

Which is associative ... but there is no identity element!

\[ \forall \sigma : A \rightarrow B. \text{id}_A; \sigma; \text{id}_B = \sigma \]

In other words, concurrent games with concurrent strategies assembles into a semicategory

\[ \text{Game}_{\text{Conc}} \]
Our model is enriched over a notion of refinement $\subseteq$ (behavior containment)
The copycat strategy $\text{copy}_A : A \rightarrow A$ behaves as the sequential identity.
Concurrent Strategies

Composition can lead to emergent behavior.

\[ \sigma \subseteq \sigma; \text{ccopy}_B \]
Concurrent Strategies

\[ \text{ccopy}_A := \parallel_{\alpha \in \tau} \text{ccopy}_A \]

Composition can lead to **emergent behavior**.

\[ \sigma \subseteq \sigma; \text{ccopy}_B \]
The Karoubi Envelope

PROPOSITION

For all concurrent game $A$ the strategy $\text{ccopy}_A : A \rightarrow A$ is idempotent, i.e.

$$\text{ccopy}_A ; \text{ccopy}_A = \text{ccopy}_A$$

Call a strategy $\sigma : A \rightarrow B$ saturated when

$$\text{ccopy}_A ; \sigma ; \text{ccopy}_B = \sigma$$

Composition of saturated strategies is associative and has as identity $\text{ccopy}_\_$. Call the resulting category of concurrent games and saturated strategies

\[ \text{Game}^{\text{Conc}} \]
Two Models of Concurrent Computation

<table>
<thead>
<tr>
<th>Game_{Conc}</th>
<th>Game_{Conc}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good for specification</td>
<td>Good for composition</td>
</tr>
</tbody>
</table>

We can convert between models:

$$\sigma : A \rightarrow B \in \text{Game}_{\text{Conc}} \xrightarrow{K_{\text{Conc}}} \text{ccopy}_A \sigma \; \text{ccopy}_B \in \text{Game}_{\text{Conc}}$$
Outline

Introduction

Compositionality

Sequentially Consistent Computation

Linearizability

Properties

Applications
**DEFINITION (ABSTRACT LINEARIZABILITY)**

We say

\[ \nu'_A : A \in \text{Game}_{\text{Conc}} \]

linearizes to

\[ \nu_A : A \in \text{Game}_{\text{Conc}} \]

when

\[ \nu'_A \subseteq K_{\text{Conc}} \nu_A \]

\[ \nu'_A \text{ is the implementation and } \nu_A \text{ the specification} \]

**DEFINITION**

A linearizable object consists of a pair

\[ (\nu'_A : A \in \text{Game}_{\text{Conc}}, \nu_A : A \in \text{Game}_{\text{Conc}}) \]

such that

\[ \nu'_A \subseteq K_{\text{Conc}} \nu_A \]
PROPOSITION (GHICA AND MURAWSKI, 2004)

\[ \sigma : A \text{ is saturated} \quad \text{if and only if} \quad \forall t \in \sigma. \forall s \in P_A.s \rightsquigarrow_A t \implies s \in \sigma \]

If \( t \in \sigma \) and \( s \) is ”more concurrent” than \( t \) then \( s \) is also in \( \sigma \)
DEFINITION

$s \in P_A$ is linearizable to $t \in P_A$ when there exists a sequence $s_O$ of Opponent moves and a sequence $s_P$ of Proponent moves such that

$$s \cdot s_P \leadsto_A t \cdot s_O$$

- $t$ need not be atomic (coincides with Herlihy-Wing when it is);
- $s_P = \text{returns}$;
- $s_O = \text{removed pending invocations (not all need be removed)}$;
- $\leadsto_A = \text{happens-before order preservation.}$
Abstract Linearizability

PROPOSITION

Let $\tau : A \in \text{Game}_{\text{Conc}}$ then

$$K_{\text{Conc}} \tau = \{s \in P_A \mid \exists t \in \tau.s \text{ linearizes to } t\}$$
Abstract Linearizability

**PROPOSITION**

Let $\tau : A \in \text{Game}_{\text{Conc}}$ then

$$K_{\text{Conc}} \tau = \{ s \in P_A | \exists t \in \tau. s \text{ linearizes to } t \}$$

**COROLLARY**

For $\sigma : A$ and $\tau : A$, $\sigma$ linearizes to $\tau \iff \sigma \subseteq K_{\text{Conc}} \tau$. 
Abstract Linearizability

PROPOSITION
Let $\tau : A \in \text{Game}_{\text{Conc}}$ then

$$K_{\text{Conc}} \tau = \{s \in P_A \mid \exists t \in \tau. s \text{ linearizes to } t\}$$

DEFINITION (ABSTRACT LINEARIZABILITY)
We say $\sigma : A \in \text{Game}_{\text{Conc}}$ linearizes to $\tau : A \in \text{Game}_{\text{Conc}}$ when

$$\sigma \subseteq K_{\text{Conc}} \tau$$
Interaction Refinement

\[ \forall B : \forall \sigma : A \rightarrow B. \sigma \in \text{Game}_{\text{Conc}} \]
Interaction Refinement: Proof (Forward)

\[ \nu' \subseteq \text{ccopy}_A \nu \Rightarrow \nu' \subseteq \nu \]

\[ \sigma \subseteq \text{ccopy}_A \sigma \]

\[ \nu' = \nu \]

\[ \nu' \subseteq \nu \]
\( \forall B. \forall \sigma : A \xrightarrow{\sigma} B. \)

\[
\sigma \\
\sigma \\
\nu'_A \\
\nu_A \\
\subseteq \\
\equiv \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq \\
\nu'_A \\
\nu_A \\
\Rightarrow \\
\subseteq
Horizontal Composition

We define a tensor product of strategies:

$$\sigma : A \quad , \quad \tau : B \quad \longrightarrow \quad \sigma \otimes \tau : A \otimes B$$

where

$$\sigma \otimes \tau = \text{all sequentially consistent interleavings of plays of } \sigma \text{ and } \tau$$

This makes $\text{Game}_{\text{Conc}}$ into a symmetric monoidal category.

($\otimes$ has a unit $1$, is associative and commutative, bifunctor, ...
Horizontal Composition: Functorial
Horizontal Composition: Monotonicity

\[ \sigma_1 : A \quad \tau_1 : B \quad \sigma_0 : A \quad \tau_0 : B \]

\[ \sigma_1 \land \tau_1 \quad \sigma_0 \land \tau_0 \]

\[ \Rightarrow \]

\[ \sigma_1 \otimes \tau_1 \quad \sigma_0 \otimes \tau_0 \]
Horizontal Composition: Order-Isomorphism

\[ \sigma_1 : A \quad \tau_1 : B \quad \sigma_0 : A \quad \tau_0 : B \]

\[ \sigma_0 : A \quad \tau_0 : B \quad \sigma_1 : A \quad \tau_1 : B \]

\[ \sigma_0 : A \quad \tau_0 : B \quad \sigma_1 : A \quad \tau_1 : B \]

\[ \sigma_0 : A \quad \tau_0 : B \quad \sigma_1 : A \quad \tau_1 : B \]

\[ \sigma_0 : A \quad \tau_0 : B \quad \sigma_1 : A \quad \tau_1 : B \]

\[ \sigma_0 : A \quad \tau_0 : B \quad \sigma_1 : A \quad \tau_1 : B \]
Locality

THEOREM

\[ \text{ccopy}_A \subseteq \nu_A \land \nu'_A \quad \text{and} \quad \text{ccopy}_B \subseteq \nu_B \land \nu'_B \quad \iff \quad \text{ccopy}_{A \otimes B} \subseteq \nu_{A \otimes B} \land \nu'_{A \otimes B} \]
Locality: Proof

\[ \text{ccopy}_{A \otimes B} \]

\[ \nu_A \otimes \nu_B \]

(Compatibility)

\[ \text{ccopy}_A \otimes \text{ccopy}_B \]

(Compatibility)

\[ \nu_A \otimes \nu_B \]

(Functoriality)
Locality: Proof

THEOREM

Holds by the order-isomorphism
Locality Proof [Herlihy and Wing, 1990]

Let $<$ be the transitive closure of the union of all $<_x$ with $<_H$. It is immediate from the construction that $<$ satisfies Conditions (1) and (2), but it remains to be shown that $<$ is a partial order. We argue by contradiction. If not, then there exists a set of operations $e_1, \ldots, e_n$, such that $e_1 < e_2 < \cdots < e_n$, $e_n < e_1$, and each pair is directly related by some $<_x$ or by $<_H$. Choose a cycle whose length is minimal.

Suppose all operations are associated with the same object $x$. Since $<_x$ is a total order, there must exist two operations $e_{i-1}$ and $e_i$ such that $e_{i-1} <_H e_i$ and $e_i <_x e_{i-1}$, contradicting the linearizability of $x$.

The cycle must therefore include operations of at least two objects. By reindexing if necessary, let $e_1$ and $e_2$ be operations of distinct objects. Let $x$ be the object associated with $e_1$. We claim that none of $e_2, \ldots, e_n$ can be an operation of $x$. The claim holds for $e_2$ by construction. Let $e_i$ be the first operation in $e_3, \ldots, e_n$ associated with $x$. Since $e_{i-1}$ and $e_i$ are unrelated by $<_x$, they must be related by $<_H$; hence the response of $e_{i-1}$ precedes the invocation of $e_i$. The invocation of $e_2$ precedes the response of $e_{i-1}$, since otherwise $e_{i-1} <_H e_2$, yielding the shorter cycle $e_2, \ldots, e_{i-1}$. Finally, the response of $e_1$ precedes the invocation of $e_2$, since $e_1 <_H e_2$ by construction. It follows that the response to $e_1$ precedes the invocation of $e_i$, hence $e_i <_H e_1$, yielding the shorter cycle $e_1, e_i, \ldots, e_n$.

Since $e_n$ is not an operation of $x$, but $e_n < e_1$, it follows that $e_n <_H e_1$. But $e_1 <_H e_2$ by construction, and because $<_H$ is transitive, $e_n <_H e_2$, yielding the shorter cycle $e_2, \ldots, e_n$, the final contradiction. \[\square\]
Implementing a Shared Queue

\[ \nu'_{\text{queue}} \]

\[ M'_{\text{queue}} \]

\[ \nu'_{\text{lock}} \]

\[ \nu'_{\text{queue}} \]

\[ M'_{\text{queue}} \]

\[ \nu'_{\text{lock}} \]

\[ \nu'_{\text{fai}} \]

\[ \nu'_{\text{counter}} \]

\[ \nu'_{\text{yield}} \]

\[ \nu_{\text{lock}} \]

\[ M_{\text{lock}} \]

\[ \nu_{\text{fai}} \]

\[ \nu_{\text{counter}} \]

\[ \nu_{\text{yield}} \]
Program Logic

- We define a program logic for showing individual programs implement linearizable objects.
- Sound for our notion of linearizability (and in particular for, interval-linearizability).
- Directly connects with our compositional theory.

**PROPOSITION (SOUNDNESS)**

If \( R[A], G[A] \models_A \{ P[A] \} M[A] \{ Q[A] \} \) and \((\nu'_E : \uparrow E, \nu_E : \uparrow E)\) is a linearizable concurrent object then

\[
\nu'_E; [M[A]] \cap \nu'_F \subseteq K_{\text{Conc}} \nu_F
\]
Conclusion

- New foundations for linearizability and its properties.
- A compositional theory for linearizability.
- Promising applications for compositional verification.

Check our paper and TR for more:

- The concurrent game semantics model
- The category-theoretic axiomatization
- Thorough comparison with previous work
- The example we described in this talk
- Full program logic description
- More...

Thank you!
Conclusion

- New foundations for linearizability and its properties.
- A compositional theory for linearizability.
- Promising applications for compositional verification.

Check our paper and TR for more:

- The concurrent game semantics model
- The category-theoretic axiomatization
- Thorough comparison with previous work
- The example we described in this talk
- Full program logic description
- More...

Thank you!
Conclusion

- New foundations for linearizability and its properties.
- A compositional theory for linearizability.
- Promising applications for compositional verification.

Check our paper and TR for more:

- The concurrent game semantics model
- The category-theoretic axiomatization
- Thorough comparison with previous work
- The example we described in this talk
- Full program logic description
- More...

Thank you!