Geometric semantics for asynchronous computability

Jérémy Ledent
joint work with Éric Goubault and Samuel Mimram

École Polytechnique

CHoCoLa
June 6, 2019
A topological approach for asynchronous computability
The two generals problem

Two divisions of the same army, commanded by general $A$ and general $B$, are surrounding an enemy fortress.
The two generals problem

Two divisions of the same army, commanded by general $A$ and general $B$, are surrounding an enemy fortress.

- They must attack simultaneously.
The two generals problem

Two divisions of the same army, commanded by general $A$ and general $B$, are surrounding an enemy fortress.

- They must attack simultaneously.
- They communicate by sending messengers.
The two generals problem

Two divisions of the same army, commanded by general $A$ and general $B$, are surrounding an enemy fortress.

- They must attack simultaneously.
- They communicate by sending messengers.
- Messengers might be captured by the enemy, in which case, the message is never received.
The two generals problem

Two divisions of the same army, commanded by general $A$ and general $B$, are surrounding an enemy fortress.

- They must attack simultaneously.
- They communicate by sending messengers.
- Messengers might be captured by the enemy, in which case, the message is never received.

How can they coordinate the attack?
The two generals problem

Two divisions of the same army, commanded by general $A$ and general $B$, are surrounding an enemy fortress.

- They must attack simultaneously.
- They communicate by sending messengers.
- Messengers might be captured by the enemy, in which case, the message is never received.

How can they coordinate the attack?
The two generals problem

Two divisions of the same army, commanded by general $A$ and general $B$, are surrounding an enemy fortress.

- They must attack simultaneously.
- They communicate by sending messengers.
- Messengers might be captured by the enemy, in which case, the message is never received.

How can they coordinate the attack?
The two generals problem

Two divisions of the same army, commanded by general $A$ and general $B$, are surrounding an enemy fortress.

- They must attack simultaneously.
- They communicate by sending messengers.
- Messengers might be captured by the enemy, in which case, the message is never received.

How can they coordinate the attack?
Asynchronous computability
a.k.a. Fault-tolerant distributed computing

**Goal:** Prove that a given concurrent task is unsolvable in a given computational model.
Asynchronous computability
a.k.a. Fault-tolerant distributed computing

Goal: Prove that a given concurrent task is unsolvable in a given computational model.

- Shared memory or message-passing
- Communication primitives: read/write, test&set, CAS...
- Type of errors: crashes, lost messages, byzantine failures...
Asynchronous computability  
a.k.a. Fault-tolerant distributed computing

**Goal:** Prove that a given *concurrent task* is unsolvable in a given computational model.

- **Shared memory or message-passing**
- *Communication primitives:* read/write, test&set, CAS...
- *Type of errors:* crashes, lost messages, byzantine failures...
- *Tasks:* Consensus, weak symmetry breaking, renaming...
Asynchronous computability
a.k.a. Fault-tolerant distributed computing

**Goal:** Prove that a given *concurrent task* is unsolvable in a given computational model.

- **Shared memory** or message-passing
- **Communication primitives:** `read/write`, `test&set`, `CAS`
- **Type of errors:** *crashes*, lost messages, byzantine failures...
- **Tasks:** *Consensus*, weak symmetry breaking, renaming...
Asynchronous computability
a.k.a. Fault-tolerant distributed computing

**Goal:** Prove that a given concurrent task is unsolvable in a given computational model.

- **Shared memory** or message-passing
- **Communication primitives:** read/write, test&set, CAS...
- **Type of errors:** crashes, lost messages, byzantine failures...
- **Tasks:** Consensus, weak symmetry breaking, renaming...

**Remark:** Usually, impossibility results come from a lack of information about the system, not from a lack of computing power.
A topological approach

Herlihy and Shavit, 1999
2004 Gödel prize

**Theorem 3.1 (Asynchronous Computability Theorem).** A decision task $(\mathcal{A}, \mathcal{C}, \Delta)$ has a wait-free protocol using read-write memory if and only if there exists a chromatic subdivision $\sigma$ of $\mathcal{A}$ and a color-preserving simplicial map

$$\mu: \sigma(\mathcal{A}) \to \mathcal{C}$$

such that for each simplex $S$ in $\sigma(\mathcal{A})$, $\mu(S) \in \Delta(\text{carrier}(S, \mathcal{A}))$. 
A topological approach

Herlihy and Shavit, 1999
2004 Gödel prize

Herlihy, Kozlov, Rajsbaum, 2013
Simplicial complexes

**Definition**

An (abstract) **simplicial complex** is a pair $\langle V, S \rangle$ where $V$ is a set of *vertices* and $S$ is a downward-closed family of subsets of $V$ called *simplices* (i.e., $X \in S$ and $Y \subseteq X$ implies $Y \in S$).
Definition

An (abstract) simplicial complex is a pair $\langle V, S \rangle$ where $V$ is a set of vertices and $S$ is a downward-closed family of subsets of $V$ called simplices (i.e., $X \in S$ and $Y \subseteq X$ implies $Y \in S$).
Definition

An (abstract) **simplicial complex** is a pair $\langle V, S \rangle$ where $V$ is a set of *vertices* and $S$ is a downward-closed family of subsets of $V$ called *simplices* (i.e., $X \in S$ and $Y \subseteq X$ implies $Y \in S$).
Definition

An (abstract) **simplicial complex** is a pair \( \langle V, S \rangle \) where \( V \) is a set of *vertices* and \( S \) is a downward-closed family of subsets of \( V \) called *simplices* (i.e., \( X \in S \) and \( Y \subseteq X \) implies \( Y \in S \)).
Example: binary input complex for 3 processes

- Every process has input value either 0 or 1.
- Every process knows its value, but not the other values.
Example: binary input complex for 3 processes

- Every process has input value either 0 or 1.
- Every process knows its value, but not the other values.

In the picture below, the three process names are represented as the colors black, grey, white:
Example: binary input complex for 3 processes

- Every process has input value either 0 or 1.
- Every process knows its value, but not the other values.

In the picture below, the three process names are represented as the colors black, grey, white:
Example: binary input complex for 3 processes

- Every process has input value either 0 or 1.
- Every process knows its value, but not the other values.

In the picture below, the three process names are represented as the colors black, grey, white:
Example: binary input complex for 3 processes

- Every process has input value either 0 or 1.
- Every process knows its value, but not the other values.

In the picture below, the three process names are represented as the colors black, grey, white:
Example: binary input complex for 3 processes

- Every process has input value either 0 or 1.
- Every process knows its value, but not the other values.

In the picture below, the three process names are represented as the colors black, grey, white:
Example: binary input complex for 3 processes

- Every process has input value either 0 or 1.
- Every process knows its value, but not the other values.

In the picture below, the three process names are represented as the colors black, grey, white:
The immediate snapshot object

\[\text{immediate\_snapshot} : \ 'a \rightarrow \ 'a\ array\]

Fix a number \(n\) of processes. We suppose given a shared array \(A\) of size \(n\). Only process \(P_i\) can write in \(A[i]\), but everyone can read it.

When \(P_i\) calls \(\text{immediate\_snapshot}(x)\):

- It writes its input value \(x\) in its own cell \(A[i]\).
- Then atomically takes a snapshot of the whole array.
The immediate snapshot object

immediate_snapshot :  'a -> 'a array

Fix a number \( n \) of processes.
We suppose given a shared array \( A \) of size \( n \).
Only process \( P_i \) can write in \( A[i] \), but everyone can read it.

When \( P_i \) calls \texttt{immediate\_snapshot(x)}:

- It writes its input value \( x \) in its own cell \( A[i] \).
- Then atomically takes a snapshot of the whole array.

**Example:** for 3 processes \( P, Q, R \) with inputs 1, 2, 3.

\[
A = \begin{bmatrix}
    & & \\
    & & \\
\end{bmatrix}
\]
The immediate snapshot object

\[
\text{immediate\_snapshot} : \ 'a \rightarrow \ 'a \text{ array}
\]

Fix a number \( n \) of processes.
We suppose given a shared array \( A \) of size \( n \).
Only process \( P_i \) can write in \( A[i] \), but everyone can read it.

When \( P_i \) calls \( \text{immediate\_snapshot}(x) \):

- It writes its input value \( x \) in its own cell \( A[i] \).
- Then atomically takes a snapshot of the whole array.

**Example:** for 3 processes \( P, Q, R \) with inputs 1, 2, 3.

\[
A = \begin{bmatrix}
  \_ & \_ & 2 \\
\end{bmatrix}
\]
The immediate snapshot object

\[ \text{immediate\_snapshot} : \ 'a \rightarrow \ 'a \text{ array} \]

Fix a number \( n \) of processes.
We suppose given a shared array \( A \) of size \( n \).
Only process \( P_i \) can write in \( A[i] \), but everyone can read it.

When \( P_i \) calls \( \text{immediate\_snapshot}(x) \):
- It writes its input value \( x \) in its own cell \( A[i] \).
- Then atomically takes a snapshot of the whole array.

**Example:** for 3 processes \( P, Q, R \) with inputs 1, 2, 3.

\[
A = \begin{array}{ccc}
\hline
& & \\
& 2 & 3 \\
\hline
\end{array}
\]
The immediate snapshot object

\[
\text{immediate\_snapshot} : \ 'a \to \ 'a \text{ array}
\]

Fix a number \( n \) of processes.
We suppose given a shared array \( A \) of size \( n \).
Only process \( P_i \) can write in \( A[i] \), but everyone can read it.

When \( P_i \) calls \( \text{immediate\_snapshot}(x) \):
- It writes its input value \( x \) in its own cell \( A[i] \).
- Then atomically takes a snapshot of the whole array.

**Example:** for 3 processes \( P, Q, R \) with inputs 1, 2, 3.

\[
A = \begin{array}{c}
\hline
& 2 & 3 \\
\hline
\end{array}
\]

\[
R\text{’s view: } \begin{array}{c}
\hline
& 2 & 3 \\
\hline
\end{array}
\]
The immediate snapshot object

\[
\text{immediate\_snapshot} : \ 'a \rightarrow \ 'a \text{ array}
\]

Fix a number \( n \) of processes.
We suppose given a shared array \( A \) of size \( n \).
Only process \( P_i \) can write in \( A[i] \), but everyone can read it.

When \( P_i \) calls \( \text{immediate\_snapshot}(x) \):

- It writes its input value \( x \) in its own cell \( A[i] \).
- Then atomically takes a snapshot of the whole array.

**Example:** for 3 processes \( P, Q, R \) with inputs 1, 2, 3.

\[
A = \begin{array}{c|c|c}
\text{ } & 2 & 3 \\
\end{array}
\]

\[
Q\text{'s view: } \begin{array}{c|c|c}
\text{ } & 2 & 3 \\
\end{array}
\]

\[
R\text{'s view: } \begin{array}{c|c|c}
\text{ } & 2 & 3 \\
\end{array}
\]
The immediate snapshot object

\[ \text{immediate\_snapshot} : \ 'a \rightarrow \ 'a\ array \]

Fix a number \( n \) of processes.
We suppose given a shared array \( A \) of size \( n \).
Only process \( P_i \) can write in \( A[i] \), but everyone can read it.

When \( P_i \) calls \( \text{immediate\_snapshot}(x) \):
- It writes its input value \( x \) in its own cell \( A[i] \).
- Then atomically takes a snapshot of the whole array.

**Example:** for 3 processes \( P, Q, R \) with inputs 1, 2, 3.

\[
A = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}
\]

\[
Q's\ view: \begin{bmatrix} & 2 & 3 \end{bmatrix} \\
R's\ view: \begin{bmatrix} 2 & 3 \end{bmatrix}
\]
The immediate snapshot object

\[ \text{immediate\_snapshot : 'a \to 'a array} \]

Fix a number \( n \) of processes.
We suppose given a shared array \( A \) of size \( n \).
Only process \( P_i \) can write in \( A[i] \), but everyone can read it.

When \( P_i \) calls \( \text{immediate\_snapshot(x)} \):

- It writes its input value \( x \) in its own cell \( A[i] \).
- Then atomically takes a snapshot of the whole array.

**Example:** for 3 processes \( P, Q, R \) with inputs 1, 2, 3.

\[
A = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}
\]

\[
\begin{array}{c|c}
\text{P's view:} & 1 & 2 & 3 \\
\text{Q's view:} & & 2 & 3 \\
\text{R's view:} & & 2 & 3 \\
\end{array}
\]
Protocol complex for immediate snapshot

Input Complex

Immediate Snapshot

Protocol Complex
Protocol complex for immediate snapshot

Input Complex

Immediate Snapshot

Protocol Complex
The (binary) consensus task

There is a fixed number \( n \) of processes. Each process \( P_i \) has a binary input \( in_i \in \{0, 1\} \). After communicating, it decides an output \( d_i \in \{0, 1\} \).
The (binary) consensus task

There is a fixed number $n$ of processes. Each process $P_i$ has a binary input $in_i \in \{0, 1\}$. After communicating, it decides an output $d_i \in \{0, 1\}$.

**Specification:**

- **Agreement:** $d_i = d_j$ for all $i, j$.
- **Validity:** $d_i \in \{in_i \mid 1 \leq i \leq n\}$ for all $i$.
The (binary) consensus task

There is a fixed number $n$ of processes. Each process $P_i$ has a binary input $in_i \in \{0, 1\}$. After communicating, it decides an output $d_i \in \{0, 1\}$.

**Specification:**

- **Agreement:** $d_i = d_j$ for all $i, j$.
- **Validity:** $d_i \in \{in_i | 1 \leq i \leq n\}$ for all $i$.

**Examples:** for 3 processes

- if the inputs are $(0, 0, 0)$, the outputs must be $(0, 0, 0)$.
- if the inputs are $(1, 0, 1)$, the outputs can be either $(0, 0, 0)$ or $(1, 1, 1)$. 
Topological definition of task solvability

Input complex
Topological definition of task solvability

Input complex

Output complex

Task specification

Input complex

Output complex

Task specification
Topological definition of task solvability
Topological definition of task solvability

Protocol complex

Output complex

Computation

Decision

Input complex

Task specification
Theorem (Herlihy and Shavit, 1999)

A task is solvable by a wait-free protocol using read/write registers if and only if there is a decision map from a subdivision of the input complex into the output complex such that [...].
Asynchronous computability theorem

Theorem (Herlihy and Shavit, 1999)

A task is solvable by a wait-free protocol using read/write registers if and only if there is a decision map from a subdivision of the input complex into the output complex such that [...].

What if:

- we replace “wait-free” by “t-resilient”?
- we use other objects instead of read/write registers?
- we use a message-passing architecture?
Asynchronous computability theorem

Theorem (Herlihy and Shavit, 1999)

A task is solvable by a wait-free protocol using read/write registers if and only if there is a decision map from a subdivision of the input complex into the output complex such that [...].

What if:

- we replace “wait-free” by “t-resilient”?
- we use other objects instead of read/write registers?
- we use a message-passing architecture?

Goal: an asynchronous computability theorem for any objects.
Specifying concurrent objects
Objects vs Tasks

“Can we solve the task $T$ using the objects $A_1, \ldots, A_k$?”
Objects vs Tasks

"Can we solve the task $T$ using the objects $A_1, \ldots, A_k$?"

**Objects:**
- Long-lived

**Tasks:**
- Used only once
Objects vs Tasks

“Can we solve the task $T$ using the objects $A_1, \ldots, A_k$?”

**Objects:**
- Long-lived
- Have a sequential flavor

**Tasks:**
- Used only once
- Intrinsically concurrently
Objects vs Tasks

“Can we solve the task $T$ using the objects $A_1, \ldots, A_k$?”

**Objects:**
- Long-lived
- Have a sequential flavor

**Tasks:**
- Used only once
- Intrinsically concurrently

But in practice:

“I can solve consensus using $X$, and I can solve $Y$ using consensus objects, so I can solve $Y$ using $X$”
Objects vs Tasks

“Can we solve the task $T$ using the objects $A_1, \ldots, A_k$?”

**Objects:**
- Long-lived
- Have a sequential flavor

**Tasks:**
- Used only once
- Intrinsically concurrently

But in practice:

“I can solve consensus using $X$, and I can solve $Y$ using consensus objects, so I can solve $Y$ using $X$”

$\rightarrow$ We would like a *composable* notion of “solving”.
Objects

From now on, everything is an object:
Objects

From now on, everything is an object:

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

"Can we implement the object B using the objects A₁,...,Aₖ?"

→ How do we specify a concurrent object?

→ What does it mean to implement an object?
From now on, everything is an object:

- Hardware: Read/Write registers, test&set, CAS, ...  
- Data structures: lists, queues, hashmaps, ...
From now on, everything is an object:

- Hardware: Read/Write registers, test&set, CAS, ...
- Data structures: lists, queues, hashmaps, ...
- Message-passing interfaces
From now on, everything is an object:

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

From now on, everything is an object:

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

“Can we implement the object $B$ using the objects $A_1, \ldots, A_k$?”
Objects

From now on, everything is an object:

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

"Can we implement the object $B$ using the objects $A_1, \ldots, A_k$?"

→ How do we specify a concurrent object?
Objects

From now on, everything is an object:
- Hardware: Read/Write registers, test&set, CAS, ...
- Data structures: lists, queues, hashmaps, ...
- Message-passing interfaces
- Immediate-snapshot, consensus, set-agreement, ...

“Can we implement the object $B$ using the objects $A_1, \ldots, A_k$?”

→ How do we specify a concurrent object?
→ What does it mean to implement an object?
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).

![Diagram showing concurrent execution of push and pop operations]
Concurrent specifications

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

\[
T = \cdot \text{push},0 \cdot \text{OK} \cdot \text{push},2 \cdot \text{pop} \cdot \text{r}^2 \cdot \text{pop} \cdot \text{r}^\text{OK} \cdot \text{r}^0
\]

**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).

![Diagram of concurrent specifications with push 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 $\sigma \subseteq \mathcal{T}$. 
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 $\sigma$. 

$$
\text{Lin}(\sigma) = \{ T \text{ concurrent trace} \mid T \text{ is linearizable w.r.t. } \sigma \} 
$$
Linearizability (Herlihy & Wing, 1990)

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

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

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

\[ \begin{align*}
P_0 & \quad [\quad ] \quad [\quad ] \\
P_1 & \quad [\quad ] \quad [\quad ] \\
P_2 & \quad [\quad ] \quad [\quad ]
\end{align*} \]
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 $\sigma$. 
**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.

![Diagram of set-linearizability](image)
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)

- Can specify every task!
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

Set-linearizability

Linearizability

stack
queue
test&set

immediate snapshot
exchanger
set-agreement
Overview

Concurrent specifications

Set-linearizability

Linearizability

- stack
- queue
- test&set

Immediate snapshot
- exchanger
- set-agreement

Validity
- write-snapshot
Overview

Concurrent specifications

Interval-linearizability

Set-linearizability

Linearizability

stack
queue
test&set

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

This talk: add a few more "desirable" properties

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
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 $x$,
We write \textbf{ConcSpec} for the set of concurrent specifications \( \sigma \subseteq T \) 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 \( t \cdot r^x_i \in \sigma \),

(5) \( \sigma \) has the expansion property.
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:

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

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,

\[
P_0 \rightarrow [a, b] \rightarrow [e, f] \rightarrow [j, k] \\
P_1 \rightarrow [c, d] \rightarrow [g, i] \\
P_2 \rightarrow [a, b] \rightarrow [e, f] \rightarrow [j, k] 
\]
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:

```
exchange(0)  FAIL
P0  [    ]  exchange(1)  FAIL
      [    ]  exchange(2)  FAIL
P1  [    ]  P2  [    ]

Hence, according to the expansion property, exchange(0)  FAIL exchange(1)  FAIL exchange(2)  FAIL should be considered correct too!
```
Example: the Exchanger object (2)

The following execution is correct:

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

Hence, according to the expansion property,

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

should be considered correct too!
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,

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

$$
P_0 \quad [\text{---}] \quad [\text{---}] \quad [\text{---}] \quad \rightarrow
$$

$$
P_1 \quad [\text{---}] \quad [\text{---}] \quad [\text{---}] \quad \rightarrow
$$

$$
P_2 \quad [\text{---}] \quad [\text{---}] \quad [\text{---}] \quad \rightarrow
$$
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,

\[ P_0 \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \]
\[ P_1 \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \]
\[ P_2 \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \]

Then any trace obtained by expanding it is still linearizable.

\[ P_0 \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \]
\[ P_1 \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \]
\[ P_2 \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \rightarrow [\quad ] \]

\[ \square \]
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,

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

Then any trace obtained by expanding it is still linearizable.

\[
\begin{align*}
P_0 & \quad [\bullet] \quad [\bullet] \quad [\bullet] \quad [\bullet] \\
P_1 & \quad [\bullet] \quad [\bullet] \quad [\bullet] \quad [\bullet] \\
P_2 & \quad [\bullet] \quad [\bullet] \quad [\bullet] \quad [\bullet] \\
\end{align*}
\]
The maps $\text{Lin}$ and $U$ form a Galois connection: for every $\sigma \in \text{SeqSpec}$ and $\tau \in \text{ConcSpec}$,

$$\text{Lin}(\sigma) \subseteq \tau \iff \sigma \subseteq U(\tau).$$
A Galois connection

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

$\text{Lin}(\sigma) \subseteq \tau \iff \sigma \subseteq U(\tau)$
A Galois connection

Theorem

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

\[
\text{Lin}(\sigma) \subseteq \tau \iff \sigma \subseteq U(\tau)
\]
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,

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

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.
We fix a set \( \{A_1, \ldots, A_k\} \) of shared objects, along with their concurrent specifications.
A computational model

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.

A protocol \( \mathcal{P} \) consists of one program for each process.

```plaintext
consensus(v) {
    b.write(v);
    x := t.test&set();
    if (x = 0)
        return v;
    else
        v' := a.read();
        return v';
}
```
The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $J_P \in \text{ConcSpec}$.

Theorem
For any wait-free protocol $P$, $J_P \in \text{ConcSpec}$.
The semantics $J_{P_0 P_1 K}$ of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $J_{P_0 P_1 K} \subseteq S$.

Theorem
For any wait-free protocol $P_0$,$P_1$,$J_{P_0 P_1 K} \in \text{ConcSpec}$. 

A computational model (2)

The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $J_P^K \subseteq S$.

Theorem For any wait-free protocol $P$, $J_P^K \in \text{ConcSpec}$.
A computational model (2)

The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification \( S \in \text{ConcSpec} \) if \( \mathcal{J}_P^K \subseteq S \).

Theorem

For any wait-free protocol \( P \), \( \mathcal{J}_P^K \in \text{ConcSpec} \).
The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $J_P \subseteq S$.

Theorem
For any wait-free protocol $P$, $J_P \in \text{ConcSpec}$. 

A computational model (2)
The semantics of a protocol is the set of execution traces it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $\mathcal{J}_p \subseteq S$.

Theorem
For any wait-free protocol $P$, $\mathcal{J}_p \in \text{ConcSpec}$.
A computational model (2)

The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification \( S \in \text{ConcSpec} \) if \( J_{P_k} \subseteq S \).

Theorem
For any wait-free protocol \( P \), \( J_{P_k} \in \text{ConcSpec} \).
The semantics of a protocol is the set of execution traces that it can produce.

It implements an object specification $S \in \text{ConcSpec}$ if $J_P^K \subseteq S$.

Theorem
For any wait-free protocol $P$, $J_P^K \in \text{ConcSpec}$. 
The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $\mathcal{J}_P \subseteq S$.

Theorem

For any wait-free protocol $P$, $\mathcal{J}_P \in \text{ConcSpec}$. 

A computational model (2)
The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $J_P \subseteq S$.

Theorem

For any wait-free protocol $P$, $J_P \in \text{ConcSpec}$.
The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $J_{P^K} \subseteq S$.

Theorem
For any wait-free protocol $P$, $J_{P^K} \in \text{ConcSpec}$.
The semantics of a protocol is the set of execution traces that it can produce.

It implements an object specification $S \in \text{ConcSpec}$ if $J_P \subseteq S$.

Theorem
For any wait-free protocol $P$, $J_P \in \text{ConcSpec}$.
A computational model (2)

The semantics of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $J_{P^K} \subseteq S$.

Theorem
For any wait-free protocol $P$, $J_{P^K} \in \text{ConcSpec}$.
A computational model (2)

The semantics $\semantics{P}$ of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \ConcSpec$ if $\semantics{P} \subseteq S$. 
A computational model (2)

The semantics $\llbracket \mathcal{P} \rrbracket$ of a protocol is the set of execution traces that it can produce. It implements an object specification $S \in \text{ConcSpec}$ if $\llbracket \mathcal{P} \rrbracket \subseteq S$.

**Theorem**

*For any wait-free protocol $\mathcal{P}$, $\llbracket \mathcal{P} \rrbracket \in \text{ConcSpec}$.***
Asynchronous computability theorem

- Tasks are now a particular kind of concurrent object:

  Tasks $\xleftarrow{}$ ConcSpec
Asynchronous computability theorem

- Tasks are now a particular kind of concurrent object:

  \[ \text{Tasks} \leftrightarrow \text{ConcSpec} \]

- Define the protocol complex for a given protocol \( \mathcal{P} \):

  \[ \text{Views of process } P_i \approx \text{States of the CFG of its program} \]
Asynchronous computability theorem

- Tasks are now a particular kind of concurrent object:

  \[ \text{Tasks} \leftrightarrow \text{ConcSpec} \]

- Define the protocol complex for a given protocol \( P \):

  \[ \text{Views of process } P_i \sim \text{States of the CFG of its program} \]

**Theorem**

A wait-free protocol implements a task if and only if there exists a decision map from the protocol complex to the output complex that makes the diagram commute.
Future work

- Get rid of the wait-free requirement:
  - $t$-resilient protocols
  - allows to model objects such as semaphores, barriers, ...
Future work

- Get rid of the wait-free requirement:
  - \( t \)-resilient protocols
  - allows to model objects such as semaphores, barriers, \ldots

- All tasks are objects, but not all objects are tasks: find a topological characterization for the other objects.
  \(\rightarrow\) Rajsbaum et al. proposed a notion of refined tasks.
Future work

- Get rid of the wait-free requirement:
  - $t$-resilient protocols
  - allows to model objects such as semaphores, barriers, …

- All tasks are objects, but not all objects are tasks: find a topological characterization for the other objects.
  - Rajsbaum et al. proposed a notion of refined tasks.

- Game semantics perspective
  - our notion of implementation looks like the composition of strategies
  - can we characterize the immediate-snapshot strategies, and deduce impossibility results from it?
Thanks!