Geometric Semantics for Asynchronous Computability

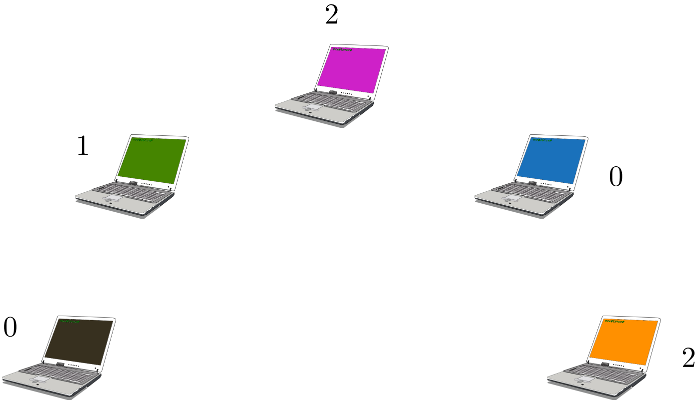
Jérémy Ledent

PhD defense – École Polytechnique

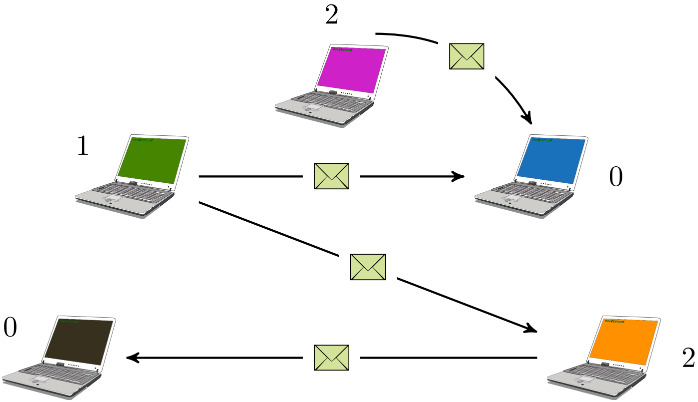
December 12, 2019

The distributed computing setting

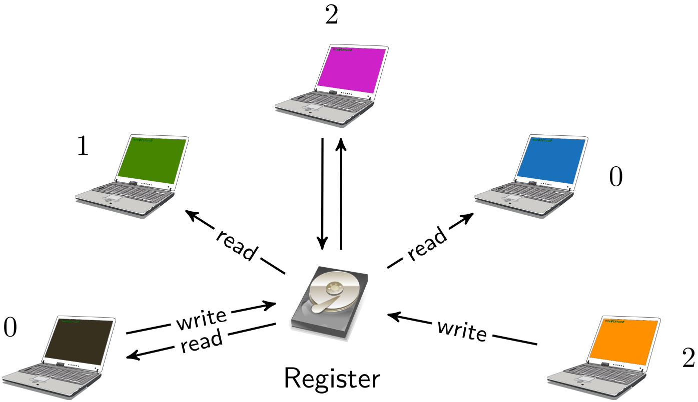
The distributed computing setting



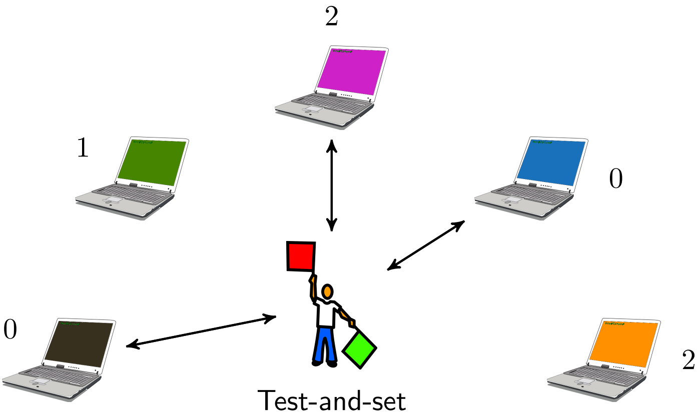
The distributed computing setting



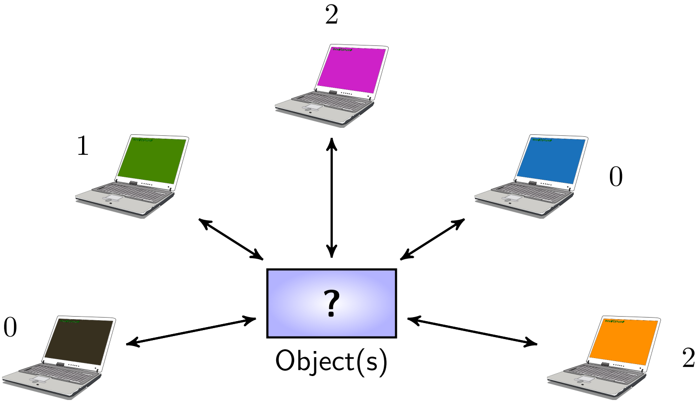
The distributed computing setting



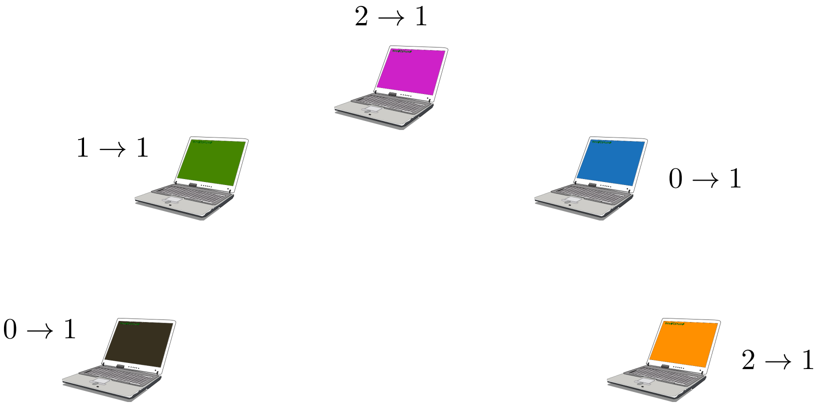
The distributed computing setting



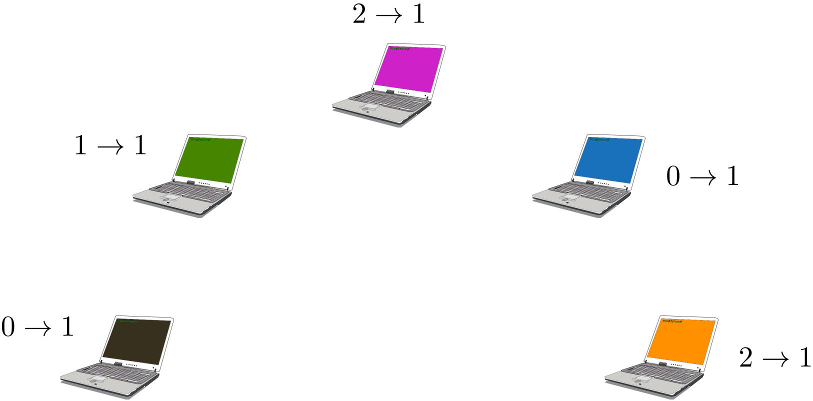
The distributed computing setting



The distributed computing setting



The distributed computing setting



Task specification: $(0, 1, 2, 0, 2) \to (1, 1, 1, 1, 1)$ ✓ or ✗ ?

Asynchronous computability

a.k.a. Fault-tolerant distributed computing

Goal: Study which concurrent tasks are solvable in various computational models.

Asynchronous computability

a.k.a. Fault-tolerant distributed computing

Goal: Study which concurrent tasks are solvable in various computational models.

- Compare the strength of objects.

Asynchronous computability

a.k.a. Fault-tolerant distributed computing

Goal: Study which concurrent tasks are solvable in various computational models.

- Compare the strength of objects.
- Compare the difficulty of solving tasks.

Asynchronous computability

a.k.a. Fault-tolerant distributed computing

Goal: Study which concurrent tasks are solvable in various computational models.

- Compare the strength of objects.
- Compare the difficulty of solving tasks.
- Failures?

Asynchronous computability

a.k.a. Fault-tolerant distributed computing

Goal: Study which concurrent tasks are solvable in various computational models.

- Compare the strength of objects.
- Compare the difficulty of solving tasks.
- Failures? $\rightarrow$ Assume protocols are wait-free / t-resilient / etc.

Asynchronous computability

a.k.a. Fault-tolerant distributed computing

Goal: Study which concurrent tasks are solvable in various computational models.

- Compare the strength of objects.
- Compare the difficulty of solving tasks.
- Failures? $\rightarrow$ Assume protocols are wait-free / t-resilient / etc.
- Synchrony vs Asynchrony

Asynchronous computability

a.k.a. Fault-tolerant distributed computing

Goal: Study which concurrent tasks are solvable in various computational models.

- Compare the strength of objects.
- Compare the difficulty of solving tasks.
- Failures? $\rightarrow$ Assume protocols are wait-free / t-resilient / etc.
- Synchrony vs Asynchrony

Our assumptions: Asynchronous and wait-free.

A topological approach

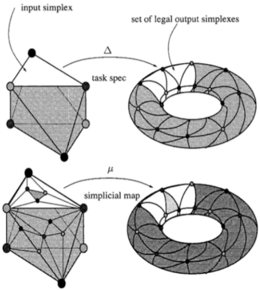


FIG. 13. Asynchronous computability theorem.

THEOREM 3.1 (ASYNCHRONOUS COMPUTABILITY THEOREM). *A decision task* $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ *has a wait-free protocol using read-write memory if and only if there exists a chromatic subdivision* σ *of* $\mathcal{I}$ *and a color-preserving simplicial map*

$$\mu: \sigma(\mathcal{I}) \to \mathcal{O}$$

such that for each simplex S *in* $\sigma(\mathcal{I})$, $\mu(S) \in \Delta(carrier(S, \mathcal{I}))$.

Herlihy and Shavit, 1999
2004 Gödel prize

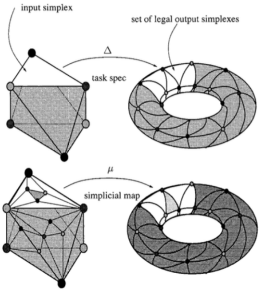


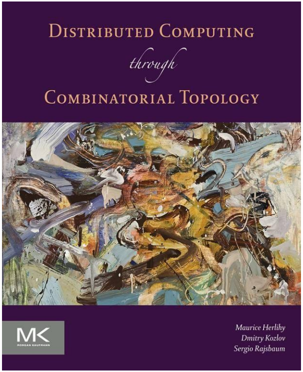
FIG. 13. Asynchronous computability theorem.

THEOREM 3.1 (ASYNCHRONOUS COMPUTABILITY THEOREM). *A decision task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ has a wait-free protocol using read-write memory if and only if there exists a chromatic subdivision σ of $\mathcal{I}$ and a color-preserving simplicial map*

$$\mu: \sigma(\mathcal{I}) \rightarrow \mathcal{O}$$

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

Herlihy and Shavit, 1999
2004 Gödel prize

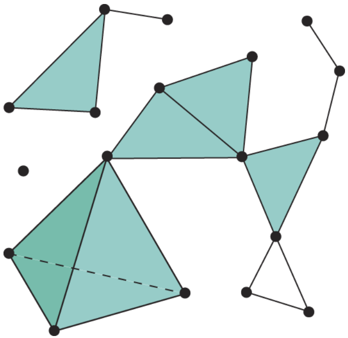


Herlihy, Kozlov, Rajsbaum, 2013

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 $Y \in S$).



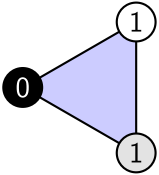
Example: binary input complex for 3 processes

- 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.

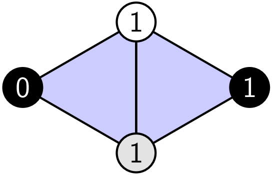
In the picture below, the three process names are represented as the colors black, grey, white:



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.

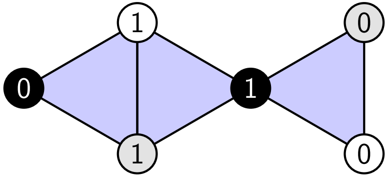
In the picture below, the three process names are represented as the colors black, grey, white:



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.

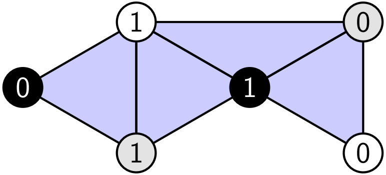
In the picture below, the three process names are represented as the colors black, grey, white:



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.

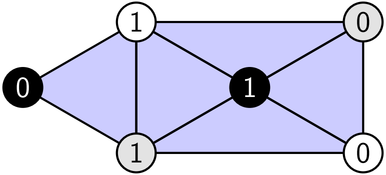
In the picture below, the three process names are represented as the colors black, grey, white:



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.

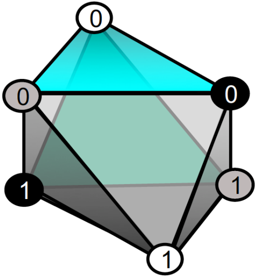
In the picture below, the three process names are represented as the colors black, grey, white:



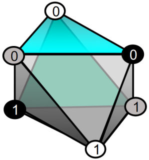
Example: binary input complex for 3 processes

- 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:

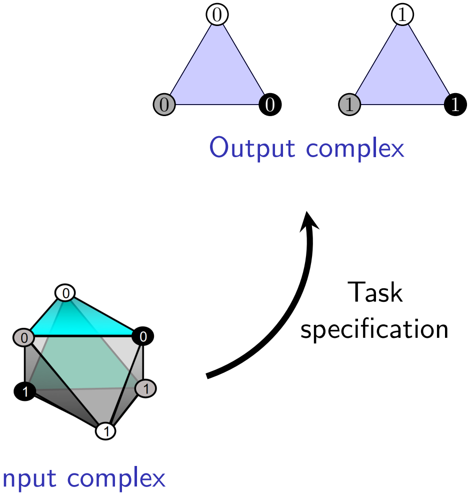


Topological characterization of task solvability

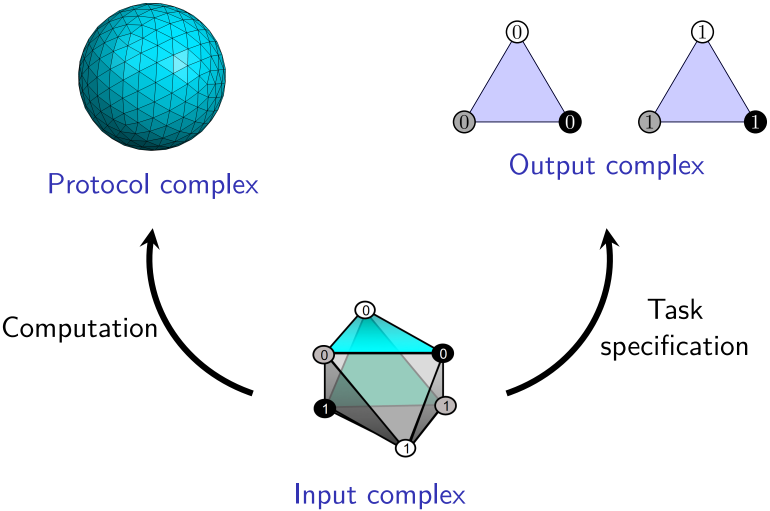


Input complex

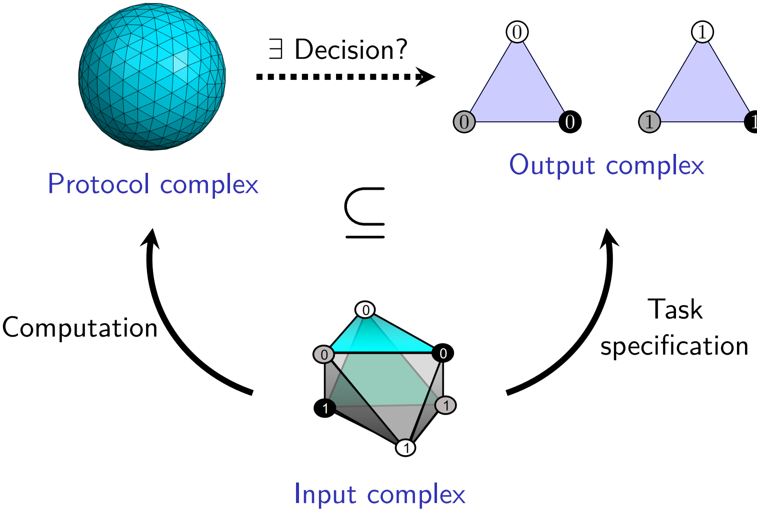
Topological characterization of task solvability



Topological characterization of task solvability



Topological characterization of task solvability



Part I: Operational Semantics

Asynchronous Computability Theorem (ACT)

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 the protocol complex into the output complex such that [...].

Asynchronous Computability Theorem (ACT)

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 the protocol complex into the output complex such that [...].

What if:

- we replace "wait-free" by "t-resilient"?

Asynchronous Computability Theorem (ACT)

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 the protocol complex into the output complex such that [...].

What if:

- we replace "wait-free" by "t-resilient"?

  $\longrightarrow$ *Asynchronous Computability Theorems for t-resilient systems*, Saraph, Herlihy, Gafni (DISC 2016).

Asynchronous Computability Theorem (ACT)

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 the protocol complex into the output complex such that [...].

What if:

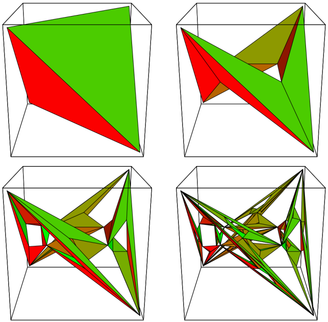
- we replace "wait-free" by "t-resilient"?

  $\longrightarrow$ *Asynchronous Computability Theorems for t-resilient systems*, Saraph, Herlihy, Gafni (DISC 2016).

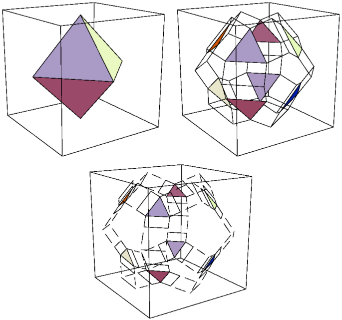
- we use other objects instead of read/write registers?

  $\longrightarrow$ Our goal here.

Protocol complexes for other objects

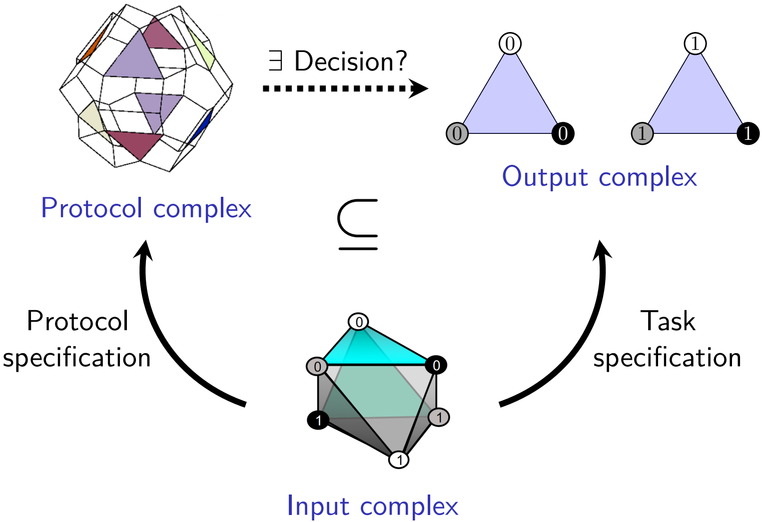


For test-and-set protocols
Herlihy, Rajsbaum, PODC'94



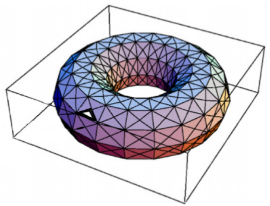
For synchronous message-passing
Herlihy, Rajsbaum, Tuttle, 2001

Topological **definition** of solvability



Benefits and drawbacks

✓ We can prove very general abstract results:



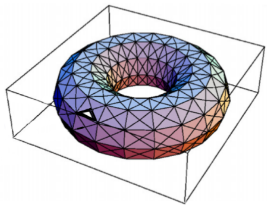
Theorem

Set-agreement is not solvable if the protocol complex is a pseudomanifold.

Herlihy, Kozlov, Rajsbaum (2013)

Benefits and drawbacks

✓ We can prove very general abstract results:



Theorem

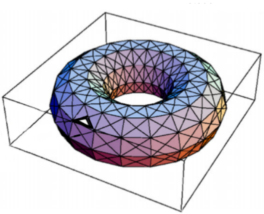
Set-agreement is not solvable if the protocol complex is a pseudomanifold.

Herlihy, Kozlov, Rajsbaum (2013)

✗ How do we know our protocol is correctly modeled?

Benefits and drawbacks

✓ We can prove very general abstract results:



Theorem

Set-agreement is not solvable if the protocol complex is a pseudomanifold.

Herlihy, Kozlov, Rajsbaum (2013)

✗ How do we know our protocol is correctly modeled?

Goal: Give a concrete meaning to "solving a task" using arbitrary objects, and prove that it agrees with the topological definition.

Outline

(1) Define a notion of concurrent object specification which is as general as possible. It should include non-linearizable objects.

Outline

(1) Define a notion of concurrent object specification which is as general as possible. It should include non-linearizable objects.

(2) Define an operational semantics for concurrent processes communicating through arbitrary shared objects.

Outline

(1) Define a notion of concurrent object specification which is as general as possible. It should include non-linearizable objects.

(2) Define an operational semantics for concurrent processes communicating through arbitrary shared objects.

(3) Define the protocol complex associated to a given protocol.

Outline

(1) Define a notion of concurrent object specification which is as general as possible. It should include non-linearizable objects.

(2) Define an operational semantics for concurrent processes communicating through arbitrary shared objects.

(3) Define the protocol complex associated to a given protocol.

(4) Prove the following:

Generalized ACT

A wait-free protocol *solves* a task if and only if there is a simplicial map from the protocol complex to the output complex which is carried by the task specification.

A Sound Foundation for the Topological Approach to Task Solvability. L., Mimram (CONCUR'19)

Outline

(1) Define a notion of concurrent object specification which is as general as possible. It should include non-linearizable objects.

(2) Define an operational semantics for concurrent processes communicating through arbitrary shared objects.

(3) Define the protocol complex associated to a given protocol.

(4) Prove the following:

Generalized ACT

A wait-free protocol *solves* a task if and only if there is a simplicial map from the protocol complex to the output complex which is carried by the task specification.

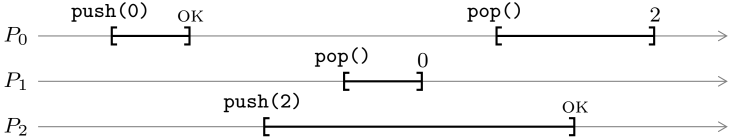
A Sound Foundation for the Topological Approach to Task Solvability. L., Mimram (CONCUR'19)

Concurrent specifications

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

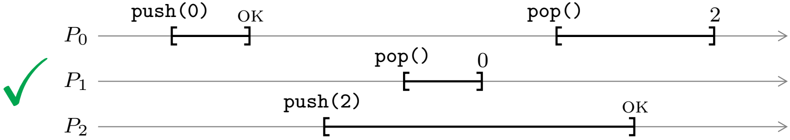
Concurrent specifications

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



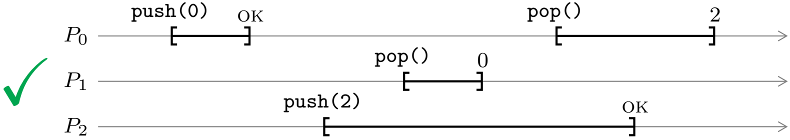
Concurrent specifications

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



Concurrent specifications

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

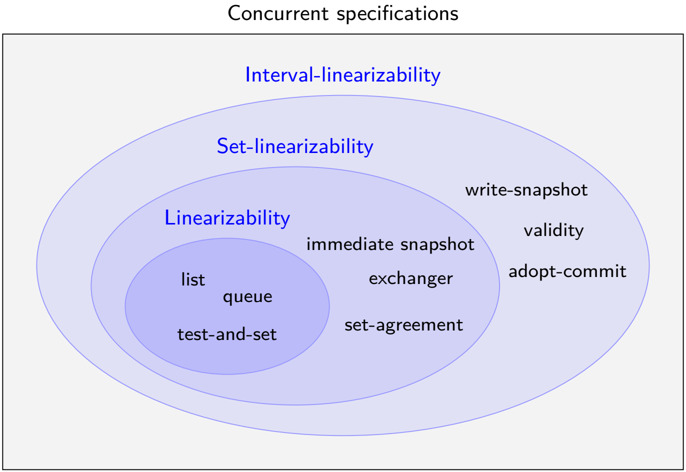


Write $\mathcal{T}$ for the set of all execution traces.

Definition

A *concurrent specification* is a subset $\sigma \subseteq \mathcal{T}$.

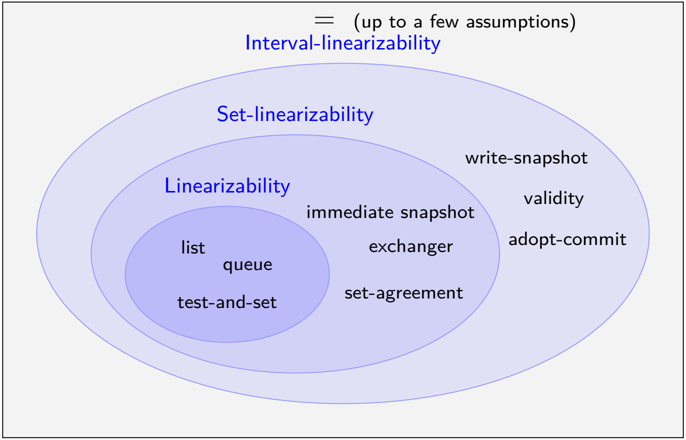
Concurrent specifications (2)



Concurrent Specifications Beyond Linearizability. Goubault, L., Mimram (OPODIS'18)

Concurrent specifications (2)

Concurrent specifications



Concurrent Specifications Beyond Linearizability. Goubault, L., Mimram (OPODIS'18)

Tasks vs Objects

Recall that a task for n processes is a relation $\Theta \subseteq \mathsf{Val}^n \times \mathsf{Val}^n$.

Tasks are less expressive than objects:

Tasks vs Objects

Recall that a task for n processes is a relation $\Theta \subseteq \mathsf{Val}^n \times \mathsf{Val}^n$.

Tasks are less expressive than objects:

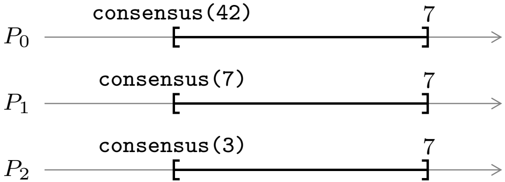
- A task is one-shot (it can be used only once),

Tasks vs Objects

Recall that a task for n processes is a relation $\Theta \subseteq \mathsf{Val}^n \times \mathsf{Val}^n$.

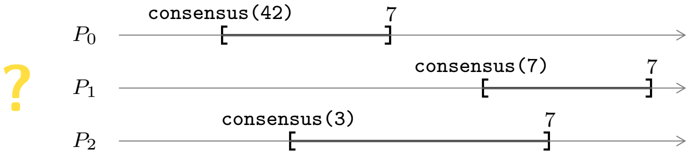
Tasks are less expressive than objects:

- A task is one-shot (it can be used only once),
- A task only specifies traces of the following form:



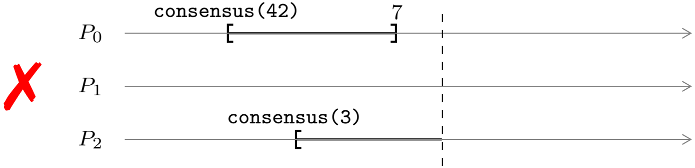
Turning a task into an object

How do we specify a consensus object?



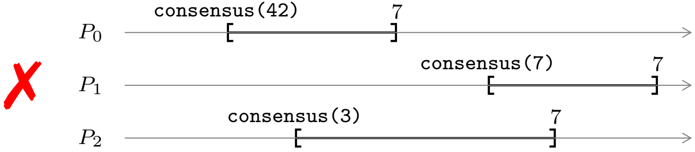
Turning a task into an object

How do we specify a consensus object?



Turning a task into an object

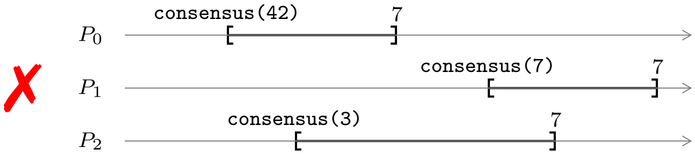
How do we specify a consensus object?



This defines a function G : Tasks $\rightarrow$ Objects.

Turning a task into an object

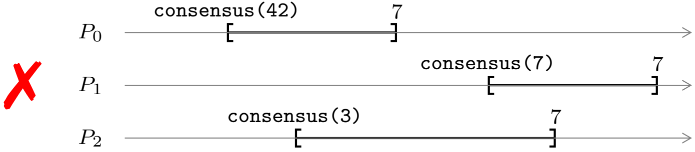
How do we specify a consensus object?



This defines a function G : Tasks $\to$ Objects.
There is also an obvious function F : Objects $\to$ Tasks.

Turning a task into an object

How do we specify a consensus object?



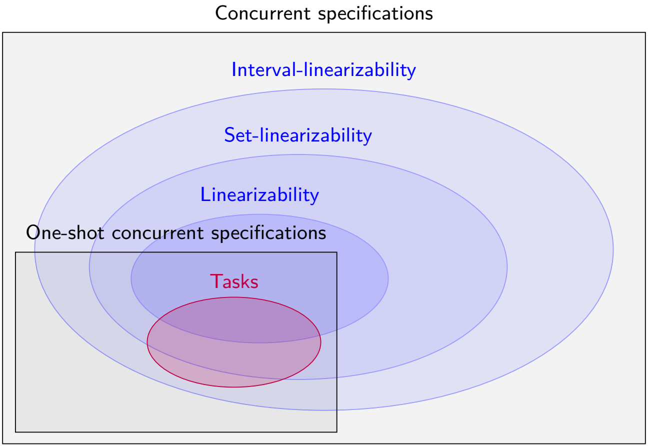
This defines a function G : Tasks $\to$ Objects.
There is also an obvious function F : Objects $\to$ Tasks.

Theorem

The functions F and G form a Galois connection:

$$\sigma \subseteq G(\Theta) \iff F(\sigma) \subseteq \Theta$$

Tasks vs Objects (2)



Unifying Concurrent Objects and Distributed Tasks: Interval-Linearizability.

Castañeda, Rajsbaum, Raynal (2018).

Generalized Asynchronous Computability Theorem

Solving a task Θ simply means implementing the object $F(\Theta)$.

Generalized Asynchronous Computability Theorem

Solving a task Θ simply means implementing the object $F(\Theta)$.

Theorem (L., Mimram – CONCUR'19)

A wait-free protocol solves a task if and only if there is a simplicial map from the protocol complex to the output complex which is carried by the task specification.

Work in progress

- **Compositionality:**

  "*If A solves B and B solves C, then A solves C.*"

$\longrightarrow$ Links with game semantics.

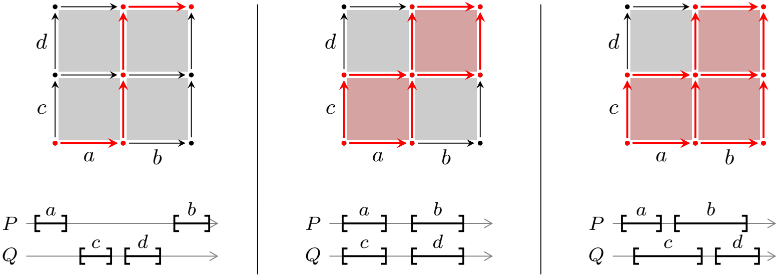
Work in progress

- **Compositionality:**

  "If A solves B and B solves C, then A solves C."

  $\longrightarrow$ Links with game semantics.

- **Directed Topology:**



Part II: Geometric Models for Epistemic Logic

Multi-agent epistemic logic

Epistemic logic is the modal logic of knowledge.

Let $\mathcal{A}$ be a finite set of *agents* and At a set of *atomic propositions*.
The syntax of formulas is:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_a\,\varphi \qquad\qquad p \in \mathsf{At},\ a \in \mathcal{A}$$

$K_a\,\varphi$ is read *"a knows φ"*.

Multi-agent epistemic logic

Epistemic logic is the modal logic of knowledge.

Let $\mathcal{A}$ be a finite set of *agents* and At a set of *atomic propositions*. The syntax of formulas is:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_a\,\varphi \mid C_B\,\varphi \qquad p \in \mathsf{At},\ a \in \mathcal{A},\ B \subseteq \mathcal{A}$$

$K_a\,\varphi$ is read *"a knows φ"*.

Common knowledge:

$$C_B\,\varphi \;\equiv\; \bigwedge_{\substack{n\in\mathbb{N}\\ a_1,\dots,a_n\in B}} K_{a_1}\dots K_{a_n}\,\varphi$$

Kripke models

A Kripke model is a tuple $M = \langle W, \sim, L \rangle$, where:

- W is a set of *worlds*
- For every $a \in \mathcal{A}$, $\sim_a \subseteq W \times W$ is an equivalence relation on W
- $L : W \to \mathscr{P}(\mathsf{At})$

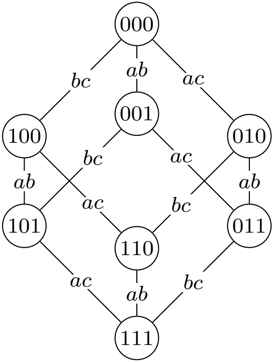
Kripke models

A Kripke model is a tuple $M = \langle W, \sim, L \rangle$, where:

- W is a set of *worlds*
- For every $a \in \mathcal{A}$, $\sim_a \subseteq W \times W$ is an equivalence relation on W
- $L : W \to \mathscr{P}(\mathsf{At})$

Example: three agents with binary inputs.

- a, b, c are agents.
- $w \sim_a w'$ is represented as an a-labeled edge between w and w'.
- 101: input values of a, b, c, in that order.

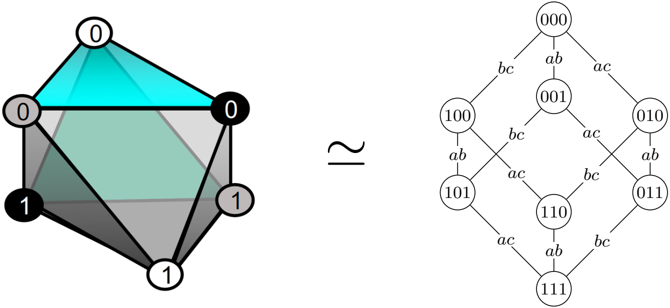


Semantics of epistemic logic formulas

Let $M = \langle W, \sim, L \rangle$ be a Kripke model and $x \in W$ a world of M. We define the truth of a formula φ in x, written $M, x \models \varphi$, by induction on φ:

$$
\begin{array}{lll}
M, x \models p & \text{iff} & p \in L(x) \\
M, x \models \neg\varphi & \text{iff} & M, x \not\models \varphi \\
M, x \models \varphi \wedge \psi & \text{iff} & M, x \models \varphi \text{ and } M, x \models \psi \\
M, x \models K_a\,\varphi & \text{iff} & \text{for all } y \in W, x \sim_a y \text{ implies } M, y \models \varphi
\end{array}
$$

An equivalence of categories



Theorem

The category of labeled pure chromatic simplicial complexes is equivalent to the category of local proper Kripke models.

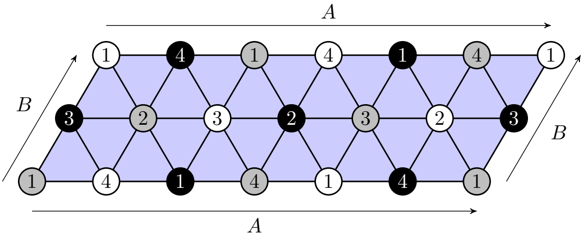
A Simplicial Complex Model for Dynamic Epistemic Logic, Goubault, L., Rajsbaum (GandALF'18)

Example: card game

Consider the following situation: *there are three agents and a deck of four cards* $\{0, 1, 2, 3\}$. *Each agent is given a card at random, and the remaining card is kept hidden.*

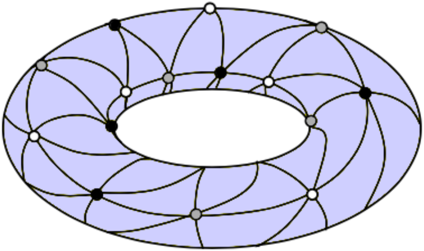
Example: card game

Consider the following situation: *there are three agents and a deck of four cards* $\{0,1,2,3\}$*. Each agent is given a card at random, and the remaining card is kept hidden.*



Example: card game

Consider the following situation: *there are three agents and a deck of four cards* $\{0, 1, 2, 3\}$*. Each agent is given a card at random, and the remaining card is kept hidden.*



Dynamic Epistemic Logic (DEL)

Syntax:
Let $\mathcal{A}$ be a finite set of *agents* and At a set of *atomic propositions*.
The syntax of formulas is:

$$\begin{array}{lcl} \varphi & ::= & p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_a\,\varphi \mid [\alpha]\,\varphi \\ \alpha & ::= & \text{“action”} \end{array}$$

$[\alpha]\,\varphi$ intuitively means *“φ will be true after the action α occurs”*.

Dynamic Epistemic Logic (DEL)

Syntax:

Let $\mathcal{A}$ be a finite set of *agents* and At a set of *atomic propositions*.
The syntax of formulas is:

$$\begin{array}{lcl} \varphi & ::= & p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_a\,\varphi \mid [\alpha]\,\varphi \\ \alpha & ::= & \text{“action”} \end{array}$$

$[\alpha]\,\varphi$ intuitively means *“φ will be true after the action α occurs”*.

Semantics:

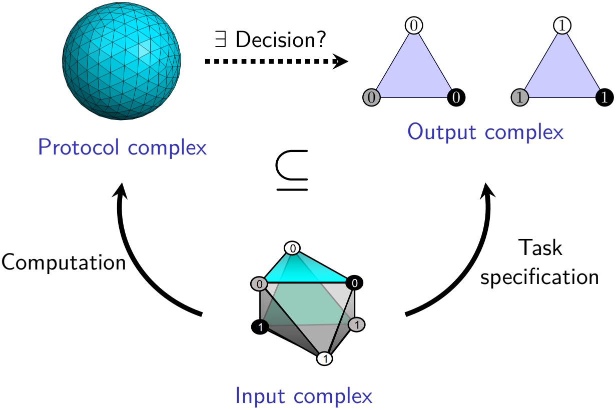
$$\begin{array}{lll} M,x \models p & \text{iff} & p \in L(x) \\ M,x \models \neg\varphi & \text{iff} & M,x \not\models \varphi \\ M,x \models \varphi \wedge \psi & \text{iff} & M,x \models \varphi \text{ and } M,x \models \psi \\ M,x \models K_a\,\varphi & \text{iff} & \text{for all } y \in W, x \sim_a y \text{ implies } M,y \models \varphi \\ M,x \models [\alpha]\,\varphi & \text{iff} & M[\alpha],x[\alpha] \models \varphi \end{array}$$

Actions in distributed computing

Motto: the product-update construction $M[\alpha]$ plays the same role as carrier maps.

Actions in distributed computing

Motto: the product-update construction $M[\alpha]$ plays the same role as carrier maps.



Actions in distributed computing

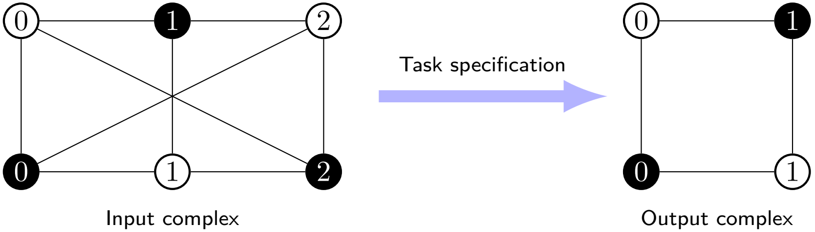
Motto: the product-update construction $M[\alpha]$ plays the same role as carrier maps.

$$
\begin{tikzcd}
\begin{array}{c}\mathcal{M}[\mathcal{P}] \\ \text{Protocol model}\end{array} \arrow[rr, dashed, "\exists\ \text{Decision?}"] \arrow[dr, bend right, "\pi_M"'] & & \begin{array}{c}\mathcal{M}[\mathcal{T}] \\ \text{Output model}\end{array} \arrow[dl, bend left, "\pi_M"] \\
& \begin{array}{c}\mathcal{M} \\ \text{Input model}\end{array} &
\end{tikzcd}
$$

Case study: the Equality Negation task

(*Nondeterministic wait-free hierarchies are not robust*, Lo and Hadzilacos, 2000)

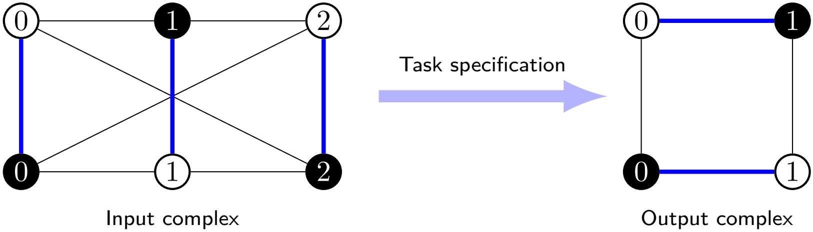
- Two processes P, Q (represented in black and white).
- Three possible inputs values $i_P, i_Q \in \{0, 1, 2\}$.
- Binary decision values $d_P, d_Q \in \{0, 1\}$.
- **Goal:** $i_P = i_Q \iff d_P \neq d_Q$.



Case study: the Equality Negation task

(*Nondeterministic wait-free hierarchies are not robust*, Lo and Hadzilacos, 2000)

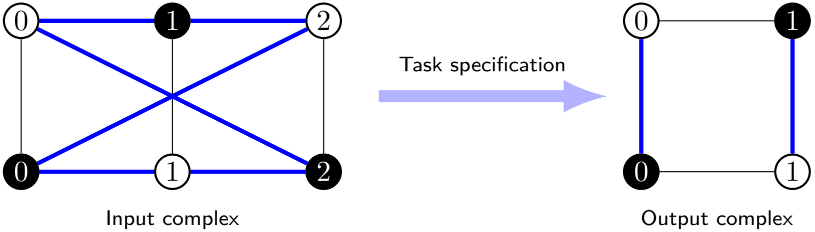
- Two processes P, Q (represented in black and white).
- Three possible inputs values $i_P, i_Q \in \{0, 1, 2\}$.
- Binary decision values $d_P, d_Q \in \{0, 1\}$.
- **Goal:** $i_P = i_Q \iff d_P \neq d_Q$.



Case study: the Equality Negation task

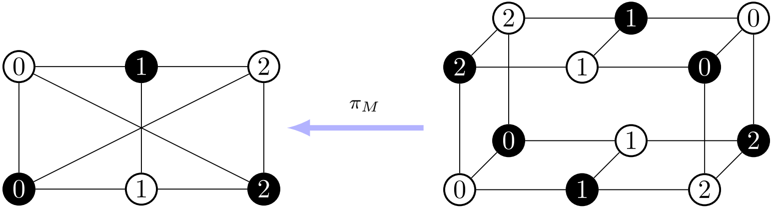
(*Nondeterministic wait-free hierarchies are not robust*, Lo and Hadzilacos, 2000)

- Two processes P, Q (represented in black and white).
- Three possible inputs values $i_P, i_Q \in \{0, 1, 2\}$.
- Binary decision values $d_P, d_Q \in \{0, 1\}$.
- **Goal:** $i_P = i_Q \iff d_P \neq d_Q$.



Case study: the Equality Negation task (2)

Using DEL, the task is modeled as follows:

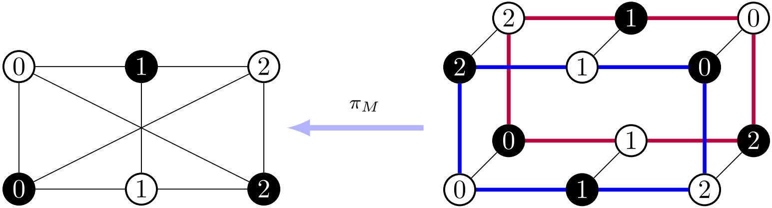


Input model M

Output model $M[T]$

Case study: the Equality Negation task (2)

Using DEL, the task is modeled as follows:

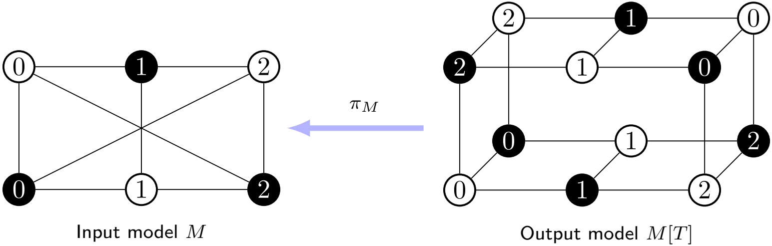


Input model M Output model $M[T]$

In blue: processes decide 0
In red: processes decide 1

Case study: the Equality Negation task (2)

Using DEL, the task is modeled as follows:



Intuitively, $M[T]$ describes the knowledge that the processes should acquire in order to solve the task.

Case study: the Equality Negation task (3)

We have two papers about this task:

- *A Dynamic Epistemic Logic Analysis of the Equality Negation Task*, Goubault, Lazić, L., Rajsbaum (DaLi'19).
  $\longrightarrow$ The reason why EN is not solvable cannot be expressed in the language of epistemic logic.

Case study: the Equality Negation task (3)

We have two papers about this task:

- *A Dynamic Epistemic Logic Analysis of the Equality Negation Task*, Goubault, Lazić, L., Rajsbaum (DaLi'19).
  $\longrightarrow$ The reason why EN is not solvable cannot be expressed in the language of epistemic logic.
- *Wait-free solvability of Equality Negation Tasks*, Goubault, Lazić, L., Rajsbaum (DISC'19).
  $\longrightarrow$ Extend the task to n processes and study its solvability.

Conclusion

A link between epistemic logic and distributed computing

- For computer scientists: we can now understand the abstract topological proofs of impossibility in terms of *knowledge*.

Conclusion

A link between epistemic logic and distributed computing

- For computer scientists: we can now understand the abstract topological proofs of impossibility in terms of *knowledge*.
- For logicians: Kripke models contain geometric information that can be used to reason about knowledge.

Conclusion

A link between epistemic logic and distributed computing

- For computer scientists: we can now understand the abstract topological proofs of impossibility in terms of *knowledge*.
- For logicians: Kripke models contain geometric information that can be used to reason about knowledge.

Future work:

- Simplicial complexes that are not pure.
  $\rightarrow$ variable number of agents

Conclusion

A link between epistemic logic and distributed computing

- For computer scientists: we can now understand the abstract topological proofs of impossibility in terms of *knowledge*.
- For logicians: Kripke models contain geometric information that can be used to reason about knowledge.

Future work:

- Simplicial complexes that are not pure.
  $\rightarrow$ variable number of agents
- Model other epistemic notions: belief, distributed knowledge.

Conclusion

A link between epistemic logic and distributed computing

- For computer scientists: we can now understand the abstract topological proofs of impossibility in terms of *knowledge*.
- For logicians: Kripke models contain geometric information that can be used to reason about knowledge.

Future work:

- Simplicial complexes that are not pure.
  $\rightarrow$ variable number of agents
- Model other epistemic notions: belief, distributed knowledge.
- Interpret bisimulation between models topologically.

Thanks!