POINTS OF VIEW
ON
ASYNCHRONOUS
COMPUTABILITY

SAMUEL MIMRAM
École Polytechnique / PPS
(with É. Goubault and C. Tasson)

WIP 2016
April 27, 2016
Asynchronous computability

In the 90s, Herlihy et al. have obtained major results on asynchronous computability.

▶ What can a bunch of processes computing in parallel can compute in the presence of failures?

▶ For instance, they show that the consensus cannot be solved.

▶ Their proofs uses geometric arguments, they construct a geometric object corresponding to the possible states and characterize those which can occur and their properties.

▶ obtain impossibility results from the fact that some maps should preserve \((n-)\)connectivity

▶ The devil lies in the details.
Unifying points of view

Here, we unify different points of view on executions:

**protocol complex**
[Herlihy, ...]

\[ \langle u_i, s_i \mid u_i u_j = u_j u_i, s_i s_j = s_j s_i \rangle \]

**partially commutative traces**

**geometric semantics**
[Goubault, ...]

**interval orders**
ASYNCHRONOUS PROTOCOLS AND TASKS
Asynchronous protocols

We consider here a model with \( n \) processes \( P_i \):
- each process has a local memory cell
- there is a global memory with \( n \) cells

\[ P_0 \quad P_1 \quad \cdots \quad P_{n-1} \]

local mem.

global mem.
Asynchronous protocols

We consider here a model with $n$ processes $P_i$:

- each process has a local memory cell
- there is a global memory with $n$ cells

At any instant a process might die and the question is: what we can compute in such a model?

(for this question we are only interested in local memories)

Each process alternatively does “rounds” made of:

- **update**: write in its global memory cell
- **scan**: read the whole global memory and update its local cell

*(immediate snapshot)*
Asynchronous protocols

We consider here a model with $n$ processes $P_i$:

- each process has a local memory cell
- there is a global memory with $n$ cells

![Diagram]

- each process alternatively does “rounds” made of
  - update: write in its global memory cell
  - scan: read the whole global memory and update its local cell (immediate snapshot)
- at any instant a process might die
Asynchronous protocols

We consider here a model with \( n \) processes \( P_i \):

- each process has a local memory cell
- there is a global memory with \( n \) cells

\[
\begin{array}{cccc}
& & & \\
\text{local mem.} & & & \\
& & & \\
\text{global mem.} & & & \\
& & & \\
\end{array}
\]

- each process alternatively does “rounds” made of
  - \textbf{update}: write in its global memory cell
  - \textbf{scan}: read the whole global memory and update its local cell
    \textit{(immediate snapshot)}

- at any instant a process might die
- and the question is: what we can compute in such a model?
  (for this question we are only interested in local memories)
Asynchronous protocols

Note that

- we do not know how the processes will be scheduled
Asynchronous protocols

Note that

- we do not know how the processes will be scheduled
- they might die: we cannot tell if a process is late or dead
Asynchronous protocols

Note that

- we do not know how the processes will be scheduled
- they might die: we cannot tell if a process is late or dead
- the local memory of each process is a partial information about the computation (called its view)
Asynchronous protocols

Note that

▶ we do not know how the processes will be scheduled
▶ they might die: we cannot tell if a process is late or dead
▶ the local memory of each process is a partial information about the computation (called its \textit{view})
▶ we are mostly interested in local memory: it contains the input and output values
Asynchronous protocols

Note that

- we do not know how the processes will be scheduled
- they might die: we cannot tell if a process is late or dead
- the local memory of each process is a partial information about the computation (called its *view*)
- we are mostly interested in local memory: it contains the input and output values
- the initial value for global memory is $\bot$ in every cell
Coherence between views

The main idea here is to introduce a semantics based on the same principles as in (hyper)coherence spaces.

A set $X \subseteq \{(i, x) \mid i \in \mathbb{N}, x \in \mathcal{V}\}$ of local memories (= views) $(i, x) \in \mathbb{N} \times \mathcal{V}$ is coherent when

$$X = \{(i, l_i)\}$$

such that there is an execution leading to a local memory $l_i$. 
Coherence between views

With 3 processes executing one round (update then scan), we typically obtain the following coherence space:
Coherence between views

With 3 processes executing one round (update then scan), we typically obtain the following coherence space:

Notice that it is simply connected.
States

Formally, we suppose fixed a number $n \in \mathbb{N}$ of processes and a set $\mathcal{V}$ of values with

- $\mathcal{I} \subseteq \mathcal{V}$: input values
- $\mathcal{O} \subseteq \mathcal{V}$: output values
- $\bot \in \mathcal{I} \cap \mathcal{O}$: the undefined value / a non-participating process
States

Formally, we suppose fixed a number $n \in \mathbb{N}$ of processes and a set $\mathcal{V}$ of **values** with

- $\mathcal{I} \subseteq \mathcal{V}$: *input values*
- $\mathcal{O} \subseteq \mathcal{V}$: *output values*
- $\bot \in \mathcal{I} \cap \mathcal{O}$: the *undefined value* / a non-participating process

A **state** consists of

- $l \in \mathcal{V}^n$: the *local memories*
- $m \in \mathcal{V}^n$: the *global memories* (always in this order)
States

Formally, we suppose fixed a number \( n \in \mathbb{N} \) of processes and a set \( \mathcal{V} \) of \textbf{values} with

- \( \mathcal{I} \subseteq \mathcal{V} \): \textit{input values}
- \( \mathcal{O} \subseteq \mathcal{V} \): \textit{output values}
- \( \bot \in \mathcal{I} \cap \mathcal{O} \): the \textit{undefined value} / a \textit{non-participating process}

A \textbf{state} consists of

- \( l \in \mathcal{V}^n \): the \textit{local memories}
- \( m \in \mathcal{V}^n \): the \textit{global memories}

(always in this order)

The \textit{standard} initial state has \( l_i = i \) and \( m_i = \bot \).
Protocols

A **protocol** $\pi$ consists of, for $0 \leq i < n$,

- $\pi_{u_i} : \mathcal{V} \rightarrow \mathcal{V}$
  the values it will write in its global memory cell depending on its local memory

- $\pi_{s_i} : \mathcal{V} \times \mathcal{V}^n \rightarrow \mathcal{V}$
  the values it will write in its local memory depending on the values of its local memory and all the global memory cells

such that

- $\pi_{s_i}(x, m) = x$ for $x \in \mathcal{O}$
  once we decide an output we don’t change our mind
The set of possible **actions** is

\[ \mathcal{A} = \{ u_i, s_i, d_i \mid 0 \leq i < n \} \]
Execution traces

The set of possible actions is

\[ \mathcal{A} = \{ u_i, s_i, d_i \mid 0 \leq i < n \} \]

an execution trace is a word in \( \mathcal{A}^* \) which is well-bracketed:

\[ \text{proj}_i(T) \in (u_is_i)^*(\varepsilon + u_id_i) \]
Execution traces

The set of possible actions is

\[ \mathcal{A} = \{ u_i, s_i, d_i \mid 0 \leq i < n \} \]

an execution trace is a word in \( \mathcal{A}^* \) which is well-bracketed:

\[ \text{proj}_i(T) \in (u_is_i)^*(\epsilon + u_id_i) \]

Given a protocol \( \pi \), its semantics

\[ \llbracket T \rrbracket_\pi : \mathcal{V}^n \times \mathcal{V}^n \to \mathcal{V}^n \times \mathcal{V}^n \]

is defined on a trace \( T \in \mathcal{A}^* \) by

\[ \llbracket u_i \rrbracket_\pi(l, m) = (l, m[i \leftarrow \pi_{u_i}(l_i))] \]

\[ \llbracket s_i \rrbracket_\pi(l, m) = (l[i \leftarrow \pi_{s_i}(l_i, m)], m) \]

\[ \llbracket d_i \rrbracket_\pi(l, m) = (l, m) \]
Execution traces

The set of possible actions is

\[ \mathcal{A} = \{ u_i, s_i, d_i \mid 0 \leq i < n \} \]

an execution trace is a word in \( \mathcal{A}^* \) which is well-bracketed:

\[ \text{proj}_i(T) \in (u_is_i)^*(\varepsilon + u_id_i) \]

Given a protocol \( \pi \), its semantics

\[ [T]_{\pi} : \mathcal{V}^n \times \mathcal{V}^n \rightarrow \mathcal{V}^n \times \mathcal{V}^n \]

is defined on a trace \( T \in \mathcal{A}^* \) by

\[ [u_i]_{\pi}(l, m) = (l, m[i \leftarrow \pi_{u_i}(l_i)]) \]
\[ [s_i]_{\pi}(l, m) = (l[i \leftarrow \pi_{s_i}(l, m)], m) \]
\[ [d_i]_{\pi}(l, m) = (l, m) \]
\[ [T \cdot T']_{\pi} = [T']_{\pi} \circ [T]_{\pi} \]
\[ [\varepsilon]_{\pi} = \text{id} \]
Execution traces

The set of possible actions is

\[ \mathcal{A} = \{u_i, s_i \mid 0 \leq i < n\} \]

an execution trace is a word in \( \mathcal{A}^* \) which is well-bracketed:

\[ \text{proj}_i(T) \in (u_is_i)^*(\epsilon + u_id_i) \]

Given a protocol \( \pi \), its semantics

\[
\llbracket T \rrbracket_\pi : \mathcal{V}^n \times \mathcal{V}^n \to \mathcal{V}^n \times \mathcal{V}^n
\]

is defined on a trace \( T \in \mathcal{A}^* \) by

- \( \llbracket u_i \rrbracket_\pi(l, m) = (l, m[i \leftarrow \pi_{u_i}(l_i)]) \)
- \( \llbracket s_i \rrbracket_\pi(l, m) = (l[i \leftarrow \pi_{s_i}(l_i, m)], m) \)

- \( \llbracket T \cdot T' \rrbracket_\pi = \llbracket T' \rrbracket_\pi \circ \llbracket T \rrbracket_\pi \)
- \( \llbracket \epsilon \rrbracket_\pi = \text{id} \)
Execution traces

With two processes executing one round each there are “essentially” three traces:

- $u_0s_0u_1s_1$:
  - $P_0$ does not see what $P_1$ has written
  - $P_1$ sees what $P_0$ has written
Execution traces

With two processes executing one round each there are “essentially” three traces:

- $u_0 s_0 u_1 s_1$:
  - $P_0$ does not see what $P_1$ has written
  - $P_1$ sees what $P_0$ has written

- $u_1 s_1 u_0 s_0$:
  - $P_0$ sees what $P_1$ has written
  - $P_1$ does not see what $P_0$ has written
Execution traces

With two processes executing one round each there are “essentially” three traces:

- \( u_0 s_0 u_1 s_1 \):
  - \( P_0 \) does not see what \( P_1 \) has written
  - \( P_1 \) sees what \( P_0 \) has written

- \( u_1 s_1 u_0 s_0 \):
  - \( P_0 \) sees what \( P_1 \) has written
  - \( P_1 \) does not see what \( P_0 \) has written

- \( u_0 u_1 s_0 s_1 / u_0 u_1 s_1 s_0 / u_1 u_0 s_0 s_1 / u_1 u_0 s_1 s_0 \):
  - \( P_0 \) sees what \( P_1 \) has written
  - \( P_1 \) sees what \( P_0 \) has written
Execution traces

These execution traces can be represented geometrically by

\[ P_0 \quad P_1 \]

\[ u_0 \quad s_0 \quad u_1 \quad s_1 \]

\[ u_0 s_0 u_1 s_1 \]

\[ u_0 u_1 s_0 s_1 \]

\[ u_1 s_1 u_0 s_0 \]

We’ll get back to this representation later on.
Execution traces

These execution traces can be represented geometrically by

We’ll get back to this representation later on.
A task $\theta$ is a relation $\theta \subseteq \mathcal{I}^n \times \mathcal{O}^n$ such that for every $l, l' \in \Theta$

- $l_i = \bot$ if and only if $l'_i = \bot$,
- there exists $l'' \in \mathcal{O}^n$ such that $(l, l'') \in \Theta$ and $(l[i \leftarrow \bot], l''[i \leftarrow \bot]) \in \Theta$.

We write $\text{dom } \Theta$ for the possible input values and $\text{codom } \Theta$ for the possible output values.
The binary consensus

In the **binary consensus** problem each process

- starts with a value in \{0, 1\}
- end with the same value, among the initial values of the alive processes.

For instance, with \( n = 2 \), we have

\[
\Theta = \{(b \perp, b \perp), (\perp b, \perp b), (bb', bb), (b' b, bb) \mid b, b' \in \{0, 1\}\}
\]
The binary consensus

In the **binary consensus** problem each process

- starts with a value in \(\{0, 1\}\)
- end with the same value, among the initial values of the alive processes.

For instance, with \(n = 2\), we have

\[
\Theta = \{(b\perp, b\perp), (\perp b, \perp b), (bb', bb), (b'b, bb) \mid b, b' \in \{0, 1\}\}
\]
The binary quasi-consensus

In the case $n = 2$, we can also consider the **binary quasi-consensus**, which is similar but restricts the output so that it cannot happen that $P_1$ decides 0 and $P_0$ decide 1 at the same time:
The way we draw tasks

Note that

- if \( l \in \text{dom} \Theta \) (the possible input values) then \( l[i \leftarrow \bot] \) also belongs to \( \text{dom} \Theta \)

dom \( \Theta \) can thus be pictured as a *simplicial complex* called the **input complex**:

\[
\begin{array}{c}
\bot 1 \\ 01 \bot \\
0 12 \\
0 \bot 2 \\
\bot \bot 2 \\
\end{array}
\]

i.e. roughly a space made of triangles, tetrahedra, etc.
(and similarly \( \text{codom} \Theta \) gives rise to the **output complex**)

The way we draw tasks

Note that

- if \( l \in \text{dom} \Theta \) (the possible input values) then
  \( l[i \leftarrow \perp] \) also belongs to \( \text{dom} \Theta \)

\( \text{dom} \Theta \) can thus be pictured as a *simplicial complex* called the

**input complex:**

\[ \begin{array}{c}
0 \perp \\
\downarrow 01 \perp \\
\downarrow 01 \downarrow 1 1 \downarrow \\
\downarrow 01 \downarrow 12 \\
\downarrow 0 \downarrow 2 \\
\downarrow \downarrow \downarrow 2
\end{array} \]

i.e. roughly a space made of triangles, tetrahedra, etc.

(and similarly \( \text{codom} \Theta \) gives rise to the **output complex**)

Note also that the vertices are **colored** by \( 0 \leq i < n \):
the only active process
A **task** $\theta$ is a relation $\theta \subseteq I^n \times O^n$ such that for every $l, l' \in \Theta$

1. $l_i = \perp$ if and only if $l'_i = \perp$,
2. there exists $l'' \in O^n$ such that $(l, l'') \in \Theta$ and $(l[i \leftarrow \perp], l''[i \leftarrow \perp]) \in \Theta$.

which means

1. $n$-simplices are in relation with $n$-simplices
2. the relation is compatible with faces
A protocol $\pi$ solves a task $\Theta$ when

- for every initial local memory $l \in \text{dom } \Theta$
- for every long enough and fair execution trace $T$

we have $l' \in \text{codom } \Theta$, where

$$(l', m') = \left[ [T]_\pi(l, \bot \ldots \bot) \right]$$
Solving tasks

A protocol $\pi$ solves a task $\Theta$ when

- for every initial local memory $l \in \text{dom } \Theta$
- for every long enough and fair execution trace $T$

we have $l' \in \text{codom } \Theta$, where

$$(l', m') = \Gamma_\pi(l, \bot \bot \ldots \bot)$$

For instance,

- the consensus cannot be solved
- the quasi-consensus can be solved

Let’s understand why.
Solving tasks

A protocol $\pi$ **solves** a task $\Theta$ when

- for every initial local memory $l \in \text{dom } \Theta$
- for every long enough and fair execution trace $T$

we have $l' \in \text{codom } \Theta$, where

$$
(l', m') = \left[ T \right]_{\pi}(l, \bot \bot \ldots \bot)
$$

For simplicity, we will suppose that $l_i = i$ initially (standard state) and thus write $\left[ T \right]_{\pi}$ instead of $\left[ T \right]_{\pi}(01 \ldots (n - 1), \bot \bot \ldots \bot)$.

For instance,

- the consensus cannot be solved
- the quasi-consensus can be solved

Let’s understand why.
A more manageable setting

In order to study tasks which can be solved by protocols we should simplify as much as possible what we consider as

- protocols
- execution traces
Restricting executions

It can be shown that we can, without loss of generality, restrict to traces which are

- well-bracketed:

\[ u_0 u_1 s_1 u_2 s_0 s_2 \quad \text{but not} \quad u_0 u_0 s_1 s_0 \]
Restricting executions

It can be shown that we can, without loss of generality, restrict to traces which are

- **well-bracketed**: 
  
  \[ u_0 u_1 s_1 u_2 s_0 s_2 \quad \text{but not} \quad u_0 u_0 s_1 s_0 \]

- **layered**: a process does not start a round before all other have finished their or died
  
  \[ u_0 s_0 u_1 s_1 u_1 u_0 s_0 s_1 \quad \text{but not} \quad u_0 u_1 s_0 u_0 s_1 s_0 \]

In particular, we have a notion of *round*. 

Restricting executions

It can be shown that we can, without loss of generality, restrict to traces which are

- **well-bracketed:**

  \[ u_0 u_1 s_1 u_2 s_0 s_2 \quad \text{but not} \quad u_0 u_0 s_1 s_0 \]

- **layered:** a process does not start a round before all other have finished their or died

  \[ u_0 s_0 u_1 s_1 u_1 u_0 s_0 s_1 \quad \text{but not} \quad u_0 u_1 s_0 u_0 s_1 s_0 \]

  In particular, we have a notion of **round**.

- **immediate snapshot:**

  \[ u_0 u_1 s_1 s_0 u_2 s_2 \quad \text{but not} \quad u_0 u_1 s_0 u_2 s_1 s_2 \]
Full-information protocols

A protocol is **full-information** when

\[ \pi_{u_i} = \text{id}_\mathcal{V} \]

We can restrict to those without loss of generality (and we will).
A category of protocols

A **morphism** \( \phi : \pi \rightarrow \pi' \) between protocols consists of functions

\[ \phi_i : \mathcal{V} \rightarrow \mathcal{V} \]

such that

\[ \phi_i(x) = x \text{ for } x \in \mathcal{I} \]

\[ \phi_i(x) \in \mathcal{O} \text{ for } x \in \mathcal{O} \]

and

\[
\begin{array}{c}
\mathcal{V} \times \mathcal{V}^n \xrightarrow{\pi_{s_i}} \mathcal{V} \\
\downarrow \phi_i \times \prod_i \phi_i \\
\mathcal{V} \times \mathcal{V}^n \xrightarrow{\pi_{s_i}} \mathcal{V}
\end{array}
\]

We say that \( \pi' \) *simulates* \( \pi \).
A category of protocols

A **morphism** \( \phi : \pi \to \pi' \) between protocols consists of functions

> \( \phi_i : \mathcal{V} \to \mathcal{V} \) translating memory

such that

> \( \phi_i(x) = x \) for \( x \in \mathcal{I} \)

> \( \phi_i(x) \in \mathcal{O} \) for \( x \in \mathcal{O} \)

> and

\[
\begin{array}{ccc}
\mathcal{V} \times \mathcal{V}^n & \xrightarrow{\pi_{si}} & \mathcal{V} \\
\phi_i \times \prod_i \phi_i \downarrow & & \phi_i \downarrow \\
\mathcal{V} \times \mathcal{V}^n & \xrightarrow{\pi_{si}} & \mathcal{V}
\end{array}
\]

We say that \( \pi' \) **simulates** \( \pi \).

Actually, we only require \( \phi_i \) and \( \phi'_i \) to be defined on reachable values for a given task.
The view protocol

Theorem (GMT)

The category of protocols admits an initial object $\pi^\triangleleft$.

Morally, the space of executions of $\pi^\triangleleft$ is the "universal cover" of the space of executions of any process $\pi$: every execution of $\pi$ corresponds to a unique execution of $\pi^\triangleleft$. 
The view protocol

We suppose that $V$ is countable so that we have an encoding $\langle x, y \rangle$ of pairs (and uples).
The view protocol

We suppose that $\mathcal{V}$ is countable so that we have an encoding $\langle x, y \rangle$ of pairs (and uples).

The initial object $\pi^\triangleleft$ is called the view protocol and is defined by

- $\pi^\triangleleft_{ui}(x) = x$ for $x \in \mathcal{V}$ (full-information),
- $\pi^\triangleleft_{si}(x, m) = \langle x, \langle m \rangle \rangle$ for $(x, m) \in \mathcal{V} \times \mathcal{V}^n$. 
The view protocol

We suppose that $\mathcal{V}$ is countable so that we have an encoding $\langle x, y \rangle$ of pairs (and uples).

The initial object $\pi^\triangleleft$ is called the view protocol and is defined by

- $\pi^\triangleleft_{ui}(x) = x$ for $x \in \mathcal{V}$ (full-information),
- $\pi^\triangleleft_{si}(x, m) = \langle x, \langle m \rangle \rangle$ for $(x, m) \in \mathcal{V} \times \mathcal{V}^n$.

Given a trace $T$, the local memory of $i$-th process after executing the trace $T$ is called its view.
The view protocol

Theorem (GMT)

The category of protocols admits an initial object $\pi^\prec$ with

$$\pi^\prec_{s_i}(x, m) = \langle x, \langle m \rangle \rangle.$$

Proof.

Suppose given a reachable memory

$$x = l_i \quad \text{with} \quad (l, m) = \llbracket T \rrbracket_{\pi^\prec}$$

Because of the definition of morphisms, we are forced to define

$$\phi_i(x) = l'_i \quad \text{with} \quad (l', m') = \llbracket T \rrbracket_{\pi}$$

It only remains to check that this definition is well-defined, i.e. it does not depend on the chosen trace $T$...
THE PROTOCOL COMPLEX
The protocol complex

Given a number $r$ of rounds for each process, the protocol complex $\chi^r(\Theta)$ is the abstract simplicial complex whose

- vertices are $x \in \mathcal{V}$ such that $x$ is the view (= local memory) of $i$-th process after executing a trace with $\pi^<$
- simplices are sets of vertices occurring together after a same execution.
The protocol complex

Suppose that we have 2 processes and the input is the standard one:

\[
0 \quad \overset{\text{}}{\longrightarrow} \quad 1
\]

The protocol complex \(\chi^1(\Theta)\) for 1 round is as follows:
The protocol complex

Suppose that we have 2 processes and the input is the standard one:

\[
\begin{array}{c}
0 \\
\downarrow
\end{array} \quad \begin{array}{c}
\downarrow
1
\end{array}
\]

The protocol complex \( \chi^1(\Theta) \) for 1 round is as follows:

\[
0, 0 \perp \quad \rightarrow \quad 1, 01
\]

After executing 1 round for each process, we have the executions

- \( u_0 s_0 u_1 s_1 \):

\[
\begin{array}{c|c}
0 & 1 \\
\hline
\perp & \perp
\end{array} \quad \xrightarrow{u_0} \quad \begin{array}{c|c}
0 & 1 \\
\hline
\perp & \perp
\end{array} \quad \xrightarrow{s_0} \quad \begin{array}{c|c}
0, 0 \perp & 1 \\
\hline
\perp & \perp
\end{array} \quad \xrightarrow{u_1} \quad \begin{array}{c|c}
0, 0 \perp & 1 \\
\hline
0 & 1
\end{array} \quad \xrightarrow{s_1} \quad \begin{array}{c|c}
0, 0 \perp & 1, 01 \\
\hline
0 & 1
\end{array}
\]
The protocol complex

Suppose that we have 2 processes and the input is the standard one:

\[ 0 \rightarrow 1 \]

The protocol complex \( \chi^1(\Theta) \) for 1 round is as follows:

\[ 0, 0 \perp \rightarrow 1, 01 \quad 0, 01 \rightarrow 1, \perp 1 \]

After executing 1 round for each process, we have the executions

\[ u_1 s_1 u_0 s_0 : \]

\[
\begin{array}{c|c}
0 & 1 \\
\hline
\perp & \perp \\
\end{array}
\xrightarrow{u_1}
\begin{array}{c|c}
0 & 1 \\
\hline
\perp & 1 \\
\end{array}
\xrightarrow{s_1}
\begin{array}{c|c}
0 & 1, \perp 1 \\
\hline
\perp & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
0 & 1, \perp 1 \\
\hline
0 & 1 \\
\end{array}
\xrightarrow{u_0}
\begin{array}{c|c}
0 & 01 \\
\hline
0 & 1 \\
\end{array}
\xrightarrow{s_0}
\begin{array}{c|c}
0, 01 & 1, \perp 1 \\
\hline
0 & 1 \\
\end{array}
\]
The protocol complex

Suppose that we have 2 processes and the input is the standard one:

\[
\begin{array}{c|c}
0 & 1 \\
\hline 
\bot & \bot \\
\end{array}
\]

The protocol complex \( \chi^1(\Theta) \) for 1 round is as follows:

\[
\begin{array}{c|c|c}
0 & 0\bot & 1,01 & 0,01 & 1,\bot1 \\
\end{array}
\]

After executing 1 round for each process, we have the executions

\[ u_1 u_1 s_0 s_1 \]:

\[
\begin{array}{c|c}
0 & 1 \\
\hline 
\bot & \bot \\
\end{array}
\quad \xrightarrow{u_0} \quad
\begin{array}{c|c}
0 & 1 \\
\hline 
0 & \bot \\
\end{array}
\quad \xrightarrow{u_1} 
\begin{array}{c|c}
0 & 1 \\
\hline 
0 & 1 \\
\end{array}
\quad \xrightarrow{s_0} 
\begin{array}{c|c|c}
0,01 & 1 \\
\hline 
0 & 1 \\
\end{array}
\quad \xrightarrow{s_1} 
\begin{array}{c|c|c}
0,01 & 1,01 \\
\hline 
0 & 1 \\
\end{array}
\quad \]
The protocol complex

Suppose that we have 2 processes and the input is the standard one:

\[ 0 \longrightarrow 1 \]

The protocol complex \( \chi^1(\Theta) \) for 1 round is as follows:

\[ 0, 0 \perp \longrightarrow 1, 01 \]

\[ 1, \perp 1 \longrightarrow 0, 01 \]
The protocol complex

Suppose that we have 2 processes and the input is the standard one:

```
0 ______________________________ 1
```

The protocol complex $\chi^1(\Theta)$ for 2 rounds is as follows:

```
0, (0⊥)1 —— 1, (0⊥)(01) —— 0, (0⊥)(01) —— 1, 0(01)
```

```
0, (01)(01)
```

```
1, (01)(01)
```

```
1, 0(⊥1) —— 0, 0(⊥1) —— 1, (01)(⊥1) —— 0, (01)1
```
The protocol complex

Suppose that we have 2 processes and the input is the standard one:

\[ 0 \rightarrow \rightarrow 1 \]

The protocol complex \( \chi^1(\Theta) \) for 2 rounds is as follows:
The protocol complex

With 3 processes and 1 one round, starting from the input complex
The protocol complex

With 3 processes and 1 one round, starting from the input complex we obtain the protocol complex
The protocol complex

With 3 processes and 1 one round, starting from the input complex we obtain the protocol complex

Notice that this is a particular subdivision of the original complex.
The chromatic subdivision

In general, the protocol complex on $r$ rounds is obtained by

- starting from the input complex
- performing a **chromatic subdivision** of it $r$ times

and this subdivision can be defined and studied independently.
The chromatic subdivision

In general, the protocol complex on $r$ rounds is obtained by

- starting from the input complex
- performing a **chromatic subdivision** of it $r$ times

and this subdivision can be defined and studied independently.

**Theorem (Herliy-Shavit, GMT, Koszlov)**

*If the input complex is contractible then the protocol complex is (in fact, collapsible).*
Solvability

Suppose that a task $\Theta$ can be solved by a protocol $\pi$:
- it can be solved in $r$ rounds
Solvability

Suppose that a task $\Theta$ can be solved by a protocol $\pi$:

- it can be solved in $r$ rounds
- there is a map $\phi : \pi^< \rightarrow \pi$ such that, for every trace $T$,

$$\phi([T]_{\pi^<}) = [T]_{\pi}$$

Theorem

If a task can be solved then there is $r$ and a simplicial map from $\chi_r(\Theta)$ to $\text{codom} \Theta$ (and, in fact, conversely).

NB: simplicial maps preserve contractibility!
Solvability

Suppose that a task \( \Theta \) can be solved by a protocol \( \pi \):

- it can be solved in \( r \) rounds
- there is a map \( \phi : \pi^< \to \pi \) such that, for every trace \( T \),
  \[
  \phi([T]_{\pi^<}) = [T]_\pi
  \]
- in particular, when the trace \( T \) has \( r \) rounds \( [T] \in \mathcal{O} \)
Solvability

Suppose that a task $\Theta$ can be solved by a protocol $\pi$:

- it can be solved in $r$ rounds
- there is a map $\phi : \pi^r \to \pi$ such that, for every trace $T$,
  \[
  \phi(\llbracket T \rrbracket_{\pi^r}) = \llbracket T \rrbracket_{\pi}
  \]
- in particular, when the trace $T$ has $r$ rounds $\llbracket T \rrbracket \in \mathcal{O}$
- [...] therefore there is a simplicial map from the $r$-iterated protocol complex to the output complex:

**Theorem**

*If a task can be solved then there is $r$ and a simplicial map from $\chi^r(\Theta)$ to $\text{codom } \Theta$ (and, in fact, conversely).*
Solvability

Suppose that a task $\Theta$ can be solved by a protocol $\pi$:

- it can be solved in $r$ rounds
- there is a map $\phi : \pi^< \rightarrow \pi$ such that, for every trace $T$,
  \[ \phi([T]_{\pi^<}) = [T]_\pi \]
- in particular, when the trace $T$ has $r$ rounds $[T] \in O$
- [...] therefore there is a simplicial map from the $r$-iterated protocol complex to the output complex:

**Theorem**

*If a task can be solved then there is $r$ and a simplicial map from $\chi^r(\Theta)$ to $\text{codom} \Theta$ (and, in fact, conversely).*

NB: simplicial maps preserve contractibility!
Consider again the **binary consensus** task:

There can be no protocol solving it (even after some rounds).
The binary quasi-consensus

Consider the **binary quasi-consensus**:

![Diagram of the binary quasi-consensus](image)
Consider the **binary quasi-consensus**:
EQUIVALENCE BETWEEN TRACES
Execution traces

The (well-bracketed) execution traces in \( \{u_i, s_i\}^* \) are semantically invariant under the congruence \( \approx \) generated by

\[
    u_j u_i \approx u_i u_j \quad \text{and} \quad s_j s_i \approx s_i s_j
\]

which means that

\[
    T \approx T' \quad \text{implies} \quad [T]_\pi = [T']_\pi
\]
Execution traces

The (well-bracketed) execution traces in \( \{u_i, s_i\}^* \) are semantically invariant under the congruence \( \equiv \) generated by

\[
  u_j u_i \equiv u_i u_j \quad s_j s_i \equiv s_i s_j
\]

which means that

\[
  T \equiv T' \quad \text{implies} \quad [T]_\pi = [T']_\pi
\]
Execution traces

The (well-bracketed) execution traces in \( \{u_i, s_i\}^* \) are semantically invariant under the congruence \( \approx \) generated by

\[
\begin{align*}
u_j u_i & \approx u_i u_j \\
s_j s_i & \approx s_i s_j
\end{align*}
\]

which means that

\[
T \approx T' \quad \text{implies} \quad [T]_\pi = [T']_\pi
\]

e.g., \( u_0 \)
Execution traces

The (well-bracketed) execution traces in \( \{u_i, s_i\}^* \) are semantically invariant under the congruence \( \approx \) generated by

\[
    u_j u_i \approx u_i u_j \quad \quad \quad \quad \quad s_j s_i \approx s_i s_j
\]

which means that

\[
    T \approx T' \quad \text{implies} \quad [T]_\pi = [T']_\pi
\]

e.g. \( u_0 u_1 \)
Execution traces

The (well-bracketed) execution traces in \( \{u_i, s_i\}^* \) are semantically invariant under the congruence \( \approx \) generated by

\[
  u_j u_i \approx u_i u_j \quad \text{and} \quad s_j s_i \approx s_i s_j
\]

which means that

\[
  T \approx T' \quad \text{implies} \quad [\pi T] = [\pi T']
\]

\begin{center}
\begin{tikzpicture}
  \node (x0) at (0,0) {\( x_0 \)};
  \node (x1) at (1,0) {\( x_1 \)};
  \node (xn1) at (2,0) {\( x_{n-1} \)};
  \node (x0prime) at (0,-1) {\( x_0' \)};
  \node (x1prime) at (1,-1) {\( x_1' \)};

  \draw[->] (x0) -- (x0prime);
  \draw[->] (x1) -- (x1prime);
  \draw[->] (xn1) -- (x1prime);

  \node (p0) at (0,-2) {\( P_0 \)};
  \node (p1) at (1,-2) {\( P_1 \)};
  \node (pn1) at (2,-2) {\( P_{n-1} \)};

  \draw[->] (x0) -- (p0);
  \draw[->] (x1) -- (p1);
  \draw[->] (xn1) -- (pn1);

  \node (local_mem) at (3,0) {local mem.};
  \node (global_mem) at (3,-1) {global mem.};
\end{tikzpicture}
\end{center}

\text{e.g.} \quad u_0 u_1 \approx u_1
Execution traces

The (well-bracketed) execution traces in \(\{u_i, s_i\}^*\) are semantically invariant under the congruence \(\approx\) generated by

\[u_j u_i \approx u_i u_j\]

\[s_j s_i \approx s_i s_j\]

which means that

\[T \approx T' \implies [T]_{\pi} = [T']_{\pi}\]

e.g.

\[u_0 u_1 \approx u_1 u_0\]
Interval orders

In a well-bracketed trace, the $u_i$ and $s_j$ form intervals:

$u_0 u_1 u_2 s_1 s_0 s_2 u_1 s_1$  $\rightsquigarrow$  

$u_0$  $u_1 s_1$  $u_1 s_1$  $u_2$  $s_2$  $s_0$
Interval orders

In a well-bracketed trace, the $u_i$ and $s_i$ form intervals:

$$
\begin{align*}
&u_0 \ u_1 \ u_2 \ s_1 \ s_0 \ s_2 \ u_1 \ s_1 \\
&\quad \leadsto
\quad \\
&u_0 \ \overbrace{u_1 \ s_1} \ u_1 \ s_1 \\
&\quad \overbrace{u_2 \ s_2} \ \\
&\quad \quad \quad \quad x_1' \\
&\leadsto
\quad x_0 \quad x_1 \quad x_2
\end{align*}
$$

An **interval order** $(X, \preceq)$ is a poset such that there exists a function $I : X \to \mathcal{P}(\mathbb{R})$ associating an interval $I_x$ to each $x$ in such a way that

$$x \prec y \quad \text{if and only if} \quad \forall s \in I_x, \forall t \in I_y, \ s < t$$
Interval orders

In a well-bracketed trace, the $u_i$ and $s_i$ form intervals:

\[
\begin{align*}
&u_0 u_1 u_2 s_1 s_0 s_2 u_1 s_1 \\
\implies
&u_0 s_0 u_1 s_1 u_1 s_1 \\
\implies
&x_0 \downarrow x_1 \downarrow x_2
\end{align*}
\]

An interval order $(X, \preceq)$ is a poset such that there exists a function $I : X \to \wp(\mathbb{R})$ associating an interval $I_x$ to each $x$ in such a way that

\[
x \prec y \quad \text{if and only if} \quad \forall s \in I_x, \forall t \in I_y, \ s \prec t
\]

There is a colored variant with $\ell : X \to \mathbb{N}$ such that $\ell(x) = \ell(y)$ implies that $x$ and $y$ are comparable.
Remark (Fishburn)

A poset is an interval order if it is "\((2 + 2)\)-free":

\[
\begin{array}{ccc}
  b & d \\
  \uparrow & \uparrow \\
  a & c
\end{array}
\]

implies

\[
\begin{array}{ccc}
  b & d \\
  \uparrow & \uparrow \\
  a & c
\end{array}
\]

or

\[
\begin{array}{ccc}
  b & d \\
  \uparrow & \uparrow \\
  a & c
\end{array}
\]
Theorem
Well-bracketed traces up to equivalence are in bijection with colored interval orders.

\[ u_0 u_1 u_2 s_1 s_0 s_2 u_1 s_1 \quad \sim \sim \quad x_0 \xrightarrow{\overleftarrow{x_0}} x_1 \xrightarrow{\overleftarrow{x_1}} x_2 \]
Views of interval orders

Suppose given two elements $x_i$ and $x_j$ of an interval order. We have the following possible situations:

$$
\begin{array}{ccc}
X_j & \uparrow & X_i \\
& X_i & \uparrow X_j \\
X_i & \uparrow & X_j \\
\end{array}
$$

which correspond to the following traces:

$$
u_is_is_j \qquad u_is_is_j \qquad u_is_j_is_i$$
Views of interval orders

Suppose given two elements $x_i$ and $x_j$ of an interval order. We have the following possible situations:

$$
\begin{array}{c}
\uparrow \\
X_j \\
\downarrow \\
X_i \\
\end{array}
\quad
\begin{array}{c}
\downarrow \\
X_i \\
\uparrow \\
X_j \\
\end{array}
\quad
\begin{array}{c}
\uparrow \\
X_j \\
\downarrow \\
X_i \\
\end{array}
\quad
\begin{array}{c}
\downarrow \\
X_i \\
\uparrow \\
X_j \\
\end{array}
$$

which correspond to the following traces:

- $u_i s_i u_j s_j$
- $u_i u_j s_i s_j$
- $u_j s_j u_i s_i$

In the two first cases, $s_j$ sees $u_i$. 

Views of interval orders

This suggests defining the \textit{i-view} of a colored interval order \((X, \preceq)\) by

1. restricting to elements which are below or independent from the maximum element \(x_i^k\) labeled by \(i\)
2. remove dependencies from \(x_i^k\)
Views of interval orders

This suggests defining the \( i \)-view of a colored interval order \((X, \preceq)\) by

1. restricting to elements which are below or independent from the maximum element \(x_i^k\) labeled by \(i\)
2. remove dependencies from \(x_i^k\)

Theorem

- an interval order can be reconstructed from all the \(i\)-views
- the execution of the \(i\)-th process in the view protocol \(\pi^i\) is uniquely determined by the \(i\)-view
Views of interval orders

For instance, with two processes, consider $u_0 u_1 s_1 u_1 s_0 s_1 u_0 s_0$:

- it corresponds to the colored interval order

\[
\begin{array}{c}
\downarrow \\
\downarrow \\
\downarrow \\
\end{array} \quad \begin{array}{c}
\leftarrow \\
\leftarrow \\
\leftarrow \\
\end{array} \quad \begin{array}{c}
\uparrow \\
\uparrow \\
\uparrow \\
\end{array}
\]

\[
x_0^1 \leftarrow x_1^1 \\
x_0^0 \\
x_0^0 \\
x_1^0
\]
Views of interval orders

For instance, with two processes, consider $u_0 u_1 s_1 u_1 s_0 s_1 u_0 s_0$:

- it corresponds to the colored interval order

\[
\begin{array}{c}
 x_0^1 & \leftarrow & x_1^1 \\
 \uparrow & \leftarrow & \uparrow \\
 x_0^0 & \downarrow & x_1^0
\end{array}
\]

- the views are

\[
\begin{array}{c}
 x_0^1 & \leftarrow & x_1^1 \\
 \uparrow & \leftarrow & \uparrow \\
 x_0^0 & x_1^0 \\
 x_0^0 & \downarrow & x_1^0
\end{array}
\]
Views of interval orders

For instance, with two processes, consider $u_0 u_1 s_1 u_1 s_0 s_1 u_0 s_0$:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$u_1$</td>
<td>0</td>
<td>$\langle 1, 01 \rangle$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$\langle 0, 0 \langle 1, 01 \rangle \rangle$</td>
<td>$\langle 1, 01 \rangle$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>$\langle 1, 01 \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_0$</td>
<td>0</td>
<td>$\langle 1, 01 \rangle$</td>
</tr>
<tr>
<td>$s_0$</td>
<td>$\langle 0, 0 \langle 1, 01 \rangle \rangle$</td>
<td>$\langle 1, 01 \rangle$</td>
</tr>
</tbody>
</table>
Views of interval orders

For instance, with two processes, consider $u_0 u_1 s_1 u_1 s_0 s_1 u_0 s_0$:

- we have a correspondence:

\[
\begin{align*}
  x^1_0 & \leftrightarrow x^1_1 \\
  x^0_0 & \leftrightarrow x^0_1 \\
\end{align*}
\]

\[
\langle\langle 0, 0\langle 1, 01\rangle\rangle, \langle 0, 0\langle 1, 01\rangle\rangle \langle 1, 01\rangle\rangle \\
\langle\langle 1, 01\rangle, 0\langle 1, 01\rangle\rangle
\]
Completeness results

From this we deduce:

**Theorem**

The equivalence is complete: given two traces $t$ and $t'$

\[ t \approx t' \iff [t]_{\pi \triangleleft} = [t']_{\pi \triangleleft} \]

**Theorem**

$\pi \triangleleft$ is actually initial in the category of protocols.
The interval order complex

Definition
The **interval order complex** is the simplicial complex whose

- **vertices** are \((i, V_i)\) where \(V_i\) is an \(i\)-view
- **maximal simplices** are \(\{(0, V_0), \ldots, (n, V_n)\}\) such that there is an interval order \((X, \prec)\) (with given number of rounds) whose \(i\)-view is \(V_i\).

Theorem
*The interval order complex is isomorphic to the protocol complex.*
DIRECTED GEOMETRIC SEMANTICS
Directed geometric semantics

The idea of geometric semantics is to formalize the dictionary:

- **program** $\leftrightarrow$ **topological space**
- **state** $\leftrightarrow$ **point of the space**
- **execution trace** $\leftrightarrow$ **path**
- **equivalent traces** $\leftrightarrow$ **homotopic paths**

so that we can import tools from (algebraic) topology in order to study concurrent programs.

We actually need to use spaces equipped with a notion of **direction** in order to take in account irreversible time.
An example

Consider two processes executing one round of update/scan, i.e.

\[ u_0.s_0 \parallel u_1.s_1 \]

The geometric semantics of this program will be

\[ \begin{array}{c}
\text{i.e. a square } [0, 1] \times [0, 1] \text{ minus two holes, which is directed componentwise.}
\end{array} \]
An example

Consider two processes executing one round of update/scan, i.e.

\[ u_0.s_0 \parallel u_1.s_1 \]

The geometric semantics of this program will be

\[ \text{directed path} : u_1 u_0 s_0 s_1 \]

i.e. a square \([0, 1] \times [0, 1]\) minus two holes, which is directed componentwise.
An example

Consider two processes executing one round of update/scan, i.e.

\[ u_0.s_0 \parallel u_1.s_1 \]

The geometric semantics of this program will be

i.e. a square \([0, 1] \times [0, 1]\) minus two holes, which is directed componentwise.

\[
\text{non directed path} : ???
\]
An example

Consider two processes executing one round of update/scan, i.e.

\[ u_0 \cdot s_0 \parallel u_1 \cdot s_1 \]

The geometric semantics of this program will be

i.e. a square \([0, 1] \times [0, 1]\) minus two holes, which is directed componentwise.

\[
\text{homotopy between paths} : \quad u_1 u_0 s_0 s_1 \approx u_0 u_1 s_0 s_1
\]
An example

Consider two processes executing one round of update/scan, i.e.

\[ u_0.s_0 \parallel u_1.s_1 \]

The geometric semantics of this program will be

i.e. a square \([0, 1] \times [0, 1]\) minus two holes, which is directed componentwise.

some paths are not homotopic
More examples

This generalizes to *more rounds*:
consider two processes executing 2 and 4 rounds of update/scan,

\[ u_0.s_0.\parallel u_0.s_0 \quad \parallel \quad u_1.s_1.u_1.s_1.u_1.s_1.u_1.s_1 \]

The geometric semantics of this program will be

![Diagram showing the geometric semantics of the processes]

NB: we will illustrate in dimension 2, where things are simpler
More examples

This generalizes to *more processes*: consider three processes executing one round of update/scan,

\[ u_0.s_0 \ || \ u_1.s_1 \ || \ u_2.s_2 \]

The geometric semantics of this program will be

NB: we will illustrate in dimension 2, where things are simpler
Directed spaces

Formally,

Definition
A pospace \((X, \leq)\) consists of a topological space \(X\) equipped with a partial order \(\leq \subseteq X \times X\), which is closed.

A dipath \(p\) is a continuous non-decreasing map \(p : [0, 1] \to X\).

A dihomotopy \(H\) from a path \(p\) to a path \(q\) is a continuous map \(H : [0, 1] \times [0, 1] \to X\) such that

\[\begin{align*}
&\text{\(H(0, t) = p(t)\) for every \(t\)} \\
&\text{\(H(1, t) = q(t)\) for every \(t\)} \\
&\text{\(t \mapsto H(s, t)\) is a dipath for every \(s\)} \\
&\text{\(s \mapsto H(s, 0)\) and \(s \mapsto H(s, 1)\) are constant}\end{align*}\]
Theorem

Fixing a number of rounds for each process, there is a bijection between

(i) directed paths up to directed homotopy in the geometric semantics

(iii) execution traces up to $\approx$

$u_1 u_0 s_0 s_1 \approx u_0 u_1 s_0 s_1$
Directed paths vs traces

**Theorem**

Fixing a number of rounds for each process, there is a bijection between

(i) directed paths up to directed homotopy in the geometric semantics

(ii) colored interval orders

(iii) execution traces up to \(\approx\)

\[
[u_0, s_0] \prec [u_1, s_1] \quad [u_0, s_0] \parallel [u_1, s_1] \quad [u_0, s_0] \succ [u_1, s_1]
\]
From geometry to the complex

One can notice in the last example that edges are in bijection with directed paths up to homotopy (and with interval orders):

\[
\begin{array}{c|c|c|c}
0, 0 \perp & 1, 01 & 1, 01 & 0, 1 \perp \\
\hline
0 < 1 & 0 < 1 & 0 > 1 & 0 > 1 \\
\end{array}
\]

(more generally maximal simplices are in bijection with maximal directed paths up to homotopy).
From geometry to the complex

This is still true for 2 processes and 2 rounds:
CONCLUSION
### Perspectives

**Links with game stuff?**

<table>
<thead>
<tr>
<th>async. comp.</th>
<th>game semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>question</td>
</tr>
<tr>
<td>scan</td>
<td>answer</td>
</tr>
<tr>
<td>view</td>
<td>view :)</td>
</tr>
<tr>
<td>interval order</td>
<td>event structure</td>
</tr>
<tr>
<td>trace up to equivalence</td>
<td>asynchronous transition system</td>
</tr>
</tbody>
</table>

**Links with quantum stuff?**

<table>
<thead>
<tr>
<th>async. comp.</th>
<th>quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Any question?**