FROM GEOMETRIC SEMANTICS TO ASYNCHRONOUS COMPUTABILITY

ÉRIC GOUBAULT
SAMUEL MIMRAM
CHRISTINE TASSON
École Polytechnique / PPS

DISC conference
October 9, 2015
General idea

Given an asynchronous protocol, we relate two associated geometric constructions:

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

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

Aim: show impossibility results!  
This work should help generalizing to more communication primitives.
ASYNCHRONOUS COMPUTABILITY
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 showing $n$ processes $P_0, P_1, \ldots, P_{n-1}$ accessing a global memory cell. Each process can update its global memory cell or scan the global memory to update its local cell.](image-url)
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

Each process alternatively does:

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

A protocol = what processes compute depending on values.
Decision tasks

We write $\mathcal{V}$ for the set of values, where $\bot$ denotes a dead process.

The question is whether a protocol can solve a task $\Theta \subseteq \mathcal{V}^n \times \mathcal{V}^n$ (where $\mathcal{V}$ is the set of values) in the presence of faults.

We suppose here that the initial value of a process is its process number (for simplicity).

Example (Consensus)

All processes must end with the same value, which is among the input values of the alive processes, i.e.

$$
\Theta = \left\{ (01, 00), (01, 11), (0\bot, 0\bot), (1\bot, 1\bot) \right\}
$$
Execution traces

An **execution trace** is determined by a word in \( \{u_i, s_i\}^* \).

**Remark**

The effect of execution traces on memory is invariant under the smallest congruence \( \approx \) such that

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

---

![Diagram](image-url)
Execution traces

An execution trace is determined by a word in \( \{u_i, s_i\}^* \).

**Remark**

The effect of execution traces on memory is invariant under the smallest congruence \( \approx \) such that

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

![Diagram](image.png)
Execution traces

An **execution trace** is determined by a word in \( \{u_i, s_i\}^* \).

**Remark**
The effect of execution traces on memory is invariant under the smallest congruence \( \approx \) such that

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

\[
\begin{array}{cccc}
  x_0 & x_1 & \cdots & x_{n-1} \\
  P_0 & P_1 & & P_{n-1} \\
  x'_0 & x'_1 & \cdots & \\
  \text{local mem.} & \text{global mem.} & \\
  \text{e.g.} & u_0u_1 & 
\end{array}
\]
Execution traces

An **execution trace** is determined by a word in \( \{u_i, s_i\}^* \).

**Remark**

The effect of execution traces on memory is invariant under the smallest congruence \( \equiv \) such that

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

![Diagram showing local and global memory with execution traces](attachment://diagram.png)

- **Local mem.**
- **Global mem.**

**e.g.**

\[
 u_0 u_1 \equiv u_1
\]
Execution traces

An execution trace is determined by a word in \( \{u_i, s_i\}^* \).

**Remark**

The effect of execution traces on memory is invariant under the smallest congruence \( \approx \) such that

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

![Diagram showing memory layout with execution traces]

- Local mem.
- Global mem.

E.g., \( u_0 u_1 \approx u_1 u_0 \)
GEOMETRIC SEMANTICS
Directed geometric semantics

The idea of geometric semantics is to formalize the dictionary:

\[ \text{program} \iff \text{topological space} \]

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

The idea of geometric semantics is to formalize the dictionary:

- **program** $\Leftrightarrow$ **topological space**
- **state** $\Leftrightarrow$ **point of the space**

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

The idea of geometric semantics is to formalize the dictionary:

\[
\begin{array}{c}
\text{program} & \Leftrightarrow & \text{topological space} \\
\text{state} & \Leftrightarrow & \text{point of the space} \\
\text{execution trace} & \Leftrightarrow & \text{path}
\end{array}
\]

so that we can import tools from (algebraic) topology in order to study concurrent programs.
The idea of geometric semantics is to formalize the dictionary:

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

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

The idea of geometric semantics is to formalize the dictionary:

<table>
<thead>
<tr>
<th>program</th>
<th>⇔</th>
<th>topological space</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>⇔</td>
<td>point of the space</td>
</tr>
<tr>
<td>execution trace</td>
<td>⇔</td>
<td>path</td>
</tr>
<tr>
<td>equivalent traces</td>
<td>⇔</td>
<td>homotopic paths</td>
</tr>
</tbody>
</table>

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

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

\[ \text{i.e. a square } [0, 1] \times [0, 1] \text{ minus two holes, which is directed componentwise.} \]

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

\[
\begin{align*}
& t_1 \\
& s_1 \\
& u_1 \\
& u_0 \\
& s_0 \\
& t_0
\end{align*}
\]

\[
\text{non directed path} : \ 
\]

???
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.

homotopy between paths \(u_1 u_0 s_0 s_1 \approx u_0 u_1 s_0 s_1\)
An example

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

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

The geometric semantics of this program will be

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

some paths are not homotopic
More examples

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

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

The geometric semantics of this program will be
More examples

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

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

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

- \(H(0, t) = p(t)\) for every \(t\)
- \(H(1, t) = q(t)\) for every \(t\)
- \(t \mapsto H(s, t)\) is a dipath for every \(s\)
- \(t \mapsto H(0, t)\) and \(t \mapsto H(1, t)\) are constant
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 \( \approx \)

\[
\begin{align*}
\langle t_1, s_1 \rangle & \quad \iff \quad \langle t_0, s_0 \rangle & & \Rightarrow \quad u_1 u_0 s_0 s_1 \approx u_0 u_1 s_0 s_1
\end{align*}
\]
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 \( \approx \)

\[
[u_0, s_0] \succ [u_1, s_1] \quad [u_0, s_0] \parallel [u_1, s_1] \quad [u_0, s_0] \prec [u_1, s_1]
\]
Interval orders

Definition
A family \((I_j)\) of intervals \(I_j = [a_j, b_j]\) of \(\mathbb{R}\) is partially ordered by \(I_j \prec I_k\) whenever \(x < y\) for every \(x \in I_j\) and \(y \in I_k\), e.g.

\[[1, 3] \prec [5, 6] \quad [1, 3] \parallel [2, 6]\]

We write \(I_j \parallel I_k\) when two elements are not comparable. A poset isomorphic to such a poset is called an interval order.
Interval orders

Definition
A family \((I_j)\) of intervals \(I_j = [a_j, b_j]\) of \(\mathbb{R}\) is partially ordered by \(I_j \prec I_k\) whenever \(x < y\) for every \(x \in I_j\) and \(y \in I_k\), e.g.

\[
[1, 3] \prec [5, 6] \quad [1, 3] \parallel [2, 6]
\]

We write \(I_j \parallel I_k\) when two elements are not comparable. A poset isomorphic to such a poset is called an interval order.

Here, we consider a colored version, where elements are of the form \((i, k)\) with \(0 \leq i < n\) a process number, such that two elements with the same labels are comparable.

\[
[u_0, s_0] \succ [u_1, s_1] \quad [u_0, s_0] \parallel [u_1, s_1] \quad [u_0, s_0] \prec [u_1, s_1]
\]
THE PROTOCOL COMPLEX
The protocol complex

The **protocol complex** is a simplicial complex associated to a protocol, introduced by Herlihy et al. as a central tool in order to characterize tasks which are solvable.

We show here how to reconstruct it from the geometric semantics.
Generic protocols

Given a protocol solving a given task, we can without loss of generality suppose that it is

- **full-information:**
  \( u_i \) writes the exact local memory in the global one

- **generic:**
  \( s_i \) adds the contents of global memory to local one (except at the end where a choice is made).

Example

With two processes, we have for instance
Generic protocols

Given a protocol solving a given task, we can without loss of generality suppose that it is

▶ full-information:
  \( u_i \) writes the exact local memory in the global one

▶ generic:
  \( s_i \) adds the contents of global memory to local one
  (except at the end where a choice is made).

Example

With two processes, we have for instance

\[
\begin{array}{c}
0 \\
\downarrow \\
P_0
\end{array} \quad \begin{array}{c}
1 \\
\downarrow \\
P_1
\end{array} \quad \begin{array}{c}
0 \\
\downarrow \\
\uparrow
\end{array} \quad \begin{array}{c}
1 \\
\downarrow \\
\uparrow
\end{array} \quad \begin{array}{c}
1 \\
\downarrow \\
\uparrow
\end{array} \quad \begin{array}{c}
1 \\
\downarrow \\
\uparrow
\end{array}
\]

\( u_1 \rightarrow \begin{array}{c}
0 \\
\downarrow \\
\uparrow
\end{array} \quad \begin{array}{c}
1 \\
\downarrow \\
\uparrow
\end{array} \quad \begin{array}{c}
0 \\
\downarrow \\
\uparrow
\end{array} \quad \begin{array}{c}
1 \\
\downarrow \\
\uparrow
\end{array} \quad \begin{array}{c}
0 \\
\downarrow \\
\uparrow
\end{array}
\]
Generic protocols

Given a protocol solving a given task, we can without loss of generality suppose that it is

▶ **full-information**:
  
  \( u_i \) writes the exact local memory in the global one

▶ **generic**:
  
  \( s_i \) adds the contents of global memory to local one

(except at the end where a choice is made).

**Example**

With two processes, we have for instance

\[
\begin{array}{c|c|c|c|c}
0 & 1 & u_1 & 0 & 1 \\
P_0 & P_1 & & u_1 & \\
\perp & \perp & & 0 & \perp \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
0 & 1 & u_1 & 0 & 1 \\
0 & 1 & & & \\
\perp & \perp & & & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
0 & 1 & & & \\
0 & 1 & & & \\
\end{array}
\]
Generic protocols

Given a protocol solving a given task, we can without loss of generality suppose that it is

▶ full-information:
  \( u_i \) writes the exact local memory in the global one

▶ generic:
  \( s_i \) adds the contents of global memory to local one (except at the end where a choice is made).

Example

With two processes, we have for instance
Generic protocols

Given a protocol solving a given task, we can without loss of generality suppose that it is

- **full-information:**
  - $u_i$ writes the exact local memory in the global one

- **generic:**
  - $s_i$ adds the contents of global memory to local one (except at the end where a choice is made).

**Example**

With two processes, we have for instance
Generic protocols

Given a protocol solving a given task, we can without loss of generality suppose that it is

- **full-information:**
  - $u_i$ writes the exact local memory in the global one
- **generic:**
  - $s_i$ adds the contents of global memory to local one (except at the end where a choice is made).

**Definition**

The local memory of a process is called its **view**. Two (or more) views are **coherent** when they can occur at the same time in some execution.
The protocol complex

Definition
Given a number of rounds, the protocol complex is the simplicial complex such that

- vertices are the possible views,
- two vertices are linked by an edge when they are coherent,
- three vertices bound a triangle when they are coherent,
- etc.

Example
With two processes, after 0 and 1 rounds, the complexes are

- 0 —— 1
- 0, 0⊥ —— 1, 01 —— 0, 01 —— 1, ⊥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):

(0, 0⊥) 0<1 1, 01 0, 01 0>1 1, ⊥1

(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:
From interval orders to the complex

Since dipaths up to dihomotopy are the same as interval orders, we can start from the latter.
From interval orders to the complex

Since dipaths up to dihomotopy are the same as interval orders, we can start from the latter.

Proposition

Given a colored interval order \((X, \preceq)\), the view \(V^k_i\) of the \(i\)-th process at round \(k\) is given by its restriction to the \(k\)-th scan of the \(i\)-th process

\[
V^k_i = \{(j, l) \mid (i, k) \parallel (j, l) \text{ or } (j, l) \prec (i, k)\}
\]
The interval order complex

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

- **vertices** are \(((i, k), V^k_i)\) where \(i\) stands for the \(i\)-th process, \(k\) for the round number and \(V^k_i\) for an interval order such that for all \((j, l) \in V^k_i\), either \((i, k) \parallel (j, l)\) or \((j, l) \prec (i, k)\),
- **maximal simplices** are \(((0, r_0), V^0_{r_0}), \ldots, ((n, r_n), V^n_{r_n})\) such that there is an interval order \((X^n_{(r)}, \prec)\) whose restriction to \((i, r_i)\) is \(V^i_{r_i}\).

Theorem
The interval order complex is isomorphic to the protocol complex.
The layered protocol complex

Generally, one considers executions which are **layered**: all processes must have finished round \( n \) (or died) before process can start round \( n + 1 \).
The layered protocol complex

Generally, one considers executions which are **layered**: all processes must have finished round $n$ (or died) before process can start round $n + 1$.

**Proposition**

Layered immediate snapshot executions correspond to the interval orders such that $J \prec K$ and $I \parallel J$ implies $I \prec K$.

Moreover, we can recover the fact that layered protocol complexes are iterated chromatic subdivisions of the standard simplex.
The layered protocol complex

Here, we can compute non-layered protocols, which would be difficult to construct by hand:

<table>
<thead>
<tr>
<th>0, ((0_)((0_)1))</th>
<th>0, ((0_))((0_1))</th>
<th>0, ((0_)((0_)1))</th>
<th>0, ((0_)1)</th>
<th>0, ((0_)((0_)1))</th>
<th>0, ((0_)((0_)1))</th>
<th>1, ((0)(01))</th>
<th>1, ((01)(01))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, ((0_)((0_1))</td>
<td>0, ((0_1))</td>
<td>1, ((0_)(01))</td>
<td>0, ((0_1))</td>
<td>1, ((0_)(01))</td>
<td>0, ((0_)(01))</td>
<td>1, ((01)(01))</td>
<td>0, ((01)(01))</td>
</tr>
<tr>
<td>1, ((0_1))</td>
<td>1, ((0)(01))</td>
<td>0, ((0_)(01))</td>
<td>0, ((0_1))</td>
<td>1, ((0)(01))</td>
<td>0, ((0)(01))</td>
<td>1, ((01)(01))</td>
<td>0, ((01)(01))</td>
</tr>
<tr>
<td>1, ((0)(01))</td>
<td>0, ((01)(01))</td>
<td>0, ((0)(01))</td>
<td>0, ((0)(01))</td>
<td>1, ((01)(01))</td>
<td>0, ((01)(01))</td>
<td>1, ((0)(01))</td>
<td>0, ((0)(01))</td>
</tr>
<tr>
<td>0, ((01)(1))</td>
<td>1, ((01)(01))</td>
<td>0, ((01)(1))</td>
<td>1, ((01)(1))</td>
<td>0, ((01)(1))</td>
<td>1, ((01)(1))</td>
<td>0, ((0)(01))</td>
<td>0, ((0)(01))</td>
</tr>
<tr>
<td>1, ((01)(1))</td>
<td>0, ((01)(1))</td>
<td>1, ((01)(1))</td>
<td>0, ((01)(1))</td>
<td>1, ((01)(1))</td>
<td>0, ((01)(1))</td>
<td>1, ((01)(1))</td>
<td>0, ((01)(1))</td>
</tr>
<tr>
<td>0, ((01)(1))</td>
<td>1, ((01)(1))</td>
<td>0, ((01)(1))</td>
<td>1, ((01)(1))</td>
<td>0, ((01)(1))</td>
<td>1, ((01)(1))</td>
<td>0, ((01)(1))</td>
<td>0, ((01)(1))</td>
</tr>
</tbody>
</table>

1, ((0(01))(01))
0, ((0)(01))((01))
0, ((0(01))(01))
0, ((0)(01))((01))
0, ((0(01))(01))
0, ((0)(01))((01))
Conclusion

We have linked geometric semantics and asynchronous computability.

The geometric semantics of many more primitives than update/scan is known (e.g. test/set, compare/swap, etc.) the next step is to try to start from the geometric semantics in order to invent the corresponding “protocol complex” (NB: interval orders were not really crucial in this work).