GEOMETRIC MODELS OF CONCURRENT COMPUTATIONS

SAMUEL MIMRAM
École Polytechnique

Habilitation à diriger des recherches
16 September 2016
Concurrent programs

We are interested in concurrent programs:

- they consist in multiple subprograms running in parallel,
- their scheduling is inherently non-deterministic.

They raise specific problems:

- how can we efficiently verify those programs?
- how can we represent the state space of those programs? (= all possible states the program can be)
We should understand the structure of computations, applications will follow.
The geometric approach

The general idea is that we are going to interpret the state space of programs as a geometric space

\[ P_a; P_b; P_c; V_a; P_f; V_c; V_b; V_f \]
\[ \parallel \]
\[ P_d; P_e; P_a; V_d; P_c; V_e; V_a; V_c \]
\[ \parallel \]
\[ P_b; P_f; V_b; P_d; V_f; P_e; V_d; V_e \]

\[ \rightsquigarrow \]

4 / 73
The geometric approach

The general idea is that we are going to interpret the state space of programs as a **geometric space**

\[
\begin{align*}
P_a; P_b; P_c; V_a; P_f; V_c; V_b; V_f \\
\parallel P_d; P_e; P_a; V_d; P_c; V_e; V_a; V_c \\
\parallel P_b; P_f; V_b; P_d; V_f; P_e; V_d; V_e
\end{align*}
\]

which helps programmers

- provides a way to visualize programs
- helps to come up with new methods
- allows the use of powerful **invariants**
  (fundamental category, curvature, homology, etc.)
The geometric approach

The general idea is that we are going to interpret the state space of programs as a **geometric space**

\[
\begin{align*}
\mathcal{P}_a; \mathcal{P}_b; \mathcal{P}_c; \mathcal{V}_a; \mathcal{P}_f; \mathcal{V}_c; \mathcal{V}_b; \mathcal{V}_f \\
\parallel \quad \mathcal{P}_d; \mathcal{P}_e; \mathcal{P}_a; \mathcal{V}_d; \mathcal{P}_c; \mathcal{V}_e; \mathcal{V}_a; \mathcal{V}_c \\
\parallel \quad \mathcal{P}_b; \mathcal{P}_f; \mathcal{V}_b; \mathcal{P}_d; \mathcal{V}_f; \mathcal{P}_e; \mathcal{V}_d; \mathcal{V}_e
\end{align*}
\]

which raises new questions

- requires adapting classical concepts in order to incorporate the **direction of time**
- provides interesting classes of spaces
Commutation of actions

In concurrent programs, some actions can be interleaved

\[ x := 5 \ || \ x := 9 \]

which means that we have the following executions:

\[
\begin{align*}
&x := 5 \\
\downarrow &\hspace{2cm} \downarrow \\
&x := 9 \\
\downarrow &\hspace{2cm} \downarrow \\
&x := 5
\end{align*}
\]

In fact, the resulting \( x \) could even be different from 5 and 9! (we should ensure that the two actions are mutually exclusive)
Commutation of actions

In concurrent programs, some actions do **commute**

\[ x := 5 \parallel y := 9 \]

in the sense that their order do not matter
Commutation of actions

Two executions which are equivalent up to reordering of commuting actions give rise to the same result:

we can reduce the state-space!
Truly concurrent semantics

The control-flow graph should incorporate this information of commutation between actions:

This is called **true concurrency**: Mazurkiewicz traces, trace monoids, asynchronous transition systems, transition systems with independence, automata with concurrency relations, ...
I

THE

TITLE

(Geometric Semantics for Concurrent Programs)
I am interested in various notions of “concurrent programs”.  

- An imperative programming language:

\[
\begin{align*}
  (\text{if } x = 3 \text{ then } y := 1 \text{ else } y := 2) & \parallel z := 5 \\
  y := 1 & \quad z := 5 \\
  z := 5 & \quad y := 1
\end{align*}
\]

We want to verify programs, reduce the state-space, find invariants, find problematic code (deadlocks / dead code), etc.
Concurrent programs
(Herlihy, ...)

I am interested in various notions of “concurrent programs”.

- Asynchronous protocols with a shared memory:

\[(U_1;S_1)^* \parallel (U_2;S_2)^* \parallel \ldots \parallel (U_n;S_n)^*\]

Which tasks can be implemented in this model, in the presence of failures?
Concurrent programs

I am interested in various notions of “concurrent programs”.

- Distributed version control systems:

  ![Diagram]

  What are the atomic operations and their rules?
  How to represent the states of the system in the case of conflicting operations?
Concurrent programs

I am interested in various notions of “concurrent programs”.

- (String) rewriting systems:
  
  \[
  \begin{align*}
  sts & \rightarrow tst \\
  ss & \rightarrow 1 \\
  tt & \rightarrow 1 \\
  
  \end{align*}
  
  We want to show confluence, reduce the number of rules, etc.
GEOMETRIC MODELS
The model of **asynchronous graphs** is already quite geometric:

![Diagram]

we have:
- 0-dimensional objects: the vertices
- 1-dimensional objects: the edges
- 2-dimensional objects: the commutation squares
Geometric models
(Pratt, van Glabbeek, ...)

In fact we could also take in account commutation of \( n \) actions:

\[
\kappa(a) = 0 \quad \kappa(a) = 1 \quad \kappa(a) = 2 \quad \kappa(a) = 3
\]

and more. In **precubical sets**, we have:

- 0-dimensional cubes: points
- 1-dimensional cubes: edges
- 2-dimensional cubes: squares
- ... 

These can be defined as a suitable presheaf category \( \hat{\Box} \): each \( n \)-cube has a source and target in dimension \( i \) for \( 0 \leq i < n \).
Geometric models

(Pratt, Goubault, Raussen, Haucourt, . . .)

These models are still very algebraic in nature. However, we can realize them as **topological spaces**:

\[
\begin{array}{c}
| - | : \hat{\square} \to \text{Top}
\end{array}
\]

For instance:
Geometric models

(Pratt, Goubault, Raussen, Haucourt, ...)

These models are still very algebraic in nature. However, we can realize them as **topological spaces**:

\[ \lvert - \rvert : \square \rightarrow \text{Top} \]

For instance:

- **dipath** = execution
- **dihomotopy** = equivalence
Geometric models

(Pratt, Goubault, Raussen, Haucourt, ...)  

These models are still very algebraic in nature. However, we can realize them as directed topological spaces:

\[ \mathcal{M} : \square \rightarrow \text{dTop} \]

For instance:

dipath = execution  
dihomotopy = equivalence
Geometric models

In order to be able to speak about quantitative properties, one would also like to consider \textbf{metric} models

\[ \vdash : \square \rightarrow \text{Met} \]

For instance:
Geometric models
(Burroni, ...)

Presentations of (higher-)categories, called polygraphs, involve relations which are not necessarily squares:

\[
\langle s, t \mid st \Rightarrow ts, sts \Rightarrow tst \rangle
\]

can be depicted as

\[
\begin{array}{c}
s \quad t \\
\Rightarrow \\
t \downarrow \\
s \\
\end{array}
\quad \quad \quad \quad \\
\begin{array}{c}
s \quad t \\
\Rightarrow \\
t \downarrow \\
s \\
\end{array}
\]
I will present *some* aspects of these models and focus on some important results:

- give an overview of the methods in the field
- give an overview of their possible applications
II

MUTEXES
AND THE
CUBE
PROPERTY
Control flow graphs

To any program one can associate a **control flow graph**:

\[
x := 43;
while x != 1 do (
    if x mod 2 != 0 then
        (x := 3*x; x := x+1)
    else
        x := x/2
); 
print("Reached 1!")
\]
How can we extend these to concurrent programs?
Mutexes

In order to prevent incompatible actions from running in parallel, one uses **mutexes**, which are **resources** on which two actions are available

- $P_a$: take the resource $a$
- $V_a$: release the resource $a$

and implementation

- guarantees that a resource has been taken at most once at any moment,
- forbids releasing a resource which as not been taken.
Mutexes

In order to prevent incompatible actions from running in parallel, one uses **mutexes**, which are **resources** on which two actions are available

- \( P_a \): *take* the resource \( a \)
- \( V_a \): *release* the resource \( a \)

and implementation

- guarantees that a resource has been taken at most once at any moment,
- forbids releasing a resource which as not been taken.

Our earlier program should be rewritten as

\[
P_a ; x := 5 ; V_a \quad \parallel \quad P_a ; x := 9 ; V_a
\]
Mutexes

In the program

\[ P_a; x := 5; V_a \parallel P_a; x := 9; V_a \]

the possible executions are

![Diagram illustrating possible executions of the program with mutexes]
Mutexes

In the program

\[ P_a; x := 5; V_a \parallel P_a; x := 9; V_a \]

the possible executions are
The cubical semantics

**Definition**
The **cubical semantics** $\check{C}_p$ of a program $p$ is

- the precubical set $C_p$ associated to $p$ by induction:

$$C_{p\parallel q} = C_p \otimes C_q$$

- with forbidden vertices removed (and iterated cofaces)
The cubical semantics

Definition
The **cubical semantics** $\check{C}_p$ of a program $p$ is

- the precubical set $C_p$ associated to $p$ by induction:

$$C_{p\parallel q} = C_p \otimes C_q$$

- with forbidden vertices removed (and iterated cofaces)

Lemma
*The dipaths starting from the initial vertex are in bijection with possible executions of the program.*

![Diagram](image)
Dihomotopy between dipaths

Definition
The **dihomotopy** relation $\sim$ between dipaths is the smallest congruence such that $A \cdot B \sim B' \cdot A'$ whenever $A \cdot B \diamond B' \cdot A'$:

![Diagram](image)

Proposition
For “coherent” programs, two dihomotopic executions lead to the same state.
III

HOMOTOOPY

VS

DIHOMOTOOPY
Dipaths

From a topological point of view, instead of considering directed paths (or dipaths)

\[ A \rightarrow B \rightarrow C \]  or  \[ A \cdot B \cdot C \]

it is more natural to consider paths

\[ A \rightarrow \leftarrow B \rightarrow C \]  or  \[ A \cdot \overline{B} \cdot C \]

where arrows can occur backward.
Homotopy

Similarly, one would usually consider **homotopy** \( \sim \) between paths: the smallest congruence, containing dihomotopy \( \hat{\sim} \) and such that for every edge

\[
x \xrightarrow{A} y
\]

we have

\[
\text{id}_x \sim A \cdot \overline{A} \quad \overline{A} \cdot A \sim \text{id}_y
\]

**Remark**

Clearly \( f \hat{\sim} g \) implies \( f \sim g \), but converse is not generally true.
Homotopy vs dihomotopy
(Fahrenberg)

Consider the following “matchbox”:

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
A_1 \\
A_4 \\
A_2 \\
A_3
\end{array}
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
C_1 \\
C_2 \\
C_3
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

where every square is filled excepting the top one:

\[
A_1 \cdot B_4 \cdot B_1 \cdot A_4
\]
Consider the following “matchbox”:

We have

\[ A_1 \cdot B_4 \sim B_1 \cdot A_4 \quad \text{but not} \quad A_1 \cdot B_4 \sim B_1 \cdot A_4 \]
Homotopy vs dihomotopy
Homotopy vs dihomotopy

\[ A_1 \cdot B_4 \quad \sim \quad C_1 \cdot \overline{C_1} \cdot A_1 \cdot B_4 \]
Homotopy vs dihomotopy

\[ A_1 \cdot B_4 \sim C_1 \cdot \overline{C_1} \cdot A_1 \cdot B_4 \]
\[ \sim \quad C_1 \cdot A_2 \cdot \overline{C_4} \cdot B_4 \]
This example cannot be obtained as the semantics of a program!
Homotopy vs dihomotopy

This example cannot be obtained as the semantics of a program!
Homotopy vs dihomotopy

$A_1 \cdot B_4 \sim C_1 \cdot \overline{C_1} \cdot A_1 \cdot B_4$
$\sim C_1 \cdot A_2 \cdot \overline{C_4} \cdot B_4$
$\sim C_1 \cdot A_2 \cdot B_3 \cdot \overline{C_3}$
$\sim C_1 \cdot B_2 \cdot A_3 \cdot \overline{C_3}$
$\sim B_1 \cdot C_2 \cdot A_3 \cdot \overline{C_3}$
Homotopy vs dihomotopy

This example cannot be obtained as the semantics of a program!
Homotopy vs dihomotopy

This example cannot be obtained as the semantics of a program!

\[ A_1 \cdot B_4 \sim C_1 \cdot C_1 \cdot A_1 \cdot B_4 \]
\[ \sim C_1 \cdot A_2 \cdot C_4 \cdot B_4 \]
\[ \sim C_1 \cdot A_2 \cdot B_3 \cdot C_3 \]
\[ \sim C_1 \cdot B_2 \cdot A_3 \cdot C_3 \]
\[ \sim B_1 \cdot C_2 \cdot A_3 \cdot C_3 \]
\[ \sim B_1 \cdot A_4 \cdot C_3 \cdot C_3 \]
\[ \sim B_1 \cdot A_4 \]
Homotopy vs dihomotopy

This example cannot be obtained as the semantics of a program!
Binary conflicts

In a situation such as

\[ P_a \parallel P_a \parallel B = \]

the vertex \( x \) is forbidden (and has to be removed).
Binary conflicts

In a situation such as

\[ P_a \parallel P_a \parallel B = \]

the vertex \( x \) is forbidden (and has to be removed).

In this case, the vertex \( y \) has to be removed too, because \( B \neq V_a \)!
The cube property

Proposition
Semantics of programs satisfy the **cube property**:

(and other more minor properties).
Homotopy vs dihomotopy

**Theorem**

In a precubical set satisfying the cube property, two dipaths are dihomotopic if and only if they are homotopic: the inclusion functor

\[ \overline{\Pi}_1(C) \hookrightarrow \Pi_1(C) \]

is faithful.

**Proof.**

Uses 2-dimensional rewriting techniques!
Some consequences

From this follows many interesting properties:

- normal forms for dihomotopy classes of paths,
- an easy definition of universal covers,
- ...
IV

NON-POSITIVELY
CURVED
PRECUBICAL
SETS
A semantics in metric spaces is desirable:

- we want to have a notion of length of paths (corresponding to the duration of an execution),
- interesting notions, such as curvature are available in this context.
Geometric realization

In order to define a geometric realization $|\cdot|$ in metric spaces, we should use the usual formula

$$|C| = \int_{n \in \Box} C_n \cdot I^n$$

However, the category of metric spaces does not have enough colimits.

We should consider Lawvere's generalized metric spaces!
Generalizing metric spaces

Definition

A **metric space** is a space \( X \) equipped with a metric \( d : X \times X \rightarrow [0, \infty] \) such that, given \( x, y, z \in X \),

1. **point equality:** \( d(x, x) = 0 \)
2. **triangle inequality:** \( d(x, z) \leq d(x, y) + d(y, z) \)
3. **finite distances:** \( d(x, y) < \infty \)
4. **separation:** \( d(x, y) = 0 \) implies \( x = y \)
5. **symmetry:** \( d(x, y) = d(y, x) \)

We consider contracting maps \( f : X \rightarrow Y \):

\[
d_Y(f(x), f(x')) \leq d_X(x, x')
\]

Unfortunately, the resulting category is not cocomplete!
Generalizing metric spaces

Definition
A **metric space** is a space $X$ equipped with a metric $d : X \times X \to [0, \infty]$ such that, given $x, y, z \in X$,

1. **point equality**: $d(x, x) = 0$
2. **triangle inequality**: $d(x, z) \leq d(x, y) + d(y, z)$
3. **finite distances**: $d(x, y) < \infty$
4. **separation**: $d(x, y) = 0$ implies $x = y$
5. **symmetry**: $d(x, y) = d(y, x)$

Intuitively, $X + Y$ should be such that

$$d(x, y) = \infty$$

for $x \in X$ and $y \in Y$. 
Generalizing metric spaces

**Definition**

A **metric space** is a space $X$ equipped with a metric $d : X \times X \to [0, \infty]$ such that, given $x, y, z \in X$,

1. **point equality:** $d(x, x) = 0$
2. **triangle inequality:** $d(x, z) \leq d(x, y) + d(y, z)$
3. **finite distances:** $d(x, y) < \infty$
4. **separation:** $d(x, y) = 0$ implies $x = y$
5. **symmetry:** $d(x, y) = d(y, x)$

Consider the relation $\approx$ on $X$ identifying a family of points $(x_i)_{i \in \mathbb{N}}$ such that $d(x_i, y) = 1/i$ for some $y$

Intuitively, in $X/\approx$, we should have $d([x_i], [y]) = 0$. 
Generalizing metric spaces

**Definition**
A **metric space** is a space $X$ equipped with a metric $d : X \times X \rightarrow [0, \infty]$ such that, given $x, y, z \in X$,

1. **point equality:** $d(x, x) = 0$
2. **triangle inequality:** $d(x, z) \leq d(x, y) + d(y, z)$
3. **finite distances:** $d(x, y) < \infty$
4. **separation:** $d(x, y) = 0$ implies $x = y$
5. **symmetry:** $d(x, y) = d(y, x)$

We can encode direction in the distance!

$$d(x, y) = \bigwedge \{ \rho - \theta \mid x = e^{i2\pi \theta}, y = e^{i2\pi \rho}, \rho \geq \theta \}$$
Generalized metric spaces

(Lawvere)

Definition
A generalized metric space is a space $X$ equipped with a metric $d : X \times X \to [0, \infty]$ such that, given $x, y, z \in X$,

1. point equality: $d(x, x) = 0$
2. triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$
Definition
A generalized metric space is a space $X$ equipped with a metric $d : X \times X \to [0, \infty]$ such that, given $x, y, z \in X$,

1. point equality: $d(x, x) = 0$
2. triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Proposition
The category $\text{GMet}$ enjoys the following:

- the category $\text{GMet}$ is complete and cocomplete,
- the forgetful functor $\text{GMet} \to \text{Set}$ has left and right adjoints,
- the forgetful functor $\text{GMet} \to \text{Top}$ preserves finite (co)limits.
Metric realization

We can define a metric realization functor

\[ |-| : \hat{\square} \rightarrow \text{GMet} \]

**Theorem**

*For a locally finite precubical set \( C \), the space \(|C|\)*

- has the usual geometric real. as underlying topological space
- is a geodesic length space.
Non-positively curved spaces

The cube property for precubical sets is analogous to Gromov’s condition for characterizing NPC cubical complexes:

**Theorem**

*Given a finite dimensional geometric precubical set $C$ satisfying the cube property, its metric realization $|C|$ is non-positively curved, i.e. locally $\text{CAT}(0)$.***
In fact, many definitions and properties of NPC cubical complexes can be carried on directly in the setting of precubical sets!
Some consequences

We have the properties of NPC spaces:

- $|C|$ is locally uniquely geodesic
- the universal cover is NPC,
- the fundamental group is automatic,
- links with geometric group theory,
- ...

This suggests generalizations of the cube property:

- what are the corresponding notion of “NPC”?
V

DISTRIBUTED VERSION CONTROL SYSTEMS
DVCS

Distributed Version Control Systems are used when working collaboratively on files and import modifications from other people...

A **patch** stores the difference between two files (i.e. the list of inserted and deleted lines).

Users can perform two actions:

- **commit** the difference between the current version and the last committed version as a patch to a server
- **update** its current version by importing all the new patches on the server

Intuitively, we have a category of files and patches.
Using DVCS

Alice

Bob

b
Using DVCS

Alice

Bob

$b$
Using DVCS

Merging modifications is naturally modeled by pushouts. In particular, this provides residual patches!
Using DVCS

Merging modifications is naturally modeled by pushouts. In particular, this provides residual patches!
Using DVCS

Merging modifications is naturally modeled by pushouts. In particular, this provides residual patches!
Merging modifications is naturally modeled by pushouts.

In particular, this provides residual patches!
Conflicts

However, not every pair of coinital morphisms has a pushout!
Conflicts

However, not every pair of coinitial morphisms has a pushout!
Conflicts

However, not every pair of coinitial morphisms has a pushout!
Handling conflicts

We should extend our model to account for “files with conflicts” and their handling.

There were many proposals for this. Instead of discussing the relative merits of each of those, we instead look for a

universal property

that this extension should satisfy: we want to

formally add pushouts.
Handling conflicts

We should extend our model to account for “files with conflicts” and their handling.

There were many proposals for this. Instead of discussing the relative merits of each of those, we instead look for a

universal property

that this extension should satisfy: we want to

formally add finite colimits.
Handling conflicts

We should extend our model to account for “files with conflicts” and their handling.

There were many proposals for this. Instead of discussing the relative merits of each of those, we instead look for a

universal property

that this extension should satisfy: we want to

formally add finite colimits which do not already exist.
Handling conflicts

We should extend our model to account for “files with conflicts” and their handling.

There were many proposals for this. Instead of discussing the relative merits of each of those, we instead look for a

universal property

that this extension should satisfy: we want to

compute a conservative finite cocompletion.
Enforcing confluence

This is a different perspective on concurrency:

- usually, we want to check the **confluence** property:

  ![Diagram](image)

- here, we want to **enforce** confluence
Let’s begin with a simplified model:

- one file
- lines can only be inserted
- lines don’t have contents
The category of files and insertions

Definition
We write $\mathcal{L}$ for the category whose

- objects are sets $\{0, \ldots, n - 1\}$ for some $n \in \mathbb{N}$,
- morphisms are injective increasing functions:

```
0 ----> 0
|   |    |
| 1  | 1 |
|____|   |
   2 ---- 2 ---- 3 ---- 4
```

Remark
This is also known as the augmented presimplicial category.
The category of files and insertions

Definition
We write $\mathcal{L}$ for the category whose
- objects are sets $\{0, \ldots, n - 1\}$ for some $n \in \mathbb{N}$,
- morphisms are injective increasing functions:

```
0  1  2  3  4
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
0  1  2  3  4
```

Proposition
This is the free monoidal category containing an object 1 and a morphism

$$\eta : 0 \to 1$$
The category of files and insertions

Definition
We write \( \mathcal{L} \) for the category whose

- objects are sets \( \{0, \ldots, n-1\} \) for some \( n \in \mathbb{N} \),
- morphisms are injective increasing functions:

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4
\end{array} \xrightarrow{\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4
\end{array}} \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4
\end{array}
\]

Remark
We can add labels to the lines by using a slice category construction.
The conservative finite cocompletion

The category $\mathcal{P}$ of “files with conflicts” should be the **free conservative finite cocompletion** of $\mathcal{L}$:

\[
\begin{array}{c}
\mathcal{L} \\
\downarrow Y \\
\mathcal{P}
\end{array} \xrightarrow{F} \begin{array}{c}
\mathcal{C} \\
\downarrow \tilde{F}
\end{array}
\]

with

- $\mathcal{P}, \mathcal{C}$ with finite colimits
- $Y, F, \tilde{F}$ preserving finite colimits
It is well-known that $\hat{\mathcal{L}}$ is the free cocompletion of $\mathcal{L}$:

\[
\begin{array}{ccc}
\mathcal{L} & \xrightarrow{F} & \mathcal{C} \\
\downarrow & & \downarrow \\
\hat{\mathcal{L}} & \xleftarrow{\tilde{F}} & \mathcal{C}
\end{array}
\]

**Theorem (Kelly)**

The free conservative cocompletion of $\mathcal{L}$ is the full subcategory of $\hat{\mathcal{L}}$ whose objects are continuous presheaves.

The finite cocompletion can be obtained by further restricting to “finite” presheaves.
From this one can extract an explicit description of $\mathcal{P}$.

**Theorem**

The free conservative finite cocompletion $\mathcal{P}$ of $\mathcal{L}$ is the category:

- **objects** $(A, <)$ are finite sets equipped with a transitive relation $<$,
- **a morphism** $f : A \to B$ is a function respecting the relation.
Computing in $\mathcal{P}$

We have all pushouts, e.g. the pushout of

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

is

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]

\[
\begin{array}{c}
a' \\
a \\
c \\
b
\end{array}
\quad \xleftarrow{\mathbf{f}_1} 
\begin{array}{c}
a \\
b
\end{array}
\quad \xrightarrow{\mathbf{f}_2}
\begin{array}{c}
a \\
d \\
b
\end{array}
\]
A free cocompletion of $\mathcal{L}$

Every object in $\mathcal{P}$ can be obtained as a colimit of objects in $\mathcal{L}$. For instance, consider the morphisms

$$\bullet \overset{s}{\rightarrow} \bullet \quad \text{and} \quad \bullet \overset{t}{\rightarrow} \bullet$$

By coproduct, we get a “sequentialization” morphism
A free cocompletion of \( \mathcal{L} \)

Every object in \( \mathcal{P} \) can be obtained as a colimit of objects in \( \mathcal{L} \). For instance, consider the morphisms

\[
\bullet \xrightarrow{s} \bullet \quad \text{and} \quad \bullet \xrightarrow{t} \bullet
\]

By coproduct, we get a “sequentialization” morphism

The pushout of

\[
\bullet \quad \xrightarrow{\text{seq}} \quad \bullet \quad \xrightarrow{\text{seq}'} \quad \bullet
\]

is
Notice that we get a way of identifying two independent lines, which can be used to solve a conflict.
There are many possible extensions of this work:

- patches with deletions,
- structured files,
- many files,
- ...
Pijul
(Meunier, Becker, ...)

http://pijul.org/
VI

PRESENTATIONS MODULO OF CATEGORIES
Presentations of $n$-categories (Burroni, ...)

An $n$-category can be presented by a polygraph, i.e. described as
- the free $n$-category on given 0-, 1, ..., $n$-generators,
- quotiented by a congruence generated by relations on $n$-cells.

\[\begin{array}{ccccccc}
P_0 & \xrightarrow{s_0} & P_1 & \xrightarrow{s_1} & P_2 & \xrightarrow{s_2} & \cdots & \xrightarrow{s_n} & P_n \\
\downarrow i_0 & & \downarrow i_1 & & \downarrow i_2 & & \cdots & & \downarrow i_n \\
P^*_0 & \xleftarrow{t_0} & P^*_1 & \xleftarrow{t_1} & P^*_2 & \xleftarrow{t_2} & \cdots & \xleftarrow{t_n} & P^*_n \\
\end{array}\]
Presentations of $n$-categories

(Burroni, ...)

An $n$-category can be *presented* by a **polygraph**, i.e. described as

- the free $n$-category on given 0-, 1, ..., $n$-generators,
- quotiented by a congruence generated by relations on $n$-cells.

\[
\begin{align*}
P_0 & \xrightleftharpoons[i_0]{s_0} P_1 \xrightleftharpoons[i_1]{s_1} P_2 \cdots \xrightleftharpoons[i_{n-1}]{s_{n-2}} P_{n-1} \xrightleftharpoons[i_n]{s_{n-1}} P_n \\
P^*_0 & \xleftleftharpoons[t_0^*]{t_0} P^*_1 \xleftleftharpoons[t_1^*]{t_1} P^*_2 \cdots \xleftleftharpoons[t_{n-2}^*]{t_{n-2}} P^*_{n-1} \xleftleftharpoons[t_n^*]{t_n} P^*_n
\end{align*}
\]

**Remark**
Since the quotient is performed on $n$-cells the underlying $(n - 1)$-category is free!

**Question**
Can we coherently quotient in lower dimensions?
We provide an answer for $n = 1$ and $n = 2$. 
Presentations of categories.
Definition

A graph $G = (V, s, t, E)$ consists of

- a set $V$ of vertices
- a set $E$ of edges
- source and target functions $s, t : E \rightarrow V$
Graphs

Definition
A graph $G = (V, s, t, E)$ consists of
- a set $V$ of vertices
- a set $E$ of edges
- source and target functions $s, t : E \to V$

The free category generated by $G$ has
- objects: vertices $V$
- morphisms: paths $E^*$ (with concatenation as composition)
Presentations of categories

**Definition**

A **presentation** $P$ of category consists of

- a graph (the *signature*)
- a set of rules relating a path with another path with same source and target

The **presented category** $\|P\|$ is the free category on the graph with paths taken modulo the congruence generated by rules.
Presentations of categories (formally)

**Definition**

A **presentation** $P$ of category consists of

- a set $P_0$ of *object generators*
- a set $P_1$ of *morphism generators*
- a set $P_2$ of *relations*

with $s_0^* \circ s_1 = s_0^* \circ t_1$ and $t_0^* \circ s_1 = t_0^* \circ t_1$

**Question**

How do we add a quotient on objects too?
Presentations modulo

**Definition**

A **presentation modulo** \((P, \tilde{P}_1)\) of category consists of

- a presentation of category \(P\),
- a set \(\tilde{P}_1 \subseteq P_1\) of *equational generators*.

\[
\begin{align*}
P_0 &= \{x_i\} \\
P_1 &= \{f_i, g_i\} \\
\tilde{P}_1 &= \{f_i\} \\
P_2 &= \{\rho\}
\end{align*}
\]
Presentations modulo

**Definition**

A *presentation modulo* \((P, \tilde{P}_1)\) of category consists of

- a presentation of category \(P\),
- a set \(\tilde{P}_1 \subseteq P_1\) of *equational generators*.

\[
\begin{align*}
P_0 & = \{x_i\} \\
P_1 & = \{f_i, g_i\} \\
\tilde{P}_1 & = \{f_i\} \\
P_2 & = \{\rho\}
\end{align*}
\]

Since, we want to consider objects modulo relations in \(\tilde{P}_1\), it is natural to suppose that

**Assumption**
The abstract rewriting system \((P_0, \tilde{P}_1)\) is convergent.
The category presented modulo $\langle P, \tilde{P}_1 \rangle$, we have three possible ways of defining the presented category from $\|P\|$: 

1. **quotient by equational generators**: turn them into identities,

2. **localize by equational generators**: turn them into isomorphisms,

3. **restrict to objects which are normal forms wrt equational generators.**
The category presented modulo

Given a presentation modulo \( (P, \tilde{P}_1) \), we have three possible ways of defining the presented category from \( \|P\| \):

1. **quotient** by equational generators: turn them into identities,
The category presented modulo

Given a presentation modulo \((P, \tilde{P}_1)\), we have three possible ways of defining the presented category from \(\|P\|\):

1. **quotient** by equational generators: turn them into identities,

2. **localize** by equational generators: turn them into isomorphisms,
The category presented modulo

\[
\begin{array}{c}
\text{The category presented modulo } \langle P, \tilde{P}_1 \rangle, \text{ we have three possible ways of defining the presented category from } \|P\|:
\end{array}
\]

1. **quotient** by equational generators: turn them into identities,

2. **localize** by equational generators: turn them into isomorphisms,

3. **restrict** to objects which are normal forms wrt equational generators.
The main result

**Theorem**

*Given a presentation modulo \((P, \tilde{P}_1)\) satisfying suitable assumptions, the three constructions are related by*

\[
\begin{align*}
\|P\| \downarrow \tilde{P}_1 & \xrightarrow{\text{iso}} \|P\|/\tilde{P}_1 \\
\xleftarrow{\text{equiv}} & \xleftarrow{\text{equiv}} P[\tilde{P}_1^{-1}]
\end{align*}
\]

**Remark**

This generalizes Ore-Dehornoy conditions ensuring that a (presented) category embeds into its enveloping groupoid.
The main result

**Theorem**

*Given a presentation modulo \( (P, \tilde{P}_1) \) satisfying suitable assumptions, the three constructions are related by*

\[
\begin{array}{c}
\| P \| \downarrow \tilde{P}_1 \quad \text{iso} \quad \| P \| / \tilde{P}_1 \\
\| P \| \downarrow \tilde{P}_1^{-1} \quad \text{equiv} \quad P[\tilde{P}_1^{-1}] \\
\end{array}
\]

**Remark**

This generalizes Ore-Dehornoy conditions ensuring that a (presented) category embeds into its enveloping groupoid.
Remarks

- One of these assumptions is a variant of the cube property
Remarks

- One of these assumptions is a variant of the cube property
- We need the notion of residual for rewriting paths
Remarks

- One of these assumptions is a variant of the cube property.
- We need the notion of residual for rewriting paths.
- The motivations (and tools) are the same as for defining the category of components (Goubault, Haucourt, ...):
Remarks

- One of these assumptions is a variant of the cube property
- We need the notion of residual for rewriting paths
- The motivations (and tools) are the same as for defining the category of components (Goubault, Haucourt, ...):

  ![Diagram](image)

- ...which is itself closely related to partial order reduction.
In dimension 2

Presentations of 1-categories is a toy case, this has since then been extended to monoidal categories (= 2-categories) to obtain a presentation of

\[ \Delta \times \Delta \]

as a product of monoidal categories (with similar properties).
VII

CONCLUSION
Omitted work

There is more work in the manuscript:
- adj. between cubical models and classical ones (Winskel)
There is more work in the manuscript:

- adj. between cubical models and classical ones (Winskel)
- relating partial order reduction and the cat. of components
Omitted work

There is more work in the manuscript:

- adj. between cubical models and classical ones (Winskel)
- relating partial order reduction and the cat. of components
- geometric models and asynchronous computability
Omitted work

There is more work in the manuscript:

- adj. between cubical models and classical ones (Winskel)
- relating partial order reduction and the cat. of components
- geometric models and asynchronous computability
- homotopical completion of rewriting systems
Omitted work

There is more work in the manuscript:

- adj. between cubical models and classical ones (Winskel)
- relating partial order reduction and the cat. of components
- geometric models and asynchronous computability
- homotopical completion of rewriting systems
- preliminary implementation of polygraphs
Unifying viewpoints and techniques

Computation

Algebra ←

Geometry

Perspectives:

- higher-dimensional categories and rewriting:
  - computational properties
  - homotopical properties
  - presentations of weak $\infty$-categories
  - directed homotopy equivalence
  - ...

- geometric invariants for asynchronous computations
  - first links with geom. sem. and partially commutative monoids
  - more invariants (Squier?)
  - ...

QUESTIONS?