DIRECTED GEOMETRIC MODELS OF CONCURRENT PROGRAMS

Samuel Mimram

École Polytechnique

Applied and Computational Algebraic Topology

Spring School

April 24th, 2017

Models of concurrent programs

- We are interested in *concurrent programs*:
  multiple threads in parallel.

Models of concurrent programs

- We are interested in *concurrent programs*:
  multiple threads in parallel.

- We want to model their *state space*:
  describe all possible executions.

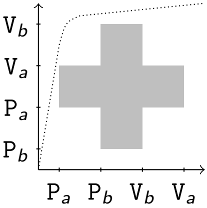
Models of concurrent programs

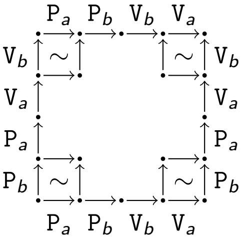
- We are interested in *concurrent programs*:
  multiple threads in parallel.
- We want to model their *state space*:
  describe all possible executions.
- We advocate here that *geometric models* can be useful:
  we can (hope to) use geometric tools.

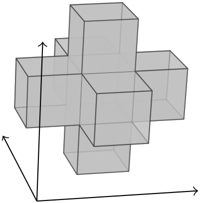
Models of concurrent programs

- We are interested in *concurrent programs*:
  multiple threads in parallel.
- We want to model their *state space*:
  describe all possible executions.
- We advocate here that *geometric models* can be useful:
  we can (hope to) use geometric tools.
- A typical application is to *verification* of programs:
  guarantee that a program will never divide by 0,
  and other problems more specific to concurrency.

Geometric models







Geometric models are interesting:

- they range from algebraic to topological flavors
- they provide useful visualizations of the state space
- we can use geometric invariants and constructions
- it raises new questions in geometry

Directed geometry

Let's consider a typical example.

For a connected space X, one considers the **fundamental group**

$$\pi_1(X, x_0)$$

Directed geometry

Let's consider a typical example.

For a connected space X, one considers the **fundamental group**

$$\pi_1(X, x_0)$$

More generally, one considers the **fundamental groupoid**

$$\Pi_1(X)$$

Directed geometry

Let's consider a typical example.

For a connected space X, one considers the **fundamental group**

$$\pi_1(X, x_0)$$

More generally, one considers the **fundamental groupoid**

$$\Pi_1(X)$$

For directed space, we have a **fundamental category**

$$\vec{\Pi}_1(X)$$

Directed geometry

Let's consider a typical example.

For a connected space X, one considers the **fundamental group**

$$\pi_1(X, x_0)$$

More generally, one considers the **fundamental groupoid**

$$\Pi_1(X)$$

For directed space, we have a **fundamental category**

$$\vec{\Pi}_1(X)$$

Other invariants have to be generalized similarly, which is not always obvious (e.g. no weak equivalences / model categories, etc.)

This talk

Here

- we focus on (simple) geometric aspects:
  more involved developments will follow in Raussen's talk
- verification is only really used here as a motivation

VERIFYING SEQUENTIAL PROGRAMS

Control flow graphs

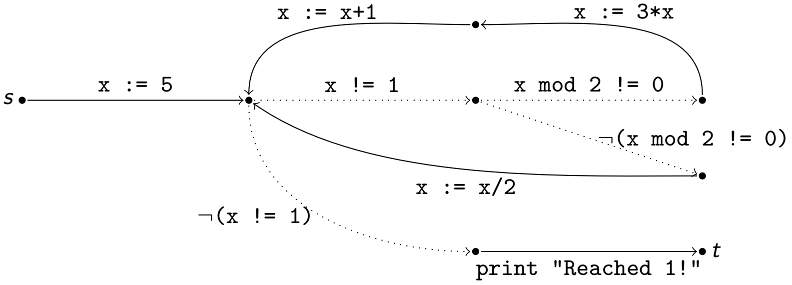
When studying sequential programs, people often already use a geometric description: **control flow graphs**.

Control flow graphs

```
x := 5;
while x != 1 do (
    if x mod 2 != 0 then
        (x := 3*x; x := x+1)
    else
        x := x/2
);
print "Reached 1!"
```

Control flow graphs

```
x := 5;
while x != 1 do (
    if x mod 2 != 0 then
        (x := 3*x; x := x+1)
    else
        x := x/2
);
print "Reached 1!"
```



A toy language

Consider a language consisting of

- *arithmetic expressions*:

$$a \quad ::= \quad x \;\mid\; n \;\mid\; a \;\texttt{+}\; a \;\mid\; a \;\texttt{*}\; a$$

A toy language

Consider a language consisting of

- *arithmetic expressions*:

$$a \quad ::= \quad x \;\mid\; n \;\mid\; a \;\texttt{+}\; a \;\mid\; a \;\texttt{*}\; a$$

- *boolean expressions*:

$$b \quad ::= \quad \texttt{true} \;\mid\; \texttt{false} \;\mid\; a \;\texttt{<}\; a \;\mid\; b \;\texttt{and}\; b \;\mid\; \neg b$$

A toy language

Consider a language consisting of

- *arithmetic expressions*:

$$a \quad ::= \quad x \mid n \mid a \texttt{ + } a \mid a \texttt{ * } a$$

- *boolean expressions*:

$$b \quad ::= \quad \texttt{true} \mid \texttt{false} \mid a \texttt{ < } a \mid b \texttt{ and } b \mid \neg b$$

- *actions*:

$$A \quad ::= \quad x \texttt{ := } a \mid \texttt{print } a$$

A toy language

Consider a language consisting of

- *arithmetic expressions*:

$$a \quad ::= \quad x \mid n \mid a \texttt{ + } a \mid a \texttt{ * } a$$

- *boolean expressions*:

$$b \quad ::= \quad \texttt{true} \mid \texttt{false} \mid a \texttt{ < } a \mid b \texttt{ and } b \mid \neg b$$

- *actions*:

$$A \quad ::= \quad x \texttt{ := } a \mid \texttt{print } a$$

- *commands / programs*:

$$\begin{aligned} c \quad ::= \quad & A \mid \\ & c\texttt{; } c \mid \texttt{if } b \texttt{ then } c \texttt{ else } c \mid \texttt{while } b \texttt{ do } c \end{aligned}$$

Graphs

A **graph** G consists of

- a set G_0 of *vertices*
- a set G_1 of *edges*
- *source* and *target* maps $\partial^-, \partial^+ : G_1 \to G_0$

Graphs

A **graph** G consists of

- a set G_0 of *vertices*
- a set G_1 of *edges*
- *source* and *target* maps $\partial^-, \partial^+ : G_1 \to G_0$

For *control flow graphs*, we moreover have

- distinguished *beginning* and *end* vertices: $s, t \in G_0$
- a *labeling*:

$$\ell \quad : \quad G_1 \quad \to \quad \mathcal{A} \sqcup \mathcal{B}$$

  into actions ($\mathcal{A}$) or boolean expressions ($\mathcal{B}$)

The control flow graph

To each program one can associate a graph by induction:

- action A:

$$G_A \quad = \quad s_A \bullet \xrightarrow{A} \bullet\, t_A$$

The control flow graph

To each program one can associate a graph by induction:

- action A:

$$G_A \quad = \quad s_A \bullet \xrightarrow{A} \bullet\, t_A$$

- sequence $p\,;q$:

The control flow graph

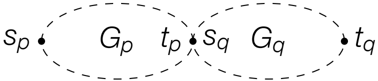
To each program one can associate a graph by induction:

- action A:

$$G_A \quad = \quad s_A \bullet \xrightarrow{A} \bullet\, t_A$$

- sequence $p\,;q$:

$$G_{p\,;q} \quad =$$



The control flow graph

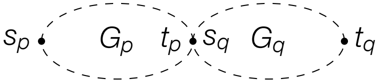
To each program one can associate a graph by induction:

- action A:

$$G_A \quad = \quad s_A \xrightarrow{A} t_A$$

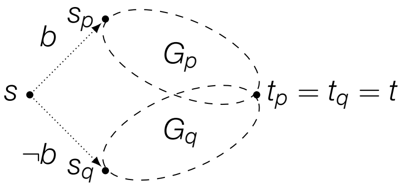
- sequence $p\,\texttt{;}\,q$:

$$G_{p;q} \quad =$$



- **`if`** b **`then`** p **`else`** q:

$$G \quad =$$



The control flow graph

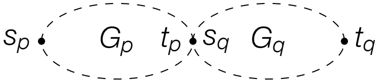
To each program one can associate a graph by induction:

- action A:

$$G_A \quad = \quad s_A \xrightarrow{A} t_A$$

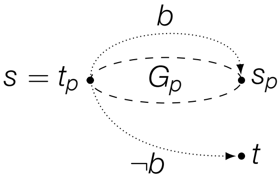
- sequence $p\,\texttt{;}\,q$:

$$G_{p;q} \quad =$$



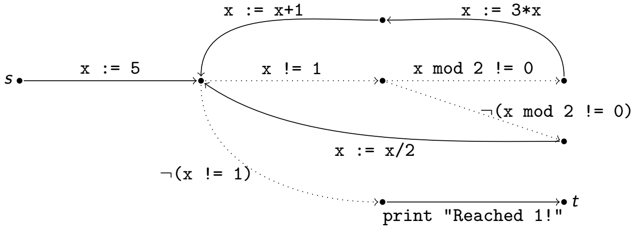
- `while` b `do` p:

$$G \quad =$$



Execution paths

An **execution path** of a program is a path starting from s_p in the graph G_p.



Denotational semantics

A **state** of the program is an element of $\Sigma = \mathbb{Z}^{\mathit{Var}}$:
variables contain integers.

Denotational semantics

A **state** of the program is an element of $\Sigma = \mathbb{Z}^{\mathit{Var}}$:
variables contain integers.

We can define a semantics by interpreting

- each arithmetic expression *a* as a function

$$[\![a]\!] \quad : \quad \Sigma \quad \to \quad \mathbb{Z}$$

e.g. in a state σ where $\sigma(\mathtt{x}) = 2$, we have

$$[\![\mathtt{x+3}]\!](\sigma) = [\![x]\!](\sigma) + [\![3]\!](\sigma) = \sigma(x) + 3 = 5$$

Denotational semantics

A **state** of the program is an element of $\Sigma = \mathbb{Z}^{\mathit{Var}}$:
variables contain integers.

We can define a semantics by interpreting

- each arithmetic expression *a* as a function
$$[\![a]\!] \quad : \quad \Sigma \quad \to \quad \mathbb{Z}$$
- each boolean expression *b* as a function
$$[\![b]\!] \quad : \quad \Sigma \quad \to \quad \{\bot, \top\}$$

Denotational semantics

A **state** of the program is an element of $\Sigma = \mathbb{Z}^{Var}$: variables contain integers.

We can define a semantics by interpreting

- each arithmetic expression a as a function

$$[\![a]\!] \quad : \quad \Sigma \quad \to \quad \mathbb{Z}$$

- each boolean expression b as a function

$$[\![b]\!] \quad : \quad \Sigma \quad \to \quad \{\bot, \top\}$$

- each action A as a function

$$[\![A]\!] \quad : \quad \Sigma \quad \to \quad \Sigma$$

e.g.

$$[\![\mathtt{x:=}a]\!](\sigma) \quad = \quad y \mapsto \begin{cases} [\![a]\!](\sigma) & \text{if } y = \mathtt{x} \\ \sigma(y) & \text{otherwise} \end{cases}$$

Denotational semantics

A **state** of the program is an element of $\Sigma = \mathbb{Z}^{Var}$: variables contain integers.

We can define a semantics by interpreting

- each arithmetic expression *a* as a function

$$[\![a]\!] \quad : \quad \Sigma \quad \to \quad \mathbb{Z}$$

- each boolean expression *b* as a function

$$[\![b]\!] \quad : \quad \Sigma \quad \to \quad \{\bot, \top\}$$

- each action *A* as a function

$$[\![A]\!] \quad : \quad \Sigma \quad \to \quad \Sigma$$

  e.g.

$$[\![A\,;B]\!](\sigma) \quad = \quad [\![B]\!]([\![A]\!](\sigma))$$

Denotational semantics

A **state** of the program is an element of $\Sigma = \mathbb{Z}^{\mathit{Var}}$:
variables contain integers.

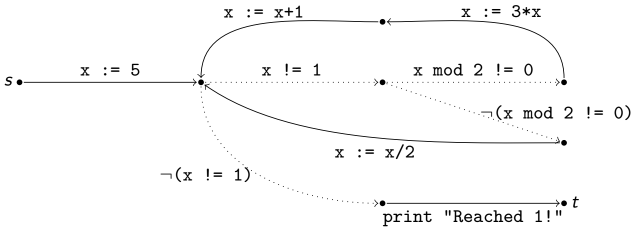
We can define a semantics by interpreting

- each arithmetic expression *a* as a function
$$[\![a]\!] \quad : \quad \Sigma \quad \to \quad \mathbb{Z}$$
- each boolean expression *b* as a function
$$[\![b]\!] \quad : \quad \Sigma \quad \to \quad \{\bot, \top\}$$
- each action *A* as a function
$$[\![A]\!] \quad : \quad \Sigma \quad \to \quad \Sigma$$

  e.g.
$$[\![\texttt{if } b \texttt{ then } A \texttt{ else } B]\!](\sigma) \quad = \quad \begin{cases} [\![A]\!](\sigma) & \text{if } [\![b]\!](\sigma) = \top \\ [\![B]\!](\sigma) & \text{otherwise} \end{cases}$$

Valid execution paths

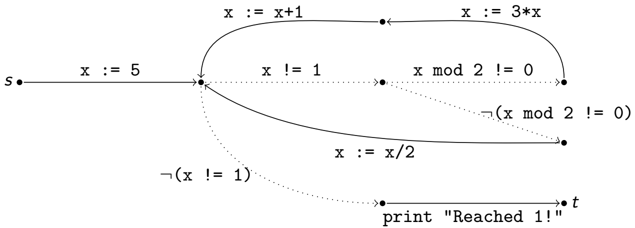
The semantics can be extended to execution paths (as the semantics of the sequence of labels, ignoring boolean conditions).



Valid execution paths

The semantics can be extended to execution paths (as the semantics of the sequence of labels, ignoring boolean conditions).

Given an initial environment, an execution path is **valid** when the boolean conditions are satisfied: these correspond to actual executions.



Verifying programs

A program with a distinguished set of *error vertices* is **correct** when there is no valid execution path with an error vertex as target.

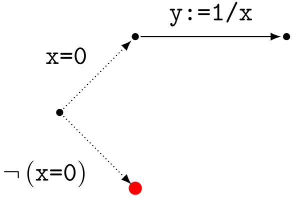
Verifying programs

A program with a distinguished set of *error vertices* is **correct** when there is no valid execution path with an error vertex as target.

Typical example:

```
if x = 0 then error else y := 1/x
```

corresponding to



Verifying programs

A program with a distinguished set of *error vertices* is **correct** when there is no valid execution path with an error vertex as target.

Reachability analysis can be performed by systematic exploration. This can be infinite because of

- loops,
- infinite sets of possible values,

and reachability is actually undecidable.

However, there are standard techniques which work well in practice such as *abstract interpretation*.

Verifying programs

Another point of view on verification consists in ensuring invariants on execution paths

$$p \quad : \quad s_p \quad \twoheadrightarrow \quad t_p$$

from the beginning to the end.

For this reason, the set of such paths is particularly important and called the **trace space**.

The terminology “space” suggests that it generally has more structure than a mere set...

CONCURRENT PROGRAMS

Concurrent programs

Concurrent programs consists in multiple processes running in parallel. In order to model this, we add a new construction to programs:

$$p \parallel q$$

means run p and q in parallel.

Sequential consistency

Assumption

Sequential consistency: the possible behaviors of $p \parallel q$ are the interleavings of the actions of p and q.

Sequential consistency

Assumption

Sequential consistency: the possible behaviors of $p \parallel q$ are the interleavings of the actions of p and q.

Example `(x1:=1; y1:=x2)` $\parallel$ `(x2:=1; y2:=x1)`

has the six following possible executions:

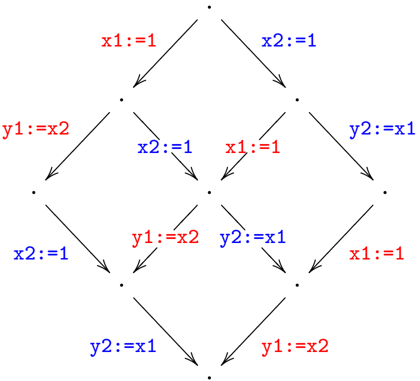
- `x1:=1; y1:=x2; x2:=1; y2:=x1`
- `x1:=1; x2:=1; y1:=x2; y2:=x1`
- `x1:=1; x2:=1; y2:=x1; y1:=x2`
- `x2:=1; x1:=1; y1:=x2; y2:=x1`
- `x2:=1; x1:=1; y2:=x1; y1:=x2`
- `x2:=1; y2:=x1; x1:=1; y1:=x2`

Sequential consistency

Assumption

Sequential consistency: the possible behaviors of $p \parallel q$ are the interleavings of the actions of p and q.

Example `(x1:=1; y1:=x2)` ‖ `(x2:=1; y2:=x1)`



Sequential consistency

Assumption

Sequential consistency: the possible behaviors of $p \parallel q$ are the interleavings of the actions of p and q.

Example `(x1:=1; y1:=x2)` $\parallel$ `(x2:=1; y2:=x1)`

Nowadays processors have **weak memory models**, because of which in the end you can even have

$$\mathtt{y1} = 0 \qquad \text{and} \qquad \mathtt{y2} = 0$$

if you are not careful...

Sequential consistency

```
#include <pthread.h>
#include <stdio.h>

volatile int x1=0, y1=0, x2=0, y2=0;

void* f1(void *arg)
{
  x1 = 1;
  y1 = x2;
  return NULL;
}

void* f2(void *arg)
{
  x2 = 1;
  y2 = x1;
  return NULL;
}
```

```
int main(void)
{
  pthread_t t1;
  pthread_t t2;
  int i;

  for (i=0; i<1000000; i++) {
    x1=0;
    x2=0;
    pthread_create(&t1, NULL, &f1, NULL);
    pthread_create(&t2, NULL, &f2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    if (y1 == 0 && y2 == 0)
      printf("Impossible case!\n");
  }

  return 0;
}
```

Semantics for parallel

This suggests that we define

$$G_{p \parallel q} \quad = \quad G_p \otimes G_q$$

where the tensor product has

- vertices: $V_{p \parallel q} = V_p \times V_q$
- edges: $E_{p \parallel q} = (E_p \times V_q) \sqcup (V_p \times E_q)$
- expected source and target

Semantics for parallel

This suggests that we define

$$G_{p \parallel q} \quad = \quad G_p \otimes G_q$$

where the tensor product has

- vertices: $V_{p \parallel q} = V_p \times V_q$
- edges: $E_{p \parallel q} = (E_p \times V_q) \sqcup (V_p \times E_q)$
- expected source and target

$$\begin{tikzcd} x \arrow[d, "e"'] \\ y \arrow[d, "f"'] \\ z \end{tikzcd} \otimes \begin{tikzcd} x' \arrow[d, "e'"'] \\ y' \arrow[d, "f'"'] \\ z' \end{tikzcd} = \begin{tikzcd}[column sep=small] & & x \otimes x' \arrow[dl, "e\otimes x'"'] \arrow[dr, "x\otimes e'"] & & \\ & y \otimes x' \arrow[dl, "f\otimes x'"'] \arrow[dr, "y\otimes e'"] & & x \otimes y' \arrow[dl, "e\otimes y'"'] \arrow[dr, "x\otimes f'"] & \\ z \otimes x' \arrow[dr, "z\otimes e'"'] & & y \otimes y' \arrow[dl, "f\otimes y'"'] \arrow[dr, "y\otimes f'"] & & x \otimes z' \arrow[dl, "e\otimes z'"] \\ & z \otimes y' \arrow[dr, "z\otimes f'"'] & & y \otimes z' \arrow[dl, "f\otimes z'"] & \\ & & z \otimes z' & & \end{tikzcd}$$

Verifying parallel programs

In theory, this is all we need to perform verification on programs, i.e. we can apply previously mentioned techniques...

Verifying parallel programs

In theory, this is all we need to perform verification on programs, i.e. we can apply previously mentioned techniques...

In practice, we face the **state space explosion problem**: given a program p of size k, the size of n copies of p in parallel

$$p \quad \otimes \quad p \quad \otimes \quad \cdots \quad \otimes \quad p$$

is

$$k^n$$

e.g.

$$\begin{tikzcd} \cdot \arrow[d] & & \cdot \arrow[d] & & & & \cdot \arrow[d] \\ \cdot \arrow[d] & \otimes & \cdot \arrow[d] & \otimes & \cdots & \otimes & \cdot \arrow[d] \\ \cdot & & \cdot & & & & \cdot \end{tikzcd}$$

making things impractical.

CUBICAL SEMANTICS OF CONCURRENT PROGRAMS

Toward geometric models

This suggests that we need to take some more structure of programs in account, i.e. study more carefully their geometry.

We will see that

- commutations between actions provide surfaces
- forbidden regions create holes

Commutation between actions

In order to face the state explosion problem, people have observed that some actions **commute**.

Commutation between actions

In order to face the state explosion problem, people have observed that some actions **commute**.

For instance, the actions of

$$\texttt{x := 1} \quad \| \quad \texttt{y := 2}$$

whose graph is

$$\begin{tikzcd}[column sep=small] & \cdot \arrow[dl, "\texttt{x:=1}"'] \arrow[dr, "\texttt{y:=2}"] & \\ \cdot \arrow[dr, "\texttt{y:=2}"'] & & \cdot \arrow[dl, "\texttt{x:=1}"] \\ & \cdot & \end{tikzcd}$$

do commute in the sense that

$$[\![\texttt{x:=1; y:=2}]\!] \quad = \quad [\![\texttt{y:=2; x:=1}]\!]$$

Commutation between actions

In order to face the state explosion problem, people have observed that some actions **commute**.

For instance, the actions of

$$\texttt{x := 1} \quad \| \quad \texttt{y := 2}$$

whose graph is

$$\begin{tikzcd}
& {} \arrow[dl, "\texttt{x:=1}"'] \arrow[dr, "\texttt{y:=2}"] & \\
{} \arrow[dr, "\texttt{y:=2}"'] & \diamond & {} \arrow[dl, "\texttt{x:=1}"] \\
& {} &
\end{tikzcd}$$

do commute in the sense that

$$[\![\texttt{x:=1; y:=2}]\!] \quad = \quad [\![\texttt{y:=2; x:=1}]\!]$$

Commutation between actions

In order to face the state explosion problem, people have observed that some actions **commute**.

For instance, the actions of

$$\texttt{x := 1} \quad \| \quad \texttt{x := 2}$$

whose graph is

$$\begin{tikzcd}
& \cdot \arrow[dl, "\texttt{x:=1}"'] \arrow[dr, "\texttt{x:=2}"] & \\
\cdot \arrow[dr, "\texttt{x:=2}"'] & & \cdot \arrow[dl, "\texttt{x:=1}"] \\
& \cdot &
\end{tikzcd}$$

do *not* commute in the sense that

$$[\![\texttt{x:=1; x:=2}]\!] \quad = \quad [\![\texttt{x:=2; x:=1}]\!]$$

Stability under refinement

Another reason why this commutation is important is that we want semantics to be stable under **refinement**.

For instance:

- the semantics of $A \parallel B$ is

$$\begin{tikzcd} \cdot \arrow[r, "A"] \arrow[d, "B"'] & \cdot \arrow[d, "B"] \\ \cdot \arrow[r, "A"'] & \cdot \end{tikzcd}$$

- if we replace A by $A_1 ; A_2$, we obtain

$$\begin{tikzcd} \cdot \arrow[r, "A_1"] \arrow[d, "B"'] & \cdot \arrow[r, "A_2"] & \cdot \arrow[d, "B"] \\ \cdot \arrow[r, "A_1"'] & \cdot \arrow[r, "A_2"'] & \cdot \end{tikzcd}$$

- the semantics of $(A_1 ; A_2) \parallel B$ is

$$\begin{tikzcd} \cdot \arrow[r, "A_1"] \arrow[d, "B"'] & \cdot \arrow[r, "A_2"] \arrow[d, "B"] & \cdot \arrow[d, "B"] \\ \cdot \arrow[r, "A_1"'] & \cdot \arrow[r, "A_2"'] & \cdot \end{tikzcd}$$

True concurrency

This idea of taking commutations in account is called **true concurrency**: Mazurkiewicz traces, asynchronous automata, etc.

True concurrency

This idea of taking commutations in account is called **true concurrency**: Mazurkiewicz traces, asynchronous automata, etc.

There is no reason why we should stop at commutation of *two* actions...

Higher commutations

- commutation of two actions is indicated by a square

$$
\begin{tikzcd}
 & {} \arrow[dl, "A"'] \arrow[dr, "B"] & \\
{} \arrow[dr, "B"'] \arrow[rr, phantom, "\Diamond"] & & {} \arrow[dl, "A"] \\
 & {} &
\end{tikzcd}
$$

Higher commutations

- commutation of two actions is indicated by a square

$$
\begin{tikzcd}
& {} \arrow[dl, "A"'] \arrow[dr, "B"] & \\
{} \arrow[dr, "B"'] & \diamond & {} \arrow[dl, "A"] \\
& {} &
\end{tikzcd}
$$

- commutation of three actions is indicated by a cube

$$
\begin{tikzcd}
{} \arrow[rr, "A"] \arrow[dr, "B"] \arrow[dd, "C"'] & & {} \arrow[dr, "B"] \arrow[dd, "C"] & \\
& {} \arrow[rr, "A"] \arrow[dd, "C"] & & {} \arrow[dd, "C"] \\
{} \arrow[rr, "A"] \arrow[dr, "B"'] & & {} \arrow[dr, "B"] & \\
& {} \arrow[rr, "A"'] & & {}
\end{tikzcd}
$$

Higher commutations

- commutation of two actions is indicated by a square

$$
\begin{tikzcd}
& {} \arrow[dl, "A"'] \arrow[dr, "B"] & \\
{} \arrow[dr, "B"'] & \diamond & {} \arrow[dl, "A"] \\
& {} &
\end{tikzcd}
$$

- commutation of three actions is indicated by a cube

$$
\begin{tikzcd}
{} \arrow[rr, "A"] \arrow[dr, "B"] \arrow[dd, "C"'] & & {} \arrow[dr, "B"] \arrow[dd, "C"] & \\
& {} \arrow[rr, "A"] \arrow[dd, "C"] & & {} \arrow[dd, "C"] \\
{} \arrow[rr, "A"] \arrow[dr, "B"'] & & {} \arrow[dr, "B"] & \\
& {} \arrow[rr, "A"'] & & {}
\end{tikzcd}
$$

- etc.

Precubical sets

A **precubical set** C consists of, for every $n \in \mathbb{N}$,

- a set C_n of *n-cubes*
- face maps
  $$\partial_i^\alpha \quad : \quad C_n \quad \to \quad C_{n-1}$$
  for $\alpha \in \{-,+\}$ and $0 \leq i < n$, such that
  $$\partial_j^\beta \partial_i^\alpha \quad = \quad \partial_i^\alpha \partial_{j+1}^\beta \quad : \quad C_{n+1} \quad \to \quad C_{n-1}$$
  for $0 \leq i \leq j < n$ and $\alpha, \beta \in \{-,+\}$.

A **precubical set** C consists of, for every $n \in \mathbb{N}$,

- a set C_n of *n-cubes*
- face maps

$$\partial_i^\alpha \quad : \quad C_n \quad \to \quad C_{n-1}$$

for $\alpha \in \{-,+\}$ and $0 \leq i < n$, such that

$$\partial_j^\beta \partial_i^\alpha \quad = \quad \partial_i^\alpha \partial_{j+1}^\beta \quad : \quad C_{n+1} \quad \to \quad C_{n-1}$$

for $0 \leq i \leq j < n$ and $\alpha, \beta \in \{-,+\}$.

For instance an element $x \in C_2$ can be pictured as

$$\begin{tikzcd}
\partial_0^- \partial_1^-(x) \arrow[r, "\partial_1^-(x)"] \arrow[d, "\partial_0^-(x)"'] \arrow[dr, phantom, "x"] & \partial_0^+ \partial_1^-(x) \arrow[d, "\partial_0^+(x)"] \\
\partial_0^- \partial_1^+(x) \arrow[r, "\partial_1^+(x)"'] & \partial_0^+ \partial_1^+(x)
\end{tikzcd}$$

Precubical sets

A **precubical set** C consists of, for every $n \in \mathbb{N}$,

- a set C_n of *n-cubes*
- face maps

  $$\partial_i^\alpha \quad : \quad C_n \quad \to \quad C_{n-1}$$

  for $\alpha \in \{-,+\}$ and $0 \leq i < n$, such that

  $$\partial_j^\beta \partial_i^\alpha \quad = \quad \partial_i^\alpha \partial_{j+1}^\beta \quad : \quad C_{n+1} \quad \to \quad C_{n-1}$$

  for $0 \leq i \leq j < n$ and $\alpha, \beta \in \{-,+\}$.

Note that there is an underlying graph with

- C_0 as vertices
- C_1 as edges
- $\partial^-, \partial^+ : C_1 \to C_0$ as source and target maps

Precubical sets

A **precubical set** C consists of, for every $n \in \mathbb{N}$,

- a set C_n of *n-cubes*
- face maps

$$\partial_i^\alpha \quad : \quad C_n \quad \to \quad C_{n-1}$$

for $\alpha \in \{-,+\}$ and $0 \leq i < n$, such that

$$\partial_j^\beta \partial_i^\alpha \quad = \quad \partial_i^\alpha \partial_{j+1}^\beta \quad : \quad C_{n+1} \quad \to \quad C_{n-1}$$

for $0 \leq i \leq j < n$ and $\alpha, \beta \in \{-,+\}$.

Terminology: a precubical set with a distinguished vertex is called an **higher-dimensional automaton** (or **HDA**)

The **tensor product**

$$C \otimes D$$

of two precubical sets has n-cubes

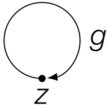
$$(C \otimes D)_n \quad = \quad \coprod_{i+j=n} C_i \times D_j$$

and boundary

$$\begin{array}{rcrcl} \partial_k^\alpha & : & (C \otimes D)_n & \to & (C \otimes D)_{n-1} \\ & & x \otimes y & \mapsto & \begin{cases} \partial_k^\alpha(x) \otimes y & \text{if } 0 \leq k < i \\ x \otimes \partial_{k-i}^\alpha(y) & \text{if } i \leq k < n \end{cases} \end{array}$$

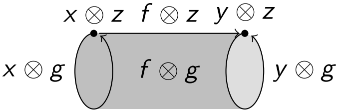
Tensor product

For instance with

$I = x \xrightarrow{f} y$ and $S^1 =$ 

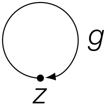
one has

- $I \otimes S^1$:



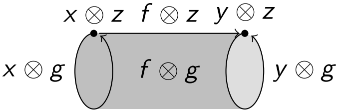
Tensor product

For instance with

$I = x \xrightarrow{f} y$ and $S^1 =$ 

one has

- $I \otimes S^1$:



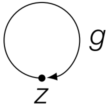
- $I \otimes I$:

$$\begin{tikzcd}
x \otimes y \arrow[r, "f \otimes y"] & y \otimes y \\
x \otimes x \arrow[u, "x \otimes f"] \arrow[r, "f \otimes x"'] \arrow[ur, phantom, "f \otimes g"] & y \otimes x \arrow[u, "y \otimes f"']
\end{tikzcd}$$

Tensor product

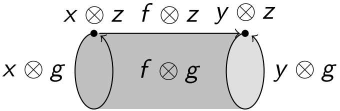
For instance with

$$I = x \xrightarrow{f} y \qquad \text{and} \qquad S^1 =$$

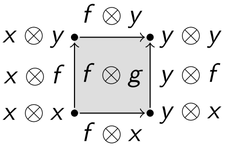


one has

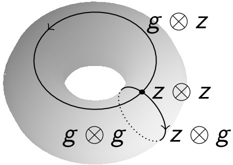
- $I \otimes S^1$:



- $I \otimes I$:



- $S^1 \otimes S^1$:



Cubical semantics

The **cubical semantics** C_p of a program p is defined as before, but in precubical sets, e.g.

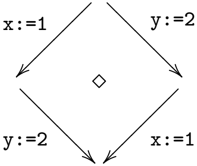
$$C_{p \parallel q} \quad = \quad C_p \otimes C_q$$

where the tensor product is now taken in precubical sets.

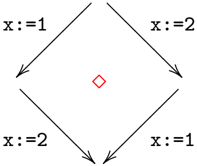
Cubical semantics

For now, the cubical semantics is too simple:

- `x := 1 || y := 2`



- `x := 1 || x := 2`



We need to carve holes!

Mutexes

In practice, systems provide primitives to ensure that

- two threads will not access a variable at the same time: *mutual exclusion*
- some sequences of actions are atomic

Mutexes

In practice, systems provide primitives to ensure that

- two threads will not access a variable at the same time: *mutual exclusion*
- some sequences of actions are atomic

A **mutex** a is a resource which you can

- *lock*: $\mathtt{P}_a$
- *release*: $\mathtt{V}_a$

and the system will guarantee that at most one process will have the resource at a given time (locks can block).

Mutexes

In practice, systems provide primitives to ensure that

- two threads will not access a variable at the same time: *mutual exclusion*
- some sequences of actions are atomic

A **mutex** a is a resource which you can

- *lock*: $\mathtt{P}_a$
- *release*: $\mathtt{V}_a$

and the system will guarantee that at most one process will have the resource at a given time (locks can block).

We add to our language actions of the form

$$\mathtt{P}_a \qquad \text{and} \qquad \mathtt{V}_a$$

More general resources

More generally, we consider **resources** a with arbitrary *capacity*

$$\kappa_a \in \mathbb{N}$$

which can be locked by at most κ_a threads at a time.

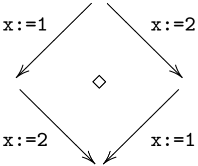
A mutex is a resource of capacity 1.

Protecting variables

The semantics of the program

```
x := 1   ||   x := 2
```

is

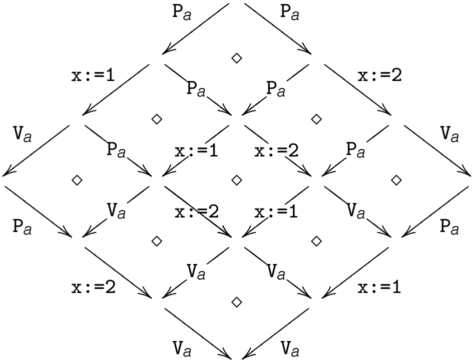


Protecting variables

The semantics of the program

$$(\mathtt{P}_a;\mathtt{x}\ \mathtt{:=}\ \mathtt{1};\mathtt{V}_a) \quad \| \quad (\mathtt{P}_a;\mathtt{x}\ \mathtt{:=}\ \mathtt{2};\mathtt{V}_a)$$

is

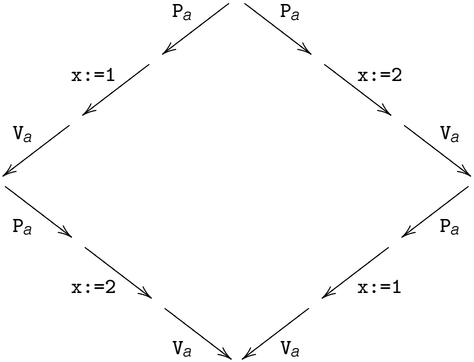


Protecting variables

The semantics of the program

$$\texttt{(P}_a\texttt{;x := 1;V}_a\texttt{)} \quad \| \quad \texttt{(P}_a\texttt{;x := 2;V}_a\texttt{)}$$

is



Protecting variables

In a program of the form

$$(\mathtt{P}_a\mathtt{;}p\mathtt{;V}_a) \quad \| \quad (\mathtt{P}_a\mathtt{;}q\mathtt{;V}_a)$$

the subprograms *p* and *q* are

- **mutually exclusive**:
  they cannot be executed at the same time

Protecting variables

In a program of the form

$$(\mathtt{P}_a;p;\mathtt{V}_a) \quad \| \quad (\mathtt{P}_a;q;\mathtt{V}_a)$$

the subprograms *p* and *q* are

- **mutually exclusive**:
  they cannot be executed at the same time
- **atomic**:
  once we start executing *p*, we go on until the end
  i.e. there is no interleaving with other threads

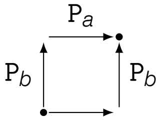
Conservative programs

A program is **conservative** when the resource consumption only depends on the vertex (not on the path reaching the vertex).

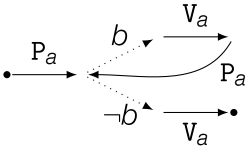
Conservative programs

A program is **conservative** when the resource consumption only depends on the vertex (not on the path reaching the vertex).

- conservative programs:



$\mathtt{P}_a \,||\, \mathtt{P}_b$

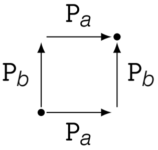


$\mathtt{P}_a\mathtt{;(while}\ b\ \mathtt{do\ (V}_a\mathtt{;P}_a\mathtt{));V}_a$

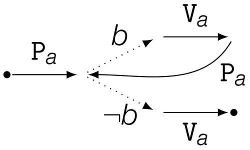
Conservative programs

A program is **conservative** when the resource consumption only depends on the vertex (not on the path reaching the vertex).

- conservative programs:

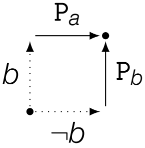


$\mathtt{P}_a \parallel \mathtt{P}_b$

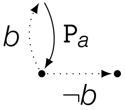


$\mathtt{P}_a\mathtt{;(while}\ b\ \mathtt{do}\ \mathtt{(V}_a\mathtt{;P}_a\mathtt{));V}_a$

- non-conservative programs:



$\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_a\ \mathtt{else}\ \mathtt{P}_b$



$\mathtt{while}\ b\ \mathtt{do}\ \mathtt{P}_a$

Conservative programs

A program is conservative when the **resource consumption**

$$\Delta(p) \quad : \quad \mathcal{R} \quad \to \quad \mathbb{Z}$$

is well defined:

$$\begin{array}{rclrcl}
\Delta(A) & = & 0 & & & \\
\Delta(\mathtt{P}_a) & = & -\delta_a & \Delta(\mathtt{V}_a) & = & \delta_a \\
\Delta(p\,\mathtt{;}\,q) & = & \Delta(p) + \Delta(q) & \Delta(p \parallel q) & = & \Delta(p) + \Delta(q)
\end{array}$$

$$\begin{array}{rcll}
\Delta(\mathtt{if}\ b\ \mathtt{then}\ p\ \mathtt{else}\ q) & = & \Delta(p) & \text{whenever } \Delta(p) = \Delta(q) \\
\Delta(\mathtt{while}\ b\ \mathtt{do}\ p) & = & 0 & \text{whenever } \Delta(p) = 0
\end{array}$$

where

$$\delta_a(b) \quad = \quad \begin{cases} 1 & \text{if } b = a \\ 0 & \text{otherwise} \end{cases}$$

Forbidden vertices

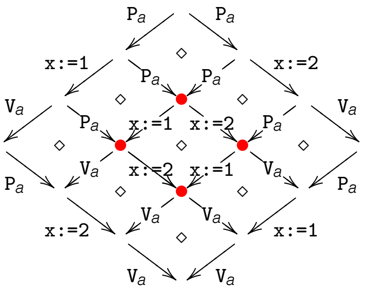
In a conservative program p a vertex $x \in C_p$ is *valid* when, given a path

$$t \quad : \quad s_p \quad \twoheadrightarrow \quad x$$

for every resource a, we have

$$0 \quad \leq \quad \kappa_a + \Delta(t)(a) \quad \leq \quad \kappa_a$$

It is **forbidden** otherwise.



Cubical semantics

The **cubical semantics** $\check{C}_p$ of a conservative program p is the precubical set

$$C_p$$

from which we have removed

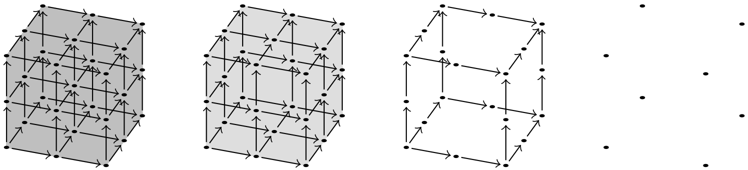
- forbidden vertices
- cubes having forbidden vertices as iterated faces

Influence of the capacity

Consider the program p:

$$\texttt{P}_a\texttt{;V}_a \quad \| \quad \texttt{P}_a\texttt{;V}_a \quad \| \quad \texttt{P}_a\texttt{;V}_a$$

Depending on the capacity of a, the geometric semantics is



$\kappa_a \geq 3$ $\qquad$ $\kappa_a = 2$ $\qquad$ $\kappa_a = 1$ $\qquad$ $\kappa_a = 0$

Equivalent paths

Given a precubical set, we write $\sim$ for the smallest equivalence relation on paths

- such that

$$A \cdot B \quad \sim \quad B' \cdot A'$$

  for every 2-cube

$$\begin{tikzcd}
& {} \arrow[dl, "A"'] \arrow[dr, "B'"] \arrow[dd, phantom, "\diamond"] & \\
{} \arrow[dr, "B"'] & & {} \arrow[dl, "A'"] \\
& {} &
\end{tikzcd}$$

- which is a congruence:

$$t \sim t' \qquad \text{and} \qquad u \sim u'$$

  implies

$$t \cdot u \quad \sim \quad t' \cdot u'$$

  for concatenable paths

Coherent programs

A program is **coherent** when all concurrent accesses to variables are protected by mutexes.

Coherent programs

A program is **coherent** when all concurrent accesses to variables are protected by mutexes.

Proposition

For coherent two equivalent paths have the same semantics:

$$t \sim t' \qquad \text{implies} \qquad [\![t]\!] = [\![t']\!]$$

Coherent programs

A program is **coherent** when all concurrent accesses to variables are protected by mutexes.

Proposition

For coherent two equivalent paths have the same semantics:

$$t \sim t' \qquad \text{implies} \qquad [\![t]\!] = [\![t']\!]$$

In order to check all the behaviors, it is enough to check one representative in each equivalence class of executions under $\sim$.

The fundamental category

Given a precubical set C, the **fundamental category** $\vec{\Pi}_1(C)$ has

- vertices of C as objects
- paths up to equivalence as morphisms

and composition is given by concatenation.

The terminology suggests that equivalence corresponds to some form of homotopy, we will get back to this.

The trace space

Given a program p, we are mostly interested in the **trace space**:

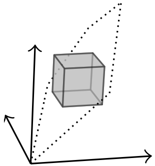
$$\vec{\Pi}_1(\check{C}_p)(s_p, t_p)$$

The trace space

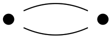
Given a program p, we are mostly interested in the **trace space**:

$$\vec{\Pi}_1(\check{C}_p)(s_p, t_p)$$

We are also interested into finer properties, e.g. how many ways there are to show that two paths are equivalent, etc.



$\rightsquigarrow$



Forgetting about values

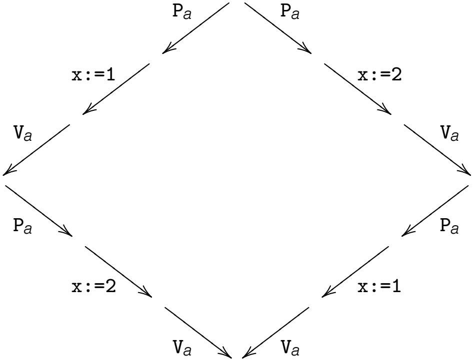
The geometry does not depend on actions manipulating values!

Forgetting about values

The geometry does not depend on actions manipulating values!

The semantics of

$$(\mathtt{P}_a;\mathtt{x\ :=\ 1};\mathtt{V}_a) \quad \| \quad (\mathtt{P}_a;\mathtt{x\ :=\ 2};\mathtt{V}_a)$$



Forgetting about values

The geometry does not depend on actions manipulating values!

The semantics of

$$(\mathtt{P}_a;\mathtt{V}_a) \quad \| \quad (\mathtt{P}_a;\mathtt{V}_a)$$

$$\begin{tikzcd}
 & & \bullet \arrow[dl, "\mathtt{P}_a"'] \arrow[dr, "\mathtt{P}_a"] & & \\
 & \bullet \arrow[dl, "\mathtt{V}_a"'] & & \bullet \arrow[dr, "\mathtt{V}_a"] & \\
\bullet \arrow[dr, "\mathtt{P}_a"'] & & & & \bullet \arrow[dl, "\mathtt{P}_a"] \\
 & \bullet \arrow[dr, "\mathtt{V}_a"'] & & \bullet \arrow[dl, "\mathtt{V}_a"] & \\
 & & \bullet & &
\end{tikzcd}$$

DIRECTED TOPOLOGICAL MODELS

Geometric realization

We write I for the **interval** topological space

$$I \quad = \quad [0,1]$$

Geometric realization

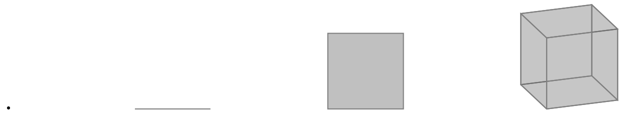
We write I for the **interval** topological space

$$I \quad = \quad [0,1]$$

The **topological *n*-cube** is defined as

$$I^n$$

For instance,



$\ldots$

$I^0 \qquad I^1 \qquad I^2 \qquad I^3 \qquad \ldots$

Geometric realization

We write I for the **interval** topological space

$$I \quad = \quad [0,1]$$

The **topological** n**-cube** is defined as

$$I^n$$

With $0 \leq i < n$ and $\alpha \in \{-,+\}$, we write

$$\iota_i^\alpha \quad : \quad I^{n-1} \to I^n$$

for the canonical inclusions:

$$\begin{aligned}
\iota_i^-(x_0,\ldots,x_{n-2}) \quad &= \quad (x_0,\ldots,0,\ldots,x_{n-2}) \\
\iota_i^+(x_0,\ldots,x_{n-2}) \quad &= \quad (x_0,\ldots,1,\ldots,x_{n-2})
\end{aligned}$$

A precubical set can be seen as a topological space,
via the **geometric realization** functor

$$|-| \quad : \quad \mathbf{PCSet} \quad \to \quad \mathbf{Top}$$

Geometric realization

A precubical set can be seen as a topological space, via the **geometric realization** functor

$$|-| \quad : \quad \mathbf{PCSet} \quad \to \quad \mathbf{Top}$$

which to a precubical set C associates the space obtained by gluing cubes as described by C:

$$|C| \quad = \quad \coprod_{n\in\mathbb{N}} (C_n \times I^n)/\approx$$

where C_n has discrete topology and

$$(\partial_i^\alpha(x), p) \quad \approx \quad (x, \iota_i^\alpha(p))$$

Geometric realization

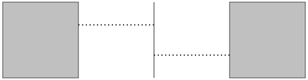
Given the precubical set

$$\begin{tikzcd} x_1 \arrow[d, "f_1"'] \arrow[dr, phantom, "\phi"] & x \arrow[l, "g_1"'] \arrow[r, "g_2"] \arrow[d, "f"] \arrow[dr, phantom, "\psi"] & x_2 \arrow[d, "f_2"] \\ y_1 & y \arrow[l, "h_1"] \arrow[r, "h_2"'] & y_2 \end{tikzcd}$$

The cubes ϕ, f, ψ will induce cubes in the geometric realization:

$$|C| \quad = \quad I^2 \sqcup I^1 \sqcup I^2 \sqcup \ldots$$

and the identification $(\partial_i^\alpha(x), p) \approx (x, \iota_i^\alpha(p))$ means



Geometric realization

Proposition

Geometric realization sends tensor product to cartesian one:

$$|C \otimes D| \quad = \quad |C| \times |D|$$

What makes **Top**
a good place to take
geometric realization?

Graphs

If we write $\square_1$ for the category

$$0 \underset{\partial^+}{\overset{\partial^-}{\rightrightarrows}} 1$$

a functor

$$G \quad : \quad \square_1^{\mathsf{op}} \quad \to \quad \mathbf{Set}$$

is characterized by two sets

$$G(0) \qquad\qquad G(1)$$

and two functions

$$G(\partial^-), G(\partial^+) \quad : \quad G(1) \quad \to \quad G(0)$$

i.e. a *graph*.

Note that every graph can be canonically obtained as a colimit involving the two graphs

$$G_0 = \cdot \qquad\qquad \text{and} \qquad\qquad G_1 = \cdot \longrightarrow \cdot$$

For instance,

$$\begin{tikzcd} \cdot \arrow[r] \arrow[rr, bend left] & \cdot \arrow[r] & \cdot \end{tikzcd}$$

is the colimit of

$$\begin{tikzcd} & & G_1 & & \\ G_0 \arrow[urr] \arrow[dr] & & G_0 \arrow[dl] \arrow[dr] & & G_0 \arrow[ull] \arrow[dl] \\ & G_1 & & G_1 & \end{tikzcd}$$

Note that every graph can be canonically obtained as a colimit involving the two graphs

$$G_0 = \cdot \qquad\qquad \text{and} \qquad\qquad G_1 = \ \cdot \longrightarrow \cdot$$

For instance,

$$\begin{tikzcd} \cdot \arrow[r] \arrow[rr, bend left] & \cdot \arrow[r] & \cdot \end{tikzcd}$$

is the colimit of

$$\begin{tikzcd} & & G_1 & & \\ G_0 \arrow[urr] \arrow[dr] & & G_0 \arrow[dl] \arrow[dr] & & G_0 \arrow[ull] \arrow[dl] \\ & G_1 & & G_1 & \end{tikzcd}$$

This means that a cocontinuous functor is uniquely determined by its image on G_0 and G_1.

Precubical sets

A precubical set can be seen as a **presheaf** functor

$$C \quad : \quad \square^{\mathrm{op}} \to \mathbf{Set}$$

where $\square$ is the category with

- $n \in \mathbb{N}$ as objects
- morphisms are generated by

$$\partial_i^\alpha \quad : \quad n-1 \quad \to \quad n$$

  quotiented by relations

$$\partial_i^\alpha \partial_j^\beta \quad = \quad \partial_{j+1}^\beta \partial_i^\alpha \quad : \quad n-1 \quad \to \quad n+1$$

To sum up: $\quad \mathbf{PCSet} \quad \cong \quad \hat{\square}$

A cube object

We can remark that we have a functor

$$F \quad : \quad \square \quad \to \quad \mathbf{Top}$$

defined by

$$F(n) \quad = \quad I^n$$

and

$$F(\partial_i^\alpha) \quad = \quad \iota_i^\alpha \quad : \quad I^{n-1} \quad \to \quad I^n$$

encoding the realization of all the standard cube and their faces.

This is all we need!

Presheaves as cocompletion

A category is **cocomplete** when it has coproducts and quotients.

Presheaves as cocompletion

A category is **cocomplete** when it has coproducts and quotients.

Proposition

The category $\hat{\square}$ is the free cocompletion of $\square$: given a functor $F : \square \to \mathcal{C}$, with $\mathcal{C}$ cocomplete, there is a unique cocontinuous functor $\tilde{F}$ such that

$$\begin{tikzcd} \square \arrow[r, "F"] \arrow[d, "Y"'] & \mathcal{C} \\ \hat{\square} \arrow[ur, dotted, "\tilde{F}"'] & \end{tikzcd}$$

where Y is the Yoneda embedding.

Presheaves as cocompletion

A category is **cocomplete** when it has coproducts and quotients.

Proposition

The category $\hat{\square}$ is the free cocompletion of $\square$: given a functor $F : \square \to \mathcal{C}$, with $\mathcal{C}$ cocomplete, there is a unique cocontinuous functor $\tilde{F}$ such that

$$\begin{tikzcd}
\square \arrow[r, "F"] \arrow[d, "Y"'] & \mathcal{C} \\
\hat{\square} \arrow[ur, dotted, "\tilde{F}"'] &
\end{tikzcd}$$

where Y is the Yoneda embedding.

We obtain geometric realization as
$$\begin{tikzcd}
\square \arrow[r, "F"] \arrow[d, "Y"'] & \mathbf{Top} \\
\hat{\square} \arrow[ur, dotted, "|-|"'] &
\end{tikzcd}.$$

Presheaves as cocompletion

A category is **cocomplete** when it has coproducts and quotients.

Proposition

The category $\hat{\square}$ is the free cocompletion of $\square$: given a functor $F : \square \to \mathcal{C}$, with $\mathcal{C}$ cocomplete, there is a unique cocontinuous functor $\tilde{F}$ such that

$$\begin{tikzcd} \square \arrow[r, "F"] \arrow[d, "Y"'] & \mathcal{C} \\ \hat{\square} \arrow[ur, dotted, "\tilde{F}"'] & \end{tikzcd}$$

where Y is the Yoneda embedding.

We obtain geometric realization as $\begin{tikzcd} \square \arrow[r, "F"] \arrow[d, "Y"'] & \mathbf{Top} \\ \hat{\square} \arrow[ur, dotted, "{|-| = \mathrm{Lan}_Y F}"'] & \end{tikzcd}$.

Presheaves as cocompletion

A category is **cocomplete** when it has coproducts and quotients.

Proposition

The category $\hat{\square}$ is the free cocompletion of $\square$: given a functor $F : \square \to \mathcal{C}$, with $\mathcal{C}$ cocomplete, there is a unique cocontinuous functor $\tilde{F}$ such that

$$\begin{tikzcd} \square \arrow[r, "F"] \arrow[d, "Y"'] & \mathcal{C} \\ \hat{\square} \arrow[ur, dotted, "\tilde{F}"'] & \end{tikzcd}$$

where Y is the Yoneda embedding.

This point of view will be useful to consider other "geometric realizations".

What about
geometric realizations
of
geometric semantics
of
concurrent programs?

Geometric semantics

The **geometric semantics** S_p of a program p is the space

$$S_p \quad = \quad \left|\check{C}_p\right|$$

Simple programs

A **simple program** p is of the form

$$p_1 \quad \| \quad p_2 \quad \| \quad \dots \quad \| \quad p_n$$

where each p_i is a sequence of actions ($\mathtt{P}_a$ / $\mathtt{V}_a$).

In this case, the geometric semantics is of the form

$$\begin{aligned} S_p &= \left|\check{C}_p\right| \\ &= |C_p \setminus F_p| \\ &= \left|I^{\otimes n} \setminus F_p\right| \\ &= I^n \setminus \bigcup_{i=1}^{l} R^i \end{aligned}$$

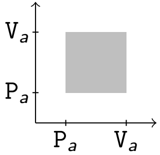
where R^i are open hyperrectangles.

Geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{V}_a \quad \| \quad \mathtt{P}_a;\mathtt{V}_a$$

with $\kappa_a = 1$, is

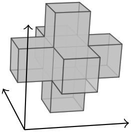


Geometric semantics

The geometric semantics of

$$\mathtt{P}_a\mathtt{;V}_a \quad \| \quad \mathtt{P}_a\mathtt{;V}_a \quad \| \quad \mathtt{P}_a\mathtt{;V}_a$$

with $\kappa_a = 1$, is

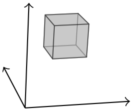


Geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{V}_a \quad \| \quad \mathtt{P}_a;\mathtt{V}_a \quad \| \quad \mathtt{P}_a;\mathtt{V}_a$$

with $\kappa_a = 2$, is

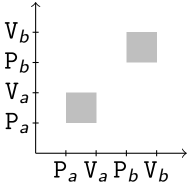


Geometric semantics

The geometric semantics of

$$\mathtt{P}_a\,;\mathtt{V}_a\,;\mathtt{P}_b\,;\mathtt{V}_b \quad \| \quad \mathtt{P}_a\,;\mathtt{V}_a\,;\mathtt{P}_b\,;\mathtt{V}_b$$

with $\kappa_a = 1$, is

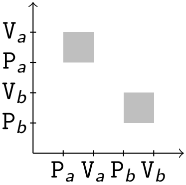


Geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{V}_a;\mathtt{P}_b;\mathtt{V}_b \quad \| \quad \mathtt{P}_b;\mathtt{V}_b;\mathtt{P}_a;\mathtt{V}_a$$

with $\kappa_a = 1$, is

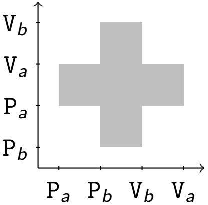


Geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{P}_b;\mathtt{V}_b;\mathtt{V}_a \quad \| \quad \mathtt{P}_b;\mathtt{P}_a;\mathtt{V}_a;\mathtt{V}_b$$

with $\kappa_a = 1$, is

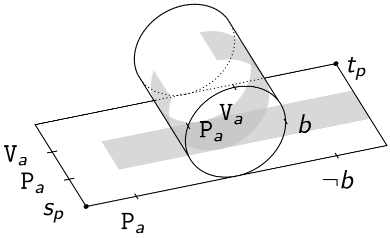


Geometric semantics

The geometric semantics of

$$\texttt{P}_a\texttt{; while } b \texttt{ do (V}_a\texttt{;P}_a\texttt{)} \quad \| \quad \texttt{P}_a\texttt{;V}_a$$

with $\kappa_a = 1$, is



Paths in geometric semantics

The geometric semantics S is equipped with beginning and end points s and t.

We can expect that

- the paths
$$p \quad : \quad I \quad \to \quad S$$
with $p(0) = s$ and $p(1) = t$ to correspond to executions of the program,

Paths in geometric semantics

The geometric semantics S is equipped with beginning and end points s and t.

We can expect that

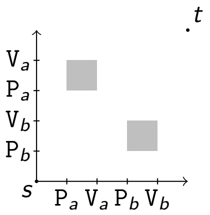
- the paths
$$p \quad : \quad I \quad \to \quad S$$
with $p(0) = s$ and $p(1) = t$ to correspond to executions of the program,
- deformations of paths correspond to equivalence between paths.

Paths in geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{V}_a;\mathtt{P}_b;\mathtt{V}_b \quad \| \quad \mathtt{P}_b;\mathtt{V}_b;\mathtt{P}_a;\mathtt{V}_a$$

is

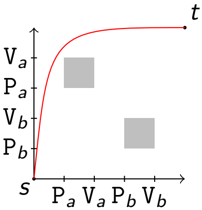


Paths in geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{V}_a;\mathtt{P}_b;\mathtt{V}_b \quad \| \quad \mathtt{P}_b;\mathtt{V}_b;\mathtt{P}_a;\mathtt{V}_a$$

is

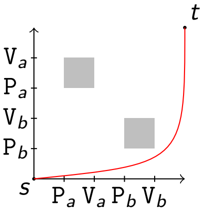


Paths in geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{V}_a;\mathtt{P}_b;\mathtt{V}_b \quad \| \quad \mathtt{P}_b;\mathtt{V}_b;\mathtt{P}_a;\mathtt{V}_a$$

is

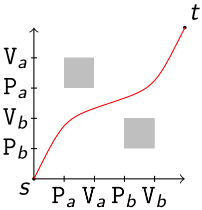


Paths in geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{V}_a;\mathtt{P}_b;\mathtt{V}_b \quad \| \quad \mathtt{P}_b;\mathtt{V}_b;\mathtt{P}_a;\mathtt{V}_a$$

is

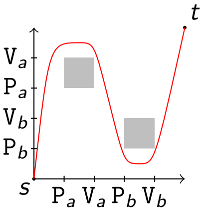


Paths in geometric semantics

The geometric semantics of

$$\mathtt{P}_a;\mathtt{V}_a;\mathtt{P}_b;\mathtt{V}_b \quad \| \quad \mathtt{P}_b;\mathtt{V}_b;\mathtt{P}_a;\mathtt{V}_a$$

is



Actual executions correspond to increasing paths!

Directed paths

We want to axiomatize a notion of **directed path**.

Directed paths

We want to axiomatize a notion of **directed path**.

In practice, we will consider $\mathbb{R}^n$ with directed paths

$$p \quad : \quad I \quad \to \quad \mathbb{R}^n$$

being those for which

$$I \xrightarrow{p} \mathbb{R}^n \xrightarrow{\pi_i} \mathbb{R}$$

is increasing for every projection π_i.

Directed spaces

Definition (Grandis)

A **d-space** (X, dX) consists of a topological space X together with a set dX of paths, called *directed*, such that

- dX contains constant paths
- dX is closed under composition with increasing reparametrizations $I \to I$
- dX is closed under concatenation

We write **dTop** for the category whose morphisms preserve directed paths.

Directed spaces

Definition (Grandis)

A **d-space** (X, dX) consists of a topological space X together with a set dX of paths, called *directed*, such that

- dX contains constant paths
- dX is closed under composition with increasing reparametrizations $I \to I$
- dX is closed under concatenation

We write **dTop** for the category whose morphisms preserve directed paths.

The category **dTop** is complete and cocomplete.

Example

Any topological space X can canonically be seen as a d-space in two ways:

- take dX the set of all paths in X, or
- take dX the set of constant paths in X

Example

Any topological space X can canonically be seen as a d-space in two ways:

- take dX the set of all paths in X, or
- take dX the set of constant paths in X

Example

The **directed interval** $\vec{I}$ is $I = [0, 1]$ with increasing functions $p : I \to I$ as directed paths.

Example

Any topological space X can canonically be seen as a d-space in two ways:

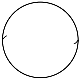
- take dX the set of all paths in X, or
- take dX the set of constant paths in X

Example

The **directed interval** $\vec{I}$ is $I = [0, 1]$ with increasing functions $p : I \to I$ as directed paths.

Example

The **directed circle** $\vec{S}^1$ and **directed complex plane** $\vec{\mathbb{C}}$:



Example

Any topological space X can canonically be seen as a d-space in two ways:

- take dX the set of all paths in X, or
- take dX the set of constant paths in X

Example

The **directed interval** $\vec{I}$ is $I = [0, 1]$ with increasing functions $p : I \to I$ as directed paths.

Remark

For a d-space (X, dX), there is a bijection

$$dX \quad \cong \quad \mathbf{dTop}(\vec{I}, X)$$

Directed geometric semantics

The category **dTop** is cocomplete and we have a functor

$$\vec{I} \quad : \quad \square \quad \rightarrow \quad \mathbf{dTop}$$

Directed geometric semantics

The category **dTop** is cocomplete and we have a functor

$$\vec{I} \quad : \quad \square \quad \to \quad \mathbf{dTop}$$

The **directed geometric realization** is its extension

$$\begin{tikzcd} \square \arrow[r, "\vec{I}"] \arrow[d, "Y"'] & \mathbf{dTop} \\ \hat{\square} \arrow[ur, dotted, "|-|"'] & \end{tikzcd}$$

i.e.

$$|C| \quad = \quad \coprod_{n \in \mathbb{N}} (C_n \times \vec{I}^n) / \approx$$

Homotopy between paths

Two paths

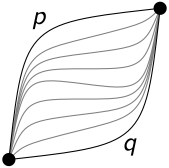
$$p, q \quad : \quad I \quad \to \quad X$$

are **homotopic** when there there is

$$h \quad : \quad I \to X^I$$

such that

- $h(0) = p$
- $h(1) = q$
- $h(t)(0)$ does not depend on t
- $h(t)(1)$ does not depend on t



Dihomotopy between paths

Two directed paths

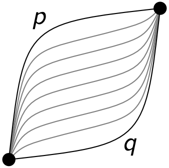
$$p, q \quad : \quad \vec{I} \quad \to \quad X$$

are **dihomotopic** when there there is

$$h \quad : \quad I \to X^{\vec{I}}$$

such that

- $h(0) = p$
- $h(1) = q$
- $h(t)(0)$ does not depend on t
- $h(t)(1)$ does not depend on t



Dihomotopy between paths

Two directed paths

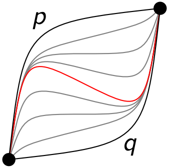
$$p, q \quad : \quad \vec{I} \quad \to \quad X$$

are **dihomotopic** when there there is

$$h \quad : \quad I \to X^{\vec{I}}$$

such that

- $h(0) = p$
- $h(1) = q$
- $h(t)(0)$ does not depend on t
- $h(t)(1)$ does not depend on t



The fundamental category

The **fundamental category** $\vec{\Pi}_1(X)$ of a d-space X has

- points of X as objects
- dipaths up to dihomotopy as morphisms

and composition is induced by concatenation.

The fundamental category

The **fundamental category** $\vec{\Pi}_1(X)$ of a d-space X has

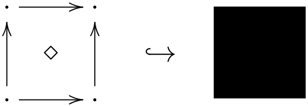
- points of X as objects
- dipaths up to dihomotopy as morphisms

and composition is induced by concatenation.

It coincides with the previous definition: for a precubical set C there is a canonical full and faithful functor

$$\vec{\Pi}_1(C) \quad \hookrightarrow \quad \vec{\Pi}_1(|C|)$$

e.g.

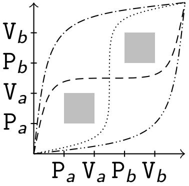


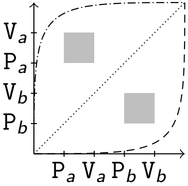
Directed vs undirected paths

The geometric realization of the programs

$\mathtt{P}_a\,;\mathtt{V}_a\,;\mathtt{P}_b\,;\mathtt{V}_b \parallel \mathtt{P}_a\,;\mathtt{V}_a\,;\mathtt{P}_b\,;\mathtt{V}_b$ and $\mathtt{P}_a\,;\mathtt{V}_a\,;\mathtt{P}_b\,;\mathtt{V}_b \parallel \mathtt{P}_b\,;\mathtt{V}_b\,;\mathtt{P}_a\,;\mathtt{V}_a$

are respectively

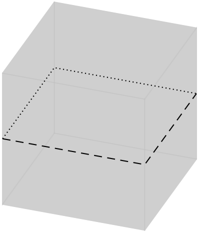




Note that the underlying spaces are homotopy equivalent!

Dihomotopy vs homotopy

Consider $\vec{I}^3$ without the interior:

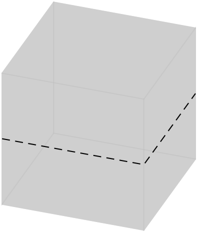


The two dipaths are

- not dihomotopic

Dihomotopy vs homotopy

Consider $\vec{I}^3$ without the interior:

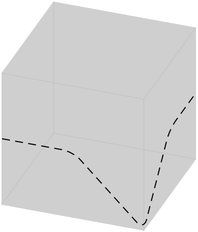


The two dipaths are

- not dihomotopic
- homotopic

Dihomotopy vs homotopy

Consider $\vec{I}^3$ without the interior:

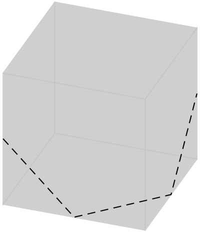


The two dipaths are

- not dihomotopic
- homotopic

Dihomotopy vs homotopy

Consider $\vec{I}^3$ without the interior:

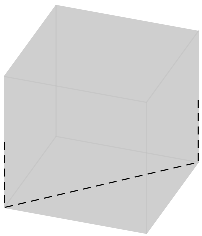


The two dipaths are

- not dihomotopic
- homotopic

Dihomotopy vs homotopy

Consider $\vec{I}^3$ without the interior:

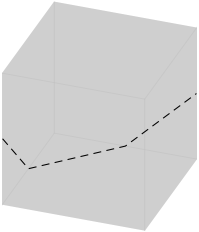


The two dipaths are

- not dihomotopic
- homotopic

Dihomotopy vs homotopy

Consider $\vec{I}^3$ without the interior:

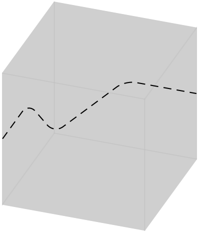


The two dipaths are

- not dihomotopic
- homotopic

Dihomotopy vs homotopy

Consider $\vec{I}^3$ without the interior:

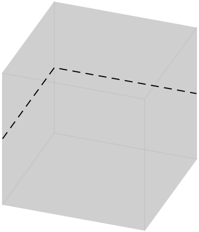


The two dipaths are

- not dihomotopic
- homotopic

Dihomotopy vs homotopy

Consider $\vec{I}^3$ without the interior:



The two dipaths are

- not dihomotopic
- homotopic

Some classical theorems
can be adapted
to this context.

The van Kampen theorem

Given $X = U \cup V$, with U, V open, the image of

$$\begin{tikzcd} U \cap V \arrow[r] \arrow[d] & U \arrow[d] \\ V \arrow[r] & X \end{tikzcd}$$

is a pushout in **Cat**:

$$\begin{tikzcd} \vec{\Pi}_1(U \cap V) \arrow[r] \arrow[d] & \vec{\Pi}_1(U) \arrow[d] \\ \vec{\Pi}_1(V) \arrow[r] & \vec{\Pi}_1(X) \end{tikzcd}$$

Dicovering spaces

A theory of **dicovering spaces** has been developed by Fajstrup.

It is useful in practice in order to unfold loops.

More later on about this (maybe).

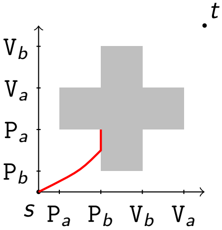
DEADLOCKS

Apart from "usual problems", concurrent programs can suffer from **deadlocks**.

In the program

$$\mathtt{P}_a;\mathtt{P}_b;\mathtt{V}_b;\mathtt{V}_a \quad \| \quad \mathtt{P}_b;\mathtt{P}_a;\mathtt{V}_a;\mathtt{V}_b$$

consider the following execution:



corresponding to

$$\mathtt{P}_a^1 \cdot \mathtt{P}_b^2 \cdot \mathtt{P}_b^1 \cdot \mathtt{P}_a^2$$

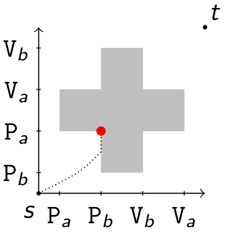
Each of the two processes is waiting for the other!

Deadlocks

Definition

A **deadlock** x in the geometric semantics is a point

- the only path
$$p \quad : \quad x \quad \twoheadrightarrow \quad y$$
with x as source is the constant path,
- x is different form the end point t_p.



Simple programs

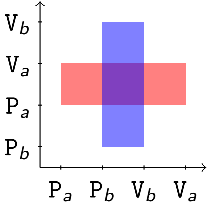
We want to find deadlocks for a simple program p with n processes:

$$S_p \quad = \quad \vec{I}^n \setminus \bigcup_{i=1}^{l} R^i$$

with

$$R^i \quad = \quad \prod_{j=1}^{n}]x^i_j, y^i_j[$$

e.g.



An algorithm for deadlocks

[Fajstrup,Goubault,Raussen]

The deadlock points can be found using the following algorithm:

An algorithm for deadlocks

[Fajstrup,Goubault,Raussen]

The deadlock points can be found using the following algorithm:

1. find n intervals $R^{i_1}, \ldots, R^{i_n}$ with $\bigcap_{i=1}^{n} R^{i_j} \neq \emptyset$

An algorithm for deadlocks

[Fajstrup,Goubault,Raussen]

The deadlock points can be found using the following algorithm:

1. find n intervals $R^{i_1}, \ldots, R^{i_n}$ with $\bigcap_{i=1}^{n} R^{i_j} \neq \emptyset$
2. compute the point z with $z_j = \max\left\{x_j^{i_1}, \ldots, x_j^{i_n}\right\}$

An algorithm for deadlocks

[Fajstrup,Goubault,Raussen]

The deadlock points can be found using the following algorithm:

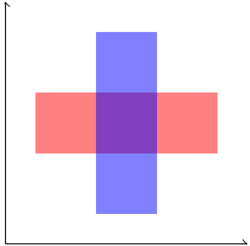
1. find n intervals $R^{i_1}, \dots, R^{i_n}$ with $\bigcap_{i=1}^{n} R^{i_j} \neq \emptyset$
2. compute the point z with $z_j = \max\left\{x_j^{i_1}, \dots, x_j^{i_n}\right\}$
3. if there is no i with $z \notin R^i$ then z is a deadlock

An algorithm for deadlocks

[Fajstrup,Goubault,Raussen]

The deadlock points can be found using the following algorithm:

1. find n intervals $R^{i_1}, \ldots, R^{i_n}$ with $\bigcap_{i=1}^{n} R^{i_j} \neq \emptyset$
2. compute the point z with $z_j = \max\left\{x_j^{i_1}, \ldots, x_j^{i_n}\right\}$
3. if there is no i with $z \notin R^i$ then z is a deadlock

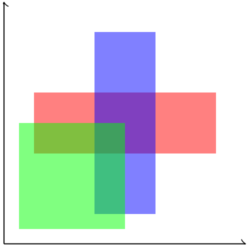


An algorithm for deadlocks

[Fajstrup,Goubault,Raussen]

The deadlock points can be found using the following algorithm:

1. find n intervals $R^{i_1}, \ldots, R^{i_n}$ with $\bigcap_{i=1}^{n} R^{i_j} \neq \emptyset$
2. compute the point z with $z_j = \max\left\{x_j^{i_1}, \ldots, x_j^{i_n}\right\}$
3. if there is no i with $z \notin R^i$ then z is a deadlock

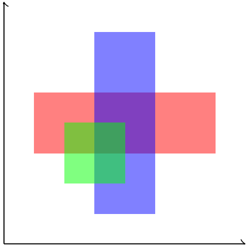


An algorithm for deadlocks

[Fajstrup,Goubault,Raussen]

The deadlock points can be found using the following algorithm:

1. find n intervals $R^{i_1}, \ldots, R^{i_n}$ with $\bigcap_{i=1}^{n} R^{i_j} \neq \emptyset$
2. compute the point z with $z_j = \max\left\{x_j^{i_1}, \ldots, x_j^{i_n}\right\}$
3. if there is no i with $z \notin R^i$ then z is a deadlock

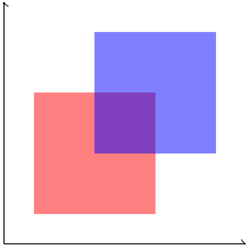


An algorithm for deadlocks

[Fajstrup,Goubault,Raussen]

The deadlock points can be found using the following algorithm:

1. find n intervals $R^{i_1}, \ldots, R^{i_n}$ with $\bigcap_{i=1}^{n} R^{i_j} \neq \emptyset$
2. compute the point z with $z_j = \max\left\{x_j^{i_1}, \ldots, x_j^{i_n}\right\}$
3. if there is no i with $z \notin R^i$ then z is a deadlock

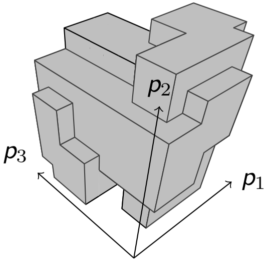


Lipski's program

Consider the program with mutexes

$$\begin{array}{ll} & \mathtt{P}_a;\mathtt{P}_b;\mathtt{P}_c;\mathtt{V}_a;\mathtt{P}_f;\mathtt{V}_c;\mathtt{V}_b;\mathtt{V}_f \\ \| & \mathtt{P}_d;\mathtt{P}_e;\mathtt{P}_a;\mathtt{V}_d;\mathtt{P}_c;\mathtt{V}_e;\mathtt{V}_a;\mathtt{V}_c \\ \| & \mathtt{P}_b;\mathtt{P}_f;\mathtt{V}_b;\mathtt{P}_d;\mathtt{V}_f;\mathtt{P}_e;\mathtt{V}_d;\mathtt{V}_e \end{array}$$

Its geometric semantics is

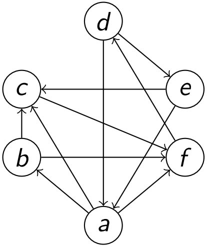


Lipski's program

Consider the program with mutexes

$$
\begin{array}{rl}
 & \mathtt{P}_a;\mathtt{P}_b;\mathtt{P}_c;\mathtt{V}_a;\mathtt{P}_f;\mathtt{V}_c;\mathtt{V}_b;\mathtt{V}_f \\
\parallel & \mathtt{P}_d;\mathtt{P}_e;\mathtt{P}_a;\mathtt{V}_d;\mathtt{P}_c;\mathtt{V}_e;\mathtt{V}_a;\mathtt{V}_c \\
\parallel & \mathtt{P}_b;\mathtt{P}_f;\mathtt{V}_b;\mathtt{P}_d;\mathtt{V}_f;\mathtt{P}_e;\mathtt{V}_d;\mathtt{V}_e
\end{array}
$$

Its *request graph* is



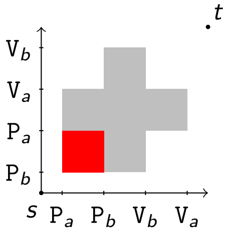
Acyclicity of the request graph implies the absence of deadlocks.

The doomed region

A point x is **doomed** when there is no path

$$p \quad : \quad x \quad \twoheadrightarrow \quad t_p$$

The **doomed region** is the subspace of doomed points.



In other words: a doomed point will eventually reach a deadlock!

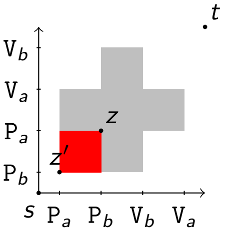
The doomed region

If z is a deadlock point as computed by the previous algorithm, then the region

$$]z', z]$$

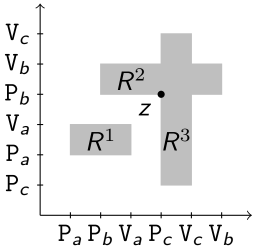
is doomed with

$$z'_j = \max\left\{x_j^{i_k} \,\middle|\, 1 \leq k \leq n, x_j^{i_k} \neq z_j\right\}$$



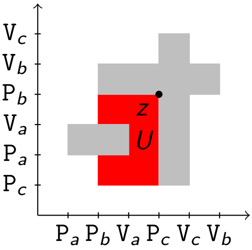
The doomed region

To compute the **doomed region**, we need need to iterate:



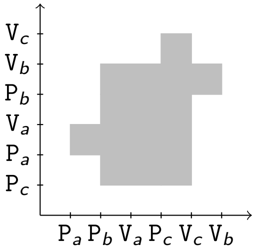
The doomed region

To compute the **doomed region**, we need need to iterate:



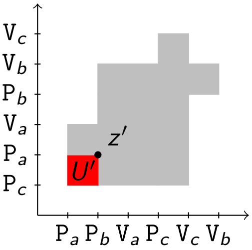
The doomed region

To compute the **doomed region**, we need need to iterate:



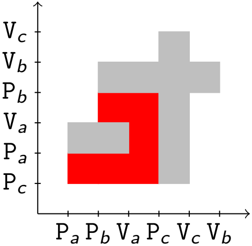
The doomed region

To compute the **doomed region**, we need need to iterate:



The doomed region

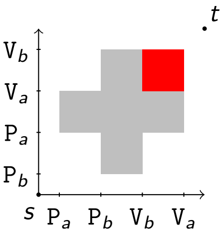
To compute the **doomed region**, we need need to iterate:



The unreachable region

Dually, the **unreachable region** consists of points x for which there is no path

$$p \quad : \quad s_e \quad \twoheadrightarrow \quad x$$



Having dead code is often an indication of a misconception.

PROGRAMS
WITH
MUTEXES
ONLY

Programs with mutexes

The most common case is the one of programs with mutexes only, i.e. $\kappa_a = 1$.

Otherwise said, all *conflicts are binary*.

Programs with mutexes

The most common case is the one of programs with mutexes only, i.e. $\kappa_a = 1$.

Otherwise said, all *conflicts are binary*.

- The cubical semantics is determined by cubes in dimension 0, 1 and 2 (all possible cubes in higher dimension are filled).

Programs with mutexes

The most common case is the one of programs with mutexes only, i.e. $\kappa_a = 1$.

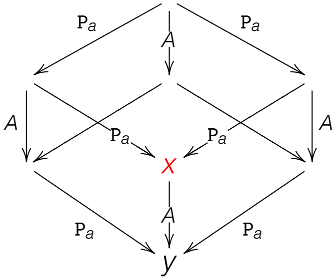
Otherwise said, all *conflicts are binary*.

- The cubical semantics is determined by cubes in dimension 0, 1 and 2 (all possible cubes in higher dimension are filled).
- The cubical semantics also satisfies an important property called the *cube property*.

The cube property

In a situation such as

$$\mathtt{P}_a \,\|\, \mathtt{P}_a \,\|\, A \qquad =$$



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

The cube property

In a situation such as

$$
\mathtt{P}_a \,\|\, \mathtt{P}_a \,\|\, A \qquad = \qquad
\begin{tikzcd}
 & & \bullet \arrow[dll, "\mathtt{P}_a"'] \arrow[d, "A"] \arrow[drr, "\mathtt{P}_a"] & & \\
\bullet \arrow[d, "A"'] \arrow[drr, "\mathtt{P}_a"] & & \bullet \arrow[dll] \arrow[drr] & & \bullet \arrow[dll, "\mathtt{P}_a"'] \arrow[d, "A"] \\
\bullet \arrow[dr, "\mathtt{P}_a"'] & & x \arrow[d, "A"] & & \bullet \arrow[dll, "\mathtt{P}_a"] \\
 & & y & &
\end{tikzcd}
$$

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

In this case, the vertex y has to be removed too, because $A \neq \mathtt{V}_a$!

The cube property

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

$$
\begin{tikzcd}
 & & {} \arrow[dll, "A"'] \arrow[d] \arrow[drr, "C'"] & & \\
{} \arrow[d, "B"'] & \diamond & {} \arrow[dll] \arrow[drr] & \diamond & {} \arrow[d, "B'"] \\
{} \arrow[drr, "C"'] & & \diamond & & {} \arrow[dll, "A'"] \\
 & & {} & &
\end{tikzcd}
\quad \Leftrightarrow \quad
\begin{tikzcd}
 & & {} \arrow[dll, "A"'] \arrow[drr, "C'"] & & \\
{} \arrow[d, "B"'] \arrow[drr] & & \diamond & & {} \arrow[d, "B'"] \arrow[dll] \\
{} \arrow[drr, "C"'] & \diamond & {} \arrow[d] & \diamond & {} \arrow[dll, "A'"] \\
 & & {} & &
\end{tikzcd}
$$

The cube property

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

$$
\begin{tikzcd}
& {} \arrow[dl, "A"'] \arrow[d] \arrow[dr, "C'"] \arrow[dl, phantom, shift left=4, "\diamond"] \arrow[dr, phantom, shift right=4, "\diamond"] & \\
{} \arrow[d, "B"'] & {} \arrow[dl] \arrow[dr] \arrow[d, phantom, "\diamond"] & {} \arrow[d, "B'"] \\
{} \arrow[dr, "C"'] & {} & {} \arrow[dl, "A'"] \\
& {} &
\end{tikzcd}
\quad \Leftrightarrow \quad
\begin{tikzcd}
& {} \arrow[dl, "A"'] \arrow[dr, "C'"] \arrow[d, phantom, "\diamond"] & \\
{} \arrow[d, "B"'] \arrow[dr] & {} & {} \arrow[d, "B'"] \arrow[dl] \\
{} \arrow[dr, "C"'] \arrow[r, phantom, "\diamond"] & {} \arrow[d] \arrow[r, phantom, "\diamond"] & {} \arrow[dl, "A'"] \\
& {} &
\end{tikzcd}
$$

and other more minor properties, e.g.

$$
\begin{tikzcd}
& {} \arrow[dl, "A"'] \arrow[dr, "B'"] \arrow[dd, phantom, "\diamond"] & \\
{} \arrow[dr, "B"'] & & {} \arrow[dl, "A'"] \\
& {} &
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
& {} \arrow[dl, "A"'] \arrow[dr, "B''"] \arrow[dd, phantom, "\diamond"] & \\
{} \arrow[dr, "B"'] & & {} \arrow[dl, "A''"] \\
& {} &
\end{tikzcd}
$$

implies $A' = A''$ and $B' = B''$.

This has many consequences!

Non-positively curved spaces

Suppose fixed a precubical set C satisfying the cube property.

Proposition

The geometric realization $|C|$ of C is non-positively curved.

Non-positively curved spaces

Suppose fixed a precubical set C satisfying the cube property.

Proposition

The geometric realization $|C|$ of C is non-positively curved.

Let me:

- recall the notion of curvature for a metric space
- recall Gromov's characterization of non-positively curved cube complexes
- explain how to put a metric on the geometric realization

Metric spaces

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

$$\begin{aligned}
&(1) & \text{point equality:} \quad & d(x,x) = 0 \\
&(2) & \text{triangle inequality:} \quad & d(x,z) \leq d(x,y) + d(y,z) \\
&(3) & \text{finite distances:} \quad & d(x,y) < \infty \\
&(4) & \text{separation:} \quad & d(x,y) = 0 \text{ implies } x = y \\
&(5) & \text{symmetry:} \quad & d(x,y) = d(y,x)
\end{aligned}$$

Metric spaces

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

$$\begin{array}{lrl} (1) & \text{point equality:} & d(x,x) = 0 \\ (2) & \text{triangle inequality:} & d(x,z) \leq d(x,y) + d(y,z) \\ (3) & \text{finite distances:} & d(x,y) < \infty \\ (4) & \text{separation:} & d(x,y) = 0 \text{ implies } x = y \\ (5) & \text{symmetry:} & d(x,y) = d(y,x) \end{array}$$

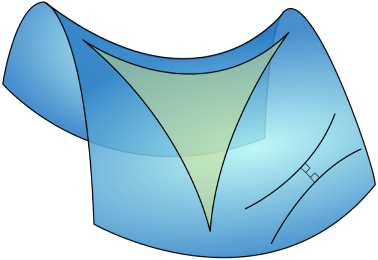
A path $p : I \to X$ is **geodesic** when

$$d(p(t), p(t')) \quad = \quad d(p(0), p(1))(t' - t)$$

for $0 \leq t \leq t' \leq 1$.

Non-positively curved spaces

- A **geodesic triangle** $\Delta(x, y, z)$ in a metric space X consists of three points x, y, z and geodesics joining any pairs.



Non-positively curved spaces

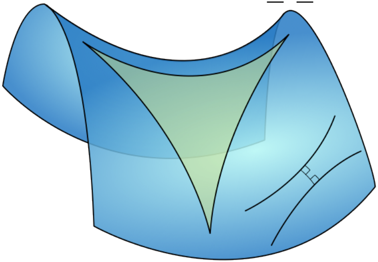
- A **geodesic triangle** $\Delta(x, y, z)$ in a metric space X consists of three points x, y, z and geodesics joining any pairs.
- A **comparison triangle** for a geodesic triangle $\Delta(x, y, z)$ consists of an isometry $\underline{\ \ } : \Delta(x, y, z) \to \mathbb{R}^2$ whose image $\underline{\Delta}(\underline{x}, \underline{y}, \underline{z})$ is a geodesic triangle.

Non-positively curved spaces

- A **geodesic triangle** $\Delta(x,y,z)$ in a metric space X consists of three points x, y, z and geodesics joining any pairs.
- A **comparison triangle** for a geodesic triangle $\Delta(x,y,z)$ consists of an isometry $\underline{\ } : \Delta(x,y,z) \to \mathbb{R}^2$ whose image $\underline{\Delta}(\underline{x},\underline{y},\underline{z})$ is a geodesic triangle.

Definition

A geodesic space is **CAT(0)** if for every geodesic triangle $\Delta(x,y,z)$, there exists a comparison triangle $\underline{\Delta}(\underline{x},\underline{y},\underline{z})$ such that for every points $p,q \in \Delta(x,y,z)$, we have $d(p,q) \leq d_{\mathbb{R}^2}(\underline{p},\underline{q})$.



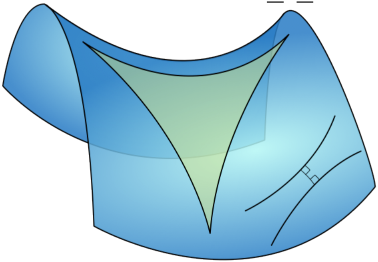
Non-positively curved spaces

- A **geodesic triangle** $\Delta(x,y,z)$ in a metric space X consists of three points x, y, z and geodesics joining any pairs.
- A **comparison triangle** for a geodesic triangle $\Delta(x,y,z)$ consists of an isometry $\underline{\ \ } : \Delta(x,y,z) \to \mathbb{R}^2$ whose image $\underline{\Delta}(\underline{x},\underline{y},\underline{z})$ is a geodesic triangle.

Definition

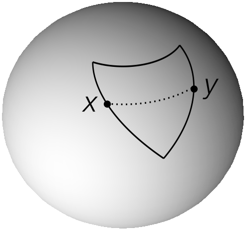
A geodesic space is **CAT(0)** if for every geodesic triangle $\Delta(x,y,z)$, there exists a comparison triangle $\underline{\Delta}(\underline{x},\underline{y},\underline{z})$ such that for every points $p,q \in \Delta(x,y,z)$, we have $d(p,q) \leq d_{\mathbb{R}^2}(\underline{p},\underline{q})$.

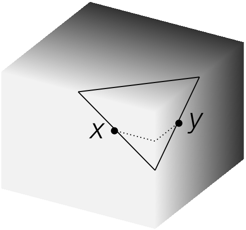
A locally CAT(0) space is called **non-positively curved** (NPC).



Positively curved spaces

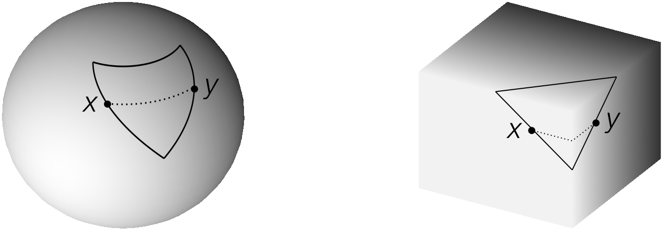
Typical examples of positively curved spaces:





Positively curved spaces

Typical examples of positively curved spaces:



For realizations of precubical sets with the cube property, we can expect that this does not happen!

Metric realization

The interval $I = [0, 1]$ can be equipped with the standard euclidean distance.

Metric realization

The interval $I = [0, 1]$ can be equipped with the standard euclidean distance.

This induces a functor

$$\begin{array}{ccc} \square & \to & \mathbf{Met} \\ n & \mapsto & I^n \end{array}$$

Metric realization

The interval $I = [0, 1]$ can be equipped with the standard euclidean distance.

This induces a functor

$$\begin{array}{ccc} \square & \to & \mathbf{Met} \\ n & \mapsto & I^n \end{array}$$

which extends as a *metric realization* functor

$$\begin{tikzcd} \square \arrow[r] \arrow[d, "Y"'] & \mathbf{Met} \\ \hat{\square} \arrow[ur, dotted] & \end{tikzcd}$$

Metric realization

The interval $I = [0, 1]$ can be equipped with the standard euclidean distance.

This induces a functor

$$\begin{array}{ccc} \square & \to & \mathbf{Met} \\ n & \mapsto & I^n \end{array}$$

which extends as a *metric realization* functor

$$\begin{tikzcd} \square \arrow[r] \arrow[d, "Y"'] & \mathbf{Met} \\ \hat{\square} \arrow[ur, dotted] & \end{tikzcd}$$

excepting that

- I did not tell you what the morphisms of **Met** are

Metric realization

The interval $I = [0, 1]$ can be equipped with the standard euclidean distance.

This induces a functor

$$\begin{array}{ccc} \square & \to & \mathbf{Met} \\ n & \mapsto & I^n \end{array}$$

which extends as a *metric realization* functor

$$\begin{tikzcd} \square \arrow[r] \arrow[d, "Y"'] & \mathbf{Met} \\ \hat{\square} \arrow[ur, dotted] & \end{tikzcd}$$

excepting that

- I did not tell you what the morphisms of **Met** are
- the category **Met** 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 $x, y, z \in X$,

$$\begin{array}{lrl} (1) & \text{point equality:} & d(x,x) = 0 \\ (2) & \text{triangle inequality:} & d(x,z) \leq d(x,y) + d(y,z) \\ (3) & \text{finite distances:} & d(x,y) < \infty \\ (4) & \text{separation:} & d(x,y) = 0 \text{ implies } x = y \\ (5) & \text{symmetry:} & d(x,y) = d(y,x) \end{array}$$

We consider contracting maps $f : X \to Y$:

$$d_Y(f(x), f(x')) \qquad \leq \qquad d_X(x, x')$$

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 $x, y, z \in X$,

$$\begin{aligned}
(1) \quad & \text{point equality:} & d(x,x) &= 0 \\
(2) \quad & \text{triangle inequality:} & d(x,z) &\leq d(x,y) + d(y,z) \\
(3) \quad & \text{finite distances:} & d(x,y) &< \infty \\
(4) \quad & \text{separation:} & d(x,y) &= 0 \text{ implies } x = y \\
(5) \quad & \text{symmetry:} & d(x,y) &= d(y,x)
\end{aligned}$$

Intuitively, $X + Y$ should be such that

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

for $x \in X$ and $y \in Y$.

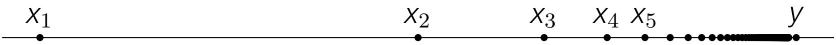
Generalizing metric spaces

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 $d([x_i], [y]) = 0$.

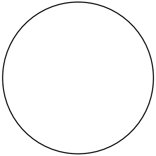
Generalizing metric spaces

Definition

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

$$
\begin{aligned}
(1) &\quad \text{point equality:} & d(x,x) &= 0 \\
(2) &\quad \text{triangle inequality:} & d(x,z) &\leq d(x,y) + d(y,z) \\
(3) &\quad \text{finite distances:} & d(x,y) &< \infty \\
(4) &\quad \text{separation:} & d(x,y) &= 0 \text{ implies } x = y \\
(5) &\quad \text{symmetry:} & d(x,y) &= d(y,x)
\end{aligned}
$$

We can encode direction in the distance!



$$d(x,y) = \bigwedge \left\{ \rho - \theta \;\middle|\; x = e^{i2\pi\theta}, y = e^{i2\pi\rho}, \rho \geq \theta \right\}$$

Generalized metric spaces

Definition (Lawvere)

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

$$
\begin{array}{rrl}
(1) & \text{point equality:} & d(x,x) = 0 \\
(2) & \text{triangle inequality:} & d(x,z) \leq d(x,y) + d(y,z)
\end{array}
$$

Generalized metric spaces

Definition (Lawvere)

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

$$\begin{aligned} (1) \quad & \text{point equality:} & d(x,x) &= 0 \\ (2) \quad & \text{triangle inequality:} & d(x,z) &\leq d(x,y) + d(y,z) \end{aligned}$$

The category **GMet** enjoys the following:

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

Directed metric realization

We write $\vec{I}$ for the **directed interval** $[0,1]$ equipped with

$$d(x,y) \quad = \quad \begin{cases} y - x & \text{if } y \geq x \\ \infty & \text{if } y < x \end{cases}$$

Directed metric realization

We write $\vec{I}$ for the **directed interval** $[0,1]$ equipped with

$$d(x,y) \quad = \quad \begin{cases} y - x & \text{if } y \geq x \\ \infty & \text{if } y < x \end{cases}$$

The product $\vec{I}^n$ is equipped with

$$d((x_1,\ldots,x_n),(y_1,\ldots,y_n)) \quad = \quad d(x_1,y_1) \vee \ldots \vee d(x_n,y_n)$$

Directed metric realization

We write $\vec{I}$ for the **directed interval** $[0,1]$ equipped with

$$d(x,y) \quad = \quad \begin{cases} y - x & \text{if } y \geq x \\ \infty & \text{if } y < x \end{cases}$$

The product $\vec{I}^n$ is equipped with

$$d((x_1,\ldots,x_n),(y_1,\ldots,y_n)) \quad = \quad d(x_1,y_1) \vee \ldots \vee d(x_n,y_n)$$

The geometric realization of a precubical set C is

$$|C| \quad = \quad \int^{n \in \square} C_n \cdot \vec{I}^n$$

Directed metric realization

We write $\vec{I}$ for the **directed interval** $[0,1]$ equipped with

$$d(x,y) \quad = \quad \begin{cases} y - x & \text{if } y \geq x \\ \infty & \text{if } y < x \end{cases}$$

The product $\vec{I}^n$ is equipped with

$$d((x_1,\ldots,x_n),(y_1,\ldots,y_n)) \quad = \quad d(x_1,y_1) \vee \ldots \vee d(x_n,y_n)$$

The geometric realization of a precubical set C is

$$|C| \quad = \quad \int^{n \in \square} C_n \cdot \vec{I}^n$$

Proposition

For finite-dimensional precubical sets, geometric realization commutes with forgetful functor **GMet** $\to$ **Top** and produces geodesic length spaces.

Colimits in **GMet** vs **Top**

Colimits in **GMet** do not necessarily coincide with those in **Top**.

Colimits in **GMet** vs **Top**

Colimits in **GMet** do not necessarily coincide with those in **Top**.

Write I_n for the space $I = [0, 1]$ equipped with $d_n(x, y) = |y - x|/n$.

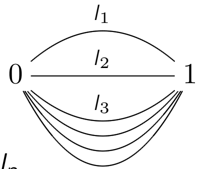
Colimits in **GMet** vs **Top**

Colimits in **GMet** do not necessarily coincide with those in **Top**.

Write I_n for the space $I = [0,1]$ equipped with $d_n(x,y) = |y-x|/n$.

Consider the colimit

$$I_\infty = \coprod_{n\in\mathbb{N}} I_n/\approx$$



where $\approx$ identifies 0 (resp. 1) in various I_n.

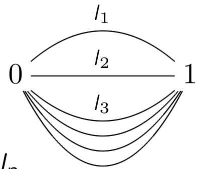
Colimits in **GMet** vs **Top**

Colimits in **GMet** do not necessarily coincide with those in **Top**.

Write I_n for the space $I = [0,1]$ equipped with $d_n(x,y) = |y-x|/n$.

Consider the colimit

$$I_\infty = \coprod_{n \in \mathbb{N}} I_n / \approx$$



where $\approx$ identifies 0 (resp. 1) in various I_n.

We have $d(0,1) = 0$ and therefore the points 0 and 1 are not separated in I_∞.

Reformulating "flag links condition" in our setting:

Theorem (Gromov)

The geometric realization of a precubical set is NPC if and only iff it satisfies the cube property.

Gromov's theorem

Reformulating "flag links condition" in our setting:

Theorem (Gromov)

The geometric realization of a precubical set is NPC if and only iff it satisfies the cube property.

Such a space is locally uniquely geodesic. In particular, directed paths are local geodesics:

an analogue of the *least action principle*

Gromov's theorem

Reformulating "flag links condition" in our setting:

Theorem (Gromov)

The geometric realization of a precubical set is NPC if and only iff it satisfies the cube property.

Such a space is locally uniquely geodesic. In particular, directed paths are local geodesics:

an analogue of the *least action principle*

Moreover, it enjoys many nice properties:

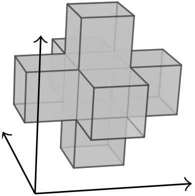
- greedy normal forms for paths,
- universal cover is CAT(0),
- fundamental group is automatic,
- …

A small example

Consider

$$\mathtt{P}_a \quad \| \quad \mathtt{P}_a \quad \| \quad \mathtt{P}_a$$

whose realization of geometric semantics is



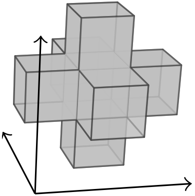
a mutex
NPC

A small example

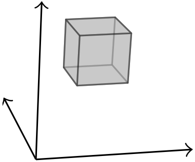
Consider

$$\mathtt{P}_a \quad || \quad \mathtt{P}_a \quad || \quad \mathtt{P}_a$$

whose realization of geometric semantics is



a mutex
NPC



a of arity 2
not NPC

Gromov's condition

Given a precubical set

$$
\begin{tikzcd}
 & & z_3 & & \\
y_1 \arrow[urr, "a_2''"] \arrow[d, "a_3'"'] & & \alpha_3 & & y_2 \arrow[ull, "a_1'"'] \arrow[d, "a_3''"] \\
z_2 & \alpha_2 & x \arrow[ull, "a_1"] \arrow[urr, "a_2"'] \arrow[d, "a_3"] & \alpha_1 & z_1 \\
 & & y_3 \arrow[ull, "a_1''"] \arrow[urr, "a_2'"'] & &
\end{tikzcd}
$$

and a vertex *x*, we can define a simplicial complex

$$
\begin{tikzcd}
a_1 \arrow[rr, dash, "\alpha_3"] \arrow[dr, dash, "\alpha_2"'] & & a_2 \arrow[dl, dash, "\alpha_1"] \\
 & a_3 &
\end{tikzcd}
$$

called the **link** of *x*.

Gromov's condition

A simplicial complex is **flag** when every boundary of an *n*-simplex is filled, for $n \geq 2$.

Gromov's condition

A simplicial complex is **flag** when every boundary of an n-simplex is filled, for $n \geq 2$.

Theorem (Gromov'87)

A finite dimensional cubed complex is NPC if and only if the link of every vertex is flag.

Gromov’s condition

A simplicial complex is **flag** when every boundary of an n-simplex is filled, for $n \geq 2$.

Theorem (Gromov’87)

A finite dimensional cubed complex is NPC if and only if the link of every vertex is flag.

$$
\begin{tikzcd}
a_1 \arrow[rr, dash, "\alpha_3"] \arrow[dr, dash, "\alpha_2"'] & & a_2 \arrow[dl, dash, "\alpha_1"] \\
& a_3 &
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & & z_3 & & \\
y_1 \arrow[urr, "a_2''"] \arrow[d, "a_3'"'] & & \alpha_3 & & y_2 \arrow[ull, "a_1'"'] \arrow[d, "a_3''"] \\
z_2 & \alpha_2 & x \arrow[ull, "a_1"] \arrow[urr, "a_2"'] \arrow[d, "a_3"] & \alpha_1 & z_1 \\
 & & y_3 \arrow[ull, "a_1''"] \arrow[urr, "a_2'"'] & &
\end{tikzcd}
$$

Gromov's condition

A simplicial complex is **flag** when every boundary of an n-simplex is filled, for $n \geq 2$.

Theorem (Gromov'87)

A finite dimensional cubed complex is NPC if and only if the link of every vertex is flag.

$$
\begin{tikzcd}
a_1 \arrow[rr, dash, "\alpha_3"] \arrow[dr, dash, "\alpha_2"'] & & a_2 \arrow[dl, dash, "\alpha_1"] \\
& a_3 &
\end{tikzcd}
\quad \Rightarrow \quad
\begin{tikzcd}
a_1 \arrow[rr, dash, "\alpha_3"] \arrow[dr, dash, "\alpha_2"'] & A & a_2 \arrow[dl, dash, "\alpha_1"] \\
& a_3 &
\end{tikzcd}
$$

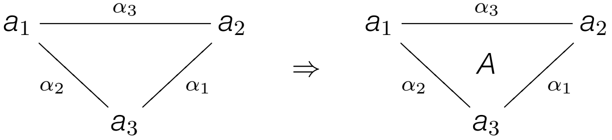
$$
\begin{tikzcd}
& z_3 & \\
y_1 \arrow[ur, "a_2''"] \arrow[d, "a_3'"'] & \alpha_3 & y_2 \arrow[ul, "a_1'"'] \arrow[d, "a_3''"] \\
z_2 \arrow[r, phantom, "\alpha_2"] & x \arrow[ul, "a_1"] \arrow[ur, "a_2"'] \arrow[d, "a_3"] \arrow[r, phantom, "\alpha_1"] & z_1 \\
& y_3 \arrow[ul, "a_1''"] \arrow[ur, "a_2'"'] &
\end{tikzcd}
$$

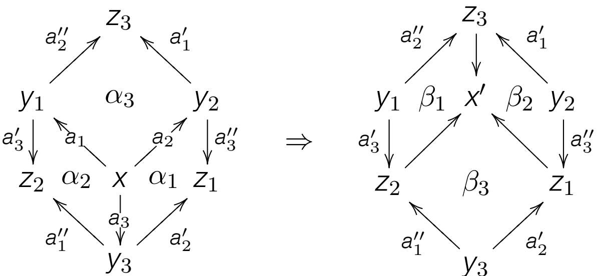
Gromov's condition

A simplicial complex is **flag** when every boundary of an *n*-simplex is filled, for $n \geq 2$.

Theorem (Gromov'87)

A finite dimensional cubed complex is NPC if and only if the link of every vertex is flag.





Some other properties
of NPC precubical sets

Dihomotopy vs homotopy

Proposition

In (the geometric realization of) C two directed paths are homotopic if and only if they are dihomotopic.

Otherwise said, there is a faithful functor

$$\vec{\Pi}_1(C) \quad \hookrightarrow \quad \Pi_1(C)$$

Universal directed covers
are easy to define
for NPC precubical sets

Covering spaces

A **covering space** of X is a space which can be obtained from X by “unrolling” some of its loops.

Covering spaces

A morphism $p : Y \to X$ is **covering** when every $x \in X$ admits a neighborhood U such that $p^{-1}(U)$ is a disjoint union of homeomorphic copies of U.

Covering spaces

A morphism $p : Y \to X$ is **covering** when every $x \in X$ admits a neighborhood U such that $p^{-1}(U)$ is a disjoint union of homeomorphic copies of U.

The **universal covering** is the initial (= most general) pointed covering of a space X.

Covering spaces

A morphism $p : Y \to X$ is **covering** when every $x \in X$ admits a neighborhood U such that $p^{-1}(U)$ is a disjoint union of homeomorphic copies of U.

The **universal covering** is the initial (= most general) pointed covering of a space X.

It is the simply connected covering of X.

Covering spaces

A morphism $p : Y \to X$ is **covering** when every $x \in X$ admits a neighborhood U such that $p^{-1}(U)$ is a disjoint union of homeomorphic copies of U.

The **universal covering** is the initial (= most general) pointed covering of a space X.

Given $x \in X$, the universal covering space of X can be described as the space of paths

$$x \twoheadrightarrow y$$

with a suitable topology.

Dicovering spaces

For **dicovering spaces** can consider:

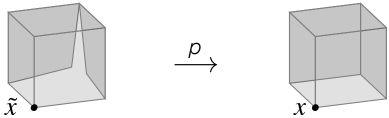
1. the universal dicovering: every $x \in X$ admits a neighborhood U such that $p^{-1}(U)$ is a disjoint union of *di*homeomorphic copies of U,
2. the space of dipaths up to dihomotopy from a fixed point.

Dicovering spaces

For **dicovering spaces** can consider:

1. the universal dicovering: every $x \in X$ admits a neighborhood U such that $p^{-1}(U)$ is a disjoint union of *di*homeomorphic copies of U,
2. the space of dipaths up to dihomotopy from a fixed point.

They do not coincide in general: the space of dipaths is



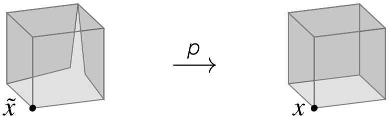
whereas identity is the only dicovering in the sense of 1.

Dicovering spaces

For **dicovering spaces** can consider:

1. the universal dicovering: every $x \in X$ admits a neighborhood U such that $p^{-1}(U)$ is a disjoint union of *di*homeomorphic copies of U,
2. the space of dipaths up to dihomotopy from a fixed point.

They do not coincide in general: the space of dipaths is



whereas identity is the only dicovering in the sense of 1.

However, for NPC spaces, they do coincide!

They correspond to
configuration spaces
of event structures

Event structures

An **event structure** $(E, \leq, \#)$ consists of

- a poset $(E, \leq)$ of *events*
- a binary relation # called *incompatibility*

such that

Event structures

An **event structure** $(E, \leq, \#)$ consists of

- a poset $(E, \leq)$ of *events*
- a binary relation # called *incompatibility*

such that

- *finite causes*: for every event e, the set

$$\{e' \in E \mid e' \leq e\}$$

  is finite,

Event structures

An **event structure** $(E, \leq, \#)$ consists of

- a poset $(E, \leq)$ of *events*
- a binary relation # called *incompatibility*

such that

- *finite causes*: for every event e, the set

  $$\{e' \in E \mid e' \leq e\}$$

  is finite,
- *hereditary incompatibility*: for every events e_1, e_2, e_2',

  $$e_1 \# e_2 \quad \text{and} \quad e_2 \leq e_2' \qquad \text{implies} \qquad e_1 \# e_2'$$

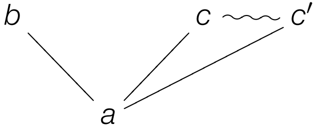
Configuration spaces

A **configuration** of an event structure is a downward closed set of events.

Configuration spaces

A **configuration** of an event structure is a downward closed set of events.

For instance, in the event structure



the configurations are:

$$\emptyset \quad \{a\} \quad \{a,b\} \quad \{a,c\} \quad \{a,c'\} \quad \{a,b,c\} \quad \{a,b,c'\}$$

Configuration spaces

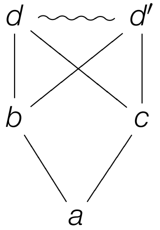
The **configuration space** of an event structure has

- vertices: the configurations
- edges: an edge $x \to y$ when $y = x \uplus \{e\}$
- n-cubes for $n \geq 2$: fill all possible cubes

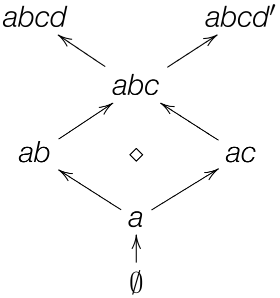
Configuration spaces

The **configuration space** of an event structure has

- vertices: the configurations
- edges: an edge $x \to y$ when $y = x \uplus \{e\}$
- n-cubes for $n \geq 2$: fill all possible cubes



$\rightsquigarrow$



Configuration spaces

The configuration space of an event structure is always a CAT(0) precubical set:

Configuration spaces

The configuration space of an event structure is always a CAT(0) precubical set:

Theorem (Chepoi, Ardila-Owen-Sullivant, Goubault-M.)
The rooted (globally) CAT(0) precubical sets are in bijection with event structures.

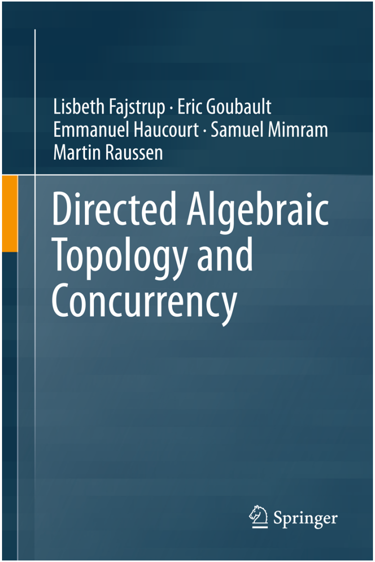
CONCLUSION

Conclusion

This was only a very brief overview, we can also mention:

- applications in verification of programs
- the geometry of tracespaces
- applications to distributed computing
- directed topology: from groupoids to categories (e.g. directed homology)
- directed homotopy type theory

Thanks!


Lisbeth Fajstrup · Eric Goubault
Emmanuel Haucourt · Samuel Mimram
Martin Raussen
Directed Algebraic Topology and Concurrency
Springer