Towards Efficient Computation of Trace Spaces of Concurrent Programs

Samuel Mimram

CEA, LIST

CHOCO Party
When verifying a concurrent program, there is a priori a large number of possible interleavings to check (exponential in the number of processes)
When verifying a concurrent program, there is a priori a large number of possible interleavings to check (exponential in the number of processes).

Many executions are equivalent: we want here to provide a minimal number of execution traces which describe all the possible cases.
When verifying a concurrent program, there is a priori a large number of possible interleavings to check (exponential in the number of processes).

Many executions are equivalent: we want here to provide a minimal number of execution traces which describe all the possible cases.

Joint work with M. Raussen, L. Fajstrup, É. Goubault, E. Haucourt and A. Lang.
Programs generate trace spaces

Consider the program

\[ x := 1 \; ; \; y := 2 \quad | \quad y := 3 \]

It can be scheduled in three different ways:

\[ y := 3 \; ; \; x := 1 \; ; \; y := 2 \quad \quad x := 1 \; ; \; y := 3 \; ; \; y := 2 \quad \quad x := 1 \; ; \; y := 2 \; ; \; y := 3 \]

Giving rise to the following graph of traces:
Programs generate trace spaces

Consider the program

\[ x := 1; y := 2 \quad | \quad y := 3 \]

It can be scheduled in three different ways:

- \( y := 3; x := 1; y := 2 \) \quad \Rightarrow \quad (x, y) = (1, 2)
- \( x := 1; y := 3; y := 2 \) \quad \Rightarrow \quad (x, y) = (1, 2)
- \( x := 1; y := 2; y := 3 \) \quad \Rightarrow \quad (x, y) = (1, 3)

Giving rise to the following graph of traces:

\[ \text{homotopy: commutation / filled square} \]
Programs generate trace spaces

Consider the program

\[ x := 1; y := 2 \quad | \quad y := 3 \]

It can be scheduled in three different ways:

\[
\begin{align*}
    &y := 3; x := 1; y := 2 & x := 1; y := 3; y := 2 & x := 1; y := 2; y := 3 \\
    (x, y) &= (1, 2) & (x, y) &= (1, 2) & (x, y) &= (1, 3)
\end{align*}
\]

Giving rise to the following graph of traces:

\[
\begin{array}{c}
\text{homotopy: commutation / filled square}
\end{array}
\]
Concurrent access to shared variables should be protected using mutexes $a, b, \ldots$:

- $P_a$: lock the mutex $a$
- $V_a$: unlock the mutex $a$
Mutexes

Concurrent access to shared variables should be protected using **mutexes** $a, b, \ldots$:

- $P_a$: lock the mutex $a$
- $V_a$: unlock the mutex $a$

$x := 1; y := 2 \mid y := 3$
Concurrent access to shared variables should be protected using **mutexes** \( a, b, \ldots \):

- \( P_a \): lock the mutex \( a \)
- \( V_a \): unlock the mutex \( a \)

\[
P_b; x:=1; V_b; P_a; y:=2; V_a | P_a; y:=3; V_a
\]
Mutexes

Concurrent access to shared variables should be protected using mutexes $a, b, \ldots$:

- $P_a$: lock the mutex $a$
- $V_a$: unlock the mutex $a$

$$P_b \cdot V_b \cdot P_a \cdot V_a \mid P_a \cdot V_a$$
Geometric semantics

A program will be interpreted as a **directed space**:

- $P_b.V_b.P_a.V_a$
Geometric semantics

A program will be interpreted as a **directed space**:

- \( P_b \cdot V_b \cdot P_a \cdot V_a \)

\[ \begin{array}{cccc}
P_b & V_b & P_a & V_a \\
\end{array} \]

- \( P_a \cdot V_a \)

\[ \begin{array}{cccc}
P_a & V_a \\
\end{array} \]
Geometric semantics

A program will be interpreted as a directed space:

- $P_b . V_b . P_a . V_a$

  \[ P_b \quad V_b \quad P_a \quad V_a \]

- $P_a . V_a$

  \[ P_a \quad V_a \]

- $P_b . V_b . P_a . V_a \quad | \quad P_a . V_a$

  \[ P_b \quad V_b \quad P_a \quad V_a \]
Geometric semantics

A program will be interpreted as a directed space:

- \( P_b \cdot V_b \cdot P_a \cdot V_a \)

\[ P_b \quad V_b \quad P_a \quad V_a \]

- \( P_a \cdot V_a \)

\[ P_a \quad V_a \]

- \( P_b \cdot V_b \cdot P_a \cdot V_a \quad | \quad P_a \cdot V_a \)

\[ P_b \quad V_b \quad P_a \quad V_a \]

\[ P_a \cdot P_b \cdot V_a \cdot V_b \cdot P_a \cdot V_a \]
Geometric semantics

A program will be interpreted as a **directed space**:

- $P_b \cdot V_b \cdot P_a \cdot V_a$

- $P_a \cdot V_a$

- $P_b \cdot V_b \cdot P_a \cdot V_a \mid P_a \cdot V_a$  \hspace{1cm} Homotopy

\[ P_a \cdot P_b \cdot V_a \cdot V_b \cdot P_a \cdot V_a \]
Geometric semantics

A program will be interpreted as a directed space:

- $P_b \cdot V_b \cdot P_a \cdot V_a$

- $P_a \cdot V_a$

- $P_b \cdot V_b \cdot P_a \cdot V_a \mid P_a \cdot V_a$

Forbidden region
A **scheduling** is the homotopy class of a path.
Schedulings

A **scheduling** is the homotopy class of a path.

We want to compute a *path in every scheduling*
A **scheduling** is the homotopy class of a path.

We want to compute *a path in every scheduling*

We do this by testing possible ways to go around forbidden regions:
The Swiss flag

\[ P_a \cdot P_b \cdot V_b \cdot V_a | P_b \cdot P_a \cdot V_a \cdot V_b \]

A forbidden region
The Swiss flag

\[ P_a P_b V_b V_a \| P_b P_a V_a V_b \]

A trace: \[ P_b P_a V_a P_a V_b P_b V_b V_a \]
A deadlock: $P_b \cdot P_a$
The Swiss flag

\[ P_a \cdot P_b \cdot V_b \cdot V_a \mid P_b \cdot P_a \cdot V_a \cdot V_b \]

An unreachable region
The Swiss flag

\[ P_a \cdot P_b \cdot V_b \cdot V_a \| P_b \cdot P_a \cdot V_a \cdot V_b \]

Here we are interested in maximal paths modulo homotopy
Plan

1. Trace semantics of programs
2. Geometric semantics of programs
3. Computation of the trace space
We suppose fixed a set $\mathcal{R}$ of resources $a$ with capacity $\kappa_a \in \mathbb{N}$.

The execution of programs are such that

1. a resource $a$ cannot be locked ($V_a$) more than $\kappa_a$ times
2. a resource $a$ cannot be freed if it has not been locked

Example

A mutex is a resource of capacity 1.
We consider programs of the form:

\[
p ::= 1 \mid P_a \mid V_a \mid p.p \mid p|p \mid p+p \mid p^*
\]
We consider programs of the form:

\[ p ::= 1 \mid P_a \mid V_a \mid p.p \mid p|p \]

We omit non-deterministic choice, loops
Programs

We consider programs of the form:

\[ p ::= 1 \mid P_a \mid V_a \mid p.p \mid p|p \]

We omit non-deterministic choice, loops, thread creation an join:

\[
\begin{align*}
A & ::= P_a \mid V_a & \text{actions} \\
t & ::= A.t \mid 1 & \text{threads} \\
p & ::= t|t|\ldots|t & \text{programs}
\end{align*}
\]
The trace semantics of a program will be an asynchronous graph:

- a graph $G = (V, E)$ labeled by actions
- with an *independence relation* $I$

relating paths of length 2
The trace semantics of a program will be an **asynchronous graph**:

- a graph $G = (V, E)$ labeled by actions
- with an *independence relation* $I$

relating paths of length 2
Trace semantics

The trace semantics of a program will be an **asynchronous graph**:

- a graph $G = (V, E)$ labeled by actions
- with an *independence relation* $I$

![Diagram of asynchronous graph]

relating paths of length 2

*Homotopy* is the smallest congruence on paths containing $I$. 
Trace semantics

To every program $p$ we associate $(U_p, b_p, e_p)$ defined by:

- $U_1$: terminal graph
- $U_{Pa}$: $b_{Pa} \xrightarrow{P_a} e_{Pa}$
- $U_{Va}$: $b_{Pa} \xrightarrow{V_a} e_{V_a}$
- $U_{p.q}$:

$U_p \cup U_q$ is the “cartesian product” of $U_p$ and $U_q$:

$$(x, y) \xrightarrow{A} (x', y) \quad \text{when} \quad x \xrightarrow{A} x' \in U_p$$

$$(x, y') \xrightarrow{B} (x, y') \quad \text{when} \quad y \xrightarrow{B} y' \in U_q$$

$$(y, x') \xrightarrow{B} (y, y')$$

$$(x, x') \xrightarrow{B} (x, y')$$
Trace semantics

Example:

\[ P_b \cdot V_b \cdot P_a \cdot V_a \mid P_a \cdot V_a \]
Trace semantics

Example:

\[ P_b \cdot V_b \cdot P_a \cdot V_a \ | \ P_a \cdot V_a \]

The **resource function** \( r_a \) associates to every vertex \( x \):

number of releases of \( a \) - number locks of \( a \)
Trace semantics

Example:

\[ P_b \cdot V_b \cdot P_a \cdot V_a \mid P_a \cdot V_a \]

The \textbf{resource function} \( r_a \) associates to every vertex \( x \):

- number of releases of \( a \) - number locks of \( a \)

Ex: \( r_a(x) = -1, \ r_b(x) = 0 \)
Trace semantics

Example:

\[ P_b \cdot V_b \cdot P_a \cdot V_a \ | \ P_a \cdot V_a \]

The resource function \( r_a \) associates to every vertex \( x \):

- number of releases of \( a \) - number locks of \( a \)

Ex: \( r_a(y) = -2 \), \( r_b(y) = 0 \)
Trace semantics

Example:

\[ P_b \cdot V_b \cdot P_a \cdot V_a \mid P_a \cdot V_a \]

The **resource function** \( r_a \) associates to every vertex \( x \):

number of releases of \( a \) - number locks of \( a \)

Ex: \( r_a(y) = -2 < -1 = \kappa_a \)
Trace semantics $T_p$:
$U_p$ where we remove vertices $x$ which do not satisfy

$$0 \leq r_a(x) + \kappa_a \leq \kappa_a$$

Example:

$$P_b . V_b . P_a . V_a \parallel P_a . V_a$$
Geometric semantics

The trace semantics is difficult to use to build intuitions. . .

In a similar way, one can define a geometric semantics where programs are interpreted by directed spaces.
**Geometric semantics**

A path in a topological space $X$ is a continuous map $I = [0, 1] \rightarrow X$.

**Definition**

A d-space $(X, dX)$ consists of

- a topological space $X$
- a set $dX$ of paths in $X$, called directed paths, such that
  - constant paths: every constant path is directed,
  - reparametrization: $dX$ is closed under precomposition with increasing maps $I \rightarrow I$, which are called reparametrizations,
  - concatenation: $dX$ is closed under concatenation.
Geometric semantics

A **path** in a topological space \( X \) is a continuous map \( I = [0, 1] \rightarrow X \).

**Definition**

A **d-space** \((X, dX)\) consists of

- a topological space \( X \)
- a set \( dX \) of paths in \( X \), called **directed paths**, such that
  - **constant paths**: every constant path is directed,
  - **reparametrization**: \( dX \) is closed under precomposition with increasing maps \( I \rightarrow I \), which are called **reparametrizations**,
  - **concatenation**: \( dX \) is closed under concatenation.

**Example**

\((X, \leq)\) space with a partial order, \( dX = \{\text{increasing maps } I \rightarrow X\} \)

\( \vec{I} \): d-space induced by \([0, 1]\)
Geometric semantics

A path in a topological space $X$ is a continuous map $I = [0, 1] \to X$.

**Definition**

A d-space $(X, dX)$ consists of

- a topological space $X$
- a set $dX$ of paths in $X$, called directed paths, such that
  - constant paths: every constant path is directed,
  - reparametrization: $dX$ is closed under precomposition with increasing maps $I \to I$, which are called reparametrizations,
  - concatenation: $dX$ is closed under concatenation.

**Example**

$S^1 = \{e^{i\theta}\} 0 \leq \theta < 2\pi$

d$S^1$: $p(t) = e^{i f(t)}$ for some increasing function $f : I \to \mathbb{R}$
Geometric semantics

To each program $p$ we associate a d-space $(H_p, b_p, e_p)$:

- $H_1$: •
- $H_{Pa} = \vec{I}$, $H_{Va} = \vec{I}$
- $H_{p,q}$:

\[
\begin{align*}
H_p & \ni b_p & H_q & \ni e_q & e_p = b_q
\end{align*}
\]

- $H_{p|q}$: $H_p \times H_q$, $b_{p|q} = (b_p, b_q)$, $e_{p|q} = (e_p, e_q)$
Geometric semantics

To each program $p$ we associate a d-space $(H_p, b_p, e_p)$:

- $H_1$: •
- $H_{P_a} = \vec{I}$, $H_{V_a} = \vec{I}$
- $H_{p,q}$:

$$
\begin{array}{c}
\begin{array}{c}
H_p \cap H_q
\end{array}
\end{array}
$$

- $H_{p|q}$: $H_p \times H_q$, $b_{p|q} = (b_p, b_q)$, $e_{p|q} = (e_p, e_q)$

Resource function: $r_a(x) \in \mathbb{N}$ for each $a \in \mathcal{R}$ and point $x$
Geometric semantics

To each program $p$ we associate a d-space $(H_p, b_p, e_p)$:

- $H_1$: •
- $H_{Pa} = \vec{I}$ $H_{Va} = \vec{I}$
- $H_{p.q}$:

\[
\begin{align*}
H_p & \quad e_p = b_q \quad H_q \\
\end{align*}
\]

- $H_{p|q}$: $H_p \times H_q$, $b_{p|q} = (b_p, b_q)$, $e_{p|q} = (e_p, e_q)$

Resource function: $r_a(x) \in \mathbb{N}$ for each $a \in \mathcal{R}$ and point $x$

Forbidden region:

$F_p = \{ x \in H_p \mid \exists a \in \mathcal{R}, \quad r_a(x) + \kappa_a < 0 \text{ or } r_a(x) > 0 \}$
Geometric semantics

To each program $p$ we associate a d-space $(H_p, b_p, e_p)$:

- $H_1$: •
- $H_{P_a} = \vec{l}$
- $H_{V_a} = \vec{l}$

- $H_{p.q}$:
  
  \begin{align*}
  b_p & \rightarrow H_p \rightarrow e_p = b_q \rightarrow H_q \rightarrow e_q
  \end{align*}

  $H_{p|q}$: $H_p \times H_q$, $b_{p|q} = (b_p, b_q)$, $e_{p|q} = (e_p, e_q)$

Resource function: $r_a(x) \in \mathbb{N}$ for each $a \in \mathcal{R}$ and point $x$

Forbidden region:

$F_p = \{ x \in H_p / \exists a \in \mathcal{R}, \ r_a(x) + \kappa_a < 0 \text{ or } r_a(x) > 0 \}$

Geometric semantics: $G_p = H_p \setminus F_p$
Examples of geometric semantics

\[ P_a \cdot V_a | P_a \cdot V_a \]
Examples of geometric semantics

\[ P_a \cdot V_a \| P_a \cdot V_a \quad P_a \cdot P_b \cdot V_b \cdot V_a \| P_b \cdot P_a \cdot V_a \cdot V_b \]
Examples of geometric semantics

\[ P_a \cdot V_a | P_a \cdot V_a \quad P_a \cdot P_b \cdot V_b \cdot V_a | P_b \cdot P_a \cdot V_a \cdot V_b \quad P_a \cdot (V_a \cdot P_a)^* | P_a \cdot V_a \]
Examples of geometric semantics

\[ P_a \cdot V_a \mid P_a \cdot V_a \mid P_a \cdot V_a \quad (\kappa_a = 2) \]

\[ P_a \cdot V_a \mid P_a \cdot V_a \mid P_a \cdot V_a \quad (\kappa_a = 1) \]
Geometric realization

The two semantics are “essentially the same”: the geometric semantics is the geometric realization of a cubical set

\[ G_p = \int_{n \in \Box} T_p(n) \cdot \vec{I}^n \]

Proposition

Given a program \( p \), with \( T_p \) as trace semantics and \( G_p \) as geometric semantics,

- every path \( \pi : b \to e \) in \( T_p \) induces a path \( \overline{\pi} : b \to e \) in \( G_p \),
- \( \pi \sim \rho \) in \( T_p \) implies \( \overline{\pi} \sim \overline{\rho} \) in \( G_p \)
- every path \( \rho \) of \( G_p \) is homotopic to a path \( \overline{\pi} \) (\( \pi \) path in \( G_p \))
Computing the trace space

Goal

Given a program $p$, we describe an algorithm to compute a trace in each equivalence class of traces $\pi : b_p \to e_p$ up to homotopy in $G_p$.

The proposition before ensures that it is the same to compute this in the trace semantics or in the geometric semantics.
The algorithm

Suppose given a program

\[ p = p_0|p_1|\ldots|p_{n-1} \]

with \textit{n threads}.
The algorithm

Suppose given a program

\[ p = p_0 | p_1 | \cdots | p_{n-1} \]

with \( n \) threads.

Under mild assumptions, the geometric semantics is of the form

\[ G_p = \vec{t}^n \setminus \bigcup_{i=0}^{l-1} R^i \]

where

\[ R^i = \prod_{j=0}^{n-1} [x_j^i, y_j^i] \]

are \( l \) open rectangles.
The algorithm

Under mild assumptions, the geometric semantics is of the form

\[ G_p = \mathcal{I}^n \setminus \bigcup_{i=0}^{l-1} R^i \]

where

\[ R^i = \prod_{j=0}^{n-1} [x^i_j, y^i_j] \]

are open rectangles.

Example

\[ P_a \cdot V_a \cdot P_b \cdot V_b \mid P_b \cdot V_b \cdot P_a \cdot V_a \]
The main idea of the algorithm is to extend the forbidden cubes downwards in various directions and look whether there is a path from $b$ to $e$ in the resulting space.

By combining those information, we will be able to compute traces modulo homotopy.
The algorithm

The main idea of the algorithm is to extend the forbidden cubes downwards in various directions and look whether there is a path from $b$ to $e$ in the resulting space.

By combining those information, we will be able to compute traces modulo homotopy.

The directions in which to extend the holes will be coded by boolean matrices $M$. 

\[ t_1 \quad t_0 \]
The index poset

\( \mathcal{M}_{l,n} \): boolean matrices with \( l \) rows and \( n \) columns.
The index poset

\( \mathcal{M}_{l,n} \): boolean matrices with \( l \) rows and \( n \) columns.

\( \chi_M \):

space obtained by extending

for every \((i, j)\) such that \( M(i, j) = 1 \)

the forbidden cube \( i \) downwards

in every direction other than \( j \)
The index poset

\( \mathcal{M}_{l,n} \): boolean matrices with \( l \) rows and \( n \) columns.

\( \chi_M \):

- space obtained by *extending* for every \((i, j)\) such that \( M(i, j) = 1 \)
- the forbidden cube \( i \) downwards in every direction other than \( j \)

\[
\begin{pmatrix}
1 & 0 \\
1 & 0
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]
The index poset

\( \mathcal{M}_{l,n} \): boolean matrices with \( l \) rows and \( n \) columns.

\( X_M \): space obtained by *extending*

for every \((i, j)\) such that \( M(i, j) = 1 \)

the forbidden cube \( i \) downwards

in every direction other than \( j \)

\[
\begin{pmatrix}
1 & 0 \\
1 & 0
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]

\( \Psi : \mathcal{M}_{l,n} \to \{0, 1\} : \)

- \( \Psi(M) = 0 \) if there is a path \( b \to e \): \( M \) is alive
- \( \Psi(M) = 1 \) if there is no path \( b \to e \): \( M \) is dead
The index poset

\[ P_a \cdot V_a \cdot P_b \cdot V_b \mid P_a \cdot V_a \cdot P_b \cdot V_b \mid P_a \cdot V_a \cdot P_b \cdot V_b \]
The index poset

- $\mathcal{M}_{l,n}$ is equipped with the pointwise ordering
- $\Psi$ is increasing: more $1 \Rightarrow$ more obstructions
- $\mathcal{M}^R_{l,n}$: matrices with non-null rows
- $\mathcal{M}^C_{l,n}$: matrices with unit column vectors
The index poset

- $\mathcal{M}_{l,n}$ is equipped with the pointwise ordering
- $\Psi$ is increasing: more $1 \Rightarrow$ more obstructions
- $\mathcal{M}^R_{l,n}$: matrices with non-null rows
- $\mathcal{M}^C_{l,n}$: matrices with unit column vectors

Definition

The **index poset** $\mathcal{C}(X) = \{ M \in \mathcal{M}^R_{l,n} \mid \Psi(M) = 0 \}$

(the alive matrices).
The index poset

- $\mathcal{M}_{I,n}$ is equipped with the pointwise ordering
- $\Psi$ is increasing: more $1 \Rightarrow$ more obstructions
- $\mathcal{M}^R_{I,n}$: matrices with non-null rows
- $\mathcal{M}^C_{I,n}$: matrices with unit column vectors

Definition
The **index poset** $C(X) = \{ M \in \mathcal{M}^R_{I,n} / \Psi(M) = 0 \}$ (the alive matrices).

Definition
The **dead poset** $D(X) = \{ M \in \mathcal{M}^C_{I,n} / \Psi(M) = 1 \}$. 
The index poset

- $\mathcal{M}_{l,n}$ is equipped with the pointwise ordering
- $\Psi$ is increasing: more 1 $\Rightarrow$ more obstructions
- $\mathcal{M}^R_{l,n}$: matrices with non-null rows
- $\mathcal{M}^C_{l,n}$: matrices with unit column vectors

**Definition**

The **index poset** $C(X) = \{ M \in \mathcal{M}_{l,n}^R / \Psi(M) = 0 \}$ (the alive matrices).

**Definition**

The **dead poset** $D(X) = \{ M \in \mathcal{M}_{l,n}^C / \Psi(M) = 1 \}$.

$D(X) \rightsquigarrow C(X) \rightsquigarrow$ homotopy classes of traces
The dead poset

Proposition

A matrix $M \in M_{l,n}^C$ is in $D(X)$ iff it satisfies

$$\forall (i, j) \in [0 : l] \times [0 : n], \quad M(i, j) = 1 \quad \Rightarrow \quad x^i_j < \min_{i' \in R(M)} y^i_{j'}$$

where $R(M)$: indexes of non-null rows of $M$. 

Example

$M$ is dead:

$$
\begin{array}{ccc}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 1 & 0 \\
\end{array}
$$

$x^0_0 = 1 < 2 = \min_{i' \in R(M)} y^0_{j'}$
The dead poset

Proposition

A matrix $M \in \mathcal{M}_{l,n}^C$ is in $D(X)$ iff it satisfies

$$\forall (i, j) \in [0 : l] \times [0 : n], \quad M(i, j) = 1 \implies x_j^i < \min_{i' \in R(M)} y_j^{i'}$$

where $R(M)$: indexes of non-null rows of $M$.

Example

$M$ is dead:

$$M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$x_1^0 = 1 < 2 = \min(y_1^0, y_1^1)$

$x_0^1 = 2 < 3 = \min(y_0^0, y_0^1)$
Proposition

A matrix $M$ is in $C(X)$ iff for every $N \in D(X)$, $N \not\leq M$. 
Proposition
A matrix $M$ is in $C(X)$ iff for every $N \in D(X)$, $N \not\leq M$.

Remark
$N \not\leq M$: there exists $(i, j)$ s.t. $N(i, j) = 1$ and $M(i, j) = 0$.

Remark
Since $C(X)$ is downward closed it will be enough to compute the set $C_{\text{max}}(X)$ of maximal alive matrices.
Connected components

\[ M \wedge N: \text{ pointwise min of } M \text{ and } N \]

**Definition**
Two matrices \( M \) and \( N \) are **connected** when \( M \wedge N \) does not contain any null row.

**Proposition**
The connected components of \( \mathcal{C}(X) \) are in bijection with homotopy classes of traces \( b \rightarrow e \) in \( X \).
Dining philosophers

\[ p_k = P_{a_k} \cdot P_{a_{k+1}} \cdot V_{a_k} \cdot V_{a_{k+1}} \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>sched.</th>
<th>ALCOOL (s)</th>
<th>ALCOOL (MB)</th>
<th>SPIN (s)</th>
<th>SPIN (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>254</td>
<td>0.1</td>
<td>0.8</td>
<td>0.3</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>510</td>
<td>0.8</td>
<td>1.4</td>
<td>1.5</td>
<td>41</td>
</tr>
<tr>
<td>10</td>
<td>1022</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>179</td>
</tr>
<tr>
<td>11</td>
<td>2046</td>
<td>32</td>
<td>9</td>
<td>42</td>
<td>816</td>
</tr>
<tr>
<td>12</td>
<td>4094</td>
<td>227</td>
<td>26</td>
<td>313</td>
<td>3508</td>
</tr>
<tr>
<td>13</td>
<td>8190</td>
<td>1681</td>
<td>58</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>14</td>
<td>16382</td>
<td>13105</td>
<td>143</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>
What is exactly the geometric information contained in the index poset?
The geometry of the trace space

The $n$-dimensional standard simplex:

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & \ldots \\
\cdot & \cdot & \cdot & \cdot & \ldots
\end{array}
\]
The geometry of the trace space

The $n$-dimensional standard simplex:

$$0 \quad 1 \quad 2 \quad 3 \quad \ldots$$

Definition

A **simplicial** set is a sequence $(X_n)$ of sets of $n$-simplices together with face maps.
The geometry of the trace space

The $n$-dimensional standard simplex:

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & \ldots \\
\end{array}
\]

Definition

A **prodsimplicial** set is a sequence $(X_n)$ of sets of products of $k$-uples $n_i$-simplices ($n = n_1 + \ldots + n_k$) together with face maps.
The geometry of the trace space

Proposition

The index poset $\mathcal{C}(X)$ is a prodsimplicial set, a matrix $M \in \mathcal{C}(X)$ representing a prodsimplex

$$\Delta_{k_0} \times \Delta_{k_2} \times \ldots \times \Delta_{k_{l-1}}$$

where $k_i + 1$ is the number of 1 on the $i$-th line of $M$.

Proposition

The geometric realization of the prodsimplicial set $\mathcal{C}(X)$ is homotopy equivalent to the trace space.
About geometric models

Was the use of the geometric model necessary?
About geometric models

Was the use of the geometric model necessary?
⇒ No: we could have formulated it directly on the trace space

⇒ Yes: it would have been very hard to think of the algorithm without "seeing" the spaces
⇒ Yes: computers are much better at manipulating numbers than complex algebraic structures
About geometric models

Was the use of the geometric model necessary?
⇒ No: we could have formulated it directly on the trace space

Was the geometric model useful?
⇒ Yes: it would have been very hard to think of the algorithm without "seeing" the spaces
⇒ Yes: computers are much better at manipulating numbers than complex algebraic structures
About geometric models

Was the use of the geometric model necessary?

⇒ No: we could have formulated it directly on the trace space

Was the geometric model useful?

⇒ Yes: it would have been very hard to think of the algorithm without “seeing” the spaces
About geometric models

Was the use of the geometric model necessary?
⇒ No: we could have formulated it directly on the trace space

Was the geometric model useful?
⇒ Yes: it would have been very hard to think of the algorithm without “seeing” the spaces
⇒ Yes: computers are much better at manipulating numbers than complex algebraic structures
Future works

We compute one execution trace in each homotopy class.
Future works

We compute **one execution trace in each homotopy class**.

What remains to do:

- use these trace to do static analysis (e.g. abstract interpretation)
- extend the methodology to program with loops
- compute schedulings compositionally
- relate this the component category
- ...