Towards Efficient Computation of Trace Spaces of Concurrent Programs

Samuel Mimram

CEA, LIST
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
Goal

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 and E. Haucourt.
Programs generate trace spaces

Consider the program

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

It can be scheduled in three different ways:

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

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

Consider the program

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

It can be scheduled in three different ways:

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

Giving rise to the following graph of traces:

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

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

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

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

$P_b \cdot V_b \cdot P_a \cdot V_a \mid P_a \cdot V_a$
A program will be interpreted as a **directed space**:

- $P_b . V_b . P_a . V_a$

```
P_b   V_b   P_a   V_a
```
Geometric semantics

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

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

- \( P_a \cdot V_a \)
A program will be interpreted as a directed space:

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

- $P_a \cdot V_a$

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

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

- $P_b. V_b. P_a. V_a$

- $P_a. V_a$

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

\[
\begin{array}{cccc}
  P_b & V_b & P_a & V_a \\
  \hline \\
  P_a & V_a \\
\end{array}
\]
A program will be interpreted as a **directed space**:

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

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

- $P_a \cdot V_a$

\[ P_a \quad V_a \]

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

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

**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.
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

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

A forbidden region
The Swiss flag


The Swiss flag

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

A deadlock: \( P_b P_a \)
The Swiss flag

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

An unreachable region
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:

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

We omit non-deterministic choice, loops
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
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 $I$

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* \( I \)

\[
\begin{array}{c}
y_1 \xrightarrow{B} z \\
A \sim A \\
x \xrightarrow{B} y_2
\end{array}
\]

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
- $U_{p_a}$: $b_{p_a} \xrightarrow{P_a} e_{p_a}$
- $U_{v_a}$: $b_{p_a} \xrightarrow{V_a} e_{v_a}$
- $U_{p.q}$:

$U_{p.q}$ is the “cartesian product” of $U_p$ and $U_q$:

\[
\begin{align*}
(x, y) & \xrightarrow{A} (x', y) \\
(x, y') & \xrightarrow{B} (x, y') \quad \text{when } y \xrightarrow{B} y' \in U_q
\end{align*}
\]

\[
\begin{align*}
(y, x') & \xrightarrow{B} (y, y') \quad \text{when } x \xrightarrow{A} x' \in U_p
\end{align*}
\]
Trace semantics

Example:

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

Example:

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

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

- number of releases of \( a \) - number locks of \( a \)
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: \( r_a(x) = -1 \), \( r_b(x) = 0 \)
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: \( r_a(y) = -2 \), \( r_b(y) = 0 \)
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:

$P_b . V_b . P_a . V_a \mid P_a . V_a$
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.
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.
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**

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

$I$: 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

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

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

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

- $H_1$:

- $H_{Pa} = \overrightarrow{I}$  \hspace{1cm} $H_{Va} = \overrightarrow{I}$

- $H_{p.q}$:

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

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

- $H_1$: $\bullet$
- $H_{pa} = \vec{I}$ \hspace{1cm} $H_{Va} = \vec{I}$
- $H_{p,q}$:

\[
\begin{align*}
H_p & \quad \bullet \quad \bullet \quad \bullet \\
\text{Forbidden region:} & \quad F_p = \{ x \in H_p / \exists a \in \mathbb{R}, r_a(x) + \kappa_a < 0 \text{ or } r_a(x) > 0 \}
\end{align*}
\]

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

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

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

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

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

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

Forbidden region:
Geometric semantics

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

- $H_1$: \[ b_p \]
- $H_{Pa} = \vec{I}$ \quad $H_{Va} = \vec{I}$
- $H_{p.q}$:
  \[ \begin{array}{c}
  b_p \\
  H_p \\
  e_p = b_q \\
  H_q \\
  e_q
  \end{array} \]

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

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

**Forbidden region:**
\[ F_p = \{ x \in H_p \mid \exists a \in \mathcal{R}, \quad r_a(x) + \kappa_a < 0 \quad \text{or} \quad r_a(x) > 0 \} \]

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

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

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

\[ P_a \cdot V_a | P_a \cdot V_a | P_a \cdot V_a \]
\[ (\kappa_a = 2) \]

\[ P_a \cdot V_a | P_a \cdot V_a | P_a \cdot V_a \]
\[ (\kappa_a = 1) \]
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

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

with \( n \) threads.
The algorithm

Suppose given a program

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

with \( n \) threads.

Under mild assumptions, the geometric semantics is of the form

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

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

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

where

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

are \( l \) open rectangles.

Example

\[
P_a \cdot V_a \cdot P_b \cdot V_b | P_b \cdot V_b \cdot P_a \cdot V_a
\]
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 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 $M$. 
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.

\( \chi_{\mathcal{M}} \):

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 \( j \)

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

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

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

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

alive \quad alive \quad alive \quad dead
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

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

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

- $\mathcal{M}_{l,n}$ is equipped with the pointwise ordering
- $\Psi$ is increasing: more $1 \Rightarrow$ more obstructions
- $\mathcal{M}_{l,n}^R$: matrices with non-null rows
- $\mathcal{M}_{l,n}^C$: 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** $D(X) = \{ M \in \mathcal{M}_{l,n}^C / \Psi(M) = 1 \}$. 
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** $D(X) = \{ M \in \mathcal{M}^C_{I,n} / \Psi(M) = 1 \}$.

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

Proposition

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

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

where \( R(M) \): indexes of non-null rows of \( M \).
The dead poset

Proposition

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

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

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

Example

$M$ is dead:

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

\[
x^0_1 = 1 < 2 = \min(y^0_1, y^1_1)
\]

\[
x^1_0 = 2 < 3 = \min(y^0_0, y^1_0)
\]
The index poset

Proposition

A matrix $M$ is in $C(X)$ iff for every $N \in D(X)$, $N \nless M$. 

Remark

$N \nless M$: 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.
The index poset

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.
Remark

The index poset contains all the geometrical information!
**Connected components**

$M \wedge N$: pointwise min of $M$ 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 $C(X)$ are in bijection with homotopy classes of traces $b \rightarrow e$ in $X$.***
$n$ processes $p_k$ in parallel:

$$p_k = P_{a_k} P_{a_{k+1}} V_{a_k} V_{a_{k+1}}$$

Dining philosophers

<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>
How do we extend this methodology to program with loops?
Loops

Given a thread $p$, we write $p^*$ for its looping: \texttt{while(...){$p$}}.
Loops

Given a thread $p$, we write $p^*$ for its looping: \texttt{while(...){}\{}p\texttt{}}.

Given a program $p$ with $n$ threads:

$$ p = p_1|p_2|\ldots|p_n $$

we write $p^*$ for

$$ p^* = p_1^*|p_2^*|\ldots|p_n^* $$
Loops

Given a thread \( p \), we write \( p^* \) for its looping: while(\( \ldots \)\{\( p \}\).

Given a program \( p \) with \( n \) threads:

\[
p = p_1 | p_2 | \ldots | p_n
\]

we write \( p^* \) for

\[
p^* = p_1^* | p_2^* | \ldots | p_n^*
\]

Notice that the geometric semantics \( X_{p^*} \) can be deduced from the semantics of \( p \) by glueing copies of \( X_p \) in every direction:

\[
p_i^* = p_i \cdot p_i \cdot p_i \ldots
\]
Notice that the geometric semantics $X_{p^*}$ can be deduced from the semantics of $p$ by glueing copies of $X_p$ in every direction.

Example
Consider the program $p = q|q|q$ with $q = P_a.V_a$ (and $a$ of arity 3):
Deloopings

Notice that the geometric semantics $X_p^*$ can be deduced from the semantics of $p$ by glueing copies of $X_p$ in every direction.

Example

Consider the program $p = q | q | q$ with $q = P_a . V_a$ (and $a$ of arity 3):

Finite deloopings:

$$X_p^{(3,2,2)} = (Y \oplus_1 Y) \oplus_2 (Y \oplus_1 Y)$$

with

$$Y = X_p \oplus_0 X_p \oplus_0 X_p$$
Similarly, given schedulings

\[ M = (1 \ 0 \ 0) \quad \text{and} \quad N = (0 \ 0 \ 1) \]

of the previous program \( p \)
Schedulings

Similarly, given schedulings

\[ M = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \quad \text{and} \quad N = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \]

of the previous program \( p \)

we write

\[ M \oplus_0 N = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \]

for the following scheduling of \( X_p^{(2,1,1)} = X_p \oplus_0 X_p \)

\[ X_{M \oplus_0 N} = \]
Similarly, given schedulings

\[ M = (1 \ 0 \ 0) \quad \text{and} \quad N = (0 \ 0 \ 1) \]

of the previous program \( p \)

we write

\[ M \oplus_0 N = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \]

for the following scheduling of \( X_p^{(2,1,1)} = X_p \oplus_0 X_p \)
Shadows

In fact, scheduling drop "shadows" on previous schedulings

\[ X_{M \oplus_0 N} = \]

\[ \neq \]

\[ = X_{M \oplus_0 X_N} \]
Shadows

In fact, scheduling drop “shadows” on previous schedulings

\[ X_{M \oplus_0 N} \neq X_{M \oplus_0 X_N} \]

Write \( X_{M|_j} \) for the shadow projected by scheduling \( M \) in direction \( j \):

\[ X_{N|_0} \]

so that

\[ X_{M \oplus_j N} = (X_M \cap X_{N|_j}) \otimes_j X_N \]
Alive matrices for programs with loops

Every scheduling $M$ of a delooping of $X_p$ is composed by glueing submatrices $(M_{i_1},...,i_n)$. 

Lemma

If a matrix $M$ is alive then all its submatrices are alive. The converse is not true!
Alive matrices for programs with loops

Every scheduling $M$ of a delooping of $X_p$ is composed by glueing submatrices $(M_{i_1},...,i_n)$.

If $X_M$ contains a deadlock then some subspace $X_{(M_{i_1},...,i_n)}$ contains a deadlock:

**Lemma**

*If a matrix $M$ is alive then all its submatrices are alive.*
Alive matrices for programs with loops

Every scheduling $M$ of a delooping of $X_p$ is composed by glueing submatrices $(M_{i_1},...,i_n)$.

If $X_M$ contains a deadlock then some subspace $X_{(M_{i_1},...,i_n)}$ contains a deadlock:

Lemma

*If a matrix $M$ is alive then all its submatrices are alive.*

The converse is not true!
Shadows can create deadlocks

The following matrices $P$ and $Q$ coding the schedulings

of $p$ are alive, however the matrix $P \oplus_0 Q$ is dead:

$$X_{P \oplus_0 Q} =$$
The shadow automaton

We construct an automaton which describes all the schedulings possible in the future (which won't create deadlocks by their shadow): given a scheduling $M$ and a direction $j$, it describes all the matrices $N$ such that $M \oplus j N$ is alive.
The shadow automaton

Definition
The **shadow automaton** of a program \( p \) is a non-deterministic automaton whose

- states are shadows
- transitions \( N \xrightarrow{j,M} N' \) are labeled by a direction \( j \) (with \( 0 \leq j < n \)) and a scheduling \( M \)
defined as the smallest automaton
  - containing the empty scheduling \( \emptyset \)
  - and such that for every state \( N' \), for every direction \( j \) and for every scheduling \( M \) such that the scheduling \( M \cup N' \) is alive, and \( M \) is maximal with this property, there is a transition
  \[
  N \xrightarrow{j,M} N' \quad \text{with} \quad N = (M \cup N')|_j.
  \]
All the states of the automaton are both initial and final.
The shadow automaton

For instance consider the program \( p = P_a \cdot V_a | P_a \cdot V_a \)

\[
X_p = t_0 \rightarrow t_1
\]
The shadow automaton

For instance consider the program $p = P_a.V_a | P_a.V_a$

$$X_p =$$

There are two maximal schedulings:

1. $t_0 \rightarrow t_1 \rightarrow t_0$
2. $t_0 \rightarrow t_1 \rightarrow t_0$
The shadow automaton

For instance consider the program $p = P_a \cdot V_a | P_a \cdot V_a$

$$X_p =$$

There are two maximal schedulings

which can drop three possible shadows
The shadow automaton

The shadow automaton of $p$ is

\[
\begin{align*}
1, & \xrightarrow{} 1, \\
0, & \xrightarrow{} 0, \\
1, & \xrightarrow{} 1, \\
0, & \xrightarrow{} 0, \\
1, & \xrightarrow{} 1, \\
0, & \xrightarrow{} 0,
\end{align*}
\]
The shadow automaton

The shadow automaton of $p$ is

![Diagram of the shadow automaton]

For instance, the transition $\begin{array}{l} \text{0, } \hline \text{0, } \hline \text{1, } \hline \end{array} \rightarrow \begin{array}{l} \text{0, } \hline \text{0, } \hline \text{1, } \hline \end{array}$ is computed as follows:

- consider the shadow $M = \begin{array}{l} \text{0, } \hline \text{0, } \hline \text{1, } \hline \end{array} \cup \begin{array}{l} \text{0, } \hline \text{0, } \hline \text{1, } \hline \end{array} = \begin{array}{l} \text{0, } \hline \text{0, } \hline \text{1, } \hline \end{array}$
- compute its shadow in direction 0: \begin{array}{l} \text{0, } \hline \text{0, } \hline \text{1, } \hline \end{array}
The shadow automaton

Theorem

Given a program $p$ to any total path in a delooping of $p$ is represented by a path in the shadow automaton of $p$ such that

- every path in the automaton comes from a total path in $X_p$
- if two total paths in $X_p$ correspond to the same path in the automaton then they are homotopic

Paths in the shadow automaton describe homotopy classes in deloopings of $p$. 
The shadow automaton

Theorem

Given a program $p$ to any total path in a delooping of $p$ is represented by a path in the shadow automaton of $p$ such that

- every path in the automaton comes from a total path in $X_p$.
- if two total paths in $X_p$ correspond to the same path in the automaton then they are homotopic.

Paths in the shadow automaton describe homotopy classes in deloopings of $p$. 

\[ t_0 \xrightarrow{0} t_1 \xrightarrow{1} \]
Reducing the size of the automaton

The shadow automaton is too big:

- we can determinize it:

```
+---+ 1,₁
|   v
+--- 1
+--- 0,₁

+--- 0,₀
|   v
+--- 0
```

- two distinct paths in the automaton can represent the same homotopy class of paths: we can quotient paths under connexity.
Reducing the size of the automaton

The shadow automaton is too big:

- we can determinize it:

- two distinct paths in the automaton can represent the same homotopy class of paths: we can quotient paths under connexity.
An application to static analysis

The program

$$p^* = \left( P_a.a := a - 1. V_a \right)^* \left| \left( P_a.(a := \frac{a}{2}). V_a \right)^* \right.$$  

induces the automaton

\[
\begin{array}{c}
0 \overset{[a := a-1]}{\longrightarrow} [a := \frac{a}{2}] \overset{[a := a-1]}{\longrightarrow} 1 \end{array}
\]
An application to static analysis

The program

\[ p^* = (P_a.a := a - 1.V_a)^* \bigg| (P_a.(a := \frac{a}{2}).V_a)^* \]

induces the automaton

\[
\begin{array}{ccc}
[\text{a:=a-1}] & [\text{a:=}\frac{a}{2}] \\
0 \quad \text{←} & \text{←} & 1 \quad \text{→} \\
[a:=\frac{a}{2}] & [a:=\text{a-1}] & [a:=\text{a-1}] \quad [a:=\text{a-1}] \quad [a:=\text{a-1}] \\
& & \text{→} & \text{→} & \text{→}
\end{array}
\]

and thus the set of equations

\[
\begin{align*}
A_0 &= I \cup (A_0 - 1) \cup (A_1 - 1) \\
A_1 &= I \cup \frac{A_1}{2} \cup \frac{A_0}{2}
\end{align*}
\]
An application to static analysis

The program

\[ p^* = \left( P_a.a := a - 1.V_a \right)^* \mid \left( P_a.(a := \frac{a}{2}).V_a \right)^* \]

induces the automaton

and thus the set of equations

\[
\begin{align*}
A_0 &= I \cup (A_0 - 1) \cup (A_1 - 1) \\
A_1 &= I \cup \frac{A_1}{2} \cup \frac{A_0}{2}
\end{align*}
\]

which admits a least fixed point

\[ A_0^\infty = A_1^\infty = \left[ -\infty, 1 \right] \]
An example: the two-phase protocol

We consider $n$ programs locking $l$ resources:

$$p_{n,l} = q|q|\ldots|q \quad \text{with} \quad q = P_{a_1}\ldots P_{a_l}\cdot V_{a_1}\ldots V_{a_l}$$
An example: the two-phase protocol

We consider \( n \) programs locking \( l \) resources:

\[
p_{n,l} = q \| q \| \ldots \| q
\]

with

\[
q = P_{a_1} \ldots P_{a_l} \cdot V_{a_1} \ldots V_{a_l}
\]

For instance, \( p_{2,2} = q \| q \) is
An example: the two-phase protocol

We consider \( n \) programs locking \( l \) resources:

\[
p_{n,l} = q|q| \ldots |q \quad \text{with} \quad q = P_{a_1} \ldots P_{a_l} \cdot V_{a_1} \ldots V_{a_l}
\]

For instance, \( p_{2,2} = q|q \) is

\[
\begin{array}{c}
t_0 \\
\uparrow \\
t_1
\end{array}
\]

We get the following results compared to spin:

<table>
<thead>
<tr>
<th>( n )</th>
<th>( l )</th>
<th>( s )</th>
<th>( t )</th>
<th>( s' )</th>
<th>( t' )</th>
<th>( s'' )</th>
<th>( t'' )</th>
<th>( s_{SPIN} )</th>
<th>( t_{SPIN} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>58</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>112</td>
<td>129</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>180</td>
<td>209</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>19</td>
<td>90</td>
<td>4</td>
<td>24</td>
<td>1</td>
<td>1</td>
<td>171</td>
<td>218</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>19</td>
<td>90</td>
<td>4</td>
<td>24</td>
<td>1</td>
<td>1</td>
<td>441</td>
<td>602</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>19</td>
<td>90</td>
<td>4</td>
<td>24</td>
<td>1</td>
<td>1</td>
<td>817</td>
<td>1128</td>
</tr>
</tbody>
</table>
About geometric models

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

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

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 booleans 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)
- speed improvements
- implementation improvements (e.g. GPU)
- lots of work remain to be done on the theoretical side