Automata minimization and glueing of categories

13 12 2017 Berkeley
Thomas Colcombet
joint work with Daniela Petrişan
Automata minimization and glueing of categories

[MFCS 2017] & [Informal presentation in SIGLOG column]

13 12 2017 Berkeley
Thomas Colcombet
joint work with Daniela Petrișan
Description of the situation
Automata
An deterministic automaton is
\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]
where
- \( Q \) is a set of **states**, 
- \( i : 1 \rightarrow Q \) is the **initial map**
- \( f : Q \rightarrow 2 \) is the **final map**
- \( \delta_a : Q \rightarrow Q \) is the **transition map**
An deterministic automaton is

$$\langle Q, i, f, (\delta_a)_{a \in A} \rangle$$

where

- $Q$ is a set of states,
- $i : 1 \rightarrow Q$ is the initial map
- $f : Q \rightarrow 2$ is the final map
- $\delta_a : Q \rightarrow Q$ is the transition map

Rabin & Scott
An **deterministic automaton** is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is a set of **states**, 
- \( i : 1 \to Q \) is the **initial map**
- \( f : Q \to 2 \) is the **final map**
- \( \delta_a : Q \to Q \) is the **transition map**

It computes the **language**:

\[ \langle A \rangle : A^* \to [1, 2] \]

\[ u \mapsto f \circ \delta_u \circ i \]
An deterministic automaton is
\[
\langle Q, i, f, (\delta_a)_{a \in A} \rangle
\]
where
- \(Q\) is a set of states,
- \(i: 1 \to Q\) is the initial map
- \(f: Q \to 2\) is the final map
- \(\delta_a: Q \to Q\) is the transition map

It computes the language:
\[
[\mathcal{A}]: A^* \to [1, 2] \approx 2
\]
\[
u \mapsto f \circ \delta_u \circ i
\]
Automata

An **deterministic automaton** is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is a set of **states**,
- \( i : 1 \to Q \) is the **initial map**
- \( f : Q \to 2 \) is the **final map**
- \( \delta_a : Q \to Q \) is the **transition map**

It computes the **language**:

\[ [A] : A^* \to [1, 2] \approx 2 \]

\[ u \mapsto f \circ \delta_u \circ i \]

A **vector automaton** is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is an \( \mathbb{R} \)-**vector space**
- \( i : \mathbb{R} \to Q \) is a **linear map**
- \( f : Q \to \mathbb{R} \) is a **linear map**
- \( \delta_a : Q \to Q \) is a **linear map**
Automata

An **deterministic automaton** is
\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]
where
- \( Q \) is a set of **states**, 
- \( i : 1 \to Q \) is the **initial map**
- \( f : Q \to 2 \) is the **final map**
- \( \delta_a : Q \to Q \) is the **transition map**

It computes the **language**:
\[
[A] : A^* \to [1, 2] \approx 2 \\
u \mapsto f \circ \delta_u \circ i
\]

A **vector automaton** is
\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]
where
- \( Q \) is an \( \mathbb{R} \)-**vector space**
- \( i : \mathbb{R} \to Q \) is a **linear map**
- \( f : Q \to \mathbb{R} \) is a **linear map**
- \( \delta_a : Q \to Q \) is a **linear map**

Schützenberger’s automata weighted over a field

Rabin & Scott
An **deterministic automaton** is
\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]
where
- \( Q \) is a set of **states**,
- \( i : 1 \to Q \) is the **initial map**
- \( f : Q \to 2 \) is the **final map**
- \( \delta_a : Q \to Q \) is the **transition map**

It computes the **language**:
\[
[A]: A^* \to [1, 2] \approx 2 \\
u \mapsto f \circ \delta_u \circ i
\]

A **vector automaton** is
\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]
where
- \( Q \) is an \( \mathbb{R} \)-**vector space**
- \( i : \mathbb{R} \to Q \) is a **linear map**
- \( f : Q \to \mathbb{R} \) is a **linear map**
- \( \delta_a : Q \to Q \) is a **linear map**

It computes the **language**:
\[
[A]: A^* \to [\mathbb{R}, \mathbb{R}] \\
u \mapsto f \circ \delta_u \circ i
\]
An **deterministic automaton** is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is a set of **states**, 
- \( i: 1 \to Q \) is the **initial map** 
- \( f: Q \to 2 \) is the **final map** 
- \( \delta_a: Q \to Q \) is the **transition map**

It computes the **language**:

\[ [A]: A^* \to [1, 2] \approx 2 \]

\[ u \mapsto f \circ \delta_u \circ i \]

A **vector automaton** is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is an \( \mathbb{R} \)-**vector space** 
- \( i: \mathbb{R} \to Q \) is a **linear map** 
- \( f: Q \to \mathbb{R} \) is a **linear map** 
- \( \delta_a: Q \to Q \) is a **linear map**

It computes the **language**:

\[ [A]: A^* \to [\mathbb{R}, \mathbb{R}] \approx \mathbb{R} \]

\[ u \mapsto f \circ \delta_u \circ i \]
An **deterministic automaton** is
\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]
where
- \( Q \) is a set of **states**,
- \( i : 1 \to Q \) is the **initial map**
- \( f : Q \to 2 \) is the **final map**
- \( \delta_a : Q \to Q \) is the **transition map**

It computes the **language**:
\[ [A] : A^* \to [1, 2] \approx 2 \]
\[ u \mapsto f \circ \delta_u \circ i \]

A **vector automaton** is
\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]
where
- \( Q \) is an \( \mathbb{R} \)-**vector space**
- \( i : \mathbb{R} \to Q \) is a **linear map**
- \( f : Q \to \mathbb{R} \) is a **linear map**
- \( \delta_a : Q \to Q \) is a **linear map**

It computes the **language**:
\[ [A] : A^* \to [\mathbb{R}, \mathbb{R}] \approx \mathbb{R} \]
\[ u \mapsto f \circ \delta_u \circ i \]
Automata

An deterministic automaton is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is a set of states,
- \( i : 1 \to Q \) is the initial map
- \( f : Q \to 2 \) is the final map
- \( \delta_a : Q \to Q \) is the transition map

It computes the language:

\[ [A] : A^* \to [1, 2] \approx 2 \]

\[ u \mapsto f \circ \delta_u \circ i \]

A vector automaton is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is an \( \mathbb{R} \)-vector space
- \( i : \mathbb{R} \to Q \) is a linear map
- \( f : Q \to \mathbb{R} \) is a linear map
- \( \delta_a : Q \to Q \) is a linear map

It computes the language:

\[ [A] : A^* \to [\mathbb{R}, \mathbb{R}] \approx \mathbb{R} \]

\[ u \mapsto f \circ \delta_u \circ i \]

These data can modeled as a functor.
Example

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

\(Q\) is an \(\mathbb{R}\)-vector space

\(i: \mathbb{R} \rightarrow Q\) is a linear map

\(f: Q \rightarrow \mathbb{R}\) is a linear map

\(\delta_a : Q \rightarrow Q\) is a linear map
Example

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise} 
\end{cases} \]

\( Q \) is an \( \mathbb{R} \)-vector space

\( i: \mathbb{R} \rightarrow Q \) is a linear map

\( f: Q \rightarrow \mathbb{R} \) is a linear map

\( \delta_a: Q \rightarrow Q \) is a linear map

\( Q = \mathbb{R}^2 \)
Example

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u_a| & \text{if } |u_b| \text{ is even, and } |u_c| = 0 \\
0 & \text{otherwise}
\end{cases} \]

\( Q \) is an \( \mathbb{R} \)-vector space

\( i: \mathbb{R} \to Q \) is a linear map

\( f: Q \to \mathbb{R} \) is a linear map

\( \delta_a: Q \to Q \) is a linear map

\( Q = \mathbb{R}^2 \)

\( i(x) = (x, 0) \)
Example

\[ L_\text{Vec}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

\[ Q \text{ is an } \mathbb{R}-\text{vector space} \]
\[ i : \mathbb{R} \rightarrow Q \text{ is a linear map} \]
\[ f : Q \rightarrow \mathbb{R} \text{ is a linear map} \]
\[ \delta_a : Q \rightarrow Q \text{ is a linear map} \]

\[ Q = \mathbb{R}^2 \]
\[ i(x) = (x, 0) \]
\[ f(x, y) = x \]
Example

\[ L_{\text{vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

\( Q \) is an \( \mathbb{R} \)-vector space
\( i: \mathbb{R} \to Q \) is a linear map
\( f: Q \to \mathbb{R} \) is a linear map
\( \delta_a: Q \to Q \) is a linear map

\( Q = \mathbb{R}^2 \)
\( i(x) = (x, 0) \)
\( f(x, y) = x \)
\( \delta_a(x, y) = (2x, 2y) \)
\( \delta_b(x, y) = (y, x) \)
\( \delta_c(x, y) = (0, 0) \)
Example

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

\[ Q \text{ is an } \mathbb{R}\text{-vector space} \]

\[ i: \mathbb{R} \rightarrow Q \text{ is a linear map} \]
\[ f: Q \rightarrow \mathbb{R} \text{ is a linear map} \]
\[ \delta_a: Q \rightarrow Q \text{ is a linear map} \]

\[ Q = \mathbb{R}^2 \]
\[ i(x) = (x, 0) \]
\[ f(x, y) = x \]
\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]

Is it possible to do better?
A better implementation

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

Solution in vector spaces

\[ Q = \mathbb{R}^2 \]
\[ i(x) = (x, 0) \]
\[ f(x, y) = x \]
\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]
A better implementation

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

Informally: use one bit for the parity to the number of b’s.

Solution in vector spaces

\[ Q = \mathbb{R}^2 \]
\[ i(x) = (x, 0) \]
\[ f(x, y) = x \]
\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]
A better implementation

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

**Informally**: use one bit for the parity to the number of b’s.

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

**Solution in vector spaces**

\[ Q = \mathbb{R}^2 \]

\[ i(x) = (x, 0) \]

\[ f(x, y) = x \]

\[ \delta_a(x, y) = (2x, 2y) \]

\[ \delta_b(x, y) = (y, x) \]

\[ \delta_c(x, y) = (0, 0) \]
A better implementation

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise} 
\end{cases} \]

Informally: use one bit for the parity to the number of b’s.

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]
\[ i(x) = (\text{even}, x) \]

Solution in vector spaces

\[ Q = \mathbb{R}^2 \]
\[ i(x) = (x, 0) \]
\[ f(x, y) = x \]
\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]
A better implementation

\[ L_{Vec}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

**Informally:** use one bit for the parity to the number of b’s.

\[
Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R})
\]

\[
i(x) = (\text{even}, x)
\]

\[
f(\text{even}, x) = x
\]

\[
f(\text{odd}, x) = 0
\]

**Solution in vector spaces**

\[
Q = \mathbb{R}^2
\]

\[
i(x) = (x, 0)
\]

\[
f(x, y) = x
\]

\[
\delta_a(x, y) = (2x, 2y)
\]

\[
\delta_b(x, y) = (y, x)
\]

\[
\delta_c(x, y) = (0, 0)
\]
A better implementation

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

Informally: use one bit for the parity to the number of b’s.

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]

Solution in vector spaces

\[ Q = \mathbb{R}^2 \]

\[ i(x) = (x, 0) \]

\[ f(x, y) = x \]

\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]
A better implementation

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

**Informally**: use one bit for the parity to the number of b’s.

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]
\[ i(x) = (\text{even}, x) \]
\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]
\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]
\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]

**Solution in vector spaces**

\[ Q = \mathbb{R}^2 \]
\[ i(x) = (x, 0) \]
\[ f(x, y) = x \]
\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]
A better implementation

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

Informally: use one bit for the parity to the number of b's.

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]

\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]

\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]
\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]

Solution in vector spaces

\[ Q = \mathbb{R}^2 \]

\[ i(x) = (x, 0) \]

\[ f(x, y) = x \]

\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]
A better implementation

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise} 
\end{cases} \]

**Informally**: use one bit for the parity to the number of b’s.

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]

\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]

\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]

\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]

\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]

\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]

\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]

**Solution in vector spaces**

\[ Q = \mathbb{R}^2 \]

\[ i(x) = (x, 0) \]

\[ f(x, y) = x \]

\[ \delta_a(x, y) = (2x, 2y) \]

\[ \delta_b(x, y) = (y, x) \]

\[ \delta_c(x, y) = (0, 0) \]

Why is it a better implementation?

Is there a good notion of such automata?

What are their properties (e.g. minimization)?
A definition via categories
Automata in a category
Automata in a category

A \((C,I,F)\)-automaton is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is a object of states,
- \( i : I \rightarrow Q \) is the initial arrow
- \( f : Q \rightarrow F \) is the final arrow
- \( \delta_a : Q \rightarrow Q \) is the transition arrow for the letter \( a \).
Automata in a category

A (C,I,F)-automaton is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is a object of states,
- \( i : I \to Q \) is the initial arrow
- \( f : Q \to F \) is the final arrow
- \( \delta_a : Q \to Q \) is the transition arrow for the letter \( a \).

The (C,I,F)-language computed is:

\[ [A] : A^* \to [I, F] \]

\[ u \mapsto f \circ \delta_u \circ i \]
Automata in a category

A (C,I,F)-automaton is

\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]

where

- \( Q \) is a object of states,
- \( i : I \to Q \) is the initial arrow
- \( f : Q \to F \) is the final arrow
- \( \delta_a : Q \to Q \) is the transition arrow for the letter \( a \).

The (C,I,F)-language computed is:

\[ [A] : A^* \to [I, F] \]

\[ u \mapsto f \circ \delta_u \circ i \]

Auto(L) is the category of (C,I,F)-automata for the (C,I,F)-language L.
Automata in a category

A \((C,I,F)\)-automaton is
\[
\langle Q, i, f, (\delta_a)_{a \in A} \rangle
\]
where

\(Q\) is a object of states,
\(i : I \to Q\) is the initial arrow
\(f : Q \to F\) is the final arrow
\(\delta_a : Q \to Q\) is the transition arrow for the letter \(a\).

The \((C,I,F)\)-language computed is:
\[
[A] : A^* \to [I, F]
\]
\[u \mapsto f \circ \delta_u \circ i\]

Auto(\(L\)) is the category of \((\mathcal{O}, \mathcal{I}, \mathcal{F})\)-automata for the \((\mathcal{O}, \mathcal{I}, \mathcal{F})\)-language \(L\).
Automata in a category

A (C,I,F)-automaton is
\[ \langle Q, i, f, (\delta_a)_{a \in A} \rangle \]
where
\[ Q \] is a object of states,
\[ i: I \to Q \] is the initial arrow
\[ f: Q \to F \] is the final arrow
\[ \delta_a: Q \to Q \] is the transition arrow for the letter \( a \).

The (C,I,F)-language computed is:
\[ [A]: A^* \to [I, F] \]
\[ u \mapsto f \circ \delta_u \circ i \]

Auto(L) is the category of (C,I,F)-automata for the (C,I,F)-language L.

\[ h: Q_A \to Q_B \]
such that tfdc:

Rk: Morphisms preserve the language.
Automata in a category

A \((C,I,F)\)-automaton is
\[
\langle Q, i, f, (\delta_a)_{a \in A} \rangle
\]
where
- \(Q\) is a object of states,
- \(i: I \rightarrow Q\) is the initial arrow
- \(f: Q \rightarrow F\) is the final arrow
- \(\delta_a: Q \rightarrow Q\) is the transition arrow for the letter \(a\).

The \((C,I,F)\)-language computed is:
\[
\{ [\mathcal{A}] : A^* \rightarrow [I, F] \}
\]
where
\[
u \mapsto f \circ \delta_u \circ i
\]

Auto\((L)\) is the category of \((C,I,F)\)-automata for the \((C,I,F)\)-language \(L\).

- \((\text{Set},1,2)\)-automata are deterministic automata
- \((\text{Rel},1,1)\)-automata are non-deterministic automata
- \((\text{Vec},K,K)\)-automata are automata weighted over a field \(K\). (more generally semi-modules)
- ...

A morphism is an arrow
\[
h: Q_A \rightarrow Q_B
\]
such that tfdc:

\[
\text{Rk: Morphisms preserve the language.}
\]
Category of disjoint unions of vector spaces

(free co-product completion of Vec)
Category of disjoint unions of vector spaces

A disjoint union of vector space is an ordered pair

\[(I, (V_i)_{i \in I})\]

where \(I\) is a set of indices, and \(V_i\) is a vector space for all \(i \in I\).
A disjoint union of vector space is an ordered pair

$$(I, (V_i)_{i \in I})$$

where $I$ is a set of indices, and $V_i$ is a vector space for all $i \in I$.

Let $\textbf{Duvs}$ be the category with
- as objects the finite unions of vector spaces
- as arrows the morphisms of finite unions of vector spaces.
Category of disjoint unions of vector spaces

A disjoint union of vector space is an ordered pair

\[(I, (V_i)_{i \in I})\]

where \(I\) is a set of indices, and \(V_i\) is a vector space for all \(i \in I\).

Let Duvs be the category with
- as objects the finite unions of vector spaces
- as arrows the morphisms of finite unions of vector spaces.

A morphism from \((I, (V_i)_{i \in I})\) to \((J, (W_i)_{i \in J})\) is the pair of:
Category of disjoint unions of vector spaces

(free co-product completion of Vec)

A disjoint union of vector space is an ordered pair

\[(I, (V_i)_{i \in I})\]

where \(I\) is a set of indices, and \(V_i\) is a vector space for all \(i \in I\).

Let \(\text{Duvs}\) be the category with

- as objects the finite unions of vector spaces
- as arrows the morphisms of finite unions of vector spaces.

A morphism from \((I, (V_i)_{i \in I})\) to \((J, (W_i)_{i \in J})\) is the pair of:

- a map \(f\) from \(I\) to \(J\)
A disjoint union of vector space is an ordered pair

\((I, (V_i)_{i \in I})\)

where \(I\) is a set of indices, and \(V_i\) is a vector space for all \(i \in I\).

Let \(\text{Duvs}\) be the category with
- as objects the finite unions of vector spaces
- as arrows the morphisms of finite unions of vector spaces.

A morphism from \((I, (V_i)_{i \in I})\) to \((J, (W_i)_{i \in J})\) is the pair of:
- a map \(f\) from \(I\) to \(J\)
- a linear map \(g_i\) from \(V_i\) to \(W_{f(i)}\) for all \(i \in I\).
A disjoint union of vector space is an ordered pair
\[(I, (V_i)_{i \in I})\]
where \(I\) is a set of indices, and \(V_i\) is a vector space for all \(i \in I\).

Let \(\text{Duvs}\) be the category with
- as objects the finite unions of vector spaces
- as arrows the morphisms of finite unions of vector spaces.

A morphism from \((I, (V_i)_{i \in I})\) to \((J, (W_i)_{i \in J})\) is the pair of:
- a map \(f\) from \(I\) to \(J\)
- a linear map \(g_i\) from \(V_i\) to \(W_{f(i)}\) for all \(i \in I\).

Remark: \(\text{Vec}\) is a subcategory of \(\text{Duvs}\).
Duvs-automata

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]

\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]

\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]
\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]
$$L_{\text{Vec}}(u) = \begin{cases} 2^{|u|_a} & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases}$$

Indices = \{\text{odd, even}\}

$$Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R})$$

$$i(x) = (\text{even}, x)$$

$$f(\text{even}, x) = x$$

$$f(\text{odd}, x) = 0$$

$$\delta_a(\text{even}, x) = (\text{even}, 2x)$$

$$\delta_a(\text{odd}, x) = (\text{odd}, 2x)$$

$$\delta_b(\text{even}, x) = (\text{odd}, x)$$

$$\delta_b(\text{odd}, x) = (\text{even}, x)$$

$$\delta_c(\text{even}, x) = (\text{even}, 0)$$

$$\delta_c(\text{odd}, x) = (\text{odd}, 0)$$
Duvs-automata

\[ L_{\text{Vec}}(u) = \begin{cases} 
2^{|u|_a} & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise} 
\end{cases} \]

Indices = \{odd, even\}

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

\[ V_{\text{odd}} = V_{\text{even}} = \mathbb{R} \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]

\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]

\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]
\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]
Duvs-automata

$L_{\text{Vec}}(u) = \begin{cases} 2^{|u|_a} & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases}$

Indices $= \{\text{odd, even}\}$

$Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R})$

$V_{\text{odd}} = V_{\text{even}} = \mathbb{R}$

$i(x) = (\text{even}, x)$

$f(\text{even}, x) = x$

$f(\text{odd}, x) = 0$

Is it minimal?

$\delta_a(\text{even}, x) = (\text{even}, 2x)$

$\delta_a(\text{odd}, x) = (\text{odd}, 2x)$

$\delta_b(\text{even}, x) = (\text{odd}, x)$

$\delta_b(\text{odd}, x) = (\text{even}, x)$

$\delta_c(\text{even}, x) = (\text{even}, 0)$

$\delta_c(\text{odd}, x) = (\text{odd}, 0)$
Duvs-automata

\[ L_{\text{vec}}(u) = \begin{cases} 2^{|u|_a} & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]
\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]
\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]
\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]

Indices = \{\text{odd, even}\}

\[ V_{\text{odd}} = V_{\text{even}} = \mathbb{R} \]

Is it minimal? No…
Duvs-automata

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]
\[ i(x) = (\text{even}, x) \]
\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]
\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]
\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]
\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]
\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]

Indices = \{\text{odd, even}\}
\[ V_{\text{odd}} = V_{\text{even}} = \mathbb{R} \]

Is it minimal? No...

(odd, 0) and (even, 0) are observationally equivalent
Duvs-automata

\[ L_{\text{Vec}}(u) = \begin{cases} 
2^{\lvert u \rvert_a} & \text{if } \lvert u \rvert_b \text{ is even, and } \lvert u \rvert_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

Indices = \{\text{odd, even}\}

\[ \begin{align*}
Q &= (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \\
i(x) &= (\text{even}, x) \\
V_{\text{odd}} &= V_{\text{even}} = \mathbb{R}
\end{align*} \]

Is it minimal? No...

\( (\text{odd}, 0) \) and \( (\text{even}, 0) \) are observationally equivalent

But the implementation is arbitrary.
Duvs-automata

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise} 
\end{cases} \]

\[ Q = \{(\text{odd}) \times \mathbb{R} \} \cup \{(\text{even}) \times \mathbb{R} \} \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]
\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]
\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]
\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]

Indices = \{\text{odd, even}\}

\[ V_{\text{odd}} = V_{\text{even}} = \mathbb{R} \]

Is it minimal? No… (\text{odd, 0}) and (\text{even, 0}) are observationally equivalent. But the implementation is arbitrary.

Can it be made minimal?
Duvs-automata

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]

\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]

\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]
\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]

Indices = \{odd, even\}

\[ V_{\text{odd}} = V_{\text{even}} = \mathbb{R} \]

Is it minimal? No…

(odd, 0) and (even, 0) are observationally equivalent

But the implementation is arbitrary.

Can it be made minimal? No…
\[
L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases}
\]

Indices = \{\text{odd, even}\}

\[Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R})\]

\[V_{\text{odd}} = V_{\text{even}} = \mathbb{R}\]

\[i(x) = (\text{even}, x)\]

\[f(\text{even}, x) = x\]
\[f(\text{odd}, x) = 0\]

\[\delta_a(\text{even}, x) = (\text{even}, 2x)\]
\[\delta_a(\text{odd}, x) = (\text{odd}, 2x)\]

\[\delta_b(\text{even}, x) = (\text{odd}, x)\]
\[\delta_b(\text{odd}, x) = (\text{even}, x)\]

\[\delta_c(\text{even}, x) = (\text{even}, 0)\]
\[\delta_c(\text{odd}, x) = (\text{odd}, 0)\]

Is it minimal? No…

(odd, 0) and (even, 0) are observationally equivalent

But the implementation is arbitrary.

Can it be made minimal? No…

Well, in fact Yes… but would be larger…
Duvs-automata

$$L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R})$$

$$i(x) = (\text{even}, x)$$

$$f(\text{even}, x) = x$$
$$f(\text{odd}, x) = 0$$

$$\delta_a(\text{even}, x) = (\text{even}, 2x)$$
$$\delta_a(\text{odd}, x) = (\text{odd}, 2x)$$

$$\delta_b(\text{even}, x) = (\text{odd}, x)$$
$$\delta_b(\text{odd}, x) = (\text{even}, x)$$

$$\delta_c(\text{even}, x) = (\text{even}, 0)$$
$$\delta_c(\text{odd}, x) = (\text{odd}, 0)$$

Indices = \{\text{odd}, \text{even}\}

$$V_{\text{odd}} = V_{\text{even}} = \mathbb{R}$$

Is it minimal? No… (odd, 0) and (even, 0) are observationally equivalent.

But the implementation is arbitrary.

Can it be made minimal? No…

Well, in fact Yes… but would be larger…

What can be done?
Minimizing automata via categories
Ingredients for the existence of a minimal automaton

Questions:
Given a (C,I,F)-automaton,
- what does it mean to be minimal?
- at what condition there exists a minimal automaton for a language?
- what do we need to effectively compute it?
Ingredients for the existence of a minimal automaton

Questions:
Given a \((C,I,F)\)-automaton,
- what does it mean to be minimal?
- at what condition there exists a minimal automaton for a language?
- what do we need to effectively compute it?

Minimal? « A DFA is \textbf{minimal} if it \textit{divides} any other automaton for the same language. »
Ingredients for the existence of a minimal automaton

Questions:
- Given a \((C,I,F)\)-automaton, what does it mean to be minimal?
- At what condition there exists a minimal automaton for a language?
- What do we need to effectively compute it?

Minimal?

\[
\text{« A DFA is minimal if it divides any other automaton for the same language. »}
\]

\[
\text{it is the quotient of a subautomaton.}
\]
Ingredients for the existence of a minimal automaton

Questions:
Given a \((C,I,F)\)-automaton,
- what does it mean to be minimal?
- at what condition there exists a minimal automaton for a language?
- what do we need to effectively compute it?

Minimal? « A DFA is **minimal** if it divides any other automaton for the same language. »

it is the **quotient** of a subautomaton.

notion of « surjection »  

notion of « injection »
Ingredients for the existence of a minimal automaton

Questions:
Given a \((C,I,F)\)-automaton,
- what does it mean to be minimal?
- at what condition there exists a minimal automaton for a language?
- what do we need to effectively compute it?

Minimal? « A DFA is minimal if it divides any other automaton for the same language. »

- It is the quotient of a subautomaton.
- It suffices to have 1. an initial automaton
  2. a final automaton
  3. a factorization system

notion of « surjection »

notion of « injection »
A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a factorization system if:
Factorization systems

A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a factorization system if:

- « epimorphisms »
- « surjections »
- « monomorphisms »
- « injections »
Factorization systems

A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a **factorization system** if:
- arrows in \(\mathcal{E}\) are closed under composition
- arrows in \(\mathcal{M}\) are closed under composition
A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a factorization system if:
- arrows in \(\mathcal{E}\) are closed under composition
- arrows in \(\mathcal{M}\) are closed under composition
- arrows that are both in \(\mathcal{E}\) and in \(\mathcal{M}\) are isomorphisms,
A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a **factorization system** if:

- arrows in \(\mathcal{E}\) are closed under composition
- arrows in \(\mathcal{M}\) are closed under composition
- arrows that are both in \(\mathcal{E}\) and in \(\mathcal{M}\) are **isomorphisms**,  
- all arrows \(f: X \to Y\) can be written

\[
f = m \circ e
\]

for some \(e: X \to Z\) in \(\mathcal{E}\) and \(m: Z \to Y\) in \(\mathcal{M}\).
Factorization systems

A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a factorization system if:
- arrows in \(\mathcal{E}\) are closed under composition
- arrows in \(\mathcal{M}\) are closed under composition
- arrows that are both in \(\mathcal{E}\) and in \(\mathcal{M}\) are isomorphisms,
- all arrows \(f: X \to Y\) can be written
  \[ f = m \circ e \]
  for some \(e: X \to Z\) in \(\mathcal{E}\) and \(m: Z \to Y\) in \(\mathcal{M}\).
Factorization systems

A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a factorization system if:
- arrows in \(\mathcal{E}\) are closed under composition
- arrows in \(\mathcal{M}\) are closed under composition
- arrows that are both in \(\mathcal{E}\) and in \(\mathcal{M}\) are isomorphisms,
- all arrows \(f: X \to Y\) can be written

\[ f = m \circ e \]

for some \(e: X \to Z\) in \(\mathcal{E}\) and \(m: Z \to Y\) in \(\mathcal{M}\).
- furthermore, this decomposition is unique up to isomorphism
  (it has in fact the stronger « diagonal property »).
Factorization systems

A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a **factorization system** if:
- arrows in \(\mathcal{E}\) are closed under composition
- arrows in \(\mathcal{M}\) are closed under composition
- arrows that are both in \(\mathcal{E}\) and in \(\mathcal{M}\) are **isomorphisms**, 
- all arrows \(f: X \rightarrow Y\) can be written
  \[f = m \circ e\]
  for some \(e: X \rightarrow Z\) in \(\mathcal{E}\) and \(m: Z \rightarrow Y\) in \(\mathcal{M}\).
- furthermore, this decomposition is unique up to **isomorphism**
  (it has in fact the stronger « **diagonal property** »).

In **Set**:

\[\begin{array}{c}
X \quad f \quad Y \\
\end{array}\]  
\[\begin{array}{c}
X \quad e \quad m \quad Img \ f \quad Y
\end{array}\]
A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a factorization system if:

- arrows in \(\mathcal{E}\) are closed under composition
- arrows in \(\mathcal{M}\) are closed under composition
- arrows that are both in \(\mathcal{E}\) and in \(\mathcal{M}\) are isomorphisms,
- all arrows \(f: X \rightarrow Y\) can be written
  \[ f = m \circ e \]
  for some \(e: X \rightarrow Z\) in \(\mathcal{E}\) and \(m: Z \rightarrow Y\) in \(\mathcal{M}\).
- furthermore, this decomposition is unique up to isomorphism (it has in fact the stronger « diagonal property »).

In **Set**:

\[
\begin{array}{ccc}
X & f & Y \\
| & & | \\
\downarrow & & \downarrow \\
\text{Img } f & & Y \\
\end{array}
\]

In **Vec**:

\[
\begin{array}{ccc}
X & e & m & Y \\
| & & | & \\
\downarrow & & \downarrow & \downarrow \\
\text{Im } f & & \text{Y} \\
\end{array}
\]
Factorization systems

A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a factorization system if:
- arrows in \(\mathcal{E}\) are closed under composition
- arrows in \(\mathcal{M}\) are closed under composition
- arrows that are both in \(\mathcal{E}\) and in \(\mathcal{M}\) are isomorphisms,
- all arrows \(f: X \rightarrow Y\) can be written
  \[ f = m \circ e \]
  for some \(e: X \rightarrow Z\) in \(\mathcal{E}\) and \(m: Z \rightarrow Y\) in \(\mathcal{M}\).
- furthermore, this decomposition is unique up to isomorphism (it has in fact the stronger « diagonal property »).

\[\text{In } \text{Set:}\]

\[\text{In } \text{Vec:}\]

\[\dim = \text{rank } f\]
Factorization system for automata
Lemma: If there is a factorization system \((\mathcal{E}, \mathcal{M})\) in a category \(\mathcal{C}\) then it can be lifted to the category of \(\mathcal{C}\)-automata for a language: these automata morphisms that belong to \(\mathcal{E}\) (resp. \(\mathcal{M}\)) as arrows in \(\mathcal{C}\).
Factorization system for automata

**Lemma:** If there is a factorization system $(\mathcal{E}, \mathcal{M})$ in a category $\mathcal{C}$ then it can be lifted to the category of $\mathcal{C}$-automata for a language: these automata morphisms that belong to $\mathcal{E}$ (resp. $\mathcal{M}$) as arrows in $\mathcal{C}$.

Hence *(Set,1,2)-automata* (i.e. DFA) have a factorization system (surjective morphisms,injective morphisms).
Factorization system for automata

**Lemma:** If there is a factorization system \((\mathcal{E}, \mathcal{M})\) in a category \(\mathcal{C}\) then it can be lifted to the category of \(\mathcal{C}\)-automata for a language: these automata morphisms that belong to \(\mathcal{E}\) (resp. \(\mathcal{M}\)) as arrows in \(\mathcal{C}\).

Hence \((\text{Set}, 1, 2)\)-automata (i.e. DFA) have a factorization system (surjective morphisms,injective morphisms).

Similarly \((\text{Vec}, K, K)\)-automata (i.e., automata weighted over a field) possess factorization system (surjective morphisms,injective morphisms).
Factorization system for automata

**Lemma:** If there is a factorization system \((\mathcal{E}, \mathcal{M})\) in a category \(\mathcal{C}\) then it can be lifted to the category of \(\mathcal{C}\)-automata for a language: these automata morphisms that belong to \(\mathcal{E}\) (resp. \(\mathcal{M}\)) as arrows in \(\mathcal{C}\).

Hence \((\text{Set},1,2)\)-automata (i.e. DFA) have a factorization system (surjective morphisms, injective morphisms).

Similarly \((\text{Vec},K,K)\)-automata (i.e., automata weighted over a field) possess factorization system (surjective morphisms, injective morphisms).

**Definition:**
- an \(\mathcal{M}\)-subobject \(X\) of \(Y\) is such that there is an \(\mathcal{M}\)-arrow \(m: X \rightarrow Y\),
- an \(\mathcal{E}\)-quotient \(X\) of \(Y\) is such that there is an \(\mathcal{E}\)-arrow \(e: Y \rightarrow X\),
- \(X (\mathcal{E}, \mathcal{M})\)-divides \(Y\) if it is a \(\mathcal{E}\)-quotient of an \(\mathcal{M}\)-subobject of \(Y\).
Minimization !
Lemma: In a category with initial object, final object, and a factorization system $(\mathcal{E}, \mathcal{M})$ then:

- there exists an object $\text{Min}$ that $(\mathcal{E}, \mathcal{M})$-divides all objects,
- furthermore $\text{Min} \approx \text{Obs}(\text{Reach}(X)) \approx \text{Reach}(\text{Obs}(X))$ for all $X$,

where

- $\text{Reach}(X)$ is the factorization of the only arrow from $I$ to $X$, and
- $\text{Obs}(X)$ is the factorization of the only arrow from $X$ to $F$. 
Lemma: In a category with initial object, final object, and a factorization system \((\mathcal{E}, \mathcal{M})\) then:
- there exists an object \(\text{Min}\) that \((\mathcal{E}, \mathcal{M})\)-divides all objects,
- furthermore \(\text{Min} \approx \text{Obs}(\text{Reach}(X)) \approx \text{Reach}(\text{Obs}(X))\) for all \(X\),

where
- \(\text{Reach}(X)\) is the factorization of the only arrow from \(I\) to \(X\), and
- \(\text{Obs}(X)\) is the factorization of the only arrow from \(X\) to \(F\).

Proof: \(\text{Min}\) is the factorization of the only arrow from \(I\) to \(F\). And…
Lemma: In a category with initial object, final object, and a factorization system \((\mathcal{E}, \mathcal{M})\) then:
- there exists an object \(\text{Min}\) that \((\mathcal{E}, \mathcal{M})\)-divides all objects,
- furthermore \(\text{Min} \simeq \text{Obs}(\text{Reach}(X)) \simeq \text{Reach}(\text{Obs}(X))\) for all \(X\),

where
- \(\text{Reach}(X)\) is the factorization of the only arrow from \(I\) to \(X\), and
- \(\text{Obs}(X)\) is the factorization of the only arrow from \(X\) to \(F\).

Proof: \(\text{Min}\) is the factorization of the only arrow from \(I\) to \(F\). And…
At this point...

We know that:
- **C-automata** and **C-languages** can be defined generally in a category $C$, yielding a category $\text{Auto}(L)$ of « C-automata for the language $L$ »

- for having a **minimal object** in a category, it is sufficient to have:
  1) an **initial** and a **final object** in the category for the language,
  2) a **factorization system** in $C$,
- that the existence of initial and final automata arise from simple assumptions on $C$,
- that the factorization system for automata is inherited from $C$,
- that standard minimization for **DFA** and **field weighted automata** are obtained this way.
At this point...

We know that:
- **C-automata** and **C-languages** can be defined generally in a category C, yielding a

  category Auto(L) of « C-automata for the language L »

- for having a **minimal object** in a category, it is sufficient to have:
  1) an initial and a final object in the category for the language,
  2) a factorization system in C,
- that the existence of initial and final automata arise from simple assumptions on C
- that the factorization system for automata is inherited from C,
- that standard minimization for **DFA** and **field weighted automata** are obtained this way.

But, what about minimizing **duvs-automata**?
Minimization of Duvs automata (wrong version)

Minimization of Duvs automata is possible (all the ingredient are there).
Minimization of Duvs automata is possible (all the ingredient are there).

However, for the definition of factorization system that works (epi,mono), the minimal automaton for

\[ L_{Vec}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

has state space

\[ Q = \mathbb{R}^2 \]

and not

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]
Glueings
Glueings

\[ L_{\text{vec}}(u) = \begin{cases} 
2|u|^a & \text{if } |u|^b \text{ is even, and } |u|^c = 0 \\
0 & \text{otherwise}
\end{cases} \]
Glueings

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\ 0 & \text{otherwise} \end{cases} \]

Vec-automaton

\( Q = \mathbb{R}^2 \)

\( i(x) = (x, 0) \)

\( f(x, y) = x \)

\( \delta_a(x, y) = (2x, 2y) \)

\( \delta_b(x, y) = (y, x) \)

\( \delta_c(x, y) = (0, 0) \)
Glueings

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

Vec-automaton

\[ Q = \mathbb{R}^2 \]
\[ i(x) = (x, 0) \]
\[ f(x, y) = x \]
\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]
Glueings

\[ L_{\text{Vec}}(u) = \begin{cases} 
2^{|u|_a} & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases} \]

**Vec-automaton**

- \( Q = \mathbb{R}^2 \)
- \( i(x) = (x, 0) \)
- \( f(x, y) = x \)
- \( \delta_a(x, y) = (2x, 2y) \)
- \( \delta_b(x, y) = (y, x) \)
- \( \delta_c(x, y) = (0, 0) \)

**Duvs-automaton**

- \( Q = \{\text{odd, even}\} \times \mathbb{R} \)
- \( i(x) = (\text{even}, x) \)
- \( f(\text{even}, x) = x \)
- \( f(\text{odd}, x) = 0 \)
- \( \delta_a(\text{even}, x) = (\text{even}, 2x) \)
- \( \delta_a(\text{odd}, x) = (\text{odd}, 2x) \)
- \( \delta_b(\text{even}, x) = (\text{odd}, x) \)
- \( \delta_b(\text{odd}, x) = (\text{even}, x) \)
- \( \delta_c(\text{even}, x) = (\text{even}, 0) \)
- \( \delta_c(\text{odd}, x) = (\text{odd}, 0) \)
Glueings

\[ L_{Vec}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise} 
\end{cases} \]

\textbf{Vec-automaton}

\[ Q = \mathbb{R}^2 \]

\[ i(x) = (x, 0) \]

\[ f(x, y) = x \]

\[ \delta_a(x, y) = (2x, 2y) \]

\[ \delta_b(x, y) = (y, x) \]

\[ \delta_c(x, y) = (0, 0) \]

\textbf{Duvs-automaton}

\[ Q = \{\text{odd, even}\} \times \mathbb{R} \]

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]

\[ f(\text{odd}, x) = 0 \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]

\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]

\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]

\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]

\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]

\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]
Glueings

\[ L_{\text{Vec}}(u) = \begin{cases} 2|u_a| & \text{if } |u_b| \text{ is even, and } |u_c| = 0 \\ 0 & \text{otherwise} \end{cases} \]

**Vec-automaton**

- \( Q = \mathbb{R}^2 \)
- \( i(x) = (x, 0) \)
- \( f(x, y) = x \)
- \( \delta_a(x, y) = (2x, 2y) \)
- \( \delta_b(x, y) = (y, x) \)
- \( \delta_c(x, y) = (0, 0) \)

**Duvs-automaton**

- \( Q = \{\text{odd, even}\} \times \mathbb{R} \)
- \( i(x) = (\text{even}, x) \)
- \( f(\text{even}, x) = x \)
- \( f(\text{odd}, x) = 0 \)

**Glue(Vec)-automaton**

- \( \delta_a(\text{even}, x) = (\text{even}, 2x) \)
- \( \delta_a(\text{odd}, x) = (\text{odd}, 2x) \)
- \( \delta_b(\text{even}, x) = (\text{odd}, x) \)
- \( \delta_b(\text{odd}, x) = (\text{even}, x) \)
- \( \delta_c(\text{even}, x) = (\text{even}, 0) \)
- \( \delta_c(\text{odd}, x) = (\text{odd}, 0) \)
Glueings

\[ L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise} 
\end{cases} \]

\text{Vec-automaton}

\[ Q = \mathbb{R}^2 \]
\[ i(x) = (x, 0) \]
\[ f(x, y) = x \]
\[ \delta_a(x, y) = (2x, 2y) \]
\[ \delta_b(x, y) = (y, x) \]
\[ \delta_c(x, y) = (0, 0) \]

\text{Duvs-automaton}

\[ Q = \{\text{odd, even}\} \times \mathbb{R} \]
\[ i(x) = (\text{even}, x) \]
\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \]
\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]
\[ \delta_b(\text{even}, x) = (\text{odd}, x) \]
\[ \delta_b(\text{odd}, x) = (\text{even}, x) \]
\[ \delta_c(\text{even}, x) = (\text{even}, 0) \]
\[ \delta_c(\text{odd}, x) = (\text{odd}, 0) \]

\text{Glue(Vec)-automaton}
Glueings

\[
L_{\text{Vec}}(u) = \begin{cases} 
2|u|_a & \text{if } |u|_b \text{ is even, and } |u|_c = 0 \\
0 & \text{otherwise}
\end{cases}
\]

**Vec-automaton**

\[
Q = \mathbb{R}^2 \\
i(x) = (x, 0) \\
f(x, y) = x \\
\delta_a(x, y) = (2x, 2y) \\
\delta_b(x, y) = (y, x) \\
\delta_c(x, y) = (0, 0)
\]

**Duvs-automaton**

\[
Q = \{\text{odd, even}\} \times \mathbb{R} \\
i(x) = (\text{even}, x) \\
f(\text{even}, x) = x \\
f(\text{odd}, x) = 0 \\
\delta_a(\text{even}, x) = (\text{even}, 2x) \\
\delta_a(\text{odd}, x) = (\text{odd}, 2x) \\
\delta_b(\text{even}, x) = (\text{odd}, x) \\
\delta_b(\text{odd}, x) = (\text{even}, x) \\
\delta_c(\text{even}, x) = (\text{even}, 0) \\
\delta_c(\text{odd}, x) = (\text{odd}, 0)
\]

**Glue(Vec)-automaton**
Defining Glue(Vec)
Defining Glue(Vec)

A glueing of vector space is
- a disjoint union of vector spaces
- together with an equivalence relation which:
  1) is trivial over each base space
  2) defines linear bijections between subspaces when restricted to pairs of base spaces.
A glueing of vector space is
- a disjoint union of vector spaces
- together with an equivalence relation which:
  1) is trivial over each base space
  2) defines linear bijections between subspaces when restricted to pairs of base spaces.

\[(p, x, 0) \sim_{\text{glue}} (q, 0, x)\]
\[(q, x, 0) \sim_{\text{glue}} (r, 0, x)\]
\[(r, x, 0) \sim_{\text{glue}} (p, 0, x)\]
Defining \textbf{Glue(Vec)}

A \textbf{glueing of vector space} is
- a disjoint union of vector spaces
- together with an equivalence relation which:
  1) is trivial over each base space
  2) defines linear bijections between subspaces when restricted to pairs of base spaces.

\[(p, x, 0) \sim_{\text{glue}} (q, 0, x)\]
\[(q, x, 0) \sim_{\text{glue}} (r, 0, x)\]
\[(r, x, 0) \sim_{\text{glue}} (p, 0, x)\]

\textbf{Morphisms} are...
complicated to describe...
Defining Glue(Vec)

A **glueing of vector space** is
- a disjoint union of vector spaces
- together with an equivalence relation which:
  1) is trivial over each base space
  2) defines linear bijections between subspaces when restricted to pairs of base spaces.

\[(p, x, 0) \sim_{\text{glue}} (q, 0, x)\]
\[(q, x, 0) \sim_{\text{glue}} (r, 0, x)\]
\[(r, x, 0) \sim_{\text{glue}} (p, 0, x)\]

**Morphisms** are... complicated to describe...

Aggregating objects from a category is a well known task in category theory: this is obtained by freely adding **colimits**.
Defining $\text{Glue}(\text{Vec})$

A **glueing of vector space** is
- a disjoint union of vector spaces
- together with an equivalence relation which:
  1) is trivial over each base space
  2) defines linear bijections between subspaces when restricted to pairs of base spaces.

$$(p, x, 0) \sim_{\text{glue}} (q, 0, x)$$
$$(q, x, 0) \sim_{\text{glue}} (r, 0, x)$$
$$(r, x, 0) \sim_{\text{glue}} (p, 0, x)$$

**Morphisms** are… complicated to describe…

Aggregating objects from a category is a well known task in category theory: this is obtained by freely adding **colimits**.

The category of **glueings of vector spaces** is the restriction of the co-completion of $\text{Vec}$ to some specific colimits: **mono-colimits**.
Defining \text{Glue}(\text{Vec})

A glueing of vector space is
- a disjoint union of vector spaces
- together with an equivalence relation which:
  1) is trivial over each base space
  2) defines linear bijections between subspaces when restricted to pairs of base spaces.

\[(p, x, 0) \sim_{\text{glue}} (q, 0, x)\]
\[(q, x, 0) \sim_{\text{glue}} (r, 0, x)\]
\[(r, x, 0) \sim_{\text{glue}} (p, 0, x)\]

Morphisms are... complicated to describe...

Aggregating objects from a category is a well known task in category theory: this is obtained by freely adding \text{colimits}.

The category of glueings of vector spaces is the restriction of the co-completion of Vec to some specific colimits: \text{mono-colimits}.

The advantage is that the concepts are well known, definition properly stated, and this can be applied to other categories than \text{Vec}.
Defining Glue(Vec) in categorical terms

Consider a category that already has colimits (for instance Vec)
Defining $\text{Glue(Vec)}$ in categorical terms

Consider a category that already has colimits (for instance $\text{Vec}$)

A **mono-co-limit diagram** is a diagram such that the universal cocone consists only of monos.
Defining \text{Glue}(\text{Vec}) in categorical terms

Consider a category that already has colimits (for instance Vec)

A \textit{mono-co-limit diagram} is a diagram such that the universal cocone consists only of monos.

For instance in Vec/Set:
Defining Glue(Vec) in categorical terms

Consider a category that already has colimits (for instance Vec)

A mono-co-limit diagram is a diagram such that the universal cocone consists only of monos.

For instance in Vec/Set:

Yes!
Defining Glue(Vec) in categorical terms

Consider a category that already has colimits (for instance Vec)

A mono-co-limit diagram is a diagram such that the universal cocone consists only of monos.

For instance in Vec/Set:

Yes!
Defining \text{Glue}(\text{Vec})

in categorical terms

Consider a category that already has colimits (for instance Vec)

A \textit{mono-co-limit diagram} is a diagram such that the universal cocone consists only of monos.

For instance in Vec/\text{Set}:

\begin{itemize}
\item coproducts are mono-colimits
\item Yes!
\item Yes!
\end{itemize}
Defining \textit{Glue(Vec)} in categorical terms

Consider a category that already has colimits (for instance Vec)

A \textbf{mono-co-limit diagram} is a diagram such that the universal cocone consists only of monos.

For instance in Vec/\textit{Set}:

Yes!  Yes!
Defining \textbf{Glue(Vec)} in categorical terms

Consider a category that already has colimits (for instance Vec)

A \textbf{mono-co-limit diagram} is a diagram such that the universal cocone consists only of monos.

For instance in Vec/Set:
Defining \text{Glue}(\text{Vec}) in categorical terms

Consider a category that already has colimits (for instance \text{Vec})

A \textbf{mono-co-limit diagram} is a diagram such that the universal cocone consists only of monos.

For instance in \text{Vec}/\text{Set}:

- Coproducts are mono-colimits:
  - Yes!

- No!
Defining \textit{Glue(Vec)} in categorical terms

Consider a category that already has colimits (for instance Vec)

A \textbf{mono-co-limit diagram} is a diagram such that the universal cocone consists only of monos.

For instance in Vec/Set:
Defining Glue(Vec) in categorical terms

Consider a category that already has colimits (for instance Vec)

A **mono-co-limit diagram** is a diagram such that the universal cocone consists only of monos.

For instance in Vec/Set:

**Definition:**

The **glueings** of a category is its free completion under mono-co-limits
Example: continued

The minimal automaton for our example is:
The **minimal automaton** for our example is:

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

with \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)
The **minimal automaton** for our example is:

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

with \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)

\[ i(x) = (\text{even}, x) \]
The minimal automaton for our example is:

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

with \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \]

\[ f(\text{odd}, x) = 0 \]
Example: continued

The **minimal automaton** for our example is:

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

with \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)

\[ i(x) = (\text{even}, x) \]

\[ f(\text{even}, x) = x \] \quad \{ \]
\[ f(\text{odd}, x) = 0 \quad \}

agrees on \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)
Example: continued

The **minimal automaton** for our example is:

\[ Q = (\{ \text{odd} \} \times \mathbb{R}) \cup (\{ \text{even} \} \times \mathbb{R}) \]

with \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)

\[
\begin{align*}
    i(x) &= (\text{even}, x) \\
    f(\text{even}, x) &= x \\
    f(\text{odd}, x) &= 0 \\
    \delta_a(\text{even}, x) &= (\text{even}, 2x) \\
    \delta_a(\text{odd}, x) &= (\text{odd}, 2x)
\end{align*}
\]
Example: continued

The *minimal automaton* for our example is:

\[ Q = \left( \{ \text{odd} \} \times \mathbb{R} \right) \cup \left( \{ \text{even} \} \times \mathbb{R} \right) \]

with \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)

\[ i(x) = (\text{even}, x) \]
\[ f(\text{even}, x) = x \]
\[ f(\text{odd}, x) = 0 \] \quad \text{agrees on} \ (\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0) \]

\[ \delta_a(\text{even}, x) = (\text{even}, 2x) \]
\[ \delta_a(\text{odd}, x) = (\text{odd}, 2x) \] \quad \text{agrees on} \ (\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0) \]
Example: continued

The **minimal automaton** for our example is:

\[ Q = (\{\text{odd}\} \times \mathbb{R}) \cup (\{\text{even}\} \times \mathbb{R}) \]

with \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)

\[
i(x) = (\text{even}, x) \\
\delta_a(\text{even}, x) = (\text{even}, 2x) \quad \delta_a(\text{odd}, x) = (\text{odd}, 2x) \]
\[
\delta_b(\text{even}, x) = (\text{odd}, x) \quad \delta_b(\text{odd}, x) = (\text{even}, x) \]
\[
\delta_c(\text{even}, x) = (\text{even}, 0) \quad \delta_c(\text{odd}, x) = (\text{odd}, 0) \]

agrees on \((\text{even}, 0) \sim_{\text{glue}} (\text{odd}, 0)\)
Properties of automata one glueings of vector spaces
Properties of automata one glueings of vector spaces

There exists an initial and a final automaton for a Glue(Vec)-language.
Properties of automata: one glueing of vector spaces.

There exists an initial and a final automaton for a Glue(Vec)-language. There is a natural factorization system « (surjection like, injection like) ». 
Properties of automata one glueings of vector spaces

There exists an initial and a final automaton for a Glue(Vec)-language. There is a natural factorization system « (surjection like, injection like) ». However, this yields wrong minimal automata:
Properties of automata one glueings of vector spaces

There exists an initial and a final automaton for a \text{Glue(Vec)}-language. There is a natural factorization system « (surjection like, injection like) ».

However, this yields wrong minimal automata:

\[ L(a^n)(x) = x \cos(n\alpha) \]
Properties of automata one glueings of vector spaces

There exists an initial and a final automaton for a Glue(Vec)-language.
There is a natural factorization system « (surjection like, injection like) ».

However, this yields wrong minimal automata:
\[ L(a^n)(x) = x \cos(n\alpha) \]
Properties of automata one glueings of vector spaces

There exists an initial and a final automaton for a Glue(Vec)-language. There is a natural factorization system « (surjection like, injection like) ».

However, this yields wrong minimal automata:

\[ L(a^n)(x) = x \cos(n\alpha) \]

For \( \alpha \) not a rational multiple of \( \pi \), the minimal automaton contains countable many copies of \( \mathbb{R} \), …one for each \( n \).
Properties of automata one glueings of vector spaces

There exists an initial and a final automaton for a Glue(Vec)-language. There is a natural factorization system « (surjection like, injection like) ».

However, this yields wrong minimal automata:

\[ L(a^n)(x) = x \cos(n\alpha) \]

For \( \alpha \) not a rational multiple of \( \pi \), the minimal automaton contains countable many copies of \( \mathbb{R} \), ...one for each \( n \)...

This is not what we wanted: we implicitly wanted to minimize among finite glueings of finite dimension vector spaces.
Properties of automata one glueings of vector spaces

There exists an initial and a final automaton for a Glue(Vec)-language. There is a natural factorization system « (surjection like, injection like) ».

However, this yields wrong minimal automata:

\[ L(a^n)(x) = x \cos(n\alpha) \]

For \( \alpha \) not a rational multiple of \( \pi \), the minimal automaton contains countable many copies of \( \mathbb{R} \), …one for each \( n \)…

This is not what we wanted: we implicitly wanted to minimize among finite glueings of finite dimension vector spaces.

**Theorem:** For Glue(Vec)-languages recognized by GlueFin(VecFin)-automata, there exists a minimal automaton for the language among GlueFin(VecFin)-automata.
The problem

\[ L(a^n)(x) = x \cos(n\alpha) \]

For \(\alpha\) not a rational multiple of \(\pi\), the minimal automaton contains countable many copies of \(\mathbb{R}\), …one for each \(n\)…

This is not what we wanted: we implicitly wanted to minimize among finite glueings of finite dimension vector spaces.
The problem

\[ L(a^n)(x) = x \cos(n\alpha) \]

For \( \alpha \) not a rational multiple of \( \pi \), the minimal automaton contains countable many copies of \( \mathbb{R} \), ...one for each \( n \)...

This is not what we wanted: we implicitly wanted to minimize among finite glueings of finite dimension vector spaces.

« implementation »
The problem

For $\alpha$ not a rational multiple of $\pi$, the minimal automaton contains countable many copies of $\mathbb{R}$, ...one for each $n$...

This is not what we wanted: we implicitly wanted to minimize among finite glueings of finite dimension vector spaces.

$L(a^n)(x) = x \cos(n\alpha)$

« implementation »

Glue(Vec) -automata for L
The problem

For $\alpha$ not a rational multiple of $\pi$, the minimal automaton contains countable many copies of $\mathbb{R}$, …one for each $n$…

This is not what we wanted: we implicitly wanted to minimize among finite glueings of finite dimension vector spaces.

$L(a^n)(x) = x \cos(n\alpha)$

« implementation »
The problem

$L(a^n)(x) = x \cos(n\alpha)$

For $\alpha$ not a rational multiple of $\pi$, the minimal automaton contains countable many copies of $\mathbb{R}$, ...one for each $n$...

This is not what we wanted: we implicitly wanted to minimize among finite glueings of finite dimension vector spaces.

« implementation »

Init

Glue(Vec) -automata for L

Fact(f)

Final

GlueFin(VecFin) -automata for L
The problem

For $\alpha$ not a rational multiple of $\pi$, the minimal automaton contains countable many copies of $\mathbb{R}$, …one for each $n$…

This is not what we wanted: we implicitly wanted to minimize among finite glueings of finite dimension vector spaces.

$L(a^n)(x) = x \cos(n\alpha)$
Idea 1: factorization through
Idea 1: factorization through $C$
Idea 1: factorization through
Idea 1: factorization through

\[ X \overset{\text{Fact}(f)}{\longrightarrow} Y \]

C
Idea 1: factorization through $X$

A factorization system through a subcategory $S$ consists of classes $(E, M)$ such that:
- $E$-arrows end in $S$ and are closer under composition
- $M$-arrows start in $S$ and are closer under composition
- all arrows that factorize through $S$ has $(E, M)$ factorization.
- the diagonal property holds.
Idea 1: factorization through $X$

A factorization system through a subcategory $S$ consists of classes $(\mathcal{E}, \mathcal{M})$ such that:

- $\mathcal{E}$-arrows end in $S$ and are closer under composition
- $\mathcal{M}$-arrows start in $S$ and are closer under composition
- all arrows that factorize through $S$ have $(\mathcal{E}, \mathcal{M})$ factorization.
- the diagonal property holds.
Idea 1: factorization through

A factorization system through a subcategory $S$ consists of classes $(E,M)$ such that:

- $E$-arrows end in $S$ and are closer under composition
- $M$-arrows start in $S$ and are closer under composition
- all arrows that factorize through $S$ has $(E,M)$ factorization.
- the diagonal property holds.
Idea 1: factorization through

A factorization system through a subcategory $S$ consists of classes $(\mathcal{E}, \mathcal{M})$ such that:
- $\mathcal{E}$-arrows end in $S$ and are closer under composition
- $\mathcal{M}$-arrows start in $S$ and are closer under composition
- all arrows that factorize through $S$ has $(\mathcal{E}, \mathcal{M})$ factorization.
- the diagonal property holds.

**Theorem:**
Glue(Vec) has a factorization system through GlueFin(VecFin)
Idea 1: factorization through

A factorization system through a subcategory $S$ consists of classes $(E, M)$ such that:
- $E$-arrows end in $S$ and are closer under composition
- $M$-arrows start in $S$ and are closer under composition
- all arrows that factorize through $S$ have $(E, M)$ factorization.
- the diagnoal property holds.

**Theorem:**

Glue(Vec) has a factorization system through GlueFin(VecFin)

(the same goes for automata)
Factorization of glueings
Theorem:
\( \text{Glue(Vec)} \) has a factorization system through \( \text{GlueFin(VecFin)} \)
Factorization of glueings

Theorem: \( \text{Glue(Vec)} \) has a factorization system through \( \text{GlueFin(VecFin)} \)

Lemma: finite unions of subspaces of a finite dimension vector space are closed under arbitrary intersection.
Factorization of glueings

Theorem: \( \text{Glue}(\text{Vec}) \) has a factorization system through \( \text{GlueFin}(\text{VecFin}) \)

Lemma: finite unions of subspaces of a finite dimension vector space are closed under arbitrary intersection.

Corollary: for all subset of a (finite dimension) vector space, there is a least finite union of vector spaces that covers it.
Factorization of glueings

**Theorem:** $	ext{Glue(Vec)}$ has a factorization system through $	ext{GlueFin(VecFin)}$

**Lemma:** finite unions of subspaces of a finite dimension vector space are closed under arbitrary intersection.

**Corollary:** for all subset of a (finite dimension) vector space, there is a least finite union of vector spaces that covers it.

This is the closure in the topology where closed sets are finite unions of subspaces.
Factorization of glueings

**Theorem:** \( \text{Glue(Vec)} \) has a factorization system through \( \text{GlueFin(VecFin)} \)

**Lemma:** finite unions of subspaces of a finite dimension vector space are closed under arbitrary intersection.

**Corollary:** for all subset of a (finite dimension) vector space, there is a least finite union of vector spaces that covers it. This is the closure in the topology where closed sets are finite unions of subspaces. This is a coarsening of Zariski topology.
Factorization of glueings

**Theorem:** $\text{Glue(Vec)}$ has a factorization system through $\text{GlueFin(VecFin)}$

**Lemma:** finite unions of subspaces of a finite dimension vector space are closed under arbitrary intersection.

**Corollary:** for all subset of a (finite dimension) vector space, there is a least finite union of vector spaces that covers it. This is the closure in the topology where closed sets are finite unions of subspaces. This is a coarsening of Zariski topology.

Proof by example in affine spaces
Factorization of glueings

**Theorem:** \( \text{Glue(Vec)} \) has a factorization system through \( \text{GlueFin(VecFin)} \)

**Lemma:** finite unions of subspaces of a finite dimension vector space are closed under arbitrary intersection.

**Corollary:** for all subset of a (finite dimension) vector space, there is a least finite union of vector spaces that covers it. This is the closure in the topology where closed sets are finite unions of subspaces. This is a coarsening of Zariski topology.

Proof by example in affine spaces
We obtain

**Theorem:** For Glue(Vec)-languages recognized by GlueFin(VecFin)-automata, there exists a minimal automaton for the language among GlueFin(VecFin)-automata.
We obtain

**Theorem:** For *Glue(Vec)*-languages recognized by *GlueFin(VecFin)*-automata, there exists a minimal automaton for the language among *GlueFin(VecFin)*-automata.

But, we do not know how to compute it…
The core algorithmic problem

Open problem: Given $n \times n$ matrices $A_1, \ldots, A_k$, compute the least finite union of subspaces of matrices that covers the generated semigroup.

For instance consider the matrix $\text{Rot}(\alpha)$ for some rational number $\alpha$.

If $\alpha$ is a rational multiple of $\pi$, it should output the (finite union) of the dimension one spaces $\text{Vec}(\text{Rot}(n\alpha))$ for integer $n$.

Otherwise, the output is the two dimension vector spaces of matrices of the form

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix}$$
The core algorithmic problem

Open problem: Given $n \times n$ matrices $A_1, \ldots, A_k$, compute the least finite union of subspaces of matrices that covers the generated semigroup.

For instance consider the matrice $\text{Rot}(\alpha)$ for some rational number $\alpha$.

If $\alpha$ is a rational multiple of $\pi$, it should output the (finite union) of the dimension one spaces $\text{Vec}(\text{Rot}(n\alpha))$ for integer $n$.

Otherwise, the output is the two dimension vector spaces of matrices of the form

$$
\begin{pmatrix}
    a & b \\
    -b & a 
\end{pmatrix}
$$

Open problem: Given $n \times n$ matrices $A_1, \ldots, A_k$, compute the Zariski closure of the semigroup generated by these matrices.
Conclusion
Contributions
Contributions

- A categorical description of why minimization is possible,
- new categorical concepts on the way,
- new ways to construct categories that yield \textit{natural classes} of
  minimizable automata using « glueings ».
Contributions

- A categorical description of why minimization is possible,
- new categorical concepts on the way,
- new ways to construct categories that yield natural classes of
  minimizable automata using « glueings ».

Related works
Contributions

- A categorical description of why minimization is possible,
- new categorical concepts on the way,
- new ways to construct categories that yield natural classes of minimizable automata using « glueings ».

Related works

- Schützenberger’s weighted automata, and its long continuations [Sakarovitch, Lombardy, Droste, Gastin, Vogler, …]
- There is a long history of categorical view of minimization [Arbib, Manes, Adamek, Milius, Silva, Panangaden, Kupke…]
Contributions

- A categorical description of why minimization is possible,
- new categorical concepts on the way,
- new ways to construct categories that yield natural classes of minimizable automata using « glueings ».

Related works

- Schützenberger’s weighted automata, and its long continuations
  [Sakarovitch, Lombardy, Droste, Gastin, Vogler, …]
- There is a long history of categorical view of minimization
  [Arbib, Manes, Adamek, Milius, Silva, Panangaden, Kupke…]

And then ?
Contributions

- A categorical description of why minimization is possible,
- new categorical concepts on the way,
- new ways to construct categories that yield **natural classes** of minimizable automata using « glueings ».

Related works

- Schützenberger’s weighted automata, and its long continuations [Sakarovitch, Lombardy, Droste, Gastin, Vogler, …]
- There is a long history of categorical view of minimization [Arbib, Manes, Adamek, Milius, Silva, Panangaden, Kupke…]

And then ?

- Make this construction effective… (generalization of sequencialization)
- tree automata
- algebras (monoids,…)
- infinite objects (ω-semigroup, o-semigroup, monads…).
Questions ?