Automata minimization and glueing of categories

Computability in Europe 2017
June 15

Thomas Colcombet
joint work with Daniela Petrișan
Automata minimization and glueing of categories

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

Computability in Europe 2017
June 15

Thomas Colcombet
joint work with Daniela Petrişan
Description of the situation
Automata
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**
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**
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**:

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

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

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

\[ u \mapsto f \circ \delta_u \circ i \]
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
\[ \langle Q, i, f, (\tilde{\delta}_a)_{a \in A} \rangle \]
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
- \( \tilde{\delta}_a : Q \to Q \) is a linear map
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
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**:

\[ [A]: A^* \to [1, 2] \approx 2 \quad u \mapsto f \circ \delta_u \circ i \]

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**:

\[ [A]: A^* \to [\mathbb{R}, \mathbb{R}] \quad u \mapsto f \circ \delta_u \circ i \]

Schützenberger’s **automata weighted over a field**

Rabin & Scott
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**:

\[
\left[ \mathcal{A} \right] : A^* \to [1, 2] \approx 2
\]

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

A **vector automaton** is

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

where

- \( Q \) is a set of \( \mathbb{R} \)-**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**:

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

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

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

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

\( Q \) 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

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

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

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

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

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

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

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

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

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

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

\( Q \) 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

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

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

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

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

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

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

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

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

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

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

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
Categories

A category has **objects** and **arrows**.
A category has objects and arrows $X, Y, Z \ldots$
A category has **objects** and **arrows**

\[ X, Y, Z \ldots \]

\[ f : X \to Y \]
A **category** has **objects** and **arrows**

\[ X, Y, Z \ldots \quad f : X \to Y \]

source \quad target
A category has objects and arrows

\( X, Y, Z \ldots \)  
\( f : X \to Y \)

- There is an identity arrow for all object:
  \( \text{Id}_X : X \to X \)
A category has **objects** and **arrows**

\[ X, Y, Z \ldots \quad f : X \to Y \]

- There is an identity arrow for all object:
  \[ \text{Id}_X : X \to X \]

- Arrows compose: for \( f : X \to Y \) and \( g : Y \to Z \) there is an arrow:
  \[ g \circ f : X \to Z \]
A category has **objects** and **arrows**

\[ X, Y, Z \ldots \quad f : X \to Y \]

- There is an identity arrow for all object:
  \[ \text{Id}_X : X \to X \]

- Arrows compose: for \( f : X \to Y \) and \( g : Y \to Z \) there is an arrow:
  \[ g \circ f : X \to Z \]

+ some associatively axioms.
A category has objects and arrows

\[ X, Y, Z \ldots \quad f : X \rightarrow Y \]

- There is an identity arrow for all object:
  \[ \text{Id}_X : X \rightarrow X \]
- Arrows compose: for \( f : X \rightarrow Y \) and \( g : Y \rightarrow Z \) there is an arrow:
  \[ g \circ f : X \rightarrow Z \]

+ some associatively axioms.

Set = (sets, maps)
Vec = (vector spaces, linear maps)
Aff = (affine spaces, affine maps)
Rel = (sets, binary relations)
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:

\[ [\mathcal{A}]: A^* \to [I, F] \]
\[ u \mapsto f \circ \delta_u \circ i \]
Automata in a category

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

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

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

A morphism is an arrow
\[
h: Q_A \to Q_B
\]
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:
\[
[A]: A^* \rightarrow [I, F]
\]
\[
u \mapsto f \circ \delta_u \circ i
\]

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

- \((\text{Set}, 1, 2)\)-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)
- \(\ldots\)

A morphism is an arrow
\[
h: Q_A \rightarrow Q_B
\]
such that tfdc:
\[
\begin{align*}
I & \xrightarrow{i_A} Q_A & Q_A & \xrightarrow{\delta_B(a)} Q_B & Q_B & \xrightarrow{f_B} F \\
\downarrow h & \quad \downarrow h & \quad \downarrow h & \quad \downarrow h & \quad \downarrow h & \\
Q_B & \xrightarrow{i_B} Q_B & Q_B & \xrightarrow{\delta_B(a)} Q_B & Q_B & \xrightarrow{f_B} F
\end{align*}
\]
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 (free co-product completion of $\text{Vec}$)

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 $i \in I$. 
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.
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 \(\text{Duvs}\) 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:
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 \textbf{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 \(Duvs\) 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 \(i \in I\).
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 \(\text{Duvs}\) 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Indices = \{odd, even\}

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

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

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

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

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

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

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

Is it minimal? No…
Duvs-automata

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

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
Duvs-automata

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

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

Indices = \{odd, even\}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Can it be made minimal?
Duvs-automata

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

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

Indices = \{odd, even\}

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

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

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

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

Is it minimal? No…

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

But the implementation is arbitrary.

Can it be made minimal? No…
Duvs-automata

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Is it minimal? No…

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

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?

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

- it is the \textbf{quotient} of a \textbf{subautomaton}.
- notion of « \textbf{surjection} »
- notion of « \textbf{injection} »
Initial and final automata

In a category, an object is
- **initial** if there is one and exactly one arrow from it to every other object
- **final** if there is one and exactly one arrow to it from every other object
In a category, an object is
- **initial** if there is one and exactly one arrow from it to every other object
- **final** if there is one and exactly one arrow to it from every other object

For **Set** and **Vec-automata**, there is an initial and a final automaton for each language.
In a category, an object is
- **initial** if there is one and exactly one arrow from it to every other object
- **final** if there is one and exactly one arrow to it from every other object

For **Set** and **Vec-automata**, there is an initial and a final automaton for each language.

**Initial (Set, 1, 2)-automaton** for L:
- states = $A^*$
- init(.) = $\varepsilon$
- final(u) = L(u)
- $\delta_a(u) = ua$
Initial and final automata

In a category, an object is
- **initial** if there is one and exactly one arrow from it to every other object
- **final** if there is one and exactly one arrow to it from every other object

For **Set** and **Vec-automata**, there is an initial and a final automaton for each language.

**Initial (Set,1,2)-automaton** for L:
- states = $A^*$
- $\text{init}(.) = \varepsilon$
- $\text{final}(u) = L(u)$
- $\delta_a(u) = ua$

**Final (Set,1,2)-automaton** for L:
- states = languages
- $\text{init}(.) = L$
- $\text{final}(R) = R(\varepsilon)$
- $\delta_a(R) = \{u : au\varepsilon \in R\}$
Initial and final automata

In a category, an object is
- **initial** if there is one and exactly one arrow from it to every other object
- **final** if there is one and exactly one arrow to it from every other object

For **Set** and **Vec-automata**, there is an initial and a final automaton for each language.

**Initial (Set,1,2)-automaton** for L:
- states = $A^*$
- init(.) = $\varepsilon$
- final(u) = $L(u)$
- $\delta_a(u) = ua$

**Final (Set,1,2)-automaton** for L:
- states = languages
- init(.) = $L$
- final(R) = $R(\varepsilon)$
- $\delta_a(R) = \{u : au \in R\}$

**Remark:** Initial and final automata exist as soon as the category has countable copowers and powers (works e.g. for **Set**, **Vec**, **Aff**, ...).
Factorization systems

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

- « epimorphisms »
- « surjections »
- « monomorphisms »
- « injections »
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 \(f: X \rightarrow Y\) can be written

\[ f = m \circ e \]

for some \(e: X \rightarrow Z\) in \(\mathcal{E}\) and \(m: Z \rightarrow Y\) in \(\mathcal{M}\).
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 \(\mathcal{M}\).
A pair of families of arrows \((\mathcal{E}, \mathcal{M})\) is a \textbf{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 \textbf{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 \textbf{isomorphism} (it has in fact the stronger « \textbf{diagonal property} »).
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 »).

In \(\text{Set}\):
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**:

\[ f: X \rightarrow Y \]

\[ e: X \rightarrow \text{Img} f \]

\[ m: \text{Img} f \rightarrow Y \]

In **Vec**:
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 \(\text{Set}\):**

- \(f\): \(X \rightarrow Y\)
- \(e\): \(X \rightarrow \text{Img } f\)
- \(m\): \(\text{Img } f \rightarrow Y\)

**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 \((\text{Set}, 1, 2)\)-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 \(m: X \to Y\),
- an \(\mathcal{E}\)-quotient \(X\) of \(Y\) is such that there is an \(\mathcal{E}\)-arrow \(e: Y \to X\),
- \(X\) \((\mathcal{E}, \mathcal{M})\)-divides \(Y\) if it is a \(\mathcal{E}\)-quotient of an \(\mathcal{M}\)-subobject of \(Y\).
Minimization!
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 $\text{Min} \cong \text{Obs(Reach}(X)) \cong \text{Reach(Obs}(X))$ 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 $F$. 
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 \(F\).

**Proof:** \(\text{Min}\) is the factorization of the only arrow from \(I\) to \(F\). And…
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 \(\text{Min} \simeq \text{Obs}(\text{Reach}(X)) \simeq \text{Reach}(\text{Obs}(X))\) 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 $tC$,
- 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 $\text{Auto}(L)$ 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 tC,
- 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**?
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

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

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

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

**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**

\[ Q = \mathbb{R} \]

\[ i(x) = x \]

\[ f(x) = x \]

\[ \delta_a(x) = x \]

\[ \delta_b(x) = x \]

\[ \delta_c(x) = x \]
Glueings

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

<table>
<thead>
<tr>
<th>Vec-automaton</th>
<th>Duvs-automaton</th>
<th>Glue(Vec)-automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q = \mathbb{R}^2 )</td>
<td>( Q = {\text{odd, even}} \times \mathbb{R} )</td>
<td></td>
</tr>
<tr>
<td>( i(x) = (x, 0) )</td>
<td>( i(x) = (\text{even}, x) )</td>
<td></td>
</tr>
<tr>
<td>( f(x, y) = x )</td>
<td>( f(\text{even}, x) = x )</td>
<td>( f(\text{odd}, x) = 0 )</td>
</tr>
<tr>
<td>( \delta_a(x, y) = (2x, 2y) )</td>
<td>( \delta_a(\text{even}, x) = (\text{even}, 2x) )</td>
<td></td>
</tr>
<tr>
<td>( \delta_b(x, y) = (y, x) )</td>
<td>( \delta_a(\text{odd}, x) = (\text{odd}, 2x) )</td>
<td></td>
</tr>
<tr>
<td>( \delta_c(x, y) = (0, 0) )</td>
<td>( \delta_b(\text{even}, x) = (\text{odd}, x) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \delta_b(\text{odd}, x) = (\text{even}, x) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \delta_c(\text{even}, x) = (\text{even}, 0) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \delta_c(\text{odd}, x) = (\text{odd}, 0) )</td>
<td></td>
</tr>
</tbody>
</table>

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

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

**Glue(Vec)-automaton**

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

A \textit{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 $\text{Glue}(\text{Vec})$

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.

Morphisms are... complicated to describe...

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

A \textit{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...

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[
f(\text{odd}, x) = 0
\]
The **minimal automaton** for our example is:

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

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

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

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

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

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

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

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

The minimal automaton for our example is:

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

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

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

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

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

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

\[
\begin{align*}
  \delta_a(\text{even}, x) &= (\text{even}, 2x) \\
  \delta_a(\text{odd}, x) &= (\text{odd}, 2x)
\end{align*}
\]

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

\[
\begin{align*}
  \delta_b(\text{even}, x) &= (\text{odd}, x) \\
  \delta_b(\text{odd}, x) &= (\text{even}, x)
\end{align*}
\]

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

\[
\begin{align*}
  \delta_c(\text{even}, x) &= (\text{even}, 0) \\
  \delta_c(\text{odd}, x) &= (\text{odd}, 0)
\end{align*}
\]

 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 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) ». 
Properties of automata one glueings of vector spaces

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

However, this yields \textit{wrong minimal automata}:
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:

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

« implementation »
Properties of automata one glueings of vector spaces

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

However, this yields \textbf{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 on 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:

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

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.
Idea 1: factorization through a subcategory

We introduce the notion of « factorization through a subcategory ».
Idea 1: factorization through

We introduce the notion of « factorization through a subcategory ». 
Idea 1: factorization through

We introduce the notion of «factorization through a subcategory». 
Idea 1: factorization through

We introduce the notion of « factorization through a subcategory ».

A factorization is a way to break an arrow in an optimally chosen middle point.
Idea 1: factorization through

We introduce the notion of « factorization through a subcategory ».

A factorization is a way to break an arrow in an optimally chosen middle point.
Idea 1: factorization through

We introduce the notion of « factorization through a subcategory ».

A factorization is a way to break an arrow in an optimally chosen middle point.

A factorization through is a way to break an arrow in an optimally chosen middle point in $S$. 
Idea 1: factorization through

We introduce the notion of « factorization through a subcategory ».

A factorization is a way to break an arrow in an optimally chosen middle point.

A factorization through is a way to break an arrow in an optimally chosen middle point in $S$. 
Idea 1: factorization through

We introduce the notion of « factorization through a subcategory ».

A factorization is a way to break an arrow in an optimally chosen middle point.

A factorization through is a way to break an arrow in an optimally chosen middle point in S.
Idea 1: factorization through

We introduce the notion of « factorization through a subcategory ».

A factorization is a way to break an arrow in an optimally chosen middle point.

A factorization through is a way to break an arrow in an optimally chosen middle point in $S$.

Minimization can now be phrased as « finding the optimal automata for the language among finite glueings of finite dimension vector spaces ».
Idea 1: factorization through

We introduce the notion of « factorization through a subcategory ».

A factorization is a way to break an arrow in an optimally chosen middle point.

A factorization through is a way to break an arrow in an optimally chosen middle point in $S$.

Minimization can now be phrased as « finding the optimal automata for the language among finite glueings of finite dimension vector spaces » (note that this distinction is not necessary for $\text{Set}$ or $\text{Vec}$)
Idea 2: unions of subspaces

**Theorem:** For Glue(Vec)-languages recognized by GlueFin(VecFin)-automata, there exists a minimal automaton for the language among GlueFin(VecFin)-automata.
Idea 2: unions of subspaces

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

**Essential idea** in the machinery: For all set $X \subseteq V$ vector space, there exists a least (for inclusion finite union of subspaces that contains $V$.)
Idea 2: unions of subspaces

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

**Essential idea** in the machinery: For all set $X \subseteq V$ vector space, there exists a least (for inclusion finite union of subspaces that contains $V$.)
Idea 2: unions of subspaces

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

**Essential idea** in the machinery: For all set $X \subseteq V$ vector space, there exists a least (for inclusion finite union of subspaces that contains $V$.)
Idea 2: unions of subspaces

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

Essential idea in the machinery: For all set $X \subseteq V$ vector space, there exists a least (for inclusion finite union of subspaces that contains $V$.)

Configurations taken by the first Vec-automaton
Idea 2: unions of subspaces

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

Essential idea in the machinery: For all set $X \subseteq V$ vector space, there exists a least (for inclusion finite union of subspaces that contains $V$.)

Configurations taken by the first $\text{Vec}$-automaton

Subspace that can be described as the glueing in 0 of two copies of $\mathbb{R}$. 
Conclusion
Contributions
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 ».
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 (omega-semigroup, o-semigroup, monads…).