# Automata minimization

## a lightweight categorical approach

Thomas Colcombet and **Daniela Petrişan**
CNRS & IRIF, Paris 7

OPCT 2017, Vienna, 29 July 2017

INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE

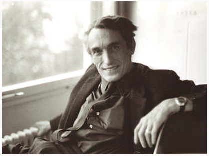
# Overview

- Motivation: hybrid set-vector automata
- Byproduct: a lightweight category-theoretic approach
- Automata are functors! Minimization in this setting.
- Examples
- Open problems!

# Motivation

# Automata for weighted languages

Once upon a time weighted automata were introduced by



[M.-P. Schützenberger, 1961]
*On the definition of a family of automata*

A minimization algorithm is also provided.

# Vector automata

An vector automaton is a tuple

$$\mathcal{A} = \langle Q, q_0, f, (\delta_a)_{a \in A} \rangle$$

- $Q$ is an $\mathbb{R}$-vector space
- $q_0$ is an initial vector in $Q$
- $f\colon Q \to \mathbb{R}$ associates to each state an output value
- for each $a \in A$, $\delta_a\colon Q \to Q$ is a linear map

The language accepted by $\mathcal{A}$ is a map $L_{\mathcal{A}}\colon A^* \to \mathbb{R}$ defined by

$$w \in A^* \mapsto f(\delta_w(q_0))$$

# Weighted languages: an example

Consider the alphabet $A = \{a, b, c\}$ and the language $L\colon A^* \to \mathbb{R}$

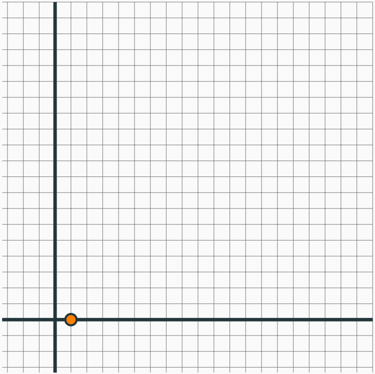
$$L(u) = \begin{cases} 2^{|u|_a} & \text{if } |u|_b \text{ is even and } |u|_c = 0, \\ 0 & \text{otherwise} \end{cases}$$

An automaton accepting this language is

$$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a \in A} \rangle$$

# Weighted languages: an example

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



$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a \in A} \rangle$

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

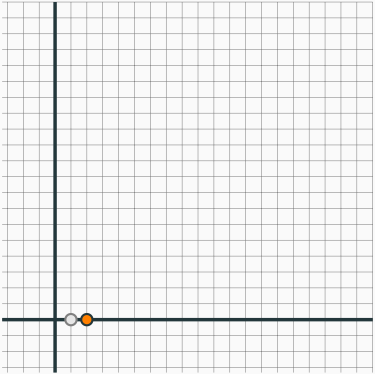
$\delta_b(x,y) = (y, x)$

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

$f(x,y) = x$

# Weighted languages: an example

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



$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a \in A} \rangle$

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

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

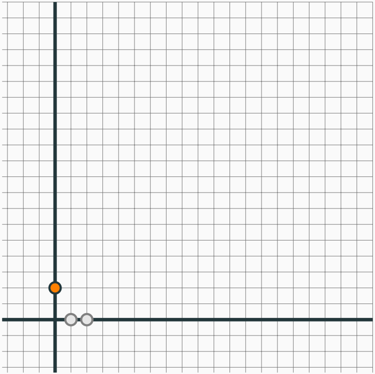
$\delta_c(x,y) = (0, 0)$

$f(x,y) = x$

*a*

# Weighted languages: an example

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



$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a \in A} \rangle$

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

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

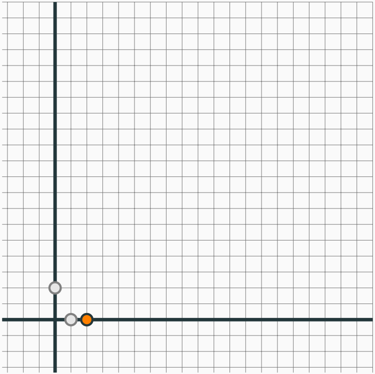
$\delta_c(x,y) = (0,0)$

$f(x,y) = x$

*ab*

# Weighted languages: an example

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



$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a\in A} \rangle$

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

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

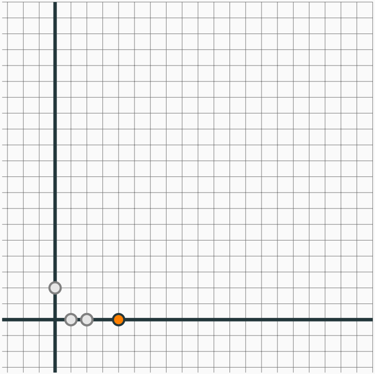
$\delta_c(x,y) = (0, 0)$

$f(x,y) = x$

*abb*

# Weighted languages: an example

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



$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a \in A} \rangle$

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

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

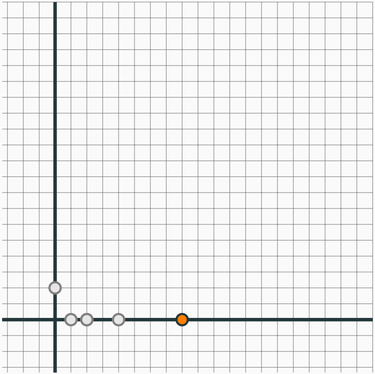
$\delta_c(x,y) = (0, 0)$

$f(x,y) = x$

*abba*

# Weighted languages: an example

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



$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a \in A} \rangle$

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

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

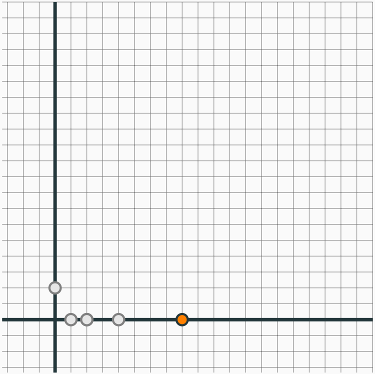
$\delta_c(x,y) = (0, 0)$

$f(x,y) = x$

*abbaa*

# Weighted languages: an example

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



$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a \in A} \rangle$

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

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

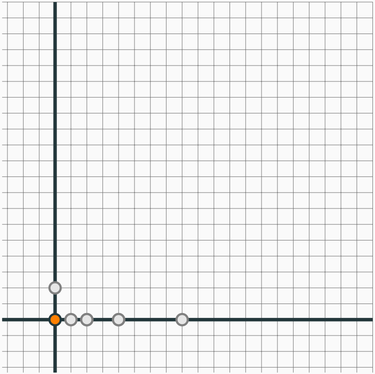
$\delta_c(x,y) = (0, 0)$

$f(x,y) = x$

$abbaa \mapsto 8$

# Weighted languages: an example

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



$\langle \mathbb{R}^2, (1,0), f, (\delta_a)_{a \in A} \rangle$

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

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

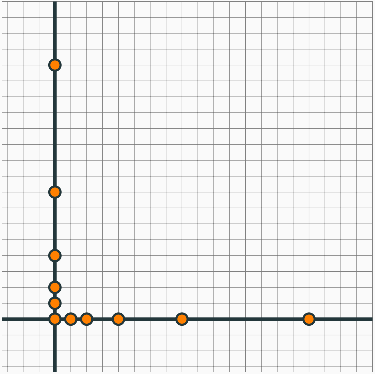
$\delta_c(x,y) = (0,0)$

$f(x,y) = x$

$abbaac \mapsto 0$

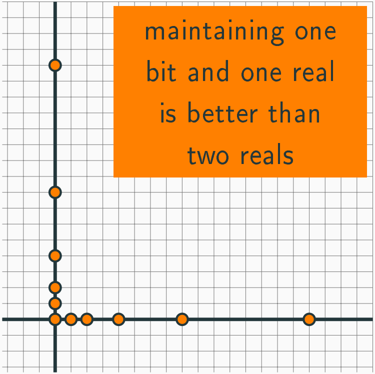
# Weighted languages: an example

The “reachable” vectors are on the “union” of two one-dimensional subspaces.



# Weighted languages: an example

The “reachable” vectors are on the “union” of two one-dimensional subspaces.



maintaining one bit and one real is better than two reals

# The starting point

Hybrid set-vector automata “have”

- a finite set of control states that evolve like DFAs
- a finite vector space for each control state

**Question.**

What is a suitable automata model so that minimisation is possible and we retrieve this “hybrid” behaviour?

# Automata as functors

# Automata in Categories: what we already knew

Automata are both

algebras for a functor + final map

and

coalgebras for a functor + initial map

# Automata in Categories: what we already knew

Automata are both
algebras for a functor + final map
and
coalgebras for a functor + initial map

Minimization can be explained via
the duality between the algebraic-coalgebraic view
(e.g. Brzozowski's algorithm)

# Automata in Categories: what we already knew

Automata are both
algebras for a functor + final map
and
coalgebras for a functor + initial map

Minimization can be explained via
the duality between the algebraic-coalgebraic view
(e.g. Brzozowski's algorithm)

The coalgebraic view brings its own advantages:
(e.g. checking NFA equivalences using
up-to techniques for bisimulations)

Thomas Colcombet
“Algèbres? Co-algèbres?
Mais ils ne sont ni l’un ni l’autre !”

An automaton processes an input, respecting its structure (word, tree, infinite word or tree, trace, . . . )

outputs a quantity in some universe of output values (Boolean values, probabilities, vector space, words, . . . )

Thomas Colcombet
“Algèbres? Co-algèbres?
Mais ils ne sont ni l’un ni l’autre !”

An automaton processes an input, respecting its structure (word, tree, infinite word or tree, trace, . . . )

outputs a quantity in some universe of output values (Boolean values, probabilities, vector space, words, . . . )

Automata are functors!!!

# Word automata

deterministic automata

$$\begin{tikzcd} 1 \arrow[r] & Q \arrow[r] \arrow["a", loop above] & 2 \end{tikzcd}$$

in Set

non-deterministic automata

$$\begin{tikzcd} 1 \arrow[r] & Q \arrow[r] \arrow["a", loop above] & 1 \end{tikzcd}$$

in Rel

weighted automata

$$\begin{tikzcd} S \arrow[r] & Q \arrow[r] \arrow["a", loop above] & S \end{tikzcd}$$

in $\mathrm{Mod}_S$

Subseq. transducers

$$\begin{tikzcd} 1 \arrow[r] & Q \arrow[r] \arrow["a", loop above] & 1 \end{tikzcd}$$

in $\mathrm{Kl}(\mathcal{T})$

# Word automata

deterministic automata

$$\begin{tikzcd} 1 \arrow[r] & Q \arrow[r] \arrow[loop above, "a"] & 2 \end{tikzcd}$$

in Set

non-deterministic automata

$$\begin{tikzcd} 1 \arrow[r] & Q \arrow[r] \arrow[loop above, "a"] & 1 \end{tikzcd}$$

in Rel

weighted automata

$$\begin{tikzcd} S \arrow[r] & Q \arrow[r] \arrow[loop above, "a"] & S \end{tikzcd}$$

in $\mathsf{Mod}_S$

Subseq. transducers

$$\begin{tikzcd} 1 \arrow[r] & Q \arrow[r] \arrow[loop above, "a"] & 1 \end{tikzcd}$$

in $\mathsf{Kl}(\mathcal{T})$

**We see a pattern emerging!**

Word automata are **functors**

$$\mathcal{A}\colon \mathcal{I} \to \mathcal{C}\,,$$

where the input category $\mathcal{I}$ is freely generated by

$$\begin{tikzcd}
\mathsf{in} \arrow[r, "\triangleright"] & \mathsf{states} \arrow[loop above, "a"] \arrow[r, "\triangleleft"] & \mathsf{out}
\end{tikzcd}$$

Word automata are **functors**

$$\mathcal{A}\colon\mathcal{I}\to\mathcal{C}\,,$$

where the input category $\mathcal{I}$ is freely generated by

$$\begin{tikzcd}
\mathsf{in} \arrow[r, "\triangleright"] & \mathsf{states} \arrow[r, "\triangleleft"] \arrow[loop above, "a"] & \mathsf{out}
\end{tikzcd}$$

deterministic automata $\mathcal{A}\colon\mathcal{I}\to\mathsf{Set}$ $\mathsf{in}\mapsto 1$ and $\mathsf{out}\mapsto 2$

Word automata are **functors**

$$\mathcal{A}:\mathcal{I} \to \mathcal{C}\,,$$

where the input category $\mathcal{I}$ is freely generated by

$$\begin{tikzcd}
\text{in} \arrow[r, "\triangleright"] & \text{states} \arrow[loop above, "a"] \arrow[r, "\triangleleft"] & \text{out}
\end{tikzcd}$$

| | | |
|---|---|---|
| deterministic automata | $\mathcal{A}:\mathcal{I} \to \mathsf{Set}$ | in $\mapsto 1$ and out $\mapsto 2$ |
| non-deterministic automata | $\mathcal{A}:\mathcal{I} \to \mathsf{Rel}$ | in $\mapsto 1$ and out $\mapsto 1$ |

Word automata are **functors**

$$\mathcal{A} \colon \mathcal{I} \to \mathcal{C}\,,$$

where the input category $\mathcal{I}$ is freely generated by

$$\begin{tikzcd}
\text{in} \arrow[r, "\triangleright"] & \text{states} \arrow[r, "\triangleleft"] \arrow[loop above, "a"] & \text{out}
\end{tikzcd}$$

| | | |
|---|---|---|
| deterministic automata | $\mathcal{A} \colon \mathcal{I} \to \mathsf{Set}$ | in $\mapsto 1$ and out $\mapsto 2$ |
| non-deterministic automata | $\mathcal{A} \colon \mathcal{I} \to \mathsf{Rel}$ | in $\mapsto 1$ and out $\mapsto 1$ |
| weighted automata | $\mathcal{A} \colon \mathcal{I} \to \mathsf{Mod}_S$ | in $\mapsto S$ and out $\mapsto S$ |

Word automata are **functors**

$$\mathcal{A}\colon \mathcal{I} \to \mathcal{C}\,,$$

where the input category $\mathcal{I}$ is freely generated by

$$\begin{tikzcd}
\mathrm{in} \arrow[r, "\triangleright"] & \mathrm{states} \arrow[loop above, "a"] \arrow[r, "\triangleleft"] & \mathrm{out}
\end{tikzcd}$$

| | | |
|---|---|---|
| deterministic automata | $\mathcal{A}\colon \mathcal{I} \to \mathsf{Set}$ | $\mathrm{in} \mapsto 1$ and $\mathrm{out} \mapsto 2$ |
| non-deterministic automata | $\mathcal{A}\colon \mathcal{I} \to \mathsf{Rel}$ | $\mathrm{in} \mapsto 1$ and $\mathrm{out} \mapsto 1$ |
| weighted automata | $\mathcal{A}\colon \mathcal{I} \to \mathsf{Mod}_S$ | $\mathrm{in} \mapsto S$ and $\mathrm{out} \mapsto S$ |
| subseq. transducers | $\mathcal{A}\colon \mathcal{I} \to \mathsf{Kl}(\mathcal{T})$ | $\mathrm{in} \mapsto 1$ and $\mathrm{out} \mapsto 1$ |

# Languages are **functors**

$$\mathcal{L} \colon \mathcal{O} \to \mathcal{C}\,,$$

where $\mathcal{O}$ is the full subcategory of $\mathcal{I}$ on objects in and out

$$\mathsf{in} \xrightarrow{\rhd w \lhd \;\; : \;\; w \in A^*} \mathsf{out}$$

# Languages are functors

$$\mathcal{L}\colon \mathcal{O} \to \mathcal{C}\,,$$

where $\mathcal{O}$ is the full subcategory of $\mathcal{I}$ on objects in and out

$$\text{in} \xrightarrow{\ \triangleright w \triangleleft \ :\ w \in A^*\ } \text{out}$$

A language $L \subseteq A^*$ can be modelled as a functor $\mathcal{L}_{\mathsf{Set}}\colon \mathcal{O} \to \mathsf{Set}$ so that $\mathcal{L}_{\mathsf{Set}}(\text{in}) = 1$ and $\mathcal{L}_{\mathsf{Set}}(\text{out}) = 2$,
For all $w \in A^*$ we have $\mathcal{L}_{\mathsf{Set}}(\triangleright w \triangleleft)\colon 1 \to 2$ in Set.

# Languages are functors

$$\mathcal{L} \colon \mathcal{O} \to \mathcal{C}\,,$$

where $\mathcal{O}$ is the full subcategory of $\mathcal{I}$ on objects in and out

$$\mathrm{in} \xrightarrow{\ \rhd w \lhd \ :\ w \in A^*\ } \mathrm{out}$$

A language $L \subseteq A^*$ can be modelled as a functor $\mathcal{L}_{\mathsf{Set}} \colon \mathcal{O} \to \mathsf{Set}$ so that $\mathcal{L}_{\mathsf{Set}}(\mathrm{in}) = 1$ and $\mathcal{L}_{\mathsf{Set}}(\mathrm{out}) = 2$,
For all $w \in A^*$ we have $\mathcal{L}_{\mathsf{Set}}(\rhd w \lhd) \colon 1 \to 2$ in Set.

Alternatively, $L \subseteq A^*$ can be modelled as a functor $\mathcal{L}_{\mathsf{Rel}} \colon \mathcal{O} \to \mathsf{Rel}$ so that $\mathcal{L}_{\mathsf{Rel}}(\mathrm{in}) = 1$ and $\mathcal{L}_{\mathsf{Rel}}(\mathrm{out}) = 1$.
For all $w \in A^*$ we have $\mathcal{L}_{\mathsf{Rel}}(\rhd w \lhd) \colon 1 \to 1$ in Rel.

# Accepting a language (the functor version)

An automaton $\mathcal{A}$ accepts a language $\mathcal{L}$ when the next diagram commutes

$$\begin{tikzcd}
\text{in} \arrow[rr, "\triangleright w \triangleleft \ : \ w \in A^*"] & & \text{out} \\
\text{in} \arrow[r, "\triangleright"] & \text{states} \arrow[r, "\triangleleft"] \arrow[loop above, "a"] & \text{out}
\end{tikzcd}$$

$$\begin{tikzcd}
\mathcal{O} \arrow[r, "\mathcal{L}"] \arrow[d, hook] & \mathcal{C} \\
\mathcal{I} \arrow[ur, "\mathcal{A}"'] &
\end{tikzcd}$$

# Accepting a language (the functor version)

An automaton $\mathcal{A}$ accepts a language $\mathcal{L}$ when the next diagram commutes

$$\begin{tikzcd}
\text{in} \arrow[rr, "\triangleright w \triangleleft \ : \ w \in A^*"] & & \text{out} \\
\text{in} \arrow[r, "\triangleright"] & \text{states} \arrow[loop above, "a"] \arrow[r, "\triangleleft"] & \text{out}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{O} \arrow[r, "\mathcal{L}"] \arrow[d, hook] & \mathcal{C} \\
\mathcal{I} \arrow[ur, "\mathcal{A}"'] &
\end{tikzcd}$$

For every language $\mathcal{L} \colon \mathcal{O} \to \mathcal{C}$ we have a category $\mathtt{Auto}_L$ of automata accepting $\mathcal{L}$.

# Automata as functors: minimization

# Minimial Automaton $\mathtt{Min}(\mathcal{L})$ for a Language

When does a 'minimal' automaton accepting a language $\mathcal{L}$ exist?

$$\begin{tikzcd}
\mathcal{O} \arrow[r, "\mathcal{L}"] \arrow[d, hook] & \mathcal{C} \\
\mathcal{I} \arrow[ur, dashed, "\mathtt{Min}(\mathcal{L})?" description] &
\end{tikzcd}$$

# Minimial Automaton $\mathtt{Min}(\mathcal{L})$ for a Language

When does a ‘minimal’ automaton accepting a language $\mathcal{L}$ exist?

$$\begin{tikzcd}
\mathcal{O} \arrow[r, "\mathcal{L}"] \arrow[d, hook] & \mathcal{C} \\
\mathcal{I} \arrow[ur, bend right, "\mathcal{A}_{\mathtt{init}}(\mathcal{L})" description] &
\end{tikzcd}$$

If the category of automata accepting $\mathcal{L}$ has

- an initial object $\mathcal{A}_{\mathtt{init}}(\mathcal{L})$,

# Minimial Automaton $\mathtt{Min}(\mathcal{L})$ for a Language

When does a ‘minimal’ automaton accepting a language $\mathcal{L}$ exist?

$$\begin{tikzcd}
\mathcal{O} \arrow[rr, "\mathcal{L}"] \arrow[ddd, hook] & & \mathcal{C} \\
& \mathcal{A}_{\mathtt{init}}(\mathcal{L}) \arrow[ur] & \\
& & \mathcal{A}_{\mathtt{final}}(\mathcal{L}) \arrow[uu] \\
\mathcal{I} \arrow[uur, dash] \arrow[urr, dash] & &
\end{tikzcd}$$

If the category of automata accepting $\mathcal{L}$ has

- an initial object $\mathcal{A}_{\mathtt{init}}(\mathcal{L})$,
- a final object $\mathcal{A}_{\mathtt{final}}(\mathcal{L})$, and,

# Minimial Automaton $\mathtt{Min}(\mathcal{L})$ for a Language

When does a ‘minimal’ automaton accepting a language $\mathcal{L}$ exist?

$$\begin{tikzcd}
\mathcal{O} \arrow[r, "\mathcal{L}"] \arrow[d, hook] & \mathcal{C} \\
\mathcal{I} \arrow[ur, bend left, "\mathcal{A}_{\mathtt{init}}(\mathcal{L})" description] \arrow[ur, dashed, "\mathtt{Min}(\mathcal{L})" description] \arrow[ur, bend right, "\mathcal{A}_{\mathtt{final}}(\mathcal{L})" description] &
\end{tikzcd}$$

If the category of automata accepting $\mathcal{L}$ has

- an initial object $\mathcal{A}_{\mathtt{init}}(\mathcal{L})$,
- a final object $\mathcal{A}_{\mathtt{final}}(\mathcal{L})$, and,
- a factorization system

then $\mathtt{Min}(\mathcal{L})$ is obtained as the factorization

$$\mathcal{A}_{\mathtt{init}}(\mathcal{L}) \twoheadrightarrow \mathtt{Min}(\mathcal{L}) \rightarrowtail \mathcal{A}_{\mathtt{final}}(\mathcal{L})\,.$$

# Trivial example

deterministic automata, i.e. $(\mathsf{Set}, 1, 2)$-automata
accepting a $(\mathsf{Set}, 1, 2)$-language

$$\begin{tikzcd}
 & A^* \arrow[dr, bend left, "\mathcal{L}?"] \arrow[d, "\mathsf{reachedState}" description] & \\
1 \arrow[ur, bend left, "\varepsilon"] \arrow[r, "i"] \arrow[dr, bend right, "\mathcal{L}"'] & Q \arrow[r, "f"] \arrow[d, "\mathsf{acceptedLanguage}" description] & 2 \\
 & 2^{A^*} \arrow[ur, bend right, "\varepsilon?"'] &
\end{tikzcd}$$

# Trivial example

deterministic automata, i.e. $(\mathsf{Set}, 1, 2)$-automata
accepting a $(\mathsf{Set}, 1, 2)$-language

$$\begin{tikzcd}
 & A^* \arrow[d, two heads] \arrow[dr, bend left, "\mathcal{L}?"] & \\
1 \arrow[ur, bend left, "\varepsilon"] \arrow[r, "i"] \arrow[dr, bend right, "\mathcal{L}"'] & \mathtt{Min}(\mathcal{L}) \arrow[r, "f"] \arrow[d, tail] & 2 \\
 & 2^{A^*} \arrow[ur, bend right, "\varepsilon?"'] &
\end{tikzcd}$$

# Minimization via epi-mono factorisations

If the output category $\mathcal{C}$ has countable powers and copowers, and, and epi-mono factorisation system, then the minimial automaton for $L$ is computed as follows

$$\begin{tikzcd}
 & \coprod_{u \in A^*} I \arrow[d, two heads] \arrow[dr, bend left, "L?"] & \\
I \arrow[ur, bend left, "\varepsilon"] \arrow[r, "i"] \arrow[dr, bend right, "L"'] & \mathtt{Min}(L) \arrow[r, "f"] \arrow[d, tail] & F \\
 & \prod_{u \in A^*} F \arrow[ur, bend right, "\varepsilon?"'] &
\end{tikzcd}$$

Thus far we have reinvented the wheel ...

Thus far we have reinvented the wheel ...

However, the wheel was a pretty awesome invention!

# What if the output category is not nice?

# Two applications

**Subsequential transducers**

the output category has copowers, factorization system, but does not have products.

# Two applications

### Subsequential transducers

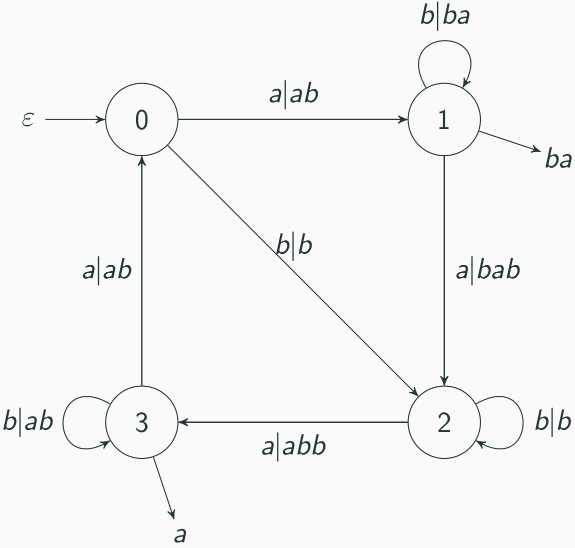
the output category has copowers, factorization system, but does not have products.

### Hybrid set-vector automata

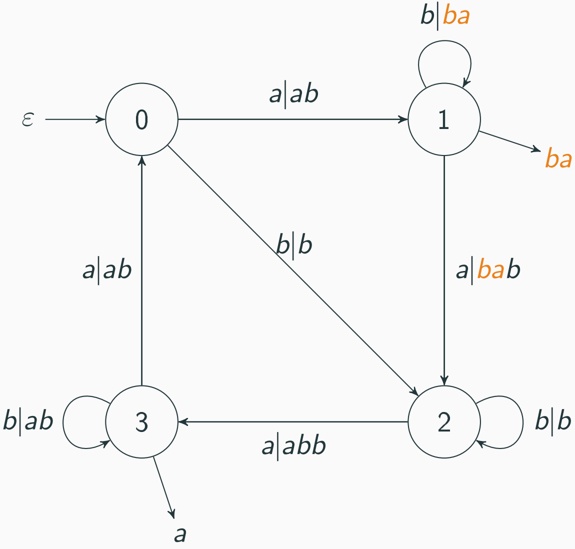
a costum-made output category that has all powers and copowers, but where the factorisation system is not “nice” enough to give a meaningful notion of minimization.

# Subsequential transducers à la Choffrut

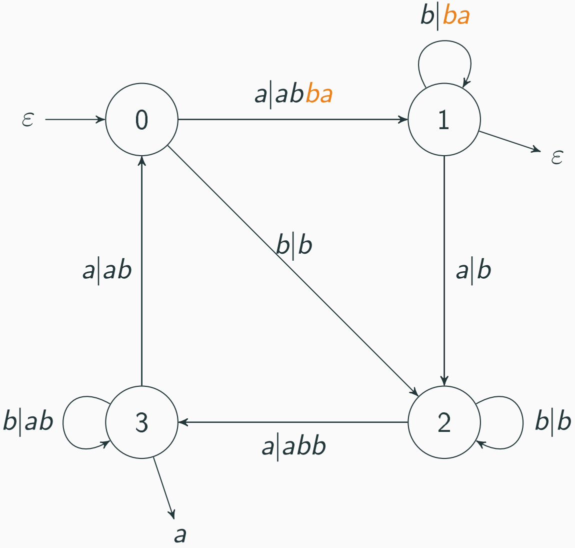
# Minimization of subsequential transducers à la Choffrut



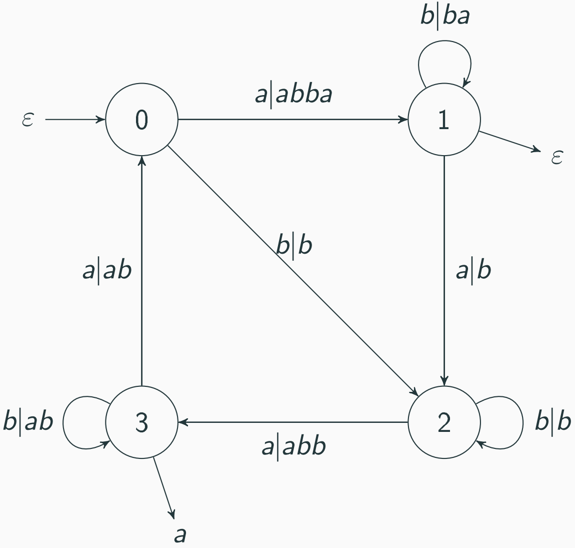
# Minimization of subsequential transducers à la Choffrut



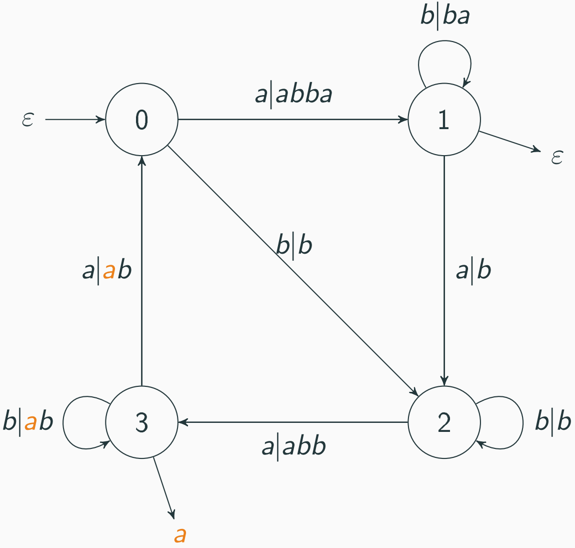
# Minimization of subsequential transducers à la Choffrut



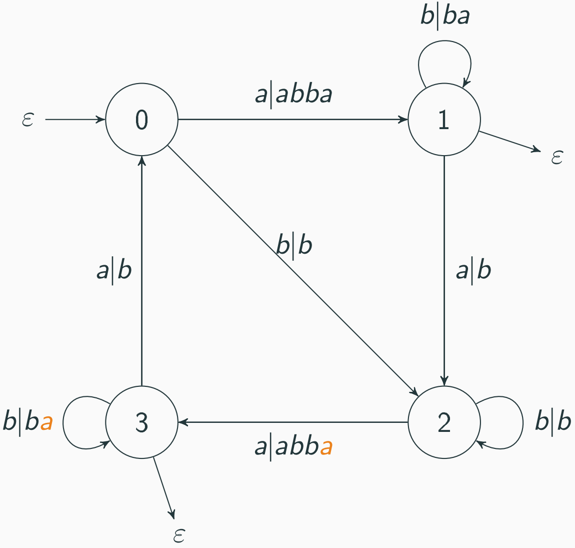
# Minimization of subsequential transducers à la Choffrut



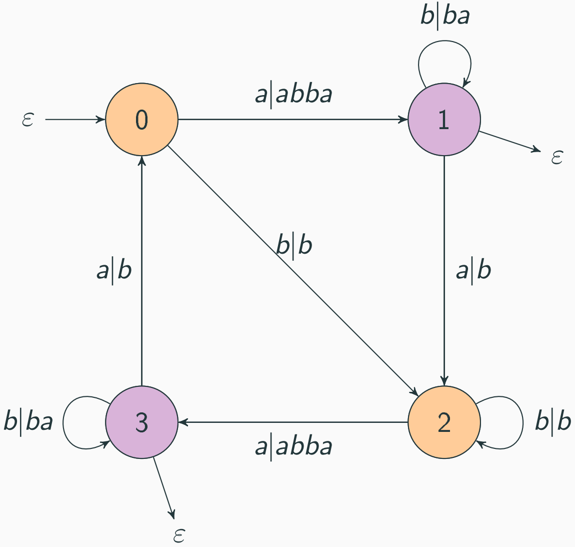
# Minimization of subsequential transducers à la Choffrut



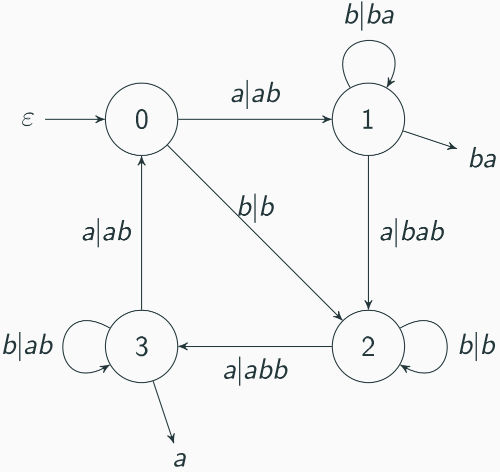
# Minimization of subsequential transducers à la Choffrut

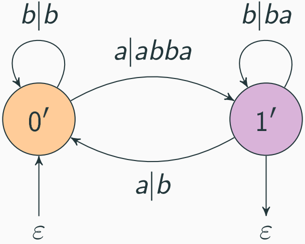


# Minimization of subsequential transducers à la Choffrut



# Minimization of subsequential transducers à la Choffrut





# Subsequential transducers as functors

A subsequential transducers with output alphabet $B$ is essentially a functor

$$\mathcal{A} : \mathcal{I} \to \mathsf{Kl}(\mathcal{T})$$

for the monad $\mathcal{T} : \mathsf{Set} \to \mathsf{Set}$ defined by

$$\mathcal{T}(X) = B^* \times X + 1 \,.$$

That is, we have the data

$$\begin{tikzcd} 1 \arrow[r] & Q \arrow[r] \arrow[loop above, "a"] & 1 & \text{in } \mathsf{Kl}(\mathcal{T}) \end{tikzcd}$$

# Subsequential transducers as functors

A subsequential transducers with output alphabet $B$ is essentially a functor

$$\mathcal{A}\colon\mathcal{I} \to \mathsf{Kl}(\mathcal{T})$$

for the monad $\mathcal{T}\colon\mathsf{Set} \to \mathsf{Set}$ defined by

$$\mathcal{T}(X) = B^* \times X + 1\,.$$

That is, we have the data

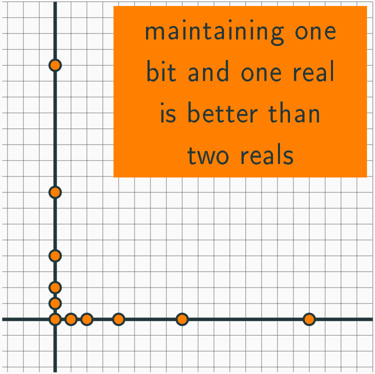
$$\begin{tikzcd} 1 \arrow[r] & Q \arrow[loop above, "a"] \arrow[r] & 1 \end{tikzcd} \qquad \text{in } \mathsf{Kl}(\mathcal{T})$$

The category $\mathsf{Kl}(\mathcal{T})$ does not have powers or products!!
This is why we cannot just use coalgebras for
$SX = (1 + B^* \times X)^{A^*} \times (1 + B^*)$, see [Hansen, 2010]

# “Glueings” of vector spaces

# Let’s backtrack to the “hybrid set-vector” automaton

The “reachable” vectors are on the “union” of two one-dimensional subspaces.

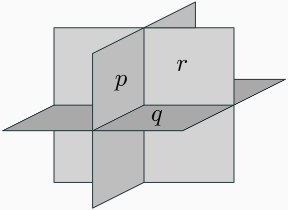


maintaining one bit and one real is better than two reals

# What is the good category to accommodate the new model?

An example of “gluings” of vector spaces

i.e. a mono-colimit in Vec



$$\begin{tikzcd}
 & & {} & & \\
\mathbb{R}^2 \arrow[rru, bend left] & & \mathbb{R}^2 \arrow[u] & & \mathbb{R}^2 \arrow[llu, bend right] \\
\mathbb{R} \arrow[u] \arrow[rru] & & \mathbb{R} \arrow[llu] \arrow[rru] & & \mathbb{R} \arrow[u] \arrow[llu]
\end{tikzcd}$$

# The category Glue($\mathcal{C}$)

A diagram $F\colon\mathcal{D}\to\mathcal{C}$ is called a mono-colimit if it has a mono-cocone in $\mathcal{C}$, that is, a cocone where all the injections are monos.

**Definition**

We define Glue($\mathcal{C}$) as the free completion of $\mathcal{C}$ under mono-colimits.

# The category $\mathsf{Glue}(\mathcal{C})$

A diagram $F\colon\mathcal{D}\to\mathcal{C}$ is called a mono-colimit if it has a mono-cocone in $\mathcal{C}$, that is, a cocone where all the injections are monos.

**Definition**

We define $\mathsf{Glue}(\mathcal{C})$ as the free completion of $\mathcal{C}$ under mono-colimits.

**Lemma**

*The category* $\mathsf{Glue}(\mathcal{C})$ *is complete and cocomplete whenever* $\mathcal{C}$ *is.*

In particular, $\mathsf{Glue}(\mathsf{Vec})$ has all the required properties so that minimisation works smoothly.

# Still, there is a catch ...

We are interested in effective minimal automata!

| | |
|---|---|
| deterministic finite automata | $\mathsf{Set}_{\mathsf{fin}}$ |
| finite-dim. vector automata | $\mathsf{Vec}_{\mathsf{fin}}$ |
| effective hybrid-set-vector automata | $\mathsf{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$ |

where $\mathsf{Glue}_{\mathsf{fin}}(\mathsf{Vec}_{\mathsf{fin}})$ is the free cocompletion of $\mathsf{Vec}_{\mathsf{fin}}$ under finite mono-colimits.

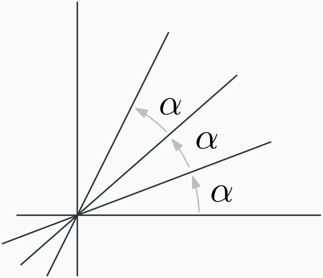
# Example

Consider the weighted language $L\colon A^* \to \mathbb{R}$ given by

$$L(u) = \cos(\alpha|u|)$$

for some $\alpha$ which is not a rational multiple of $\pi$.

The minimal automaton in Glue(Vec) is a countable colimit of one-dimensional spaces.



It seems we have “broken” the minimisation wheel ...

It seems we have "broken" the minimisation wheel ...

The fix: a factorisation **through** system

# Conclusions

# Conclusions

Our contribution: a new automata model!

The category-theoretic perspective helps with the accurate description of the hybrid set-vector automata model.

# Conclusions

Our contribution: a new automata model!

The category-theoretic perspective helps with the accurate description of the hybrid set-vector automata model.

Quite a few questions remain to be answered...

Can we characterise the presheaves that are mono-colimits of representables? (some partial results, e.g. we proved that they preserve equalisers, but that is not sufficient)

# Conclusions

Our contribution: a new automata model!

The category-theoretic perspective helps with the accurate description of the hybrid set-vector automata model.

Quite a few questions remain to be answered...

Can we characterise the presheaves that are mono-colimits of representables? (some partial results, e.g. we proved that they preserve equalisers, but that is not sufficient)

How do we effectively minimise hybrid-set-vector automata?

# Conclusions

Adjunctions between output categories lift to adjunctions for “adjoint transpose” languages. Unifying explanation for

- determinization of NFAs
- generalised powerset construction
- reversing automata

What other uses can we find for the “minimization wheel”?

- syntactic monoids, algebras
- minimization by duality
- syntactic spaces with internal monoids

  [Gehrke, P., Reggio, ICALP’16, LICS’17]
- minimization of subsequential transducers (à la Choffrut)

# References

[Colcombet, P., ACM SIGLOG april 2017]
*Automata and minimization.*

[Colcombet, P., MFCS 2017]
*Automata in the Category of Glued Vector Spaces*

[Colcombet, P., CALCO 2017]
*Automata Minimization: a Functorial Approach*