A categorical approach to automata learning and minimization – part 1

Daniela Petrişan

Université Paris Cité, IRIF, France

TACL'24, Barcelona, 24-28 June 2024

INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE

Université de Paris

TANCL'07 in Oxford

- the first conference I attended
- organized by Mai Gehrke and Hilary Priestley
- some wonderful talks Samson Abramsky, Alexander Kurz, Jean-Éric Pin, etc.
- in particular, Jean-Éric Pin taked about automata, semigroups and duality ...

TANCL'07 in Oxford

- the first conference I attended
- organized by Mai Gehrke and Hilary Priestley
- some wonderful talks Samson Abramsky, Alexander Kurz, Jean-Éric Pin, etc.
- in particular, Jean-Éric Pin taked about automata, semigroups and duality ...
- my first slide at TACL'2019, organized by Mai Gehrke in Nice !

References for Lecture 1

T. Colcombet and D. Petrişan. *Automata minimization: a functorial approach*. Log. Methods Comput. Sci., 16(1), 2020

J. E. Pin (Ed.) *Handbook of Automata Theory*, EMS Press, 2021

This tutorial is about ...

the interplay between category theory and automata theory.

This tutorial is about ...

the interplay between category theory and automata theory.

In particular, we will see how the category-theoretic approach

- provides a unifying framework for modelling various forms of automata,
- for obtaining generic algorithms for learning algorithms,

This tutorial is about ...

the interplay between category theory and automata theory.

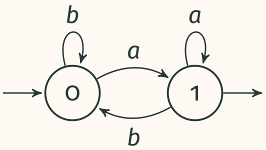
In particular, we will see how the category-theoretic approach

- provides a unifying framework for modelling various forms of automata,
- for obtaining generic algorithms for learning algorithms,
- highlights the link between automata learning and minimization.

Automata – the basics

A complete deterministic finite automaton over some finite alphabet A is a tuple $\mathcal{A} = (Q, q_0, F, (\delta_a)_{a \in A})$ where

- Q is a finite set of states
- q_0 is an element of Q, called initial state
- $F \subseteq Q$ is a subset of accepting states
- for every letter $a \in A$, $\delta_a : Q \to Q$ is a transition function

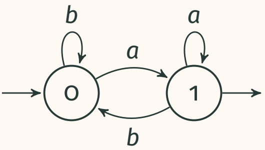


For each word $w = a_1 \ldots a_n \in A^*$, we put $\delta_w = \delta_{a_n} \circ \ldots \circ \delta_{a_1}$ and $\delta_\varepsilon = id_Q$.

Automata – the basics

A complete deterministic finite automaton over some finite alphabet A is a tuple $\mathcal{A} = (Q, q_0, F, (\delta_a)_{a \in A})$ where

- Q is a finite set of states
- q_0 is an element of Q, called initial state
- $F \subseteq Q$ is a subset of accepting states
- for every letter $a \in A$, $\delta_a : Q \to Q$ is a transition function

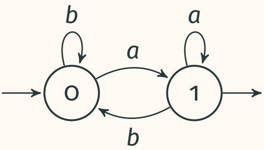


For each word $w = a1 \ldots a_n \in A^*$, we put $\delta_w = \delta_{a_n} \circ \ldots \circ \delta_{a_1}$ and $\delta_\varepsilon = id_Q$. A word $w \in A^*$ is accepted by $\mathcal{A}$ when $\delta_w(q_0) \in F$.

Automata – the basics

A complete deterministic finite automaton over some finite alphabet A is a tuple $\mathcal{A} = (Q, q_0, F, (\delta_a)_{a \in A})$ where

- Q is a finite set of states
- q_0 is an element of Q, called initial state
- $F \subseteq Q$ is a subset of accepting states
- for every letter $a \in A$, $\delta_a \colon Q \to Q$ is a transition function



For each word $w = a_1 \ldots a_n \in A^*$, we put $\delta_w = \delta_{a_n} \circ \ldots \circ \delta_{a_1}$ and $\delta_\varepsilon = id_Q$.
A word $w \in A^*$ is accepted by $\mathcal{A}$ when $\delta_w(q_0) \in F$.
The language of $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A})$ of words over A^* accepted by $\mathcal{A}$.

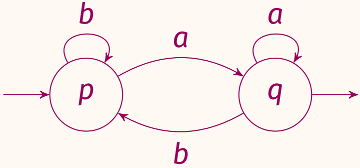
Regular languages

... form a very robust class – described in a multitude of ways.

Example: “Last letter is a.”

$(a+b)^*a$

the language of a regular expression (Kleene theorem)



recognised by a DFA or an NFA

regular languages

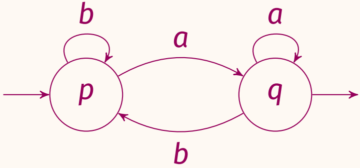
Regular languages

… form a very robust class – described in a multitude of ways.

Example: "Last letter is *a*."

$(a+b)^*a$

the language of a regular expression (Kleene theorem)



recognised by a DFA or an NFA

regular languages

$\phi\colon A^* \to \{1, a, b\}$

the preimage of a monoid morphism

	1	a	b
1	1	a	b
a	a	a	b
b	b	a	b

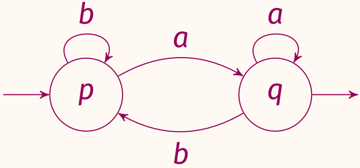
Regular languages

… form a very robust class – described in a multitude of ways.

Example: "Last letter is a."

$(a+b)^*a$

the language of a regular expression (Kleene theorem)



recognised by a DFA or an NFA

regular languages

$\phi\colon A^* \to \{1, a, b\}$

the preimage of a monoid morphism

	1	a	b
1	1	a	b
a	a	a	b
b	b	a	b

$\exists x.\ \neg(\exists y.\ x < y) \wedge Q_a x$

definable in MSO (Büchi-Elgot-Trakhtenbrot)

Minimization

Given a language $L \subseteq A^*$ and a word $u \in A^*$ the left quotient $u^{-1}L$ is the set

$$\{v \in A^* \mid uv \in L\}$$

The Myhill-Nerode equivalence is defined by

$$u \cong_L v \text{ iff } u^{-1}L = v^{-1}L$$

Minimization

Given a language $L \subseteq A^*$ and a word $u \in A^*$ the left quotient $u^{-1}L$ is the set

$$\{v \in A^* \mid uv \in L\}$$

The Myhill-Nerode equivalence is defined by

$$u \cong_L v \text{ iff } u^{-1}L = v^{-1}L$$

Theorem (Myhill-Nerode). A language L is regular iff it has only finitely many left quotients iff $\cong_L$ has finite index.

Minimization

Given a language $L \subseteq A^*$ and a word $u \in A^*$ the left quotient $u^{-1}L$ is the set

$$\{v \in A^* \mid uv \in L\}$$

The Myhill-Nerode equivalence is defined by

$$u \cong_L v \text{ iff } u^{-1}L = v^{-1}L$$

Theorem (Myhill-Nerode). A language L is regular iff it has only finitely many left quotients iff $\cong_L$ has finite index.

Proof. $\Rightarrow$ If an automaton $\mathcal{A} = (Q, q_0, F, (\delta_a)_{a \in A})$ accepts a language L, then the automaton $(Q, \delta_u(q_0), F, (\delta_a)_{a \in A})$ accepts $u^{-1}L$.

Minimization

Given a language $L \subseteq A^*$ and a word $u \in A^*$ the left quotient $u^{-1}L$ is the set

$$\{v \in A^* \mid uv \in L\}$$

The Myhill-Nerode equivalence is defined by

$$u \cong_L v \text{ iff } u^{-1}L = v^{-1}L$$

Theorem (Myhill-Nerode). A language L is regular iff it has only finitely many left quotients iff $\cong_L$ has finite index.

Proof. $\Rightarrow$ If an automaton $\mathcal{A} = (Q, q_0, F, (\delta_a)_{a \in A})$ accepts a language L, then the automaton $(Q, \delta_u(q_0), F, (\delta_a)_{a \in A})$ accepts $u^{-1}L$.

$\Leftarrow$ Consider the Nerode automaton of L, that is $(Q, q_0, F, (\delta_a)_{a \in A})$, where

- $Q = \{u^{-1}L \mid u \in A^*\}$,
- $q_0 = L$
- $F = \{u^{-1}L \mid u \in L\}$ and
- $\delta_a(u^{-1}L) = (ua)^{-1}L$.

Minimization

How do we minimize an automaton $\mathcal{A}$?

- remove all states that are not accessible from the initial state. We obtain the reachable sub-automaton $\mathrm{Reach}(\mathcal{A})$.
- Merge all states that accept the same language, we obtain the observable quotient $\mathrm{Obs}(\mathrm{Reach}(\mathcal{A}))$.

Minimization

How do we minimize an automaton $\mathcal{A}$?

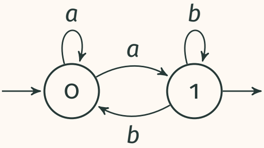
- remove all states that are not accessible from the initial state. We obtain the reachable sub-automaton $\mathrm{Reach}(\mathcal{A})$.
- Merge all states that accept the same language, we obtain the observable quotient $\mathrm{Obs}(\mathrm{Reach}(\mathcal{A}))$.

There are several algorithms for minimizing automata:
Moore, Hopcroft, Brzozowski.

Non-deterministic automata

A non-deterministic finite automaton over some finite alphabet A is a tuple $\mathcal{A} = (Q, I, F, \delta)$ where

- Q is a finite set of states
- $I \subseteq Q$ is a subset of initial states
- $F \subseteq Q$ is a subset of accepting states
- $\delta \subseteq Q \times A \times Q$ is set of transitions

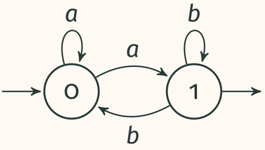


A word $w \in A^*$ is accepted by $\mathcal{A}$ when there is a path labelled by w starting from an initial state and finishing in an accepting state.

Non-deterministic automata

A non-deterministic finite automaton over some finite alphabet A is a tuple $\mathcal{A} = (Q, I, F, \delta)$ where

- Q is a finite set of states
- $I \subseteq Q$ is a subset of initial states
- $F \subseteq Q$ is a subset of accepting states
- $\delta \subseteq Q \times A \times Q$ is set of transitions



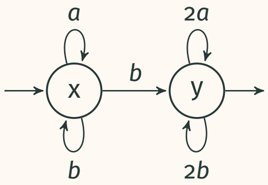
A word $w \in A^*$ is accepted by $\mathcal{A}$ when there is a path labelled by w starting from an initial state and finishing in an accepting state.

Proposition. Every NFA is equivalent to a DFA.

Weighted automata over a semiring

Given a semiring S, an S-weighted automaton over some finite alphabet $A$is a tuple $\mathcal{A} = (Q, i, f, \delta)$ where

- Q is a finite set of states
- $i\colon Q \to S$ assigns an initial value to each state
- $f\colon Q \to S$ is a subset a final value to each state
- $\delta\colon Q \times A \times Q \to S$ assigns to each transition a value in S

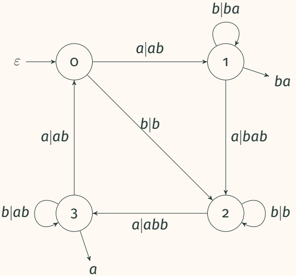


- Let $w \in A^*$. For an accepting path labelled by w compute its weight using the multiplication of the semiring.
- We add the weights of all accepting pathes labelled by w to obtain $\mathcal{L}(\mathcal{A})(w)$.

Sequential transducers

A sequential transducer with input alphabet A and output alphabet B consists of:

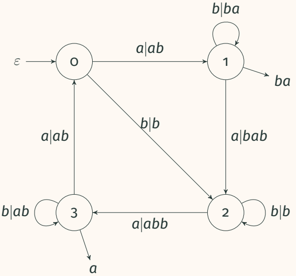
- a finite set of states Q
- an initial state with an initial output in B^*, or an undefined initial state



Sequential transducers

A sequential transducer with input alphabet A and output alphabet B consists of:

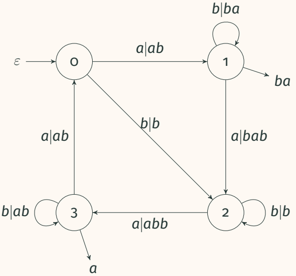
- a finite set of states Q
- an initial state with an initial output in B^*, or an undefined initial state
- for each $a \in A$ a transition function $Q \to B^* \times Q + 1$



Sequential transducers

A sequential transducer with input alphabet A and output alphabet B consists of:

- a finite set of states Q
- an initial state with an initial output in B^*, or an undefined initial state
- for each $a \in A$ a transition function $Q \to B^* \times Q + 1$
- for each state in Q, either an output word in B^* or undefined.



A unifying framework for automata minimization

Very basic notions of category theory

Definition. A category $\mathcal{C}$ consists of the following data:

- a class of objects $A, B, \ldots$

Very basic notions of category theory

Definition. A category $\mathcal{C}$ consists of the following data:

- a class of objects $A, B, \ldots$
- for every pair of objects (A, B) a set $\mathcal{C}(A, B)$ of morphisms or arrows
  We write $f\colon A \to B$ or $A \xrightarrow{f} B$ for $f \in \mathcal{C}(A, B)$
- for every object A, an identity morphism $1_A\colon A \to A$

Very basic notions of category theory

Definition. A category $\mathcal{C}$ consists of the following data:

- a class of objects $A, B, \ldots$
- for every pair of objects (A, B) a set $\mathcal{C}(A, B)$ of morphisms or arrows

  We write $f\colon A \to B$ or $A \xrightarrow{f} B$ for $f \in \mathcal{C}(A, B)$
- for every object A, an identity morphism $1_A\colon A \to A$
- a partial composition $\circ : \mathcal{C}(A, B) \times \mathcal{C}(B, C) \to \mathcal{C}(A, C)$

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend left, "g \circ f"] & B \arrow[r, "g"] & C \end{tikzcd}$$

Additionally the composition should satisfy unit and associativity axioms.

Examples of categories

To get the gist of the remaining slides, we basically need to understand 4-5 examples of categories :

- Set – the category of sets and functions

Examples of categories

To get the gist of the remaining slides, we basically need to understand 4-5 examples of categories :

- Set – the category of sets and functions
- Rel – the category of sets and relations

Examples of categories

To get the gist of the remaining slides, we basically need to understand 4-5 examples of categories :

- Set – the category of sets and functions
- Rel – the category of sets and relations
- Vec – the category of vector spaces and linear transformations

Examples of categories

To get the gist of the remaining slides, we basically need to understand 4-5 examples of categories :

- Set – the category of sets and functions
- Rel – the category of sets and relations
- Vec – the category of vector spaces and linear transformations
- $\mathcal{T}$ – the category of free partial actions of some free monoid B^* and their morphisms

Examples of categories

To get the gist of the remaining slides, we basically need to understand 4-5 examples of categories :

- Set – the category of sets and functions
- Rel – the category of sets and relations
- Vec – the category of vector spaces and linear transformations
- $\mathcal{T}$ – the category of free partial actions of some free monoid B^* and their morphisms
- the free category on a graph, in particular

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

Many more, that you have surely encountered: (semi)groups, monoids, topological spaces, etc.

Word automata

deterministic automata $\quad 1 \longrightarrow Q \longrightarrow 2 \quad$ in Set

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

Word automata

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

non-deterministic automata $$\begin{tikzcd} 1 \arrow[r] & Q \arrow[loop above, "a"] \arrow[r] & 1 \end{tikzcd}$$ in Rel

Word automata

deterministic automata	$\begin{tikzcd} 1 \arrow[r] & Q \arrow[loop above, "a"] \arrow[r] & 2 \end{tikzcd}$	in Set
non-deterministic automata	$\begin{tikzcd} 1 \arrow[r] & Q \arrow[loop above, "a"] \arrow[r] & 1 \end{tikzcd}$	in Rel
weighted automata	$\begin{tikzcd} K \arrow[r] & Q \arrow[loop above, "a"] \arrow[r] & K \end{tikzcd}$	in Vec_K

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

Seq. transducers $\begin{tikzcd} 1 \arrow[r] & Q \arrow[r] \arrow["a", loop above] & 1 \end{tikzcd}$ in $\mathcal{T}$

The output category for subsequential transducers

We consider partial actions for the free monoid B^*.

The output category for subsequential transducers

We consider partial actions for the free monoid B^*.

We consider a category $\mathcal{T}$ with

- objects: sets $X, Y, Z, \ldots$
- arrows: $f : X \nrightarrow Y$, where $f : X \to B^* \times Y + 1$ is a function

The output category for subsequential transducers

We consider partial actions for the free monoid B^*.

We consider a category $\mathcal{T}$ with

- objects: sets $X, Y, Z, \ldots$
- arrows: $f : X \nrightarrow Y$, where $f : X \to B^* \times Y + 1$ is a function

Composition of arrows in $\mathcal{T}$ is defined using the monoid multiplication in B^*.

If $f : X \nrightarrow Y$ and $g : Y \nrightarrow Z$ then $g \circ f : X \nrightarrow Z$ (i.e. $g \circ f : X \to B^* \times Z + 1$) is given by $g \circ f(x) = \begin{cases} (uv, z) & \text{if } f(x) = (u, y) \text{ and } g(y) = (v, z) \\ \perp & \text{otherwise.} \end{cases}$

The output category for subsequential transducers

We consider partial actions for the free monoid B^*.

We consider a category $\mathcal{T}$ with

- objects: sets $X, Y, Z, \dots$
- arrows: $f\colon X \nrightarrow Y$, where $f\colon X \to B^* \times Y + 1$ is a function

Composition of arrows in $\mathcal{T}$ is defined using the monoid multiplication in B^*.

If $f\colon X \nrightarrow Y$ and $g\colon Y \nrightarrow Z$ then $g \circ f\colon X \nrightarrow Z$ (i.e. $g \circ f\colon X \to B^* \times Z + 1$) is given by $g \circ f(x) = \begin{cases} (uv, z) & \text{if } f(x) = (u, y) \text{ and } g(y) = (v, z) \\ \perp & \text{otherwise.} \end{cases}$

This is the Kleisli category for the monad $T\colon \mathrm{Set} \to \mathrm{Set}$ given by $T(X) = B^* \times X + 1$, which associates to each set X the free partial action of B^* on X.

The output category for subsequential transducers

Interpretting the arrows

$$\begin{tikzcd} 1 \arrow[r, "i"] & Q \arrow[r, "f"] \arrow[loop above, "\delta_a"] & 1 \end{tikzcd} \qquad \text{in } \mathcal{T}$$

ammounts to give

The output category for subsequential transducers

Interpretting the arrows

$$
\begin{tikzcd}
1 \arrow[r, "i"] & Q \arrow[loop above, "\delta_a"] \arrow[r, "f"] & 1
\end{tikzcd}
\qquad \text{in } \mathcal{T}
$$

ammounts to give

- a function $i\colon 1 \to B^* \times Q + 1$, i.e. an initial state with an initial output in B^*, or an undefined initial state

The output category for subsequential transducers

Interpretting the arrows

$$\begin{tikzcd} 1 \arrow[r, "i"] & Q \arrow[loop above, "\delta_a"] \arrow[r, "f"] & 1 \end{tikzcd} \qquad \text{in } \mathcal{T}$$

ammounts to give

- a function $i\colon 1 \to B^* \times Q + 1$, i.e. an initial state with an initial output in B^*, or an undefined initial state
- for each $a \in A$ a function $\delta_a\colon Q \to B^* \times Q + 1$

The output category for subsequential transducers

Interpretting the arrows

$$\begin{tikzcd} 1 \arrow[r, "i"] & Q \arrow[r, "f"] \arrow[loop above, "\delta_a"] & 1 \end{tikzcd} \qquad \text{in } \mathcal{T}$$

ammounts to give

- a function $i : 1 \to B^* \times Q + 1$, i.e. an initial state with an initial output in B^*, or an undefined initial state
- for each $a \in A$ a function $\delta_a : Q \to B^* \times Q + 1$
- a final map $f : Q \to B^* \times 1 + 1$, i.e. for each state in Q either an output word in B^* or undefined.

What does “interpretting” mean?

“Interpretting” means moving from one category to another.
It’s all about functors – which are to categories what functions are to sets !!

What does “interpretting” mean?

“Interpretting” means moving from one category to another.
It’s all about functors – which are to categories what functions are to sets ‼

Definition. Given categories $\mathcal{C}$ and $\mathcal{D}$, a functor $F : \mathcal{C} \to \mathcal{D}$ consists of the following data:

- for each object A of $\mathcal{C}$, an object FA of $\mathcal{D}$
- for each arrow $f : A \to B$ in $\mathcal{C}$, an arrow $Ff : FA \to FB$ in $\mathcal{D}$

such that identities and composition are preserved:
$F(1_A) = 1_{FA}$ and $Ff \circ Fg = F(f \circ g)$ when $f \circ g$ is defined.

Word automata as functors

Word automata on A^* are **functors** $\mathcal{A}\colon\mathcal{I} \to \mathcal{C}$, where the input category $\mathcal{I}$ is freely generated by

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

The data given by the functor $\mathcal{A}$ is a tuple $\langle Q, i, f, (\delta_a)_{a\in A}\rangle$, where

- Q is an object of $\mathcal{C}$.
- $i\colon I \to Q$ is the «initial» arrow, for some object I of $\mathcal{C}$
- $f\colon Q \to F$ is the «final» arrow, for some object F of $\mathcal{C}$
- $\delta_a\colon Q \to Q$ is the «transition» arrow for each $a \in A$

Word automata as functors

Word automata on A^* are **functors** $\mathcal{A}\colon \mathcal{I} \to \mathcal{C}$, where the input category $\mathcal{I}$ is freely generated by

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

The data given by the functor $\mathcal{A}$ is a tuple $\langle Q, i, f, (\delta_a)_{a \in A} \rangle$, where

- Q is an object of $\mathcal{C}$.
- $i\colon I \to Q$ is the «initial» arrow, for some object I of $\mathcal{C}$
- $f\colon Q \to F$ is the «final» arrow, for some object F of $\mathcal{C}$
- $\delta_a\colon Q \to Q$ is the «transition» arrow for each $a \in A$

The language accepted by $\mathcal{A}$ is a map $L_{\mathcal{A}}\colon A^* \to \mathcal{C}(I, F)$ that associates to a word $w = a_1 \ldots a_n$ the composite morphism

$$I \xrightarrow{i} Q \xrightarrow{\delta_{a_1}} Q \xrightarrow{\delta_{a_2}} \ldots \xrightarrow{\delta_{a_n}} Q \xrightarrow{f} F$$

Automata and languages as functors

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

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

Automata and languages as functors

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

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

For every language $\mathcal{L}\colon \mathcal{O} \to \mathcal{C}$ we consider a category $\mathrm{Auto}_{\mathcal{L}}$ of automata accepting $\mathcal{L}$.

$\mathcal{O}$ can be seen as an “observation” subcategory of $\mathcal{I}$.

Much of the ensuing theory can be developed independently on the precise shape of $\mathcal{I}$.

Automata in a category: minimization

Minimzation of $\mathcal{C}$-automata

- What does it mean for a $\mathcal{C}$-automaton to be minimal?
- What are sufficient conditions on $\mathcal{C}$ so that a minimal automaton for a language exists?
- Can we compute the minimal automaton effectively?

Minimzation of $\mathcal{C}$-automata

- What does it mean for a $\mathcal{C}$-automaton to be minimal?
- What are sufficient conditions on $\mathcal{C}$ so that a minimal automaton for a language exists?
- Can we compute the minimal automaton effectively?

A DFA is minimal when it divides any other automaton accepting the same language.

Minimzation of $\mathcal{C}$-automata

- What does it mean for a $\mathcal{C}$-automaton to be minimal?
- What are sufficient conditions on $\mathcal{C}$ so that a minimal automaton for a language exists?
- Can we compute the minimal automaton effectively?

A DFA is minimal when it divides any other automaton accepting the same language. Here divides = «is a quotient of a sub-automaton of»

Minimzation of $\mathcal{C}$-automata

- What does it mean for a $\mathcal{C}$-automaton to be minimal?
- What are sufficient conditions on $\mathcal{C}$ so that a minimal automaton for a language exists?
- Can we compute the minimal automaton effectively?

A DFA is minimal when it divides any other automaton accepting the same language. Here divides = «is a quotient of a sub-automaton of»

Thus we need a notion of «quotient» (surjection for sets) and «sub-object» (injection for sets), i.e. a factorization system.

Three more category-theoretic notions

- An initial object in a category $\mathcal{C}$ is an object X such that for any object A of $\mathcal{C}$ there is a unique morphism $! \colon X \to A$.
  Question: what is the initial object in Set*? And in* Rel*?*

Three more category-theoretic notions

- An initial object in a category $\mathcal{C}$ is an object X such that for any object A of $\mathcal{C}$ there is a unique morphism $! \colon X \to A$.
  Question: what is the initial object in Set*? And in* Rel*?*
- A final object in a category $\mathcal{C}$ is an object Y such that for any object A of $\mathcal{C}$ there is a unique morphism $! \colon A \to Y$.
  Question: what is the final object in Set*? And in* Rel*?*

Three more category-theoretic notions

- An initial object in a category $\mathcal{C}$ is an object X such that for any object A of $\mathcal{C}$ there is a unique morphism $! \colon X \to A$.
  Question: what is the initial object in Set*? And in* Rel*?*
- A final object in a category $\mathcal{C}$ is an object Y such that for any object A of $\mathcal{C}$ there is a unique morphism $! \colon A \to Y$.
  Question: what is the final object in Set*? And in* Rel*?*
- A factorization system provides the category-theoretic generalizations for the notions of “quotients” and “subobjects”, definition on next slide...

The three ingredients for minimization

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

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

The three ingredients for minimization

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

left Kan ext?

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

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

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

The three ingredients for minimization

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

$$
\begin{tikzcd}
\mathcal{O} \arrow[r, "\mathcal{L}"] \arrow[d, hook] & \mathcal{C} \\
\mathcal{I} \arrow[ur, bend left=50, "\mathcal{A}_{\mathtt{init}}(\mathcal{L})"] \arrow[ur, bend right=50, "\mathcal{A}_{\mathtt{final}}(\mathcal{L})"'] &
\end{tikzcd}
$$

right Kan extension?

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,

The three ingredients for minimization

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

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

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

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

The three ingredients for minimization

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

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

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

- an initial object $\mathcal{A}_{\mathtt{init}}(\mathcal{L})$, ✓ when $\mathcal{C}$ has copowers
- a final object $\mathcal{A}_{\mathtt{final}}(\mathcal{L})$, and, ✓ when $\mathcal{C}$ has powers
- a factorization system ✓ when $\mathcal{C}$ has one

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

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

Factorization systems

Factorization systems are a generalization of the next situation: Every function $f\colon X \to Y$ can we written as a composite

$$X \xrightarrow{e} \!\!\!\!\rightarrow Z \rightarrowtail Y$$

with e a surjection and m and injection, and, moreover, such a decomposition is unique up-to isomorphism.

Factorization systems

Factorization systems are a generalization of the next situation: Every function $f\colon X \to Y$ can we written as a composite

$$X \xrightarrow{e} \mathrel{\!\!\!\!\twoheadrightarrow} Z \rightarrowtail \xrightarrow{m} Y$$

with e a surjection and m and injection, and, moreover, such a decomposition is unique up-to isomorphism.

In a category $\mathcal{C}$, a factorization system consists of two classes of morphisms, E and M, so that:

Factorization systems

Factorization systems are a generalization of the next situation: Every function $f\colon X \to Y$ can we written as a composite

$$X \overset{e}{\twoheadrightarrow} Z \overset{m}{\rightarrowtail} Y$$

with e a surjection and m and injection, and, moreover, such a decomposition is unique up-to isomorphism.

In a category $\mathcal{C}$, a factorization system consists of two classes of morphisms, E and M, so that:

- E and M contain the isomorphisms and are closed under composition;

Factorization systems

Factorization systems are a generalization of the next situation: Every function $f: X \to Y$ can we written as a composite

$$X \xrightarrow{e} \!\!\!\!\twoheadrightarrow Z \rightarrowtail^{m} Y$$

with e a surjection and m and injection, and, moreover, such a decomposition is unique up-to isomorphism.

In a category $\mathcal{C}$, a factorization system consists of two classes of morphisms, E and M, so that:

- E and M contain the isomorphisms and are closed under composition;
- every morphism $f: X \to Y$ can we written as a composite $e \circ m$ with $e \in E$ and $m \in M$;

Factorization systems

Factorization systems are a generalization of the next situation: Every function $f: X \to Y$ can we written as a composite

$$X \xrightarrow{e} \!\!\!\!\rightarrow Z \rightarrowtail^{m} Y$$

with e a surjection and m and injection, and, moreover, such a decomposition is unique up-to isomorphism.

In a category $\mathcal{C}$, a factorization system consists of two classes of morphisms, E and M, so that:

- E and M contain the isomorphisms and are closed under composition;
- every morphism $f: X \to Y$ can we written as a composite $e \circ m$ with $e \in E$ and $m \in M$;
- the decomposition is functorial, i.e. any two decompositions are isomorphic

The three ingredients for minimization

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

The three ingredients for minimization

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

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

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

The three ingredients for minimization

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

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

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

The three ingredients for minimization

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

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

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

The three ingredients for minimization

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

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

- an initial object $\mathcal{A}_{\mathtt{init}}(\mathcal{L})$, ✓when $\mathcal{C}$ has copowers
- a final object $\mathcal{A}_{\mathtt{final}}(\mathcal{L})$, and, ✓when $\mathcal{C}$ has powers
- a factorization system ✓when $\mathcal{C}$ has one

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: minimizing DFAs

The initial automaton $\mathcal{A}_{\mathtt{init}}$ for Set-automata accepting a language L is the following :

$$\begin{tikzcd} 1 \arrow[r, "\varepsilon"] & A^* \arrow[loop above, "w \mapsto wa"] \arrow[r, "L?"] & 2 \end{tikzcd}$$

Trivial example: minimizing DFAs

The initial automaton $\mathcal{A}_{\mathtt{init}}$ for Set-automata accepting a language L is the following :

$$\begin{tikzcd} 1 \arrow[r, "\varepsilon"] & A^* \arrow[loop above, "w \mapsto wa"] \arrow[r, "L?"] & 2 \end{tikzcd}$$

The final automaton $\mathcal{A}_{\mathtt{final}}$ for Set-automata accepting a language L is the following :

$$\begin{tikzcd} 1 \arrow[r, "L"] & 2^{A^*} \arrow[loop above, "K \mapsto a^{-1}K"] \arrow[r, "\varepsilon?"] & 2 \end{tikzcd}$$

Trivial example: minimizing DFAs accepting L

The unique map from the initial to the final automaton is given by $! \colon A^* \to 2^{A^*}$, defined by $w \mapsto w^{-1}L$.

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

Trivial example: minimizing DFAs accepting L

The unique map from the initial to the final automaton is given by $!\colon A^* \to 2^{A^*}$, defined by $w \mapsto w^{-1}L$.

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

Another trivial example

$\mathbb{R}$-weighted automata, i.e. $(\mathsf{Vec}, \mathbb{R}, \mathbb{R})$-automata accepting a $(\mathsf{Vec}, \mathbb{R}, \mathbb{R})$-language

$$\begin{tikzcd}
 & \bigoplus\limits_{u \in A^*} \mathbb{R} \arrow[d, "\mathsf{reachedState}"] \arrow[dr, bend left, "L?"] & \\
\mathbb{R} \arrow[ur, bend left, "\varepsilon"] \arrow[r, "i"] \arrow[dr, bend right, "L"'] & Q \arrow[r, "F"] \arrow[d, "\mathsf{acceptedLanguage}"] & \mathbb{R} \\
 & \prod\limits_{u \in A^*} \mathbb{R} \arrow[ur, bend right, "\varepsilon?"'] &
\end{tikzcd}$$

Another trivial example

$\mathbb{R}$-weighted automata, i.e. $(\mathsf{Vec}, \mathbb{R}, \mathbb{R})$-automata
accepting a $(\mathsf{Vec}, \mathbb{R}, \mathbb{R})$-language

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

The minimal transducer in a picture

We obtain $\mathtt{Min}(\mathcal{L})$ – the minimal subsequential transducer as obtained by Choffrut!

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

The minimal transducer in a picture

We obtain $\mathtt{Min}(\mathcal{L})$ – the minimal subsequential transducer as obtained by Choffrut! In fact it also works if we replace B^* by a trace monoid.

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

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

The automaton $\mathtt{Min}(\mathcal{L})$ divides any other automaton accepting $\mathcal{L}$.

$$\begin{tikzcd}
 & \mathcal{A} \arrow[dr] & \\
\mathcal{A}_{\mathtt{init}}(L) \arrow[ur] & & \mathcal{A}_{\mathtt{final}}(L)
\end{tikzcd}$$

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

The automaton $\mathtt{Min}(\mathcal{L})$ divides any other automaton accepting $\mathcal{L}$.

$$
\begin{tikzcd}
& & \mathcal{A} \arrow[drr, bend left] & & \\
\mathcal{A}_{\mathtt{init}}(L) \arrow[r, two heads] \arrow[urr, bend left] & \mathtt{reach}(\mathcal{A}) \arrow[ur, tail] & & & \mathcal{A}_{\mathtt{final}}(L)
\end{tikzcd}
$$

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

The automaton $\mathtt{Min}(\mathcal{L})$ divides any other automaton accepting $\mathcal{L}$.

$$\begin{tikzcd}
 & & \mathcal{A} \arrow[drr, bend left] & \\
\mathcal{A}_{\mathtt{init}}(L) \arrow[r, two heads] \arrow[urr, bend left] & \mathtt{reach}(\mathcal{A}) \arrow[r, two heads] \arrow[ur, tail] & \mathtt{obs}(\mathtt{reach}(\mathcal{A})) \arrow[r, tail] & \mathcal{A}_{\mathtt{final}}(L)
\end{tikzcd}$$

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

The automaton $\mathtt{Min}(\mathcal{L})$ divides any other automaton accepting $\mathcal{L}$.

$$
\begin{tikzcd}
 & & \mathcal{A} \arrow[dr, bend left] & \\
\mathcal{A}_{\mathtt{init}}(L) \arrow[urr, bend left] \arrow[r, two heads] \arrow[drr, bend right, two heads] & \mathtt{reach}(\mathcal{A}) \arrow[ur, tail] \arrow[r, two heads] & \mathtt{obs}(\mathtt{reach}(\mathcal{A})) \arrow[r, tail] & \mathcal{A}_{\mathtt{final}}(L) \\
 & & \mathtt{Min}(L) \arrow[ur, bend right, tail] &
\end{tikzcd}
$$

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

The automaton $\mathtt{Min}(\mathcal{L})$ divides any other automaton accepting $\mathcal{L}$.

$$
\begin{tikzcd}
 & & \mathcal{A} \arrow[dr, bend left] & \\
\mathcal{A}_{\mathtt{init}}(L) \arrow[urr, bend left] \arrow[r, two heads] \arrow[drr, two heads, bend right] & \mathtt{reach}(\mathcal{A}) \arrow[ur, tail] \arrow[r, two heads] & \mathtt{obs}(\mathtt{reach}(\mathcal{A})) \arrow[r, tail] \arrow[d, dashed, no head] & \mathcal{A}_{\mathtt{final}}(L) \\
 & & \mathtt{Min}(L) \arrow[ur, tail, bend right] &
\end{tikzcd}
$$

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

The automaton $\mathtt{Min}(\mathcal{L})$ divides any other automaton accepting $\mathcal{L}$.

$$
\begin{tikzcd}
 & & \mathcal{A} \arrow[dr, bend left] & \\
\mathcal{A}_{\mathtt{init}}(L) \arrow[urr, bend left] \arrow[r, two heads] \arrow[drr, two heads, bend right] & \mathtt{reach}(\mathcal{A}) \arrow[ur, tail] \arrow[r, two heads] & \mathtt{obs}(\mathtt{reach}(\mathcal{A})) \arrow[r, tail] \arrow[d, dashed, no head] & \mathcal{A}_{\mathtt{final}}(L) \\
 & & \mathtt{Min}(L) \arrow[ur, tail, bend right] &
\end{tikzcd}
$$

Thus far we identified simple sufficient conditions on $\mathcal{C}$ so that minimization of $\mathcal{C}$-automata is guaranteed!

Lifting adjunctions between output categories to automata

Lifting adjunctions

Suppose we have the ‘same’ language interpretted in two different categories related by an adjunction $F \dashv U$:

$$L_C\colon A^* \to C(X, UY) \text{ and } L_D\colon A^* \to D(FX, Y).$$

$$\begin{tikzcd}
Auto(L_C) \arrow[rr, bend left] \arrow[rr, phantom, "\perp"] \arrow[d, "States"] & & Auto(L_D) \arrow[ll, bend left] \arrow[d, "States"] \\
C \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] & & D \arrow[ll, bend left, "U"]
\end{tikzcd}$$

Lifting adjunctions – determinization

Suppose we have the 'same' regular language interpretted in two different categories (Set and Rel) related by an adjunction $F \dashv U$:

$$L_{\mathrm{Set}}: A^* \to \mathrm{Set}(1, U1) \text{ and } L_{\mathrm{Rel}}: A^* \to \mathrm{Rel}(F1, 1).$$

$$\begin{tikzcd}
Auto(L_{\mathrm{Set}}) \arrow[rr, bend left] \arrow[rr, phantom, "\perp"] \arrow[d, "States"] & & Auto(L_{\mathrm{Rel}}) \arrow[ll, bend left] \arrow[d, "States"] \\
\mathrm{Set} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] & & \mathrm{Rel} \arrow[ll, bend left, "U"]
\end{tikzcd}$$

Corollary 1. The determinization of NFA is a right adjoint to inclusions of DFA in NFA.

Lifting adjunctions – determinization

Suppose we have the ‘same’ regular language interpretted in two different categories (Set and Rel) related by an adjunction $F \dashv U$:

$$L_{\mathrm{Set}} \colon A^* \to \mathrm{Set}(1, U1) \text{ and } L_{\mathrm{Rel}} \colon A^* \to \mathrm{Rel}(F1, 1).$$

$$\begin{tikzcd}
Auto(L_{\mathrm{Set}}) \arrow[rr, bend left] \arrow[d, "States"] \arrow[rr, phantom, "\perp"] & & Auto(L_{\mathrm{Rel}}) \arrow[ll, bend left] \arrow[d, "States"] \\
\mathrm{Set} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] & & \mathrm{Rel} \arrow[ll, bend left, "U"]
\end{tikzcd}$$

Corollary 1. The determinization of NFA is a right adjoint to inclusions of DFA in NFA.

Corollary 2. Initial automata for free in Kleisli valued automata.