

A categorical approach to automata learning and minimization – part 2

Daniela Petrişan

Université Paris Cité, IRIF, France

TACL'24, Barcelona, 24-28 June 2024



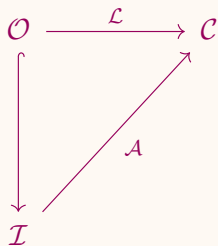
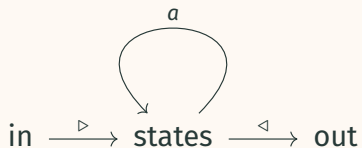
INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE



The setting: Automata and languages as functors

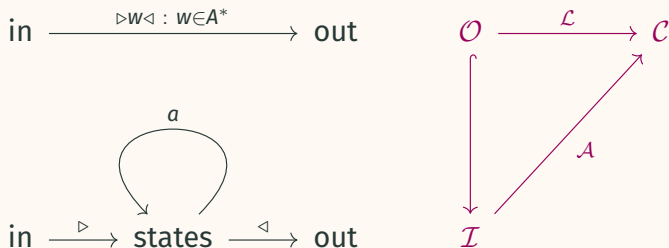
An automaton \mathcal{A} **accepts** a language \mathcal{L} when the next diagram commutes

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



The setting: Automata and languages as functors

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



For every language $\mathcal{L}: \mathcal{O} \rightarrow \mathcal{C}$ we consider a category **Auto** $_{\mathcal{L}}$ of automata accepting \mathcal{L} .

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

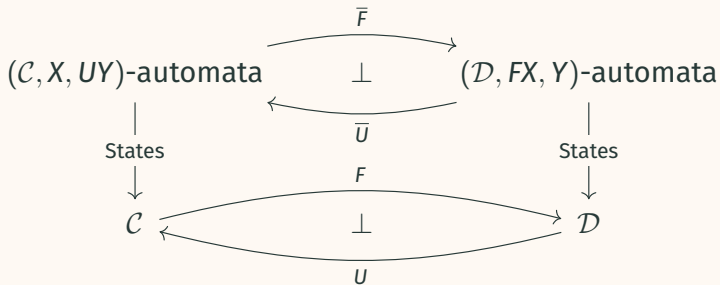
Much of the ensuing theory can be developed independently on the precise shape of \mathcal{I} .

A useful lemma

An adjunction $F \dashv U: \mathcal{C} \rightarrow \mathcal{D}$ lifts to an adjunction between functor categories $[\mathcal{I}, \mathcal{C}]$ and $[\mathcal{I}, \mathcal{D}]$.

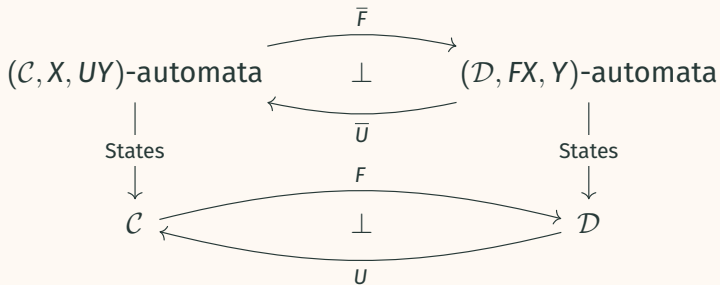
A useful lemma

An adjunction $F \dashv U: \mathcal{C} \rightarrow \mathcal{D}$ lifts to an adjunction between functor categories $[\mathcal{I}, \mathcal{C}]$ and $[\mathcal{I}, \mathcal{D}]$. We can refine this for any objects X in \mathcal{C} and Y in \mathcal{D} to a lifting:



A useful lemma

An adjunction $F \dashv U: \mathcal{C} \rightarrow \mathcal{D}$ lifts to an adjunction between functor categories $[\mathcal{I}, \mathcal{C}]$ and $[\mathcal{I}, \mathcal{D}]$. We can refine this for any objects X in \mathcal{C} and Y in \mathcal{D} to a lifting:



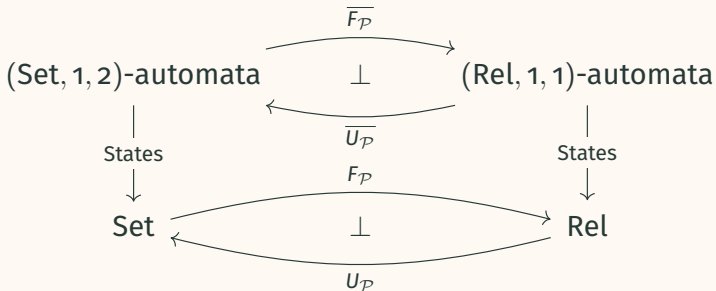
such that, furthermore, the lifted functors preserve the accepted languages up to isomorphism (since $\mathcal{C}(X, UY) \cong \mathcal{D}(FX, Y)$).

An instance: Determinization

The **powerset construction is a right adjoint** to the inclusion functor of deterministic automata into non-deterministic automata.

Recall the adjunction between Set and Rel. We have

$$U_{\mathcal{P}}(1) = \mathcal{P}(1) = 2 \text{ and } F_{\mathcal{P}}(1) = 1.$$

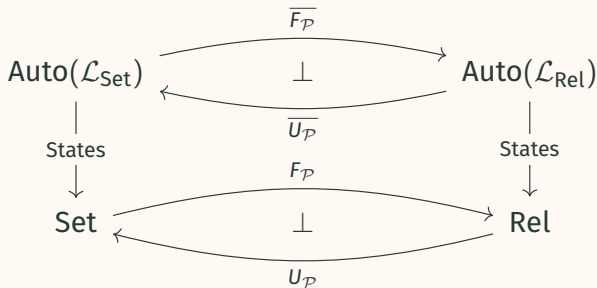


An instance: Determinization

The **powerset construction is a right adjoint** to the inclusion functor of deterministic automata into non-deterministic automata.

Recall the adjunction between Set and Rel. We have

$$U_{\mathcal{P}}(1) = \mathcal{P}(1) = 2 \text{ and } F_{\mathcal{P}}(1) = 1.$$



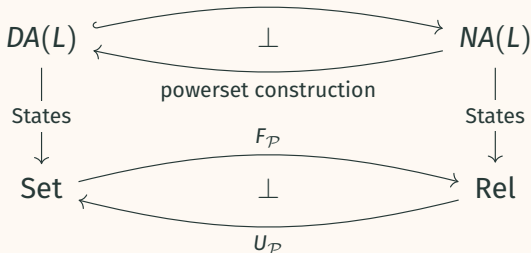
The same language L can be seen as a Set-valued functor \mathcal{L}_{Set} , and equivalently, as a Rel-valued functor \mathcal{L}_{Rel} .

An instance: Determinization

The **powerset construction is a right adjoint** to the inclusion functor of deterministic automata into non-deterministic automata.

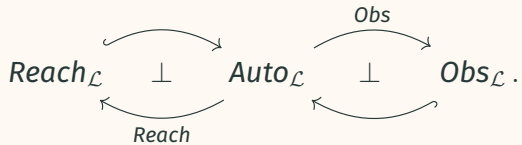
Recall the adjunction between Set and Rel. We have

$$U_{\mathcal{P}}(1) = \mathcal{P}(1) = 2 \text{ and } F_{\mathcal{P}}(1) = 1.$$



The same language L can be seen as a Set-valued functor \mathcal{L}_{Set} , and equivalently, as a Rel-valued functor \mathcal{L}_{Rel} .

Minimization via adjunctions



Brzowski's minimization algorithm

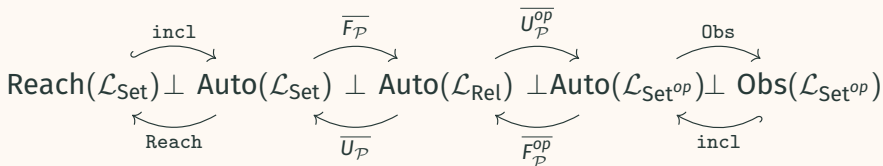
$$\min(\mathcal{A}) = \text{determinize}(\text{transpose}(\text{determinize}(\text{transpose}(\mathcal{A}))))),$$

where

- **determinize** applies a powerset construction to a non-deterministic automaton, and restricts to the reachable states, yielding a deterministic automaton, and
- **transpose** reverses all the edges of a non-deterministic automaton, and swaps the role of initial and final states (it accepts the mirrored language).

Brzowski's minimization algorithm

$$\min(\mathcal{A}) = \text{determinize}(\text{codeterminize}(\mathcal{A})),$$



Syntactic Monoids

Syntactic Monoid

Let L be a regular language over some finite alphabet A .

The **syntactic monoid** of L is the *minimal* monoid recognizing L .

Syntactic Monoid

Let L be a regular language over some finite alphabet A .

The **syntactic monoid** of L is the *minimal* monoid recognizing L .

The syntactic monoids via duality

Let $\mathcal{B}(L)$ denote the Boolean subalgebra of $\mathcal{P}(A^*)$ generated by the quotients of L , i.e. by the sets

$$w^{-1}Lv^{-1} = \{u \in A^* \mid wuv \in L\}$$

Theorem

The syntactic monoid of L is the **dual** of $\mathcal{B}(L)$.

Monoid and biaction recognizers

We are interested in

Monoid recognizers

A monoid morphism $\phi: A^* \rightarrow M$ and $F \subseteq M$.

Monoid and biaction recognizers

We are interested in

Monoid recognizers

A monoid morphism $\phi: A^* \rightarrow M$ and $F \subseteq M$.

However, we can easily work with **unary contexts**, so in fact we will represent as functors:

A^* -biaction recognizers

A biaction morphism $\phi: A^* \rightarrow X$ and $F \subseteq X$.

Monoid and biaction recognizers

We are interested in

Monoid recognizers

A monoid morphism $\phi: A^* \rightarrow M$ and $F \subseteq M$.

However, we can easily work with **unary contexts**, so in fact we will represent as functors:

A^* -biaction recognizers

A biaction morphism $\phi: A^* \rightarrow X$ and $F \subseteq X$.

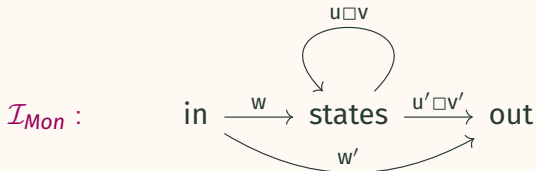
A monoid recognizer induces an A^* -biaction recognizer. Conversely ...

Lemma

Surjective A^ -biactions recognizers are in one-to-one correspondence with surjective monoid recognizers.*

We change the input category

We will represent A^* -bifunction recognizers as *Set*-valued functors from a different input category \mathcal{I}_{Mon}



A functor

$$\mathcal{A}: \mathcal{I}_{Mon} \rightarrow \text{Set}$$

is just an A^* -bifunction recognizer.

The three ingredients for minimization

- initial automaton
- final automaton
- factorization system

The three ingredients for minimization

- initial automaton ✓
 - final automaton
 - factorization system
-
- exists because Set is cocomplete
we can compute it as a colimit

The three ingredients for minimization

- initial automaton ✓
 - final automaton ✓
 - factorization system
-
- exists because Set is cocomplete
we can compute it as a colimit
 - exists because Set is complete
we can compute it as a limit

The three ingredients for minimization

- initial automaton ✓
- final automaton ✓
- factorization system ✓

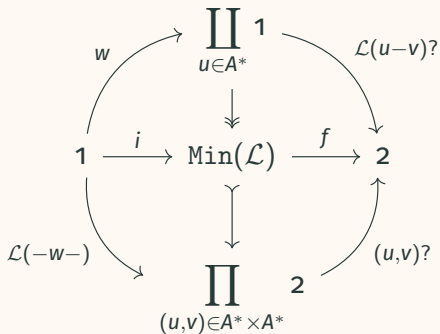
- exists because Set is cocomplete
we can compute it as a colimit
- exists because Set is complete
we can compute it as a limit
- lift the factorization system from Set

The syntactic monoid

Fact

The syntactic A^* -bivariation recognizer

is exactly the syntactic monoid of a given language \mathcal{L} .



The syntactic monoid

Fact

The syntactic A^* -bifunction recognizer

is exactly the syntactic monoid of a given language \mathcal{L} .

