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

Université de Paris

The setting: Automata and languages as functors

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

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

The setting: Automata and languages as functors

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

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

For every language $\mathcal{L} \colon \mathcal{O} \to \mathcal{C}$ we consider a category $\mathsf{Auto}_{\mathcal{L}}$ 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 $F \dashv U \colon \mathcal{C} \to \mathcal{D}$ lifts to an adjunction between functor categories $[\mathcal{I}, \mathcal{C}]$ and $[\mathcal{I}, \mathcal{D}]$.

A useful lemma

An adjunction $F \dashv U \colon \mathcal{C} \to \mathcal{D}$ 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:

$$\begin{tikzcd}
(\mathcal{C}, X, UY)\text{-automata} \arrow[rr, bend left, "\overline{F}"] \arrow[rr, phantom, "\perp"] \arrow[d, "\text{States}"] & & (\mathcal{D}, FX, Y)\text{-automata} \arrow[ll, bend left, "\overline{U}"] \arrow[d, "\text{States}"] \\
\mathcal{C} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] & & \mathcal{D} \arrow[ll, bend left, "U"]
\end{tikzcd}$$

A useful lemma

An adjunction $F \dashv U \colon \mathcal{C} \to \mathcal{D}$ 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:

$$
\begin{tikzcd}
(\mathcal{C}, X, UY)\text{-automata} \arrow[rr, bend left, "\overline{F}"] \arrow[rr, phantom, "\perp"] \arrow[d, "\mathrm{States}"] & & (\mathcal{D}, FX, Y)\text{-automata} \arrow[ll, bend left, "\overline{U}"] \arrow[d, "\mathrm{States}"] \\
\mathcal{C} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] & & \mathcal{D} \arrow[ll, bend left, "U"]
\end{tikzcd}
$$

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

$$
\begin{tikzcd}
(\mathsf{Set}, 1, 2)\text{-automata} \arrow[rr, bend left, "\overline{F_{\mathcal{P}}}"] \arrow[rr, phantom, "\perp"] \arrow[d, "\mathsf{States}"'] & & (\mathsf{Rel}, 1, 1)\text{-automata} \arrow[ll, bend left, "\overline{U_{\mathcal{P}}}"] \arrow[d, "\mathsf{States}"] \\
\mathsf{Set} \arrow[rr, bend left, "F_{\mathcal{P}}"] \arrow[rr, phantom, "\perp"] & & \mathsf{Rel} \arrow[ll, bend left, "U_{\mathcal{P}}"]
\end{tikzcd}
$$

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

$$
\begin{tikzcd}
\mathsf{Auto}(\mathcal{L}_{\mathsf{Set}}) \arrow[rr, bend left, "\overline{F_{\mathcal{P}}}"] \arrow[rr, phantom, "\perp"] \arrow[d, "\mathsf{States}"] & & \mathsf{Auto}(\mathcal{L}_{\mathsf{Rel}}) \arrow[ll, bend left, "\overline{U_{\mathcal{P}}}"] \arrow[d, "\mathsf{States}"] \\
\mathsf{Set} \arrow[rr, bend left, "F_{\mathcal{P}}"] \arrow[rr, phantom, "\perp"] & & \mathsf{Rel} \arrow[ll, bend left, "U_{\mathcal{P}}"]
\end{tikzcd}
$$

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

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

$$
\begin{tikzcd}[column sep=huge, row sep=large]
DA(L) \arrow[r, hook, bend left] \arrow[r, phantom, "\perp"] \arrow[d, "\text{States}" description] & NA(L) \arrow[l, bend left, "\text{powerset construction}"] \arrow[d, "\text{States}" description] \\
\mathsf{Set} \arrow[r, bend left, "F_{\mathcal{P}}"] \arrow[r, phantom, "\perp"] & \mathsf{Rel} \arrow[l, bend left, "U_{\mathcal{P}}"]
\end{tikzcd}
$$

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

Minimization via adjunctions

$$
\begin{tikzcd}
\mathit{Reach}_{\mathcal{L}} \arrow[rr, hook, bend left] \arrow[rr, phantom, "\perp"] & & \mathit{Auto}_{\mathcal{L}} \arrow[ll, bend left, "\mathit{Reach}"] \arrow[rr, bend left, "\mathit{Obs}"] \arrow[rr, phantom, "\perp"] & & \mathit{Obs}_{\mathcal{L}} \arrow[ll, hook, bend left]
\end{tikzcd}
.
$$

Brzozowski's minimization algorithm

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

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).

Brzozowski's minimization algorithm

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

$$\begin{tikzcd}
\mathsf{Reach}(\mathcal{L}_{\mathsf{Set}}) \arrow[r, hook, bend left, "\texttt{incl}"] \arrow[r, phantom, "\perp"] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Set}}) \arrow[l, bend left, "\texttt{Reach}"] \arrow[r, bend left, "\overline{F_{\mathcal{P}}}"] \arrow[r, phantom, "\perp"] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Rel}}) \arrow[l, bend left, "\overline{U_{\mathcal{P}}}"] \arrow[r, bend left, "\overline{U^{op}_{\mathcal{P}}}"] \arrow[r, phantom, "\perp"] & \mathsf{Auto}(\mathcal{L}_{\mathsf{Set}^{op}}) \arrow[l, bend left, "\overline{F^{op}_{\mathcal{P}}}"] \arrow[r, bend left, "\texttt{Obs}"] \arrow[r, phantom, "\perp"] & \mathsf{Obs}(\mathcal{L}_{\mathsf{Set}^{op}}) \arrow[l, hook, bend left, "\texttt{incl}"]
\end{tikzcd}$$

Syntactic Monoids

Syntactic Monoid

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

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

Syntactic Monoid

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

The synatctic 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 $\phi\colon A^* \to M$ and $F \subseteq M$.

Monoid and biaction recognizers

We are interested in

Monoid recognizers
A monoid morphism $\phi\colon A^* \to M$ 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 $\phi\colon A^* \to X$ and $F \subseteq X$.

Monoid and biaction recognizers

We are interested in

Monoid recognizers
A monoid morphism $\phi\colon A^* \to M$ 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 $\phi\colon A^* \to X$ 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^*-biaction recognizers as *Set*-valued functors from a different input category $\mathcal{I}_{Mon}$

$$\mathcal{I}_{Mon}: \quad \begin{tikzcd} \mathrm{in} \arrow[r, "\mathrm{w}"] \arrow[rr, bend right, "\mathrm{w}'"'] & \mathrm{states} \arrow[loop above, "\mathrm{u}\Box\mathrm{v}"] \arrow[r, "\mathrm{u}'\Box\mathrm{v}'"] & \mathrm{out} \end{tikzcd}$$

A functor

$$\mathcal{A}\colon \mathcal{I}_{Mon} \to \mathbf{Set}$$

is just an A^*-biaction 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^*-biaction recognizer
is exactly the syntactic monoid of a given language $\mathcal{L}$.

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

The syntactic monoid

Fact

The syntactic A^*-biaction recognizer
is exactly the syntactic monoid of a given language $\mathcal{L}$.

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