A categorical approach to automata learning and minimization

Daniela Petrişan

Université Paris Cité, IRIF, France

EPIT’25, Aussois, 20 May 2025

INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE

Université de Paris

Some references

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

T. Colcombet, D. Petrisan, R. Stabile, *Learning Automata and Transducers: A Categorical Approach.* CSL 2021

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

Further reading:

Q. Aristote, S. van Gool, D. Petrişan, M. Shirmohammadi, *Learning Weighted Automata over Number Rings, Concretely and Categorically* LICS 2025

`https://arxiv.org/pdf/2504.16596`

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.

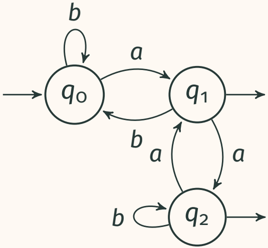
In particular, we will see how the category-theoretic approach

- 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 with effects

Complete Deterministic Finite Automata

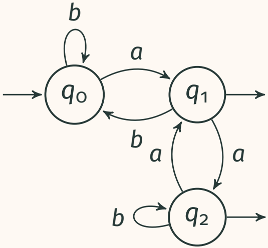
Let’s rewrite the definition of a complete DFA…



$$\begin{tikzcd}
1 \arrow[r, "q_0"] & \{q_0, q_1, q_2\} \arrow[loop above, "\delta_a"] \arrow[loop below, "\delta_b"] \arrow[r, "\chi_{q_1,q_2}"] & 2
\end{tikzcd}$$

Complete Deterministic Finite Automata

Let’s rewrite the definition of a complete DFA...



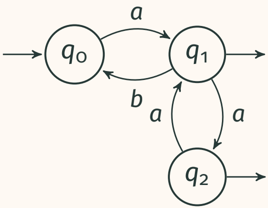
$$\begin{tikzcd}
1 \arrow[r, "q_0"] & \{q_0, q_1, q_2\} \arrow[loop above, "\delta_a"] \arrow[loop below, "\delta_b"'] \arrow[r, "\chi_{q_1,q_2}"] & 2
\end{tikzcd}$$

To give a complete DFA over A amounts to give a set Q and the functions depicted below.

$$\begin{tikzcd}
1 \arrow[r, "q_0"] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)"] \arrow[r, "\chi_F"] & 2
\end{tikzcd}$$

Deterministic Finite Automata

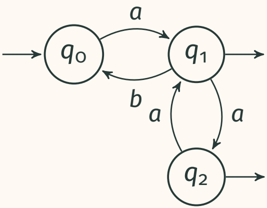
Let’s rewrite the definition of a DFA...



$$\begin{tikzcd} 1 \arrow[r, "q_0", "\shortmid" marking] & \{q_0, q_1, q_2\} \arrow[loop above, "\delta_a", "\shortmid" marking] \arrow[loop below, "\delta_b", "\shortmid" marking] \arrow[r, "\chi_{q_1,q_2}", "\shortmid" marking] & 1 \end{tikzcd}$$

Deterministic Finite Automata

Let’s rewrite the definition of a DFA...



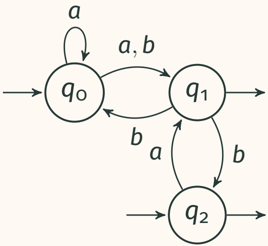
$$\begin{tikzcd}
1 \arrow[r, "q_0", "/" marking] & \{q_0, q_1, q_2\} \arrow[loop above, "\delta_a", "/" marking] \arrow[loop below, "\delta_b", "/" marking] \arrow[r, "\chi_{q_1,q_2}", "/" marking] & 1
\end{tikzcd}$$

To give a DFA over A amounts to give a set Q and the **partial functions** depicted below.

$$\begin{tikzcd}
1 \arrow[r, "q_0", "/" marking] & Q \arrow[loop above, "\delta_\sigma (\sigma \in A)", "/" marking] \arrow[r, "\chi_F", "/" marking] & 1
\end{tikzcd}$$

Nondeterministic Finite Automata

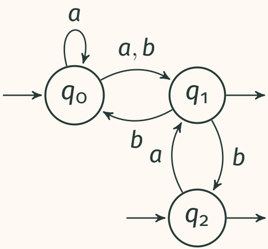
Let's rewrite the definition of an NFA...



$$\begin{tikzcd}
1 \arrow[r, "q_0"] & \{q_0, q_1, q_2\} \arrow[loop above, "\delta_a"] \arrow[loop below, "\delta_b"] \arrow[r, "\chi_{q_1, q_2}"] & 1
\end{tikzcd}$$

Nondeterministic Finite Automata

Let’s rewrite the definition of an NFA…

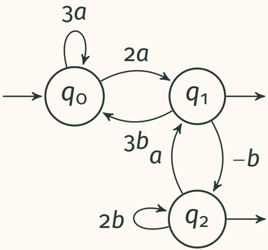


$$\begin{tikzcd}
1 \arrow[r, "q_0", "\bullet"{marking}] & \{q_0, q_1, q_2\} \arrow[loop above, "\delta_a", "\bullet"{marking}] \arrow[loop below, "\delta_b"', "\bullet"{marking}] \arrow[r, "\chi_{q_1,q_2}", "\bullet"{marking}] & 1
\end{tikzcd}$$

To give an NFA amounts to give a set Q and the **relations** depicted below.

$$\begin{tikzcd}
1 \arrow[r, "I", "\bullet"{marking}] & Q \arrow[loop above, "\delta_\sigma\,(\sigma \in A)", "\bullet"{marking}] \arrow[r, "\chi_F", "\bullet"{marking}] & 1
\end{tikzcd}$$

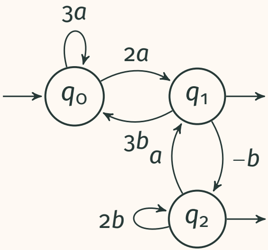
Weighted automata over a semiring



$$\begin{tikzcd} R \arrow[r, "i"] & R^{\{q_0,q_1,q_2\}} \arrow[r, "fin"] \arrow[loop above, "\delta_a"] \arrow[loop below, "\delta_b"'] & R \end{tikzcd}$$

e.g., $\delta_a(q_0) = 3q_0 + 2q_1$,
$fin(q_1) = fin(q_2) = 1$, etc...

Weighted automata over a semiring



$$\begin{tikzcd} R \arrow[r, "i"] & R^{\{q_0,q_1,q_2\}} \arrow[loop above, "\delta_a"] \arrow[loop below, "\delta_b"] \arrow[r, "fin"] & R \end{tikzcd}$$

e.g., $\delta_a(q_0) = 3q_0 + 2q_1$,
$fin(q_1) = fin(q_2) = 1$, etc...

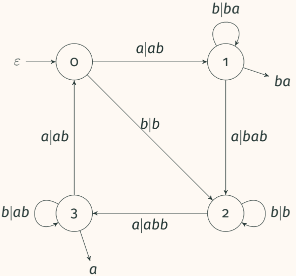
To give a WA over R amounts to give a free module R^Q and the **linear maps** depicted below.

$$\begin{tikzcd} R \arrow[r, "i"] & R^Q \arrow[loop above, "\delta_\sigma (\sigma \in A)"] \arrow[r, "f"] & R \end{tikzcd}$$

Sequential transducers

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

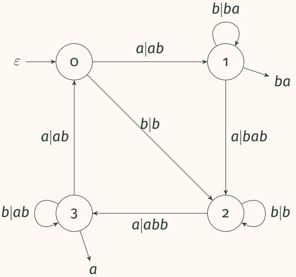
- a finite set of states Q
- an initial state with an initial output in B^*, or an undefined initial state



Sequential transducers

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

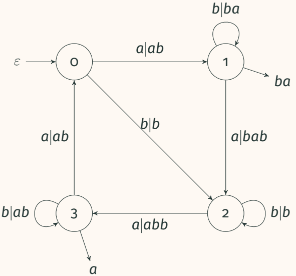
- a finite set of states Q
- an initial state with an initial output in B^*, or an undefined initial state
- for each $a \in A$ a transition function $Q \to B^* \times Q + 1$



Sequential transducers

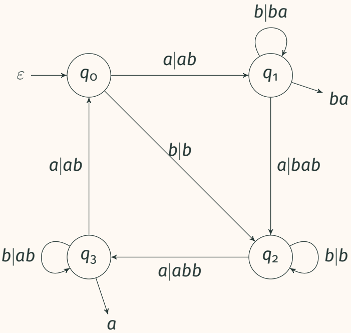
A sequential transducer with input alphabet A and output alphabet B consists of:

- 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.



Sequential Transducers

Let's rewrite the definition of a sequential transducer with input alphabet $\{a, b\}$ and output alphabet $B = \{a, b\}$.

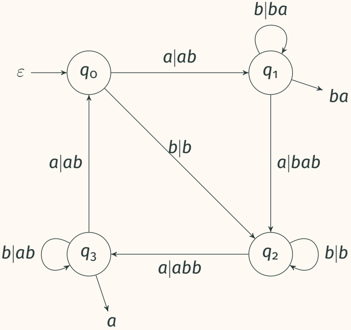


$$\begin{tikzcd}
1 \arrow[r, "q_0", "\blacklozenge" marking] & \{q_0, q_1, q_2, q_3\} \arrow[loop above, "\delta_a", "\blacklozenge" marking] \arrow[loop below, "\delta_b"', "\blacklozenge" marking] \arrow[r, "\chi_{q_1,q_2}", "\blacklozenge" marking] & 1
\end{tikzcd}$$

Note that an arrow $X \xrightarrow{\blacklozenge} Y$ is actually a function $X \to 1 + B^* \times Y$.

Sequential Transducers

Let's rewrite the definition of a sequential transducer with input alphabet $\{a, b\}$ and output alphabet $B = \{a, b\}$.



$$\begin{tikzcd} 1 \arrow[r, "q_0"] & \{q_0, q_1, q_2, q_3\} \arrow[r, "\chi_{q_1,q_2}"] \arrow[loop above, "\delta_a"] \arrow[loop below, "\delta_b"] & 1 \end{tikzcd}$$

Note that an arrow $X \longrightarrow Y$ is actually a function $X \to 1 + B^* \times Y$.

To give a seq. transducer amounts to give a set Q and arrows:

$$\begin{tikzcd} 1 \arrow[r, "q_0"] & Q \arrow[r, "\chi_F"] \arrow[loop above, "\delta_\sigma (\sigma \in A)"] & 1 \end{tikzcd}$$

Word automata

$$\begin{tikzcd}
1 \arrow[r, "q_0"] & Q \arrow[r, "\chi_F"] \arrow[loop above, "\delta_\sigma\,(\sigma\in A)"] & 2
\end{tikzcd}$$

complete DFAs — Set (sets and functions)

Word automata

$$\begin{tikzcd}
1 \arrow[r, "q_0"] & Q \arrow[r, "\chi_F"] \arrow[loop above, "\delta_\sigma\,(\sigma\in A)"] & 2
\end{tikzcd}$$

complete DFAs — Set (sets and functions)

$$\begin{tikzcd}
1 \arrow[r, "q_0", "/" marking] & Q \arrow[r, "\chi_F", "/" marking] \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "/" marking] & 1
\end{tikzcd}$$

DFAs — Set$_\bullet$ (sets and partial functions)

Word automata

$$\begin{tikzcd}
1 \arrow[r, "q_0"] & Q \arrow[r, "\chi_F"] \arrow[loop above, "\delta_\sigma\,(\sigma\in A)"] & 2
\end{tikzcd}$$

complete DFAs — Set (sets and functions)

$$\begin{tikzcd}
1 \arrow[r, "q_0", "/" marking] & Q \arrow[r, "\chi_F", "/" marking] \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "/" marking] & 1
\end{tikzcd}$$

DFAs — $\mathrm{Set}_\bullet$ (sets and partial functions)

$$\begin{tikzcd}
1 \arrow[r, "I", "\bullet" marking] & Q \arrow[r, "\chi_F", "\bullet" marking] \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\bullet" marking] & 1
\end{tikzcd}$$

NFA — Rel (sets and relations)

Word automata

$$\begin{tikzcd}
1 \arrow[r, "q_0"] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)"] \arrow[r, "\chi_F"] & 2
\end{tikzcd}$$

complete DFAs — Set (sets and functions)

$$\begin{tikzcd}
1 \arrow[r, "q_0", "/" marking] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "/" marking] \arrow[r, "\chi_F", "/" marking] & 1
\end{tikzcd}$$

DFAs — $\mathsf{Set}_\bullet$ (sets and partial functions)

$$\begin{tikzcd}
1 \arrow[r, "I", "\bullet" marking] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\bullet" marking] \arrow[r, "\chi_F", "\bullet" marking] & 1
\end{tikzcd}$$

NFA — Rel (sets and relations)

$$\begin{tikzcd}
R \arrow[r, "i", "\circ" marking] & R^Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\circ" marking] \arrow[r, "f", "\circ" marking] & R
\end{tikzcd}$$

WAs over R — $\mathsf{FreeMod}_R$ (R-modules and linear maps)

Word automata

$$\begin{tikzcd}
1 \arrow[r, "q_0"] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)"] \arrow[r, "\chi_F"] & 2
\end{tikzcd}$$

complete DFAs — Set (sets and functions)

$$\begin{tikzcd}
1 \arrow[r, "q_0", "/" marking] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "/" marking] \arrow[r, "\chi_F", "/" marking] & 1
\end{tikzcd}$$

DFAs — $\text{Set}_\bullet$ (sets and partial functions)

$$\begin{tikzcd}
1 \arrow[r, "I", "\bullet" marking] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\bullet" marking] \arrow[r, "\chi_F", "\bullet" marking] & 1
\end{tikzcd}$$

NFA — Rel (sets and relations)

$$\begin{tikzcd}
R \arrow[r, "i", "\circ" marking] & R^Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\circ" marking] \arrow[r, "f", "\circ" marking] & R
\end{tikzcd}$$

WAs over R — FreeMod_R (R-modules and linear maps)

$$\begin{tikzcd}
1 \arrow[r, "q_0", "\blacklozenge" marking] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\blacklozenge" marking] \arrow[r, "\chi_F", "\blacklozenge" marking] & 1
\end{tikzcd}$$

Sequential transducers — ?

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: $f\colon X \longrightarrow\!\!\!\!\blacklozenge\!\!\!\rightarrow Y$, where $f\colon X \to B^* \times Y + 1$ 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 $X, Y, Z, \dots$
- arrows: $f\colon X \relbar\joinrel\blacklozenge\joinrel\rightarrow Y$, where $f\colon X \to B^* \times Y + 1$ is a function

How to compose $f\colon X \relbar\joinrel\blacklozenge\joinrel\rightarrow Y$ and $g\colon Y \relbar\joinrel\blacklozenge\joinrel\rightarrow Z$?

$g \circ f\colon X \relbar\joinrel\blacklozenge\joinrel\rightarrow Z$ (i.e. $g \circ f\colon X \to B^* \times Z + 1$) is given by

$$g \circ f(x) = \begin{cases} (uv, z) & \text{if } f(x) = (u, y) \text{ and } g(y) = (v, z) \\ \perp & \text{otherwise.} \end{cases}$$

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:** $f\colon X \mathrel{-\!\!\blacklozenge\!\!\rightarrow} Y$, where $f\colon X \to B^* \times Y + 1$ is a function

How to compose $f\colon X \mathrel{-\!\!\blacklozenge\!\!\rightarrow} Y$ and $g\colon Y \mathrel{-\!\!\blacklozenge\!\!\rightarrow} Z$?

$g \circ f\colon X \mathrel{-\!\!\blacklozenge\!\!\rightarrow} Z$ (i.e. $g \circ f\colon X \to B^* \times Z + 1$) is given by

$$g \circ f(x) = \begin{cases} (uv, z) & \text{if } f(x) = (u, y) \text{ and } g(y) = (v, z) \\ \bot & \text{otherwise.} \end{cases}$$

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.

Word automata

$$\begin{tikzcd}
1 \arrow[r, "q_0"] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)"] \arrow[r, "\chi_F"] & 2
\end{tikzcd}$$

complete DFAs — Set (sets and functions)

$$\begin{tikzcd}
1 \arrow[r, "q_0", "/"{marking}] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "/"{marking}] \arrow[r, "\chi_F", "/"{marking}] & 1
\end{tikzcd}$$

DFAs — Set$_\bullet$ (sets and partial functions)

$$\begin{tikzcd}
1 \arrow[r, "q_0", "\bullet"{marking}] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\bullet"{marking}] \arrow[r, "\chi_F", "\bullet"{marking}] & 1
\end{tikzcd}$$

NFA — Rel (sets and relations)

$$\begin{tikzcd}
R \arrow[r, "i", "\circ"{marking}] & R^Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\circ"{marking}] \arrow[r, "f", "\circ"{marking}] & R
\end{tikzcd}$$

WAs over R — FreeMod$_R$ (R-modules and linear maps)

$$\begin{tikzcd}
1 \arrow[r, "q_0", "\blacklozenge"{marking}] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)", "\blacklozenge"{marking}] \arrow[r, "\chi_F", "\blacklozenge"{marking}] & 1
\end{tikzcd}$$

Sequential transducers — $\mathcal{T}$

$$\begin{tikzcd}
I \arrow[r, "i"] & Q \arrow[loop above, "\delta_\sigma\,(\sigma\in A)"] \arrow[r, "f"] & O
\end{tikzcd}$$

$(\mathcal{C}, I, O)$-automata — $\mathcal{C}$

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

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

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 $w=a_1\dots a_n$ the composite morphism

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

Automata and languages as functors

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

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

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

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 $\mathcal{I}$.

The output categories we have seen so far

What do these categories have in common ?

- Set – the category of sets and functions
- $\mathbf{Set}_\bullet$ – the category of sets and partial 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 output categories we have seen so far

What do these categories have in common ?

- Set – the category of sets and functions
- Set$_\bullet$ – the category of sets and partial 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

Answer. They are categories of free algebras (aka Kleisli categories) for monads specifying some effect:

- the identity monad
- the Maybe monad (aka option)
- the powerset monad – non-determinism
- the monad of partial free actions of B^*.

Changing output categories

Adjunctions – Recap

Having an adjunction

$$\begin{tikzcd} C \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & D \arrow[l, bend left, "U"] \end{tikzcd}$$

means we have isomorphisms $C(X, UY) \cong D(FX, Y)$ natural in both X and Y.

$f\colon FX \to Y$ yields $f_\flat\colon X \to UY$

$g\colon X \to UY$ yields $g^\sharp\colon FX \to Y$

Adjunctions – example 1

Exercise. Describe an adjunction between Set and $\mathrm{Set}_\bullet$.

$$\begin{tikzcd}
\mathrm{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathrm{Set}_\bullet \arrow[l, bend left, "U"]
\end{tikzcd}$$

means we have isomorphisms $Set(X, UY) \cong \mathrm{Set}_\bullet(FX, Y)$ natural in both X and Y.

$$f\colon FX \not\longrightarrow Y \quad \text{in } \mathrm{Set}_\bullet \text{ yields } f_\flat\colon X \to UY \text{ in Set}$$

$$g\colon X \to UY \text{ in Set yields } \quad g^\sharp\colon FX \not\longrightarrow Y \quad \text{in } \mathrm{Set}_\bullet$$

Adjunctions – example 1

Exercise. Describe an adjunction between Set and $\mathsf{Set}_\bullet$.

$$\begin{tikzcd}
\mathsf{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathsf{Set}_\bullet \arrow[l, bend left, "U"]
\end{tikzcd}$$

means we have isomorphisms $Set(X, UY) \cong \mathsf{Set}_\bullet(FX, Y)$ natural in both X and Y.

Answer. $FX = X$, $UX = 1 + X$...

$$f{:}X \mathrel{\ooalign{$\longrightarrow$\cr\hfil$/$\hfil}} Y \quad \text{in } \mathsf{Set}_\bullet \text{ yields } f_\flat{:}X \to 1 + Y \text{ in Set}$$

$$g{:}X \to 1 + Y \text{ in Set yields } \quad g^\sharp{:}X \mathrel{\ooalign{$\longrightarrow$\cr\hfil$/$\hfil}} Y \quad \text{in } \mathsf{Set}_\bullet$$

Adjunctions – example 2

Exercise. Describe an adjunction between Set and Rel.

$$\begin{tikzcd}
\mathrm{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathrm{Rel} \arrow[l, bend left, "U"]
\end{tikzcd}$$

means we have isomorphisms $Set(X, UY) \cong \mathrm{Rel}(FX, Y)$ natural in both X and Y.

$$f\colon FX \longrightarrow\!\!\bullet\!\!\longrightarrow Y \quad \text{in Rel yields } f_\flat\colon X \to UY \text{ in Set}$$

$$g\colon X \to UY \text{ in Set yields } \quad g^\sharp\colon FX \longrightarrow\!\!\bullet\!\!\longrightarrow Y \quad \text{in Rel}$$

Adjunctions – example 2

Exercise. Describe an adjunction between Set and Rel.

$$\begin{tikzcd}
\mathrm{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathrm{Rel} \arrow[l, bend left, "U"]
\end{tikzcd}$$

means we have isomorphisms $Set(X, UY) \cong \mathrm{Rel}(FX, Y)$ natural in both X and Y.

Answer. $FX = X$, $UX = \mathcal{P}X$...

$$f : X \longrightarrow\!\!\bullet\!\!\longrightarrow Y \quad \text{in Rel yields } f_\flat : X \to \mathcal{P}Y \text{ in Set}$$

$$g : X \to \mathcal{P}Y \text{ in Set yields} \quad g^\sharp : X \longrightarrow\!\!\bullet\!\!\longrightarrow Y \quad \text{in Rel}$$

Adjunctions – example 3

The above adjunctions are all standard in category theory:
They are adjunctions between Set and the Kleisli category for a monad.

$$\begin{tikzcd}
\mathrm{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathrm{Kl}(T) \arrow[l, bend left, "U"]
\end{tikzcd}$$

with F identity on objects and $UX = TX$.

Getting rid of effects – or lifting adjunctions

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

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

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

Lifting adjunctions – completing DFAs

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

$$L_{\mathrm{Set}} : A^* \to \mathrm{Set}(1,2) \text{ and } L_{\mathrm{Set}_\bullet} : A^* \to \mathrm{Set}_\bullet(1,1).$$

$$\begin{tikzcd}
\mathit{Auto}(L_{\mathrm{Set}}) \arrow[r, bend left, "\mathit{inclusion}"] \arrow[r, phantom, "\perp"] \arrow[d, "\mathit{States}" description] & \mathit{Auto}(L_{\mathrm{Set}_\bullet}) \arrow[l, bend left, "\mathit{completion}"] \arrow[d, "\mathit{States}" description] \\
\mathrm{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathrm{Set}_\bullet \arrow[l, bend left, "U"]
\end{tikzcd}$$

Corollary 1. The completion of a DFA is a right adjoint to inclusions of complete DFA in DFA.

Lifting adjunctions – determinization

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

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

$$\begin{tikzcd}
\mathit{Auto}(L_{\mathsf{Set}}) \arrow[r, bend left] \arrow[r, phantom, "\perp"] \arrow[d, "\mathit{States}" description] & \mathit{Auto}(L_{\mathsf{Rel}}) \arrow[l, bend left] \arrow[d, "\mathit{States}" description] \\
\mathsf{Set} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathsf{Rel} \arrow[l, bend left, "U"]
\end{tikzcd}$$

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 $F \dashv U$:

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

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

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.

Automata in a category: minimization

DFA 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

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

DFA 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.

DFA 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 $u^{-1}L$.

DFA 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
- $\delta_a(u^{-1}L) = (ua)^{-1}L$.

DFA 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 Reach$(\mathcal{A})$.
- Merge all states that accept the same language, we obtain the observable quotient Obs(Reach$(\mathcal{A})$).

DFA 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.

Minimization 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?

Minimization 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.

Minimization 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»

Minimization 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?

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

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?

$$\begin{tikzcd}
\mathcal{O} \arrow[rr, "\mathcal{L}"] \arrow[dddd, hook] & & \mathcal{C} \\
& \mathcal{A}_{\texttt{init}}(\mathcal{L}) \arrow[ur] & \\
& \texttt{Min}(\mathcal{L}) \arrow[uur, dashed] & \\
& \mathcal{A}_{\texttt{final}}(\mathcal{L}) \arrow[uuur] & \\
\mathcal{I} \arrow[uuur] \arrow[uur, dashed] \arrow[ur] & &
\end{tikzcd}$$

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,
- a factorization system

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

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

The three ingredients for minimization

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

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

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

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

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

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

Factorization systems

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

$$\begin{tikzcd} X \arrow[r, two heads, "e"] & Z \arrow[r, tail, "m"] & Y \end{tikzcd}$$

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 $f : X \to Y$ can we written as a composite

$$X \xrightarrow{e} \mathrel{\mkern-14mu}\twoheadrightarrow Z \rightarrowtail\mathrel{\mkern-14mu}\xrightarrow{m} Y$$

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

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

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;

Factorization systems

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

$$\begin{tikzcd} X \arrow[r, two heads, "e"] & Z \arrow[r, tail, "m"] & Y \end{tikzcd}$$

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 $f\colon X \to Y$ can we written as a composite $m \circ e$ 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

$$X \xrightarrow{e} \mathrel{\mkern-14mu}\twoheadrightarrow Z \rightarrowtail \mathrel{\mkern-14mu}\xrightarrow{m} Y$$

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:

- E and M contain the isomorphisms and are closed under composition;
- every morphism $f : X \to Y$ can we written as a composite $m \circ e$ 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 $\mathcal{A}_{\mathtt{init}}(\mathcal{L})$,
- 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?

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 :

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

Trivial example: minimizing DFAs

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

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

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

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

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 $w \mapsto w^{-1}L$.

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

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

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

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

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

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

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

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

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

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

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

Minimal Automaton $\texttt{Min}(\mathcal{L})$ for a Language

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

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

Minimal Automaton $\texttt{Min}(\mathcal{L})$ for a Language

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

$$\begin{tikzcd}
 & & \mathcal{A} \arrow[rrd, bend left] & & \\
\mathcal{A}_{\texttt{init}}(L) \arrow[rru, bend left] \arrow[r, two heads] \arrow[rrd, two heads, bend right] & \texttt{reach}(\mathcal{A}) \arrow[ru, tail] \arrow[rr, two heads] & & \texttt{obs}(\texttt{reach}(\mathcal{A})) \arrow[r, tail] & \mathcal{A}_{\texttt{final}}(L) \\
 & & \texttt{Min}(L) \arrow[rru, tail, bend right] & &
\end{tikzcd}$$

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

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

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

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

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

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

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

Learning

The L^*-algorithm

- **Goal:** learn a regular language of words L.

The L^*-algorithm

- **Goal:** learn a regular language of words L.
- The algorithm interacts with a teacher who knows L by asking two types of queries:

The L^*-algorithm

- **Goal:** learn a regular language of words L.
- The algorithm interacts with a teacher who knows L by asking two types of queries:
  1. Membership queries: Does a word belong to the language ?

The L^*-algorithm

- **Goal:** learn a regular language of words L.
- The algorithm interacts with a teacher who knows L by asking two types of queries:
  1. Membership queries: Does a word belong to the language ?
  2. Equivalence queries: It gives the teacher a hypothesis automaton and asks whether it accepts the language L.

The L^*-algorithm

- **Goal:** learn a regular language of words L.
- The algorithm interacts with a teacher who knows L by asking two types of queries:
  1. Membership queries: Does a word belong to the language ?
  2. Equivalence queries: It gives the teacher a hypothesis automaton and asks whether it accepts the language L. If no, the teacher provides a counter-example.

The L^*-algorithm

- **Goal:** learn a regular language of words L.
- The algorithm interacts with a teacher who knows L by asking two types of queries:
  1. Membership queries: Does a word belong to the language ?
  2. Equivalence queries: It gives the teacher a hypothesis automaton and asks whether it accepts the language L. If no, the teacher provides a counter-example.
- The algorithm stops when the teacher agrees that the hypothesis automaton accepts the language L.

The L^*-algorithm: some definitions

- At each step, we maintain a pair of sets of words (Q, T), starting with $(\{\epsilon\}, \{\epsilon\})$.
    - Q —> potential states for the hypothesis automaton
    - T —> test words used to define an equivalence relation coarser than the Myhill-Nerode equivalence.

The L^*-algorithm: some definitions

- At each step, we maintain a pair of sets of words (Q, T), starting with $(\{\epsilon\}, \{\epsilon\})$.
  - Q —> potential states for the hypothesis automaton
  - T —> test words used to define an equivalence relation coarser than the Myhill-Nerode equivalence.
- the T-equivalence relation: $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$

The L^*-algorithm: some definitions

- At each step, we maintain a pair of sets of words (Q, T), starting with $(\{\epsilon\}, \{\epsilon\})$.
    - Q —> potential states for the hypothesis automaton
    - T —> test words used to define an equivalence relation coarser than the Myhill-Nerode equivalence.
- the T-equivalence relation: $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$
- closedness : $\forall q \in Q . \forall a \in A . \exists p \in Q . \; p \sim_T qa$.

The L^*-algorithm: some definitions

- At each step, we maintain a pair of sets of words (Q, T), starting with $(\{\epsilon\}, \{\epsilon\})$.
  - Q —> potential states for the hypothesis automaton
  - T —> test words used to define an equivalence relation coarser than the Myhill-Nerode equivalence.
- the T-equivalence relation: $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$
- closedness : $\forall q \in Q . \forall a \in A . \exists p \in Q . \ p \sim_T qa$.
- consistency : $\forall q, q' \in Q . \forall a \in A . \ q \sim_T q' \Rightarrow qa \sim_T q'a$

The L^*-algorithm: some definitions

- At each step, we maintain a pair of sets of words (Q, T), starting with $(\{\epsilon\}, \{\epsilon\})$.
  - Q —> potential states for the hypothesis automaton
  - T —> test words used to define an equivalence relation coarser than the Myhill-Nerode equivalence.
- the T-equivalence relation: $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$
- closedness : $\forall q \in Q\,.\forall a \in A\,.\exists p \in Q\,.\; p \sim_T qa.$
- consistency : $\forall q, q' \in Q\,.\forall a \in A\,.\;\; q \sim_T q' \Rightarrow qa \sim_T q'a$
- When (Q, T) is closed and consistent it is possible to build a hypothesis automaton $\mathcal{H}(Q, T)$

L^*-revisited

- At the (Q,T) stage of the algorithm the learner only has access to a fragment of the language:

$$L_{Q,T} : QAT \cup QT \rightarrowtail A^* \xrightarrow{L} 2$$

L^*-revisited

- At the (Q, T) stage of the algorithm the learner only has access to a fragment of the language:

$$L_{Q,T} : QAT \cup QT \rightarrowtail A^* \xrightarrow{L} 2$$

- This can be represented by a notion of (Q, T)-biautomaton

$$\begin{tikzcd}
1 \arrow[r, "\triangleright q", "(q \in Q)"'] & Q_1 \arrow[r, shift left, "a \ (a \in A)"] \arrow[r, shift right, "\varepsilon"'] & Q_2 \arrow[r, "t \triangleleft", "(t \in T)"'] & 2
\end{tikzcd}$$

such that the following coherence diagrams commute

$$\begin{tikzcd}
 & Q_1 \arrow[dr, "a"] & \\
1 \arrow[ur, "\triangleright q"] \arrow[dr, "\triangleright qa"'] & & Q_2 \\
 & Q_1 \arrow[ur, "\varepsilon"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & Q_2 \arrow[dr, "t \triangleleft"] & \\
Q_1 \arrow[ur, "a"] \arrow[dr, "\varepsilon"'] & & 2 \\
 & Q_2 \arrow[ur, "at \triangleleft"'] &
\end{tikzcd}$$

Minimal (Q, T)-biautomaton and the hypothesis automaton

Closure and consistency for the pair (Q, T) can be encoded categorically via the minimal (Q, T)-biautomaton.

Minimal (Q, T)-biautomaton and the hypothesis automaton

Closure and consistency for the pair (Q, T) can be encoded categorically via the minimal (Q, T)-biautomaton.

We obtain a generic FunL* algorithm that instantiates to

- Angluin's original algorithm,
- the weighted automata over fields variant of L^*
- the sequential transducer variant

Further details: Thomas Colcombet, Daniela Petrisan, Riccardo Stabile: Learning Automata and Transducers: A Categorical Approach. CSL 2021

Perspectives

Minimization/learning for free ?

Are there some conditions on a monad T so that $\mathrm{Kl}(T)$ has all the required properties required for the existence of minimization/learning of $\mathrm{Kl}(T)$-automata ?

Minimization/learning for free ?

Are there some conditions on a monad T so that $\mathrm{Kl}(T)$ has all the required properties required for the existence of minimization/learning of $\mathrm{Kl}(T)$-automata ?

Move to Eilenberg-Moore algebras when $\mathrm{Kl}(T)$ is not good enough, e.g., to the category of join sup-semilattices.

We see a Rel-valued automaton as a JSL-valued automaton.

Minimization/learning for free ?

Are there some conditions on a monad T so that $\mathrm{Kl}(T)$ has all the required properties required for the existence of minimization/learning of $\mathrm{Kl}(T)$-automata ?

Move to Eilenberg-Moore algebras when $\mathrm{Kl}(T)$ is not good enough, e.g., to the category of join sup-semilattices.

We see a Rel-valued automaton as a JSL-valued automaton.

Extension to tree automata

Minimization/learning for free ?

Are there some conditions on a monad T so that $\mathrm{Kl}(T)$ has all the required properties required for the existence of minimization/learning of $\mathrm{Kl}(T)$-automata ?

Move to Eilenberg-Moore algebras when $\mathrm{Kl}(T)$ is not good enough, e.g., to the category of join sup-semilattices.

We see a Rel-valued automaton as a JSL-valued automaton.

Extension to tree automata

Weighted automata over number rings –> see `https://arxiv.org/pdf/2504.16596`

Minimization/learning for free ?

Are there some conditions on a monad T so that $\mathrm{Kl}(T)$ has all the required properties required for the existence of minimization/learning of $\mathrm{Kl}(T)$-automata ?

Move to Eilenberg-Moore algebras when $\mathrm{Kl}(T)$ is not good enough, e.g., to the category of join sup-semilattices.

We see a Rel-valued automaton as a JSL-valued automaton.

Extension to tree automata

Weighted automata over number rings –> see `https://arxiv.org/pdf/2504.16596`

Learning nominal automata, building on Victor Iwaniack's work on automata in toposes.

More details for learning

Minimal (Q, T)-biautomaton and the hypothesis automaton

We can compute the minimal (Q, T)-biautomaton in an arbitrary category* using off-the-shelf results from (Colcombet,P., 2017).

$$\begin{tikzcd}
1 \arrow[r, "\triangleright q_{min}"] & Q/{\sim_{T \cup AT}} \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] & (Q \cup QA)/{\sim_T} \arrow[r, "t\triangleleft_{min}"] & 2
\end{tikzcd}$$

Recall $w \sim_T v$ iff $\forall u \in T.\quad wu \in L \Leftrightarrow vu \in L$

* under mild assumptions

Minimal (Q, T)-biautomaton and the hypothesis automaton

We can compute the minimal (Q, T)-biautomaton in an arbitrary category* using off-the-shelf results from (Colcombet,P., 2017).

$$\begin{tikzcd}
1 \arrow[r, "\triangleright q_{min}"] & Q/{\sim_{T \cup AT}} \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] & (Q \cup QA)/{\sim_T} \arrow[r, "t\triangleleft_{min}"] & 2
\end{tikzcd}$$

Recall $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$

$\triangleright q_{min}(*) = [q]_{\sim_{T \cup AT}}$

Minimal (Q,T)-biautomaton and the hypothesis automaton

We can compute the minimal (Q,T)-biautomaton in an arbitrary category* using off-the-shelf results from (Colcombet,P., 2017).

$$\begin{tikzcd}
1 \arrow[r, "\triangleright q_{min}"] & Q/{\sim_{T\cup AT}} \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] & (Q \cup QA)/{\sim_T} \arrow[r, "t\triangleleft_{min}"] & 2
\end{tikzcd}$$

Recall $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$

$$\triangleright q_{min}(*) = [q]_{\sim_{T\cup AT}} \qquad a_{min}([q]_{\sim_{T\cup AT}}) = [qa]_{\sim_T}$$

$$\varepsilon_{min}([q]_{\sim_{T\cup AT}}) = [q]_{\sim_T}$$

Minimal (Q,T)-biautomaton and the hypothesis automaton

We can compute the minimal (Q,T)-biautomaton in an arbitrary category* using off-the-shelf results from (Colcombet,P., 2017).

$$\begin{tikzcd} 1 \arrow[r, "\triangleright q_{min}"] & Q/{\sim_{T \cup AT}} \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] & (Q \cup QA)/{\sim_T} \arrow[r, "t \triangleleft_{min}"] & 2 \end{tikzcd}$$

Recall $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$

$\triangleright q_{min}(*) = [q]_{\sim_{T \cup AT}}$ $\quad a_{min}([q]_{\sim_{T \cup AT}}) = [qa]_{\sim_T}$ $\quad t \triangleleft_{min} ([q]_{\sim_T}) = L_{Q,T}(qt)$

$\varepsilon_{min}([q]_{\sim_{T \cup AT}}) = [q]_{\sim_T}$ $\quad t \triangleleft_{min} ([qa]_{\sim_T}) = L_{Q,T}(qat)$

Minimal (Q, T)-biautomaton and the hypothesis automaton

We can compute the minimal (Q, T)-biautomaton in an arbitrary category* using off-the-shelf results from (Colcombet,P., 2017).

$$\begin{tikzcd} 1 \arrow[r, "\triangleright q_{min}"] & Q/{\sim_{T \cup AT}} \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] & (Q \cup QA)/{\sim_T} \arrow[r, "t \triangleleft_{min}"] & 2 \end{tikzcd}$$

Recall $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$

$\triangleright q_{min}(*) = [q]_{\sim_{T \cup AT}}$ $\quad a_{min}([q]_{\sim_{T \cup AT}}) = [qa]_{\sim_T}$ $\quad t \triangleleft_{min} ([q]_{\sim_T}) = L_{Q,T}(qt)$

$\varepsilon_{min}([q]_{\sim_{T \cup AT}}) = [q]_{\sim_T}$ $\quad t \triangleleft_{min} ([qa]_{\sim_T}) = L_{Q,T}(qat)$

Minimal (Q,T)-biautomaton and the hypothesis automaton

We can compute the minimal (Q,T)-biautomaton in an arbitrary category* using off-the-shelf results from (Colcombet,P., 2017).

$$\begin{tikzcd} 1 \arrow[r, "\triangleright q_{min}"] & Q/{\sim_{T\cup AT}} \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] & (Q\cup QA)/{\sim_T} \arrow[r, "t\triangleleft_{min}"] & 2 \end{tikzcd}$$

Recall $w \sim_T v$ iff $\forall u \in T.\quad wu \in L \Leftrightarrow vu \in L$

$\triangleright q_{min}(*) = [q]_{\sim_{T\cup AT}}$ $\qquad a_{min}([q]_{\sim_{T\cup AT}}) = [qa]_{\sim_T}$ $\qquad t\triangleleft_{min}([q]_{\sim_T}) = L_{Q,T}(qt)$

$\varepsilon_{min}([q]_{\sim_{T\cup AT}}) = [q]_{\sim_T}$ $\qquad t\triangleleft_{min}([qa]_{\sim_T}) = L_{Q,T}(qat)$

- ε_{min} is surjective iff (Q,T) is closed
- ε_{min} is injective iff (Q,T) is consistent

```
$Q = T := \{\varepsilon\}$
repeat
  while $(Q,T)$ not closed and consistent
    if $(Q,T)$ is not closed enlarge $Q$
      $(\forall q \in Q.\forall a \in A.\exists p \in Q.\ p \sim_T qa)$
    if $(Q,T)$ is not consistent enlarge $T$
      $(\forall q,q' \in Q.\forall a \in A.\ \ q \sim_T q' \Rightarrow qa \sim_T q'a)$
  ask an equivalence query for $\mathcal{H}(Q,T)$
  if the answer is no then
    add the counterexample and its
    prefixes to $Q$
until the answer is yes
return $\mathcal{H}(Q,T)$
```

Minimal (Q,T)-biautomaton and the hypothesis automaton

We can compute the minimal (Q,T)-biautomaton in an arbitrary category* using off-the-shelf results from (Colcombet,P., 2017).

$$\begin{tikzcd}
1 \arrow[r, "\triangleright q_{min}"] & Q/{\sim_{T\cup AT}} \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] & (Q \cup QA)/{\sim_T} \arrow[r, "t\triangleleft_{min}"] & 2
\end{tikzcd}$$

Recall $w \sim_T v$ iff $\forall u \in T. \quad wu \in L \Leftrightarrow vu \in L$

$$\triangleright q_{min}(*) = [q]_{\sim_{T\cup AT}} \qquad a_{min}([q]_{\sim_{T\cup AT}}) = [qa]_{\sim_T} \qquad t \triangleleft_{min} ([q]_{\sim_T}) = L_{Q,T}(qt)$$

$$\varepsilon_{min}([q]_{\sim_{T\cup AT}}) = [q]_{\sim_T} \qquad t \triangleleft_{min} ([qa]_{\sim_T}) = L_{Q,T}(qat)$$

- ε_{min} is surjective iff (Q,T) is closed
- ε_{min} is injective iff (Q,T) is consistent
- If ε_{min} is an isomorphism we merge the two states of the (Q,T)-biautomaton and obtain $\mathcal{H}(Q,T)$.

```
$Q = T := \{\varepsilon\}$
repeat
  while $(Q,T)$ not closed and consistent
    if $(Q,T)$ is not closed enlarge $Q$
      ( $\forall q \in Q . \forall a \in A . \exists p \in Q .\ p \sim_T qa$)
    if $(Q,T)$ is not consistent enlarge $T$
      ($\forall q, q' \in Q . \forall a \in A .\ \ q \sim_T q' \Rightarrow qa \sim_T q'a$)
  ask an equivalence query for $\mathcal{H}(Q,T)$
  if the answer is no then
    add the counterexample and its
    prefixes to $Q$
until the answer is yes
return $\mathcal{H}(Q,T)$
```

The $FunL^*$-algorithm

```
input: teacher of the target language $L$                       in a catgeory $\mathcal{C}$
output: $Min(L)$
$Q := T := \{\varepsilon\}$
repeat
   while $\varepsilon_{min}$ is not an isomorphism do           Iso = $E \cap M$
      if $\varepsilon_{min} \notin E$ then                      $(E, M)$ fact. system
         add $QA$ to $Q$
      if $\varepsilon_{min} \notin M$ then
         add $AT$ to $T$
   ask an equivalence query for the hypothesis automaton $\mathcal{H}(Q, T)$
   if the answer is no then
      add the counterexample and all its prefixes to $Q$
until the answer is yes
return $\mathcal{H}(Q, T)$
```