RFSA via Functors

Quentin Schroeder

June 30, 2024

Abstract

This report presents a new perspective on the minimization of Residual Finite State Automata (RFSA) using the functorial framework of [4]. Our method essentially attempts to explain the relationship between the category of automata over complete join semi-lattices (JSLA) and non deterministic automata, which are modeled over the category whose objects are sets and morphisms are relations. The expected result was that the canonical RFSA [6] can be obtained by taking the irreducibles of the the canonical JSLA. However this is only partly true. The algebraic description of a canonical RFSA turns out to be more subtle than originally thought. We found a canonical lax functor $\mathcal{J}$ from the category of join semi-lattices to the category of relations. The functor being lax means that it does not preserve composition of morphisms, but one has an inclusion instead. This laxness poses some subtle problems, but we can nevertheless show that despite the canonical RFSA not being in the image of that functor, it instead be freely generated by the action of $\mathcal{J}$ on the transitions.

1 Introduction

In this report we will give a new perspective on the minimization of residual finite state automata [6] in the functorial framework of [4] – which neatly combines the algebraic and coalgebraic approaches to automata theory.

Context

There has been many approaches to unify big parts of automata theory through the language of category theory. The oldest of which is probably the coalgebraic approach [9], but there has also been very recent approaches using monads to capture the notion of recognizability [2]. Or the fibrational approach of [11], which gives another potential account of how to deal with non deterministic automata. Lastly there is the functorial approach to minimization of [4], which is the context in which we find ourselves for this report.

The latter gives a general framework of viewing automata minimization through the language of category theory, expressing automata as certain kinds of functors and minimization as a result of image factorization. A powerful example they give is

a fully abstract explanation for the Brzozowski algorithm for minimization of deterministic automata via a chain of adjunctions. This approach works very well for categories satisfying certain mild assumptions, namely the existence of some products and coproducts as well as a reasonable factorization system. This is not the case for the category of relations, which is used to model non-deterministic automata. This explains why minimization of non-deterministic automata is problematic. Furthermore there is a large body of work trying to give different accounts of minimal non-deterministic automata [3, 7, 10, 14]. Fundamental for our development is the paper of [6] on residual finite state automata. Closely related to our approach is the work of Adámek et al. [1] on closure spaces. Our goal was to explain RFSA using the functorial framework and to show the relationship to the minimization of join-semilattice automata.
This report was made in the context of the "travail de recherche encadré" at the Master Parisien de Recherche en Informatique under the supervision of Daniela Petrişan. It was a 6 ECTS course which is usually completed alongside the second semester of the first year of the master.

Contributions

The contributions of this report are an interpretation of RFSA in the functorial framework via a (lax) functor from the category of join-sup-semilattices to the category of relations.

2 Functorial approach

We start with some preliminaries. We will sometimes leave out complete technical definitions and proofs for the sake of readability. If the reader is interested in a more formal account, they can be found in the references.

Definition 2.1 (Category [12])**.** A **category** C is a collection of objects $ob(C)$ together with a set of morphisms $C(x,y)$ for each pair of objects x,y and a composition operation for all objects x,y,z:

$$\circ : C(y,z) \times C(x,y) \to C(x,z)$$

Furthermore each set $C(x,x)$ contains a morphism $id_x : x \to x$ called the **identity** on x. This data is subject to unity and associtivity axioms. Namely, for any $f : x \to y \in C(x,y)$, $f \circ id_x = f = id_y \circ f$. Lastly, for $f : x \to y$, $g : y \to z$, $h : z \to w$, we have:

$$h \circ (g \circ f) = (h \circ g) \circ f$$

Notice the abuse of notation, we will often write $f : x \to y \in C$ to mean $f \in C(x,y)$ and $x \in C$ for $x \in ob(C)$. This should never be ambiguous from context.

Remark 2.1. One major advantage of working with categories is the proof technique they give rise to, the diagram chase. We will often encounter situations where we have equations between morphisms. A quick example would be the unity axioms of 2.1:

$$\begin{tikzcd}
x \arrow[r, "f"] & y \\
x \arrow[u, "id_x"] \arrow[ur, "f"'] &
\end{tikzcd}$$

To understand what this means, fix two verticies off the graph, for example the bottom left x and the top right y and read off each label the edges along a directed path of the graph. Such a path corresponds to a composition of morphisms, so the path $x - x - y$ corresponds to $f \circ id_x$. We say this diagram commutes since the other direct path $x - y$ corresponds to f and $f \circ id_x = f$.
A diagram commutes if each directed path with the same start and end points corresponds to the same composite morphism.

Example 2.1.1. Fix some alphabet Σ, and denote the set of words over Σ by Σ^*.

1. SET has sets as objects and functions between them as morphisms, the identity on every set X is the function $id_X : X \to X$, mapping $x \mapsto x$. The composition is function composition which is associative and the unity is easy to check.

2. REL has sets as objects and relations as morphisms, the identity on every set X is the diagonal relation $id_X : X \nrightarrow X$ where $(x, y) \in id_X$ iff $x = y$. Composition is given by composition of relations $S \circ R = \{(x, z) \mid \exists y : xRy \texttt{ and } ySz\}$.

3. VECT has vector spaces as objects and linear maps as morphisms.

4. For any directed multigraph G, we can build the **free category** $Free(G)$, which has vertices of G as objects and directed paths in G as morphisms. Composition is given by joining two paths at their end and start points. Identities are given by empty paths.

5. We will use the category $\mathcal{O}$, which is the free category on the following multigraph, where each word $w \in \Sigma^*$ has an associated arrow from $\in$ to out. We will see later that this multigraph can be used to understand the language accepted by an automaton:

$$\begin{tikzcd}
in \arrow[r, "\forall w \in \Sigma^* : \rhd w \lhd"] & out
\end{tikzcd}$$

6. The category $\mathcal{I}$ which is the free category following graph having three verticies in, st, out and where for each input letter in $a \in \Sigma$ we have an edge $a : st \to st$:

$$\begin{tikzcd}
in \arrow[r, "\rhd"] & st \arrow[r, "\lhd"] \arrow[loop below, "{\forall a\in\Sigma, a:st\to st}"] & out
\end{tikzcd}$$

The starting point for the functorial approach of [4] is that various forms of automata can be obtained by *interpreting* the edges in last item of Example 2.1.1 as morphisms of various categories capturing a certain effect of the automaton such as non-determinism. For example a DFA can be modelled by interpreting the interpreting the edges as functions between sets:

$$\begin{tikzcd}
\mathbb{1} \arrow[r, "q_0"] & Q \arrow[r, "F"] \arrow[loop below, "{\forall \delta_a\in\Sigma, a:st\to st}"] & \{0,1\}
\end{tikzcd}$$

Similarly we can represent an NFA by interpreting the edges in the graph above as relations:

$$\begin{tikzcd}
\mathbb{1} \arrow[r, "I"] & Q \arrow[r, "F"] \arrow[loop below, "{\forall \delta_a\in\Sigma, a:st\to st}"] & \mathbb{1}
\end{tikzcd}$$

From a category theoretic point of view ‘interpreting’ edges amounts to giving a functor – a notion that we introduce next.

Definition 2.2 (Functor)**.** A **functor** $F : C \to D$ from a category C to a category D consists of a mapping $F : ob(C) \to ob(D)$ on objects and for each pair of objects x, y in C, a function $F_{x,y} : C(x, y) \to D(F(x), F(y))$ such that:

- $\forall x \in ob(C) : F(id_x) = id_{F(x)}$
- $\forall f : x \to y, g : y \to z \in C : F(g \circ f) = F(g) \circ F(f)$

Example 2.2.1.

- $\mathcal{P}$: SET → SET the powerset functor, takes a function $f : X \to Y$ to $\mathcal{P}(f) : \mathcal{P}(X) \to \mathcal{P}(Y)$, with $S \mapsto f(S)$.
- $\iota : \mathcal{O} \to \mathcal{I}$, is the inclusion taking $\rhd\ w \lhd$ to $\lhd \circ a_n \circ \cdots \circ a_0 \circ \rhd$, where $w = a_0 a_1 \dots a_n$. This inclusion will be relevant to understand the language accepted by an automaton.

- Given any two functors $F : A \to B$ and $G : B \to C$, we can compose them to get a new functor $G \circ F : A \to C$, which sends $a \in ob(A)$ to $G(F(a))$ and $f \in A(x, y)$ to $G(F(f))$.
- For any category C, the identity functor $id_C : C \to C$ is the functor that sends $x \in ob(C)$ to x and $f \in C(x, y)$ to f.

Definition 2.3 (Automata)**.** Given a category C, a C-**automaton** A is a functor $A : \mathcal{I} \to C$, where $\mathcal{I}$ is is the free category described in Example 2.1.1. The language accepted by such an automaton A is the composite functor $L := A \circ \iota : \mathcal{O} \to C$, where ι is the inclusion defined in Example 2.2.1.

The intuition behind this definition is that $\mathcal{O}$ is a full subcategory of $\mathcal{I}$ that abstracts away the inner structure of the automaton and only looks at its observable behavior, that is its language.

Example 2.3.1. So far we restated the definitions from traditional automata theory via category theory, in the usual cases of SET and REL we recover the usual notions of deterministic automaton, non-deterministic automaton and the languages they accept.

- SET-automata $A : \mathcal{I} \to$ SET with $A(in) = \mathbb{1}$ and $A(out) = \mathcal{B} := \{\bot, \top\}$ correspond to deterministic automata.
  Indeed, $A(st)$ gives the set of states Q for the automaton, specifying a map $A(\rhd) : \mathbb{1} \to A(st)$ amounts to picking out the initial state q_0 of the automaton. Furthermore $A(\lhd) : A(st) \to \mathcal{B}$ corresponds the final states F. Lastly for each $a \in \Sigma$, $A(a) : A(st) \to A(st)$ is the usual transition map of a deterministic automaton δ_a. A word is accepted if $A(\rhd \circ a_n \circ \cdots \circ a_0 \circ \lhd)(\star) = \top$, that is if $A(w)(q_0) = \delta^*(w, q_0) \in F$, which is exactly the usual acceptance condition for deterministic automata.
- REL-automata $A : \mathcal{I} \to$ REL such that $A(in) = \mathbb{1}$ and $A(out) = \mathbb{1}$ correspond to non-deterministic automata.
  The relation $A(\rhd) : \mathbb{1} \nrightarrow A(st)$ corresponds to the set of initial states I of the automaton. And the relation $A(\lhd) : A(st) \nrightarrow \mathbb{1}$ corresponds the final states F of the automaton. Lastly for each $a \in \Sigma$, $A(a) : A(st) \nrightarrow A(st)$ is just another way of stating the usual transition relation δ_a. Here again we can check that A accepts w iff $A(w)[I] \cap F \neq \varnothing$, which is the usual notion of acceptance for non-deterministic automata.

Definition 2.4 (Natural Transformation)**.** For functors $F, G : C \to D$, a **natural transformation** $\alpha : F \to G$ is a family of morphisms $\alpha_x : F(x) \to G(x) \in D$ for each object $x \in ob(C)$ such that for any $f : x \to y \in C$, the following diagram commutes:

$$\begin{tikzcd}
Fx \arrow[r, "\alpha_x"] \arrow[d, "Ff"'] & Gx \arrow[d, "Gf"] \\
Fy \arrow[r, "\alpha_y"'] & Gy
\end{tikzcd}$$

We say that α is natural in x.
This definition tends to be a bit abstract. The idea is that it should not matter if we change from the image of F to G before or after applying a morphism in C. So it is a way of coherently transforming the image of F to the image of G.

Example 2.4.1.

- For the powerset functor $\mathcal{P}$: SET → SET, the natural transformation η : $id_{\mathsf{SET}} \Rightarrow \mathcal{P}$ is given by $\eta_X : X \to \mathcal{P}(X)$, $x \mapsto \{x\}$.
  This is a way of embedding a set into its powerset.
- For the powerset functor $\mathcal{P}$: SET → SET, the natural transformation ϵ : $\mathcal{P} \circ \mathcal{P} \Rightarrow \mathcal{P}$ is given by $\mu_X : \mathcal{P}(\mathcal{P}(X)) \to \mathcal{P}(X)$, $S \mapsto \bigcup S$.
  This is a way of removing a layer of nesting in the powerset.
- For SET-automata $A, B : \mathcal{I} \to$ SET, an atomaton morphism is a natural transformation $\alpha : A \Rightarrow B$ such that its components at in, out are the identity, that is we have three morphisms:
  - $\alpha_{in} : A(in) \to B(in)$, which is the identity of $\mathbb{1}$
  - $\alpha_{st} : A(st) \to B(st)$, which is some function mapping on states
  - $\alpha_{out} : A(out) \to B(out)$, which is the identity of $\mathcal{B}$

  Now we will see an example of naturality in action, for $\rhd$: $in \to st$, we have the following naturality square:

$$\begin{tikzcd}
A(in) \arrow[r, "\alpha_{in}"] \arrow[d, "A(\rhd)"'] & B(in) \arrow[d, "B(\rhd)"] \\
A(st) \arrow[r, "\alpha_{st}"'] & B(st)
\end{tikzcd}$$

  Notice that the top arrow is just identity so we can identity that part of the diagram to obtain:

$$\begin{tikzcd}
& \mathbb{1} \arrow[dl, "A(\rhd)"'] \arrow[dr, "B(\rhd)"] & \\
A(st) \arrow[rr, "\alpha_{st}"'] & & B(st)
\end{tikzcd}$$

Through a similar argument we find that the following also commutes:

$$\begin{tikzcd}
 & \mathcal{B} & \\
A(st) \arrow[ur, "A(\lhd)"] \arrow[rr, "\alpha_{st}"'] & & B(st) \arrow[ul, "B(\lhd)"']
\end{tikzcd}$$

And lastly for each $a \in \Sigma$, we have the following commutative square:

$$\begin{tikzcd}
A(st) \arrow[r, "\alpha_{st}"'] \arrow[d, "A(a)"'] & B(st) \arrow[d, "B(a)"] \\
A(st) \arrow[r, "\alpha_{st}"'] & B(st)
\end{tikzcd}$$

These three diagrams specify that α is given by a morphism between the state spaces of the automata, which behaves well with respect to the initial and final state as well as with the transitions. In all the examples considered we retrieve the usual notion of morphism of automata.

- More generally for any C-automata $A, B : \mathcal{I} \to C$, a **morphism of C-automata** is a natural transformation $\alpha : A \Rightarrow B$ such that $\alpha_{in} = id_{A(in)}$ and $\alpha_{out} = id_{A(out)}$.

Proposition 2.1. *C-automata form a category.*

Definition 2.5 (Initial, Terminal object)**.**
An object x in a category C is **initial** if for any object $y \in ob(C)$ there is a unique morphism $! : x \to y$.
An object x in a category C is **terminal** if for any object $y \in ob(C)$ there is a unique morphism $! : y \to x$.

Example 2.5.1.

- In SET, the empty set is initial and any singleton set is terminal.
- In REL, the empty set is initial and terminal.
- In JSL3.2, the initial and terminal objects are both $\mathcal{B}$, the two element lattice $\{\bot, \top\}$ with $\bot \leqslant \top$.

Remark 2.2 (Universal Constructions). A common theme in category theory is the idea of universal constructions, which are ways of defining objects in a category by their relationship to other objects.
The initial and terminal objects are the simplest examples of this, but there are many more, which for the sake of brevity we will not go into here.
That said we have to mention the following:

- **products** in a category are a generalization of the cartesian product of sets. We usually denote the product of a family of objects $(X_i)_i$ by $\prod\limits_i X_i$.

- **coproducts** in a category are a generalization of the disjoint union of sets. We usually denote the coproduct of a family of objects $(X_i)_i$ by $\coprod\limits_i X_i$.

- **left Kan extension** is the least constrained of extending the domain of a functor along another functor.

- dually a **right Kan extension** is the most constrained of extending the domain of a functor along another functor.

For full accounts of these we refer the reader to [12].

The intuition of a Kan extension can be illustrated by the following simpler example in the setting of partially ordered sets X, Y, Z and order preserving maps $f : X \to Z$ and $k : X \to Y$ we have two reasonable ways (assuming the right meets and joins exists) of constructing an order preserving function $g : Y \to Z$.

The first is to approximate elements of Y through k from below and then apply f, explicitly $g(y) = \bigvee_{k(x) \leqslant y} f(x)$, this corresponds to the left Kan extension of f along k and gives us that $f \leqslant g \circ k$.[1]

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "k"'] & Z \\
Y \arrow[ur, "g"', ""{name=G}] &
\arrow[Rightarrow, from=1-1, to=G, "\leqslant"']
\end{tikzcd}$$

The second is to approximate elements of Y through k from above and then apply f, explicitly $g(y) = \bigwedge_{y \leqslant k(x)} f(x)$, this corresponds to the right Kan extension of f along k and gives us that $g \circ k \leqslant f$.

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "k"'] & Z \\
Y \arrow[ur, "g"', ""{name=G}] &
\arrow[Rightarrow, from=G, to=1-1, "\geqslant"]
\end{tikzcd}$$

The left and right Kan extension give us two "optimal" ways of approximating a functor f through a functor k.

The Kan extensions are of interest to us because they give two universal ways of finding automata accepting a given language L, namely, the initial, respectively

[1] Note that a partially ordered set $(X, \leqslant)$ can be seen as a very simple category whose objects are the elements of X and with at most one morphisms from x to y, whenever $x \leqslant y$.

the terminal automaton in the category of automata accepting L.

$$\begin{tikzcd}
\mathcal{O} \arrow[r, "L"] \arrow[d, "\iota"'] & C \\
\mathcal{I} \arrow[ur, dotted, bend left] \arrow[ur, dotted, bend right] &
\end{tikzcd}$$

Proposition 2.2. *The left Kan extension of a C-language along the inclusion $\iota : \mathcal{O} \to \mathcal{I}$ is the initial C-automaton accepting that language and exists if C has countable products*[2]*.*

Dually the right Kan extension of a C-language along the inclusion $\iota : \mathcal{O} \to \mathcal{I}$ is the terminal C-automaton accepting that language and it exists if C has countable coproducts.

Example 2.5.2.

Furthermore, Kan extensions of languages along the inclusion can be computed explicitly in many cases.

- The initial SET-automaton accepting L is given by:

$$\begin{aligned} A(st) &= \Sigma^* \\ A(\rhd)(\top) &= \epsilon \\ A(\lhd)(w) &= \top \text{ iff } w \in L \\ A(a)(w) &= wa \end{aligned}$$

- The terminal SET-automaton accepting L is given by:

$$\begin{aligned} A(st) &= \mathcal{P}(\Sigma^*) \\ A(\rhd) &= L \\ A(\lhd)(S) &= \top \text{ iff } \epsilon \in S \\ A(a)(S) &= a^{-1}S = \{w \mid wa \in S\} \end{aligned}$$

- The initial REL-automaton accepting L is given by:

$$\begin{aligned} A(st) &= \Sigma^* \\ A(\rhd) &= \{\epsilon\} \\ A(\lhd) &= L \\ A(a) &= \{(w, wa) \mid w \in \Sigma^*\} \end{aligned}$$

[2] In [4] this is stated using powers and copowers, which is more general.

- The terminal REL-automaton accepting L is given by:

$$\begin{aligned}
A(st) &= \Sigma^* \\
A(\rhd) &= L \\
A(\lhd) &= \{\epsilon\} \\
A(a) &= \{(wa, w) \mid w \in \Sigma^*\}
\end{aligned}$$

Notice that neither the initial, nor the terminal SET-automaton accepting a language L are the minimal automaton for this language. But they are nevertheless relevant for minimization. It was noticed in [4] that the unique map from the initial SET-automaton to the terminal one is given by the function $\Sigma^* \to \mathcal{P}(()\Sigma^*)$ mapping $w \in \Sigma^*$ to $w^{-1}L$. Hence two such words w, w' are identified by this map when $w^{-1}L = w'^{-1}L$, hence when they are equivalent with respect to the Myhill-Nerode equivalence relation. So the image of this map is precisely the state space of the minimal automaton for L. This situation generalizes to weighted automata, subsequential transducers, syntactic monoids, etc. To move away from SET to other categories, we need a category theoretic way of generalizing quotients or images of functions. This is given by the notion of factorization system below.

Definition 2.6 (factorization system [13])**.** A **factorization system** in a category C is a pair of classes of morphisms $(\mathcal{E}, \mathcal{M})$ such that each class contains all morphisms and is closed under composition. Furthermore we require each $f : X \to Y \in C$ to have a factorization of the form:

$$\begin{tikzcd}
X \arrow[r, "e"] \arrow[rr, bend left, "f"] & Im(f) \arrow[r, "m"] & Y
\end{tikzcd}$$

Where e is in $\mathcal{E}$ and m is in $\mathcal{M}$.
And lastly for each square:

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, two heads, "e"] & Z \arrow[d, tail, "m"] \\
Y \arrow[r, "g"] \arrow[ur, dotted, "\exists!"] & W
\end{tikzcd}$$

where $e \in \mathcal{E}$, $m \in \mathcal{M}$ and f, g are C morphisms, we have a unique morphism going from Y to Z making the square commute. The intuition is what is most important here, we are simply stating that any morphism can be factored into a surjection followed by an injection.

We will later see that the notion of factorization system lets us define a minimization procedure for automata in a very general way.

Example 2.6.1.

- In SET, the factorization system is given by $\mathcal{E}$ the class of surjections and $\mathcal{M}$ the class of injections.
- The category REL has trivial factorization systems, but they do not behave well with respect to cardinality of sets, hence they are not useful in practice.

Definition 2.7 (Divisibility)**.** Given a factorization system $(\mathcal{E}, \mathcal{M})$ in a category C and objects $X, Y \in C$, we say that X **$(\mathcal{E}, \mathcal{M})$-divides** Y if there is an object A such that:

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

Where $e \in \mathcal{E}$ and $m \in \mathcal{M}$.
An object X is $(\mathcal{E}, \mathcal{M})$-**minimal** if for any object Y such that X $(\mathcal{E}, \mathcal{M}) - divides$ Y.

Remark 2.3. In the category SET, we have that if X $(\mathcal{E}, \mathcal{M})$-divides Y, then the cardinality of X is less than or equal to the cardinality of Y. So divisibility encodes the notion of size in SET. Which we will use to be able to define a minimization procedure for automata.

Proposition 2.3. *Given a factorization system $(\mathcal{E}, \mathcal{M})$ in a category C, we can construct a factorization system $(\mathcal{E}', \mathcal{M}')$ on the category of C-automata.*

Proposition 2.4 (sufficient structure for minimization)**.** *If a category C has a factorization system $(\mathcal{E}, \mathcal{M})$ an initial object and a final object, then there is a $(\mathcal{E}, \mathcal{M})$-minimal object M in C. And it can be obtained via the image factorization of the unique morphism $I \to F$, where I is the initial object and F is the final object of C.*

Proposition 2.5 (lifting structure for minimization [4])**.** *If a category C has a factorization system $(\mathcal{E}, \mathcal{M})$ and C has countable powers and copowers then the category of C-automata has a factorization system $(\mathcal{E}', \mathcal{M}')$, an initial and terminal automaton and thus by 2.4 a $(\mathcal{E}', \mathcal{M}')$-minimal object.*

Example 2.7.1. For SET-automata, the minimal object is the following automaton accepting L:

$$\begin{aligned} A(st) &= \{w^{-1}L \mid w \in\} \\ A(\rhd) &= L \\ A(\lhd) &= \{\epsilon\} \\ A(a) &= a^{-1}L \end{aligned}$$

This is exactly the Myhill-Nerode minimization procedure for DFAs. Our goal will be to use a similar procedure with JSL-automata and then relate them to the canonical residual finite state automata of [6].

Definition 2.8 (Reachability Obserability)**.** Given a category C with initial object I and a factorization system $(\mathcal{E}, \mathcal{M})$ and an object $X \in C$, we define $reach(X)$ to the image of the factorization:

$$\begin{tikzcd}
I \arrow[r, two heads] \arrow[rr, bend left, "!"] & reach(X) \arrow[r, tail] & X
\end{tikzcd}$$

We say that X is **reachable** if $reach(X) \cong X$.

Dually if C has a terminal object F and a factorization system $(\mathcal{E}, \mathcal{M})$ and an object $X \in C$, we define $obs(X)$ to the image of the factorization:

$$\begin{tikzcd}
X \arrow[r, two heads] \arrow[rr, bend left, "!"] & obs(X) \arrow[r, tail] & F
\end{tikzcd}$$

We say that X is **observable** if $obs(X) \cong X$.

Example 2.8.1. These definition are a bit too general, we will just note that in SET-automata, for an SET-automaton A is reachable precisely when every state is reachable from the initial state. Similarly for REL-automata, an REL-automaton A is reachable precisely when every state is reachable from the set of initial states.

3 Minimization in JSL

Our goal will be to give a full presentation of the minimal join-semi-lattice automaton for a language. We begin with an account of the basic properties of join-semilattices, distribute lattices and frames, which are the main objects of study in this section.

Definition 3.1 (Lattice)**.** A **lattice** is a partially ordered set $(L, \leqslant)$ in which every pair of elements $x, y \in L$ has a greatest lower bound $x \wedge y$ and a least upper bound $x \vee y$.

Definition 3.2 (JSL)**.** A **complete join semi-lattice (JSL)** $(L, \leqslant, \bigvee)$ is a partially ordered set $L, \leqslant$ together with a join operation $\bigvee : \mathcal{P}(L) \to L$ such that for any subset S of L:

1. for any $s \in S$, $s \leqslant \bigvee S$
2. for any $x \in L$, if $\forall s \in S : s \leqslant x$, then $\bigvee S \leqslant x$

The first condition states that $\bigvee S$ is an upper bound of S.
The second condition states that $\bigvee S$ is the least upper bound of S.
This definition also implies that L has a greatest element $\top = \bigvee L$ and a least element $\bot = \bigvee \varnothing$.

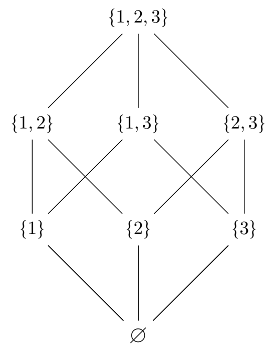
Remark 3.1.

- The join operation is idempotent, commutative and associative.
- The join operation is monotone in each argument.
- Every complete jsl is a complete lattice, so it has an operation $\bigwedge$ which corresponds to the greatest lower bound of a set.
- Every complete lattice is of course also a lattice.
- A finite JSL is a lattice with a top and bottom element.

Example 3.2.1. Let $\mathcal{P}(X)$ denote the powerset of a set X. The following are examples of complete join semi-lattices.

- the booleans: $\mathcal{B} = \{\bot, \top\}$ with $\bot \leqslant \top$ and $\top \vee \bot = \top$
- the powerset of any set X with the join given by unions: $(\mathcal{P}(X), \subseteq, \bigcup)$
- the unit interval with the supremum operation $([0,1], \leqslant, sup)$
- the natural numbers with an adjoined top element $\top$ with the maximum operation $(\mathcal{N}^{\top}, \leqslant, max)$
- logical propositions over some set of atoms, with the existential quantifier
- the opens of a topological space with the union operation

Definition 3.3 (Hasse Diagram)**.** For any finite JSL L, we can represent its elements via **Hasse diagrams**, which place elements from least to greatest with respect to the order relation. Here instead of trying to give a fully formal account[3] we will instead just give an example of a lattice $\mathcal{P}(\{1,2,3\})$:



[3] A complete account of lattice theory and Hasse diagrams can be found in [5]

Definition 3.4 (JSL Morphism)**.** A function $f : L \to M$ between two JSLs is a **JSL morphism** if for any $S \subseteq L$, $f(\bigvee S) = \bigvee f(S)$ in M.

Proposition 3.1 (Properties of JSL)**.** *Denote* JSL *the category of JSLs and JSL morphisms.*
JSL *has an appropriate factorization system and has products and coproducts.*
We list the properties we will use here:

- *products :* $\prod L_i$ *is the set product with pointwise operations and the usual projections*
- *coproducts :* $\coprod L_i$ *has the underlying set* $\prod L_i$ *again with pointwise operations and inclusions:*

$$\begin{aligned} \iota_j : L_j &\to \coprod L_i \\ l &\mapsto (l_k)_{k \in I} \end{aligned}$$

$$\textit{where } l_k = \begin{cases} l & \textit{if } k = j \\ \perp & \textit{otherwise} \end{cases}$$

- *Every* $f : X \to Y$ *in* JSL *can be factored into a surjection* $e : X \to Im(f)$ *followed by an injection* $m : Im(f) \to Y$ *such that* $f = m \circ e$*.*
- *Any two* JSL *morphisms* $f, g : X \to Y$ *can be compared pointwise denoted* $f \leqslant g$*, so* JSL(X, Y) *forms a partially ordered set.*

Definition 3.5 (Comparison functor)**.** The **comparison functor** $\mathcal{K}$: REL → JSL takes $X \mapsto \mathcal{P}(X)$ and $(R : X \nrightarrow Y) \mapsto R[-] : \mathcal{P}(X) \to \mathcal{P}(Y)$, the direct image of the relation.
Furthermore for $R, S : X \nrightarrow Y$, $R \subseteq S$ implies $K(R) \leqslant K(S)$.

Proposition 3.2. *The* **minimal** JSL-**automaton** A^{JSL}_{min} *for a language* L *is given by the following construction:*

$$\begin{aligned} &A(st) = \{\bigcup_{w \in S} w^{-1}L \mid S \subseteq \Sigma^*\} \\ &A(\rhd)(\top) = L \\ &A(\lhd)(q) = \top \;\; \boldsymbol{if} \;\; \epsilon \in q \\ &A(a)(q) = a^{-1}q \end{aligned}$$

Where we note that $a^{-1}\bigcup_{w \in S} w^{-1}L = \bigcup_{w \in S} a^{-1}w^{-1}L$*.*
We can also compute for any JSL-automaton A*, its automaton with only the reachable states,* $reach(A)$ *which has the states* $\{\bigvee_{w \in S} A(\rhd w)(\top) \mid S \subseteq \Sigma^*\}$*. Similarly we can compute the observable automaton* $obs(A)$*, which has the state set* $\{w \in \Sigma^* \mid A(w \lhd)(q) = \top\}$*.*

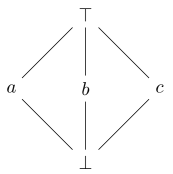
Definition 3.6 (Distributive lattice)**.** A **distributive lattice** is a JSL L in which the binary lattice operations distribute over each other.

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \tag{1}$$

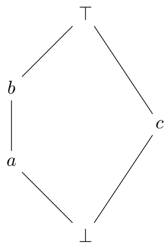
Example 3.6.1.

- The lattice $\mathcal{P}(\{1, 2, 3\})$ is distributive.
- The lattice $\mathcal{P}(X)$ is distributive.
- The opens of a topological space with binary unions and intersections is distributive.

Proposition 3.3 (M3-N5)**.** *If a lattice L contains an embedded copy of the M_3 or N_5 lattice, then L is not distributive.*
Where M_3 is given by the following Hasse diagram:

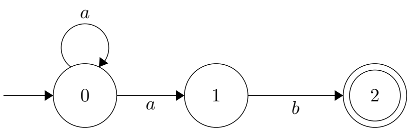


And N_5 is given by the following Hasse diagram:

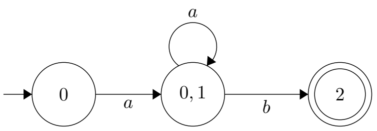


Proposition 3.4. *Given an NFA A as a functor $A : \mathcal{I} \to$ REL accepting a language L, $K \circ A$ is a JSL-automaton accepting the same language.*
It can be explicitly constructed via the subset construction for NFA's and closing the set of states under the join operation.
If the determinization is trimmed, then the resulting JSL-automaton is generates the $reach(K \circ A)$.

Example 3.6.2. We can find examples where the JSL-automaton corresponding to a trimmed determinized NFA has a non distribute underlying lattice.



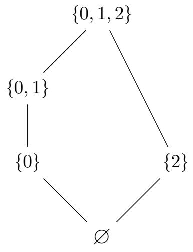
We get an N_5 lattice after determinizing and taking the reachable subautomaton in JSL.



The corresponding transition table of this JSL-automaton is below.

δ	a	b
$\varnothing$	$\varnothing$	$\varnothing$
$\{0\}$	$\{0,1\}$	$\varnothing$
$\{0,1\}$	$\{0,1\}$	$\{2\}$
$\{2\}$	$\varnothing$	$\varnothing$
$\{0,1,2\}$	$\{0,1\}$	$\{2\}$

Below is the Hasse diagram of the reachable JSL-automaton.



4 Minimal RFSA via Functors

Now that we have seen the construction for minimal JSLA, the next step will be to define the $\mathcal{J}$ functor and show that the canonical RFSA is deeply related to the minimal JSLA.

Definition 4.1 (Frame)**.** A **frame** [8] is a complete lattice $(L, \leqslant, \bigvee)$ such that for any $x \in L$ and $S \subseteq L$, $x \wedge \bigvee S = \bigvee\{x \wedge s \mid s \in S\}$. When a frame is finite they coincide with finite distributive lattices.

Example 4.1.1.

- The lattice $\mathcal{P}(X)$ is a frame.
- The unit interval with the supremum operation is a frame.
- The lattice of open sets of a topological space is a frame.

Definition 4.2 (Completely join-irreducible element)**.** An element $x \neq \bot$ in a JSL L is **completely join-irreducible** if for any $S \subseteq L$, $x = \bigvee S$ implies that $x \in S$. We denote the set of completely join-irreducible elements of L as $\mathcal{J}(L)$.

Example 4.2.1. The set of join irreducibles can be seen as the best attempt at finding a minimal amount of elements from which we can regenerate the original join semi lattice. What is more, in many cases the join irreducibles completely determine the lattice [4].

- In the lattice $\mathcal{P}(\{1, 2, 3\})$ from before, the completely join-irreducible elements are $\{1\}, \{2\}, \{3\}$.
- More generally in the lattice $\mathcal{P}(X)$, the completely join-irreducible elements are the singletons of X.
- For the unit interval with the supremum operation, there are no completely join irreducible elements since any non zero element can be written as a limit of points before it.
- For the natural numbers with adjoined $\top$, with the maximum operation, every non zero, non $\top$ element is completely join-irreducible.

Definition 4.3 (Completely Join-prime element)**.** An element x in a JSL L is **completely join-prime** if for any $S \subseteq L$ such that $x \leqslant \bigvee S$, there is some $s \in S$ such that $x \leqslant s$.

Proposition 4.1. *In a frame L, the completely join-irreducible elements are exactly the join-prime elements.*

Definition 4.4 (Join dense set)**.** A set $D \subseteq L$ is **join-dense** in a JSL L if for any $x \in L$, there is $Q \subseteq D$ such that $x = \bigvee Q$.

Example 4.4.1.

- In the lattice $\mathcal{P}(\{1, 2, 3\})$ from before, the set $\{\{1\}, \{2\}, \{3\}\}$ is join-dense.
- More generally in the lattice $\mathcal{P}(X)$, the set of singletons of X is join-dense.
- For the unit interval with the supremum operation, the set of diadic rationals is join-dense.

Proposition 4.2. *If a JSL L is finite, then the set of completely join-irreducible elements $\mathcal{J}(L)$ is join-dense.*

[4] The join irreducibles together with the meet irreducibles are what is commonly studied in the representation theory of lattices [5]

Definition 4.5 (Irreducibles lax functor)**.** We define the **irreducibles (lax) functor** $\mathcal{J}$ as:

$$\begin{aligned} \mathcal{J} : \mathsf{JSL} &\to \mathsf{REL} \\ L &\mapsto \{x \in L \mid x \texttt{ join irreducible}\} \\ f : L \to L' &\mapsto \mathcal{J}(f) := \{(x,y) \in \mathcal{J}(L) \times \mathcal{J}(L') \mid y \leqslant f(x)\} \end{aligned}$$

[5]

Proposition 4.3. *J is almost a functor, in fact for* JSL*-morphisms g, f that can be composed, we have :*

1. $J(g) \circ J(f) \subseteq J(g \circ f)$
2. $J(id_L) = id_{\mathcal{J}(L)}$

Example 4.5.1. A take the lattices $\mathcal{B}, \mathcal{P}(\mathcal{B}), N_5$, where N_5 has irreducibles a, b, c with $a < b$.
The maps $f : \mathcal{B} \to \mathcal{P}(\mathcal{B})$, $g : \mathcal{P}(\mathcal{B}) \to N_5$ are given by $f(\top) = \{\top, \bot\}$ and $g(\{\top\}) = a$, $g(\{\bot\}) = c$ We then have $\mathcal{J}(f) = \{(\top, \{\bot\}), (\top, \{\top\})\}$, $\mathcal{J}(g) = \{(\{\bot\}, a), (\{\top\}, c)\}$. Thus $\mathcal{J}(g \circ f) = \{(\top, a), (\top, b), (\top, c)\}$, but $\mathcal{J}(g) \circ \mathcal{J}(f) = \{(\top, a), (\top, c)\}$!
Hence even in the finite case $\mathcal{J}$ is not necessarily strict.

Proposition 4.4. *For $f : X \to Y$ and $g : Y \to Z$, with Y being such that $\mathcal{J}(Y)$ is join dense in Y and Z a frame, then $\mathcal{J}(g \circ f) = \mathcal{J}(g) \circ \mathcal{J}(f)$.*

The notion of residual finite state automaton (RFSA) was introduced in [6] to address the problem of finding a good notion of minimal automaton for non deterministic finite automata (NFA). In [4] they constructed a category of NFA and noticed that their procedure gives us a good reason why NFA are so badly behaved. Namely the category that generates them, REL has very few limits and colimits, which makes it fail to be a regular category. This means that we have no obvious notion of image factorization that we can use to construct a minimal automaton.
With this motivation in mind it makes sense to look for a restricted version of NFA that has a good notion of minimal automaton. We quickly recall the definition of RFSA from [6].

Definition 4.6. A **residual finite state automaton** RFSA over an alphabet Σ accepting a language L is an NFA $A = (Q, \delta, I, F)$ accepting L such that for state $q \in Q$, there is a word u such that $L_q = u^{-1}L$. Where L_q is the language recognized from q.

[5] This is in fact a lax functor once we view REL and JSL in a 2-categorical setting

Definition 4.7 (Prime Residual)**.** Given a regular language L, we define the residual lattice of L as the set of all language derivatives $u^{-1}L$, together with set union as lattice structure. We denote this set as $Res(L)$.
A **prime residual** of L is a join irreducible element of $Res(L)$.

Example 4.7.1. First recall that any regular language has a finite number of residuals.
Let $L = a^*b^* + b^*a^*$, then $Res(L)$ is computed via:

$$\begin{aligned}
a^{-1}L &= a^*b^* \\
b^{-1}L &= b^*a^* \\
(ab)^{-1}L &= b^* \\
(ba)^{-1}L &= a^*
\end{aligned}$$

The prime residuals are then a^*b^*, b^*a^*, b^* and a^*, since we cannot write them as unions of other residuals.

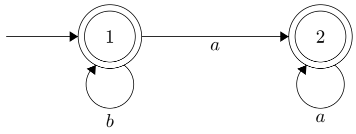
Definition 4.8 (Minimal RFSA)**.** [6, Section 4.5] Given a regular language L, the **minimal RFSA** $A = (Q, \delta, I, F)$ recognizing L is given by:

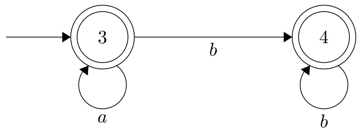
$$\begin{aligned}
&Q = \{w^{-1}L \mid w^{-1}L \texttt{ prime residual}\} = \mathcal{J}(Res(L)) \\
&I = \{w^{-1}L \mid w^{-1}L \subseteq L\} \\
&F = \{w^{-1}L \mid \epsilon \in w^{-1}L\} \\
&\delta_a(w^{-1}L) = \{v^{-1}L \in Q \mid v^{-1}L \subseteq (wa)^{-1}L\}
\end{aligned}$$

Proposition 4.5. *$\mathcal{J} \circ A_{min}$, where A is the minimal JSL-automaton accepting L is generates the canonical RFSA recognizing L. This automaton denoted $\mathcal{J}_g(L)$ is generated by each letter transition of $\mathcal{J} \circ A_{min}$ via its graph.*

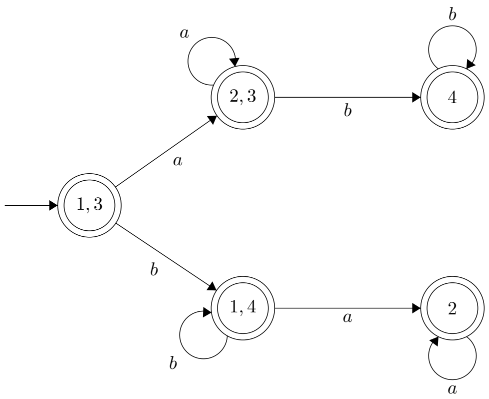
Lemma 4.6. *$J \circ A$ and A accept the same language.*

Example 4.8.1. We will now give a worked example from the original paper [6] using our framework.
We start off with the following RFSA recognizing $a^*b^* + b^*a^*$:





Then we determinize and keep the reachable part of the JSLA:



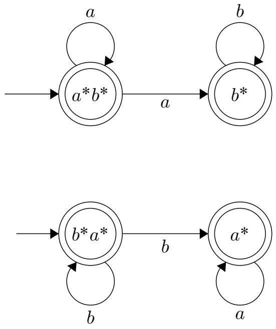
Now we can compute the minimal RFSA, by computing the language derivatives by hand:

$$
\begin{aligned}
&L = a^*b^* + b^*a^* \\
&a^{-1}L = a^*b^* \\
&b^{-1}L = b^*a^* \\
&(ab)^{-1}L = b^* \\
&(ba)^{-1}L = a^*
\end{aligned}
$$

We see that the only prime derivatives $\mathcal{J}(Res(L))$ are:

$$\begin{aligned}
&a^{-1}L = a^*b^*\\
&b^{-1}L = b^*a^*\\
&(ab)^{-1}L = b^*\\
&(ba)^{-1}L = a^*
\end{aligned}$$

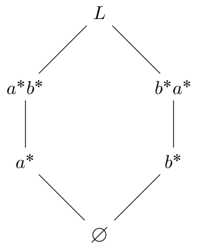
These then become our states:



Now we can do a sanity check and apply the observable states check. We get a mapping defined on the join irreducibles:

$$\begin{aligned}
&\{1,3\} \mapsto L\\
&\{2,3\} \mapsto a^*b^*\\
&\{4\} \mapsto b^*\\
&\{1,4\} \mapsto b^*a^*\\
&\{2\} \mapsto a^*
\end{aligned}$$

Note that the image of this map generates a new lattice:



With the exact 4 join irreducibles which we would expect.

We would like to view RFSA as NFA with extra structure. Notice that the definition of RFSA 4.6 has some some sort of assignment from states q to words w such that $L_q = w^{-1}L$. What we want to do is state everything related to RFSA in terms of REL morphisms. We propose the following definition of RFSA.

Definition 4.9. An RFSA over an alphabet Σ accepting a language L is an NFA A accepting L such that there is a relation $char : A(st) \nrightarrow A^{init}(st)$ such that the following commutes in REL, where the morphisms denoted by ! are the unique morphisms into the final object:

$$\begin{tikzcd}
A(st) \arrow[rr, "\shortmid" marking, "char"'] \arrow[dr, "\shortmid" marking, "!^{Q}_{st}"] & & A^{init}(st) \arrow[dl, "\shortmid" marking, "!^{init}_{st}"'] \\
& A^{final}(st) &
\end{tikzcd}$$

RFSA then denotes the full subcategory of NFA satisfying the existence of such a morphism.

We would now like to see if this definition makes sense:

Proposition 4.7. *The categorical definition and the usual one coincide.*

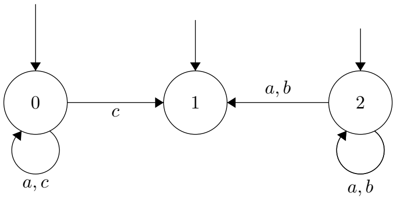
Proposition 4.8. *If $f : L \to L'$ is a regular* JSL *epimorphism where L satisfies DCC, that is a surjective jsl morphism, then $\mathcal{J}(f)$ is a surjective relation. Moreover, for each $y \in \mathcal{J}(Y)$ there is some $x \in \mathcal{J}(X)$ such that $f(x) = y$.*

Lemma 4.9. *K and J act strictly on initial and final automata, furthermore they preserve them.*

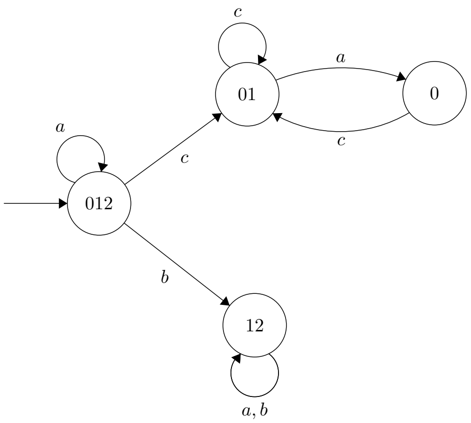
Proposition 4.10. *Given the automaton accepting the language L defined by $\mathcal{J}_g(L)$ is in fact an RFSA and coincides with the canonical RFSA.*

One could hope that the extra structure of the minimal jsla could be enough to make sure the applying $\mathcal{J}$ would give back a strict functor.
But we can find very small examples where even this fails.

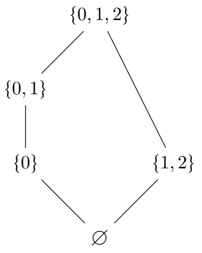
Example 4.9.1. We want to have an automaton which determinizes to an N_5 lattice. And has a top element as its initial state. We construct a similar counterexample as in 4.5.1. We want the initial state to be mapped to itself.
Take the following NFA A:



Which determinizes into $reach(\mathcal{K} \circ A)$:



This gives us an N_5 lattice:



We give the table for δ:

δ	a	b	c
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$\{0\}$	$\{0\}$	$\varnothing$	$\{0, 1\}$
$\{0, 1\}$	$\{0\}$	$\varnothing$	$\{0, 1\}$
$\{1, 2\}$	$\{1, 2\}$	$\{1, 2\}$	$\varnothing$
$\{0, 1, 2\}$	$\{0, 1, 2\}$	$\{1, 2\}$	$\{0, 1\}$

We could choose final states to make sure nothing gets identified in the obs step. So now applying $\mathcal{J}$, we get that $\mathcal{J}(\delta_a)[\{0,1,2\}] = \{\{0\},\{1,2\}\}$, but $\mathcal{J}(\delta_a(\{0,1,2\})) = \{\{0\},\{0,1\},\{1,2\}\}$. After seeing this example, it is not too hard to extend it to an example where one transition sends every non-empty state to $\{0,1,2\}$ playing the role of the initial map here. This means that in general $\mathcal{J}$ is not strict on the minimal JSLA for a language!

5 Conclusion

We noticed that making canonical NFA work in our framework quickly introduced the added complexity of having to consider the ordering that relations of the same domain and codomain introduce, which quickly lead us to find the right level of generality to deal with this problem. A lot of the problems are that we have no good way of interpreting the fact that the functor $\mathcal{J}$ is lax, this making any automaton composed with $\mathcal{J}$ potentially lax. The majority of the project was spent trying to understand the $\mathcal{J}$ functor on when it behaves nicely. By using this lax functor we found that the minimization of join-semilattices is deeply related to the minimization of RFSA, which confirms a certain folklore theorem about RFSA that had been flying around. The fact that strictness of the functor $\mathcal{J}$ and that it lead to questions about what a lax automaton could be shows that there is still a lot of work to be done in this area and that this topic is still worth exploring. We plan on continuing our research, especially on how to interpret the fact that the RFSA is generated by a lax functor and whether this is a more general phenomenon which requires further investigation. There is also interest in studying the $\mathcal{J}$ functor on its own, since it seems to be an example of a very lax kind of adjunction.

References

[1] Jiří Adámek, Robert S. R. Myers, Henning Urbat, and Stefan Milius. On Continuous Nondeterminism and State Minimality. *Electron. Notes Theor. Comput. Sci.*, 308:3–23, October 2014.

[2] Mikołaj Bojańczyk. Recognisable Languages over Monads. In *Developments in Language Theory*, pages 1–13. Springer, Cham, Switzerland, July 2015.

[3] Janusz Brzozowski and Hellis Tamm. Theory of átomata. *Theoretical Computer Science*, 539:13–27, 2014.

[4] Thomas Colcombet and Daniela Petrişan. Automata Minimization: a Functorial Approach. *Logical Methods in Computer Science*, 16, Issue 1, March 2020.

[5] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, England, UK, April 2002.

[6] François Denis, Aurélien Lemay, and Alain Terlutte. Residual Finite State Automata. *Fundam. Inform*, 51:339–368, January 2002.

[7] Pierre Ganty, Elena Gutiérrez, and Pedro Valero. A Quasiorder-based Perspective on Residual Automata. *arXiv*, July 2020.

[8] A. Joyal and M. Tierney. An extension of the Galois theory of Grothendieck. *Mem. Amer. Math. Soc.*, 1984.

[9] C. Kupke and Y. Venema. Coalgebraic Automata Theory: Basic Results. *Logical Methods in Computer Science*, 4, Issue 4, November 2008.

[10] Hendrik Maarand and Hellis Tamm. Yet Another Canonical Nondeterministic Automaton. In *Descriptional Complexity of Formal Systems*, pages 184–196. Springer, Cham, Switzerland, August 2022.

[11] Paul-André Melliès and Noam Zeilberger. Parsing as a lifting problem and the Chomsky-Schützenberger representation theorem, July 2022. [Online; accessed 13. Jun. 2024].

[12] E. Riehl. *Category theory in context*. Aurora: Dover modern math originals. Dover Publications, 2017.

[13] E. Riehl and E. Riehl. FACTORIZATION SYSTEMS, 2008. [Online; accessed 7. Mar. 2024].

[14] Hellis Tamm. Generalization of the Double-Reversal Method of Finding a Canonical Residual Finite State Automaton. In *Descriptional Complexity of Formal Systems*, pages 268–279. Springer, Cham, Switzerland, June 2015.