

# **Bachelor-Thesis**

# **3-Valued Automata**

**Alexander Loitzl**

A thesis presented in partial fulfillment for the degree of

**Bachelor of Science**



Department of Computer Science  
Faculty of Natural Sciences  
Paris-Lodron-University Salzburg  
Austria  
August 2021

**Advisor: Assoz. Prof. Ana Sokolova**

## Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass für die vorliegende Arbeit keine anderen als die angegebenen Quellen und Hilfsmittel verwendet wurden. Die Arbeit wurde selbstständig verfasst und Teile welche inhaltlich oder wörtlich aus den angegebenen Quellen stammen, wurden entsprechend gekennzeichnet.

Diese Arbeit wurde bisher noch nicht in ähnlicher oder gleicher Form als Bachelorarbeit zur Beurteilung eingereicht.

Salzburg, August 2021

---

Alexander Loitzl

## **Zusammenfassung**

$\mathfrak{Z}$ -wertige Automaten sind eine Erweiterung endlicher Automaten, welche es ermöglichen Ungewissheit auszudrücken. Ursprünglich inspiriert durch die koalgebraische Perspektive definieren wir deterministische und nicht-deterministische Automaten welche die gleiche Klasse von Sprachen beschreiben. Wir zeigen, dass die sogenannten  $\mathfrak{Z}$ -regulären Sprachen abgeschlossen bezüglich der definierten Operationen sind. Weiters, bildet die Klasse der  $\mathfrak{Z}$ -regulären Sprachen mit ausgewählten Operationen eine Kleene Algebra. Wir führen  $\mathfrak{Z}$ -reguläre Ausdrücke ein und zeigen, mithilfe eines adaptierten Kleene Theorems, deren Äquivalenz zu den  $\mathfrak{Z}$ -regulären Sprachen. Ziel der Arbeit ist es als erster Schritt beim Entwurf eines allgemeineren Verband (Lattice) Automaten zu dienen.

## Abstract

$\mathfrak{Z}$ -valued automata are a simple extension of finite automata, allowing to express uncertainty. Drawing inspiration from the coalgebraic perspective we define deterministic as well as nondeterministic automata and give an adapted determinization procedure. We reproduce some of the standard results on regular languages for our new class of languages. We show closure under the defined operations and that the class of so-called  $\mathfrak{Z}$ -regular languages together with our operations satisfies the properties of a Kleene Algebra. We introduce  $\mathfrak{Z}$ -regular expressions and show their equivalence to  $\mathfrak{Z}$ -regular languages through a theorem similar to the Kleene Theorem. The work is intended to serve as a first step in introducing a more general class of lattice automata.

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Preliminaries</b>	<b>2</b>
1.1 Finite Automata . . . . .	2
1.1.1 Deterministic Finite Automata . . . . .	2
1.1.2 Nondeterministic Finite Automata . . . . .	4
1.1.3 Regular Languages . . . . .	7
1.2 Few notions from lattice theory . . . . .	8
1.2.1 Lattice as a poset . . . . .	8
1.2.2 Lattice as an algebra . . . . .	9
1.2.3 Three element semilattices . . . . .	10
1.3 Automata from a coalgebraic perspective . . . . .	12
<b>2 <math>\mathfrak{J}</math>-valued automata and <math>\mathfrak{J}</math>-regular languages</b>	<b>14</b>
2.1 $\mathfrak{J}$ -valued Automata . . . . .	14
2.1.1 Deterministic $\mathfrak{J}$ -valued Automata . . . . .	14
2.1.2 Nondeterministic $\mathfrak{J}$ -valued Automata . . . . .	17
2.1.3 Determinization . . . . .	18
2.1.4 Language equivalence of automata . . . . .	19
2.2 $\mathfrak{J}$ -regular Languages . . . . .	22
2.2.1 Operators and Properties of $\mathfrak{J}$ -regular languages . . . . .	23
2.2.2 $\mathfrak{J}$ -regular expressions . . . . .	26
<b>3 Results on automata and language</b>	<b>28</b>
3.1 Closure Properties . . . . .	28
3.1.1 Union . . . . .	28
3.1.2 Concatenation . . . . .	29
3.1.3 Kleene star . . . . .	31
3.1.4 Complement . . . . .	33
3.1.5 Intersection . . . . .	33
3.2 Kleene Theorem for $\mathfrak{J}$ -regular languages . . . . .	34
3.3 Kleene Algebra . . . . .	35
3.3.1 Idempotent Semiring . . . . .	36
3.3.2 Axioms involving Kleene Star . . . . .	40
<b>Conclusion</b>	<b>43</b>

# Introduction

With the ever-growing use and demand of computer systems by the industry, but also by us in our everyday life, the complexity and use cases rise. To be able to reason about their behavior, several computational models exist, each providing certain desirable aspects in the field they are used in. We are particularly interested in a special case of a Moore Machine [9], which are computational models that are similar to deterministic finite automata, but differ in their output. While deterministic finite automata have two output values, namely accept and reject, Moore Machines have a finite set of output values. We introduce an additional output value  $\perp$  representing an uncertain output.

Such a model already exists, see e.g. [1] where it is used in the context of runtime verification allowing to give answers of uncertainty. Moore Machines with three output values are used to evaluate  $LTL_3$  formulae, an extension of linear temporal logic (LTL), allowing to evaluate neither to true or false when only given a finite prefix of an infinite trace. The focus lies on the application in the specified setting, whereas in this work we pay attention to the semantics of such automata and the languages they recognize. Moreover, to introduce nondeterminism for our so-called  $\mathfrak{Z}$ -valued automata, we regard the set of outputs to have the structure of a join-semilattice with bottom.

Moore Machines have also been studied in the field of Coalgebra, which provides an abstract framework allowing the definition of several dynamic systems. In this framework a simple definition of Moore Machines exists and finite automata are a special case with only two outputs, see e.g. [6]. In [10], an approach is presented which allows deriving generalized expressions for several kinds of coalgebraic systems. This approach is also applicable to Moore Machines, but the expressions are not equipped with the standard notion of concatenation or Kleene star.

Building on what already exists, in this thesis we focus on giving a concise definition of a  $\mathfrak{Z}$ -valued automaton and their languages. We show that standard results on finite automata also hold for our new class of automata. We give a determinization algorithm and define  $\mathfrak{Z}$ -regular expressions including a notion of concatenation and Kleene Star. We then show the equivalence of  $\mathfrak{Z}$ -regular expressions and  $\mathfrak{Z}$ -regular languages with a theorem similar to the Kleene Theorem for regular languages. Moreover, we show that the resulting languages, with accordingly defined operations, satisfy all properties of a Kleene algebra. This work shall serve as a first step in the ongoing work of studying lattice automata, in particular their languages and expressions.

# 1 | Preliminaries

## 1.1 Finite Automata

In this chapter we review the theory of finite automata and recall few basic notions from algebra that we will need later. The upcoming section is meant as a brief introduction and recollection of the standard notions. In addition, the section aims to highlight the important properties that are put into focus when introducing a new class of languages. The notation, sometimes tweaked to our advantage, is based on the standard literature on finite automata [12, 5].

A finite automaton is a simple computational model which processes an input word, solely by scanning it from left to right. It then decides if it should be accepted or not. A word  $w$  is a finite concatenation of symbols from an alphabet  $\Sigma$ . We write  $w \in \Sigma^*$ , where  $\Sigma^*$  is the set containing all finite sequences of symbols in  $\Sigma$ . Note that also the sequence of 0 symbols, the so-called empty word denoted by  $\varepsilon$ , is in the set  $\Sigma^*$ . A finite automaton, in its "simplest" deterministic form, consists of states, including one initial state and any number of accepting states. In each state, for every symbol of the alphabet  $\Sigma$ , a unique transition is defined. These specify how one can move from one state to another. A computation always starts in the initial state and for every read symbol, a transition is performed. These can lead to another state, but also loops are allowed. If all symbols have been read, the state in which the computation ended is considered: if it is an accepting state, the automaton accepts the input, otherwise the input is rejected.

In Figure 1.1 we can already see an example of a finite automaton. The incoming arrow, labeled start, identifies the initial state  $q_0$ . Each remaining arrow depicts the transition that is performed after reading the symbol it is labeled with. The words 1001 and 0101 are being accepted as they reach the doubly lined accepting state, whereas the inputs 1100 and 0011 are being rejected, as the computation ends in the non-accepting states  $q_1$  and  $q_0$  respectively. The set of all such accepted words is called the language of the automaton.

### 1.1.1 Deterministic Finite Automata

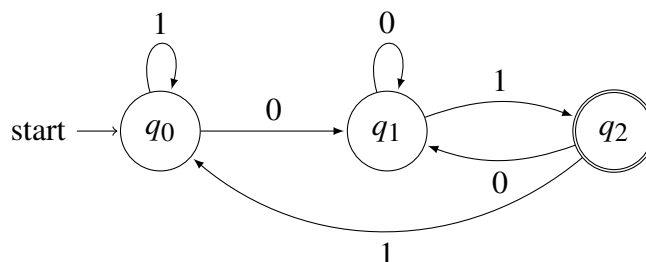


Figure 1.1: Transition diagram of a DFA

When trying out the example inputs for the automaton, we can observe that for every input there is a unique way of transitioning through the system. This concept is called determinism, and we ensure this by having exactly one transition for every symbol leaving a state, i.e., a deterministic choice of a next state for each symbol that has been read.

**Definition 1.1.1.** A deterministic finite automaton (DFA) is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  with:

- $Q$  a finite set of states,
- $\Sigma$  a finite set of input symbols,
- $\delta : Q \times \Sigma \rightarrow Q$  the transition function,
- $q_0 \in Q$  denoting the initial or starting state, and
- $F \subseteq Q$  the set of accepting states.

We write, as usual,  $p \xrightarrow{a} q$  for  $p, q \in Q$ ,  $a \in \Sigma$  and  $\delta(p, a) = q$ .

**Definition 1.1.2.** A path  $\pi$  of length  $n$  in an automaton  $M = (Q, \Sigma, \delta, q_0, F)$  is a finite alternating sequence of states and symbols, denoted  $\pi = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n$ , such that  $p_i \in Q$  for  $i = 0, \dots, n$ ,  $a_i \in \Sigma$  for  $i = 1, \dots, n$  and  $\delta(p_i, a_{i+1}) = p_{i+1}$ , i.e.,  $p_i \xrightarrow{a_{i+1}} p_{i+1}$ , for  $i = 1, \dots, n$ .

Note that for a path  $\pi = p_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} p_n$ , the last state  $p_n$  is reachable by a sequence of transitions

$$p_n = \delta(\dots(\delta(\delta(p_0, a_1), a_2) \dots), a_n).$$

We now extend our transition function  $\delta$ , to the so-called extended transition function  $\delta^*$  that can take entire words as input, allowing for a concise way to talk about the states that are reached on execution.

**Definition 1.1.3.** The extended transition function  $\delta^*(q, w)$  for a state  $q \in Q$  and a word  $w \in \Sigma^*$  is defined inductively:

$$\delta^*(q, w) = \begin{cases} q & \text{if } w = \varepsilon, \\ \delta(\delta^*(q, x), a) & \text{if } w = xa, x \in \Sigma^*, a \in \Sigma. \end{cases}$$

### Language of a DFA

Having defined  $\delta^*$ , we now have a simple way to talk about the language of a state.

**Definition 1.1.4.** The language of a state  $q \in Q$  in a DFA  $M = (Q, \Sigma, \delta, q_0, F)$  is denoted by  $L(q)$  and given by

$$L(q) = \{w \in \Sigma^* \mid \delta^*(q, w) \in F\}.$$

Clearly, this gives us a language map  $L : Q \rightarrow \mathcal{P}(\Sigma^*)$ .

Hence, the language of a state  $q$  consists of all words from which an accepting state is reached, starting from  $q$ . With these definitions in place, we can finally talk about the language of an automaton, i.e., the set of all accepted words.

**Definition 1.1.5.** *The recognized language  $L$  of a deterministic finite automaton  $M$ , the language of  $M$  for short, is the set of words accepted from the initial state and denoted by  $L(M)$ , i.e.,*

$$L(M) = L(q_0).$$

All such languages, for which a recognizing DFA exists, form the class of regular languages.

**Definition 1.1.6.** *A language  $L$  is a regular language if there exists a DFA  $M$  with  $L(M) = L$ .*

We can now turn our attention back to our example automaton of Figure 1.1 and give the formal definition  $M = (Q, \Sigma, \delta, q_0, F)$  with:

- $Q = \{q_0, q_1, q_2\}$ ,
- $\Sigma = \{0, 1\}$ ,
- $\delta$  defined with a so-called transition table, given in Table 1.1, and
- $F = \{q_2\}$ .

$\delta$	0	1
$q_0$	$q_1$	$q_0$
$q_1$	$q_1$	$q_2$
$q_2$	$q_1$	$q_0$

Table 1.1: Transition table of  $\delta$  for the DFA in Figure 1.1

With some observation one can also find a concise way of defining the language  $L(M)$  of our automaton, namely  $L(M) = \{w \in \Sigma^* \mid w \text{ ends with } 01\}$ . Whatever the prefix of any word, the transition  $q_1 \xrightarrow{1} q_2$  is required to reach an accepting state. We can only reach  $q_1$  after reading a 0. Moreover, we leave the only accepting state  $q_2$ , if we were to read another symbol.

It is also important to understand that if two automata recognize the same language, they are not necessarily identical. Not only can they of course differ in the labeling of states, but also in the number of states and the overall structure.

**Definition 1.1.7.** *Two automata  $M_1$  and  $M_2$  are equivalent, iff  $L(M_1) = L(M_2)$ .*

## 1.1.2 Nondeterministic Finite Automata

In a DFA, in every state, there was a unique transition for each symbol. This is no longer true in case of nondeterminism. A nondeterministic finite automaton can follow several paths of computation and there is not necessarily a unique transition for a given symbol. It can also make a transition without reading an input symbol.

**Definition 1.1.8.** A *nondeterministic finite automaton (NFA)* is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  with:

- $Q$  a finite set of states,
- $\Sigma$  a finite set of input symbols,
- $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$  the transition function, where  $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ ,
- $q_0 \in Q$  denoting the initial or starting state, and
- $F \subseteq Q$  the set of accepting states.

Note that now the transition function maps from  $Q \times \Sigma_\epsilon$  to the powerset  $\mathcal{P}(Q)$  rather than to  $Q$ , as in a deterministic automaton. This allows several arrows labeled with the same symbol from a given state, as transitions map to a subset of  $Q$ . To be precise, we write  $p \xrightarrow{a} q$  for  $p, q \in Q$ ,  $a \in \Sigma_\epsilon$  and  $q \in \delta(p, a)$ . Note that additionally  $\epsilon$ -transitions are introduced, which allows for a transition without reading an input symbol.

The NFA in Figure 1.2 models both newly introduced concepts. In  $q_0$ , when reading a 0, we can make a nondeterministic choice to either stay in  $q_0$  or make a step to  $q_1$ . In  $q_1$  we can always decide to move back to  $q_0$  without reading a symbol. In  $q_1$  the outgoing arrow labeled by 0 is omitted, as  $\delta(q_1, 0) = \emptyset$ . The same applies to  $q_2$  for both symbols 0, 1. This automaton is fairly similar to the DFA in Figure 1.1, it even recognizes the same language! We only redefine  $\delta$  in Table 1.2 as it is the only component that is subject to change.

$\delta$	0	1	$\epsilon$
$q_0$	$\{q_0, q_1\}$	$\{q_0\}$	$\emptyset$
$q_1$	$\emptyset$	$\{q_2\}$	$\{q_0\}$
$q_2$	$\emptyset$	$\emptyset$	$\emptyset$

Table 1.2: Transition table for  $\delta$  of the NFA in Figure 1.2

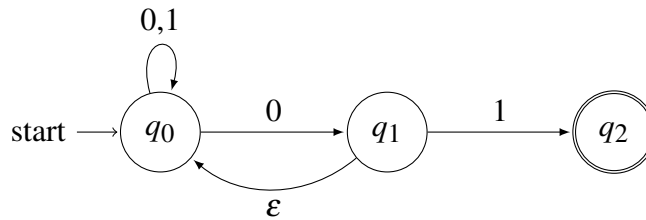


Figure 1.2: Transition diagram of an NFA

In order to define the extended transition function of an NFA we first need to introduce the epsilon closure of a state  $q$ , denoted  $E(q)$ . It includes all states that can be reached from  $q$  by following any number of transitions without reading any input symbol. In our example NFA in Figure 1.2,  $E(q_1)$  is  $\{q_0, q_1\}$ , as we have an explicit  $\epsilon$ -transition from  $q_1$  to  $q_0$  and as without reading a symbol one stays in the same state per default. We next give the formal definition of  $E(q)$ .

**Definition 1.1.9.** *The epsilon closure of a state  $q$ ,  $E(q)$  is defined inductively:*

- $q \in E(q)$ ,
- $p \in E(q)$ , if there exists  $r \in E(q)$  with  $r \xrightarrow{\varepsilon} p$ .

We now also define the extended transition function for an NFA. Here  $\varepsilon$ -transitions are of relevance, as one can always follow such a transition without reading any input. Therefore after every step, the epsilon closure of all reachable states is considered, rather than just the states themselves.

**Definition 1.1.10.** *The extended transition function  $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$  of an NFA for a state  $q \in Q$  and a word  $w \in \Sigma^*$  is defined inductively:*

$$\delta^*(q, w) = \begin{cases} E(q) & \text{if } w = \varepsilon, \\ E\left(\bigcup_{p \in \delta^*(q, x)} \delta(p, a)\right) & \text{if } w = xa, x \in \Sigma^* \text{ and } a \in \Sigma, \end{cases}$$

where for a set  $X \subseteq Q$ ,  $E(X) = \bigcup_{q \in X} E(q)$ .

**Definition 1.1.11.** *The language  $L$  of a state  $q \in Q$  in an NFA  $M$  is now*

$$L(q) = \{w \in \Sigma^* \mid \delta^*(q, w) \cap F \neq \emptyset\}$$

and the language of an NFA  $M$  is again simply  $L(M) = L(q_0)$ .

Hence, a word is accepted by an NFA if and only if there exists a path with this word from the initial state to an accepting state. In other words, we only reject the input, if all possible paths end in rejecting states.

Although NFA seem to be more powerful than DFA, one can show that the class of languages accepted by NFA is the same as those by their deterministic counterpart. This can be done, by giving a procedure to construct a DFA  $D$  from any given NFA  $N$ , with  $L(D) = L(N)$ . This mentioned procedure is called determinization or the powerset construction. Therefore we get the following theorem, whose proof we omit at this point. In Section 2.1.3 we show that this also holds for our new class of automata via a proof similar to that for finite automata.

**Theorem 1.1.12 ([5]).** *The following are equivalent, for a Language  $L$ .*

1.  $L$  is regular.
2.  $L$  is being recognized by a DFA.
3.  $L$  is being recognized by an NFA.

### 1.1.3 Regular Languages

We already know that a language  $L$  is regular if there exists a finite automaton recognizing it. Now we introduce a new concept called regular expressions and briefly discuss the equivalence of languages generated by regular expressions and the class of regular languages. But first, we define some operations on regular languages.

#### Operators and Properties of Regular Languages

The three regular operators are union, concatenation, and Kleene star. Furthermore, we will give the definitions of the complement of a language and the intersection of two regular languages.

**Definition 1.1.13.** *Let  $L, L'$  be regular languages over a common alphabet  $\Sigma$ .*

- *The union of  $L$  and  $L'$  is just their set union, i.e.,  $L \cup L' = \{w \mid w \in L \vee w \in L'\}$ .*
- *The concatenation of  $L$  and  $L'$  is the language  $L \cdot L' = \{w \cdot w' \mid w \in L \wedge w' \in L'\}$ .*
- *The Kleene star of  $L$  is  $L^* = \{w_1 w_2 \cdots w_n \mid n \in \mathbb{N} \wedge \forall i \leq n. w_i \in L\}$ .*
- *The complement of  $L$  is  $L^c = \{w \in \Sigma^* \mid w \notin L\}$ .*
- *The intersection of  $L$  and  $L'$  is just their set intersection, i.e.,  $L \cap L' = \{w \mid w \in L \wedge w \in L'\}$ .*

**Theorem 1.1.14** ([5]). *The class of regular languages is closed under union, intersection, concatenation, complement, and Kleene star.*

These properties are called the closure properties. All, see e.g. [12, 5], break down to constructing a suitable NFA accepting the language.

#### Regular Expressions

**Definition 1.1.15.** *For a given alphabet  $\Sigma$  we call  $R$  a regular expression, if it is either:*

1.  $a$ , where  $a \in \Sigma$ ,
2.  $\varepsilon$ ,
3.  $\emptyset$ ,
4.  $(R_1 \cup R_2)$ , for regular expressions  $R_1, R_2$ ,
5.  $(R_1 \cdot R_2)$ , for regular expressions  $R_1, R_2$ ,
6.  $(R_1)^*$ , for regular expression  $R_1$ ,

*and its language is given by:*

1.  $L(a) = \{a\}$ ,
2.  $L(\varepsilon) = \{\varepsilon\}$ ,
3.  $L(\emptyset) = \emptyset$ ,
4.  $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$ ,

5.  $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2)$ ,
6.  $L(R_1^*) = L(R_1)^*$ .

As usual, we often identify a regular expression with the language it denotes, e.g., we speak of the language  $0^*$ . The equivalence of regular expressions and regular languages is a well-known result of Kleene.

**Theorem 1.1.16** (Kleene's Theorem [7]). *A language  $L$  over an alphabet  $\Sigma$  is a regular language iff it is the language of a regular expression.*

**Example 1.1.17.** We give some simple examples of languages defined by regular expressions.

$$\begin{aligned} 0^* &= \{w \mid w \text{ contains only } 0s\}. \\ (0 \cup 1)^* 0 &= \{w \in \{0, 1\}^* \mid w \text{ ends with a } 0\}. \\ (0 \cup 1)^* 1 (0 \cup 1)^* &= \{w \in \{0, 1\}^* \mid w \text{ contains at least one } 1\}. \end{aligned}$$

## 1.2 Few notions from lattice theory

Before we can proceed to introduce our new class of automata, we will review basic lattice theory, or rather give the most important definitions. This enables a nice and easy way to define the properties of our automata and allows us to talk about them in a concise way. Lattices can be defined either as a poset or as an algebra. We are particularly interested in their algebraic properties. Nonetheless, we shall first introduce them as a special partial order, as it provides more of an intuitive sense of the overall structure.

### 1.2.1 Lattice as a poset

A poset (a partially ordered set) is called a join-semilattice if for every two-element subset there exists a least upper bound, a supremum. For two elements  $a, b \in S$ , of a join-semilattice  $L = (S, \leq)$ , we denote the supremum or join of the two as  $a \sqcup b$ . Dually, if for every two-element subset a greatest lower bound or infimum exists, we get the notion of a meet-semilattice. We denote the infimum or meet of two elements as  $a \sqcap b$ . [13]

We can now give a simple example of a semilattice, namely the subset-inclusion relation  $\subseteq$  on  $\mathcal{P}(\{0, 1\})$  in Figure 1.3. We can see, that this partial order is a join-semilattice, where for the join of two elements  $a, b \in \mathcal{P}(\{0, 1\})$ , we get  $a \sqcup b = a \cup b$ . Moreover, it is also a meet-semilattice with  $a \sqcap b = a \cap b$ , for any  $a, b \in \mathcal{P}(\{0, 1\})$ .

**Definition 1.2.1** ([13]). *A lattice is a partially ordered set that is both a meet- and join-semilattice.*

**Definition 1.2.2** ([13]). *A complete lattice is a lattice in which there exists a join and meet for every subset.*

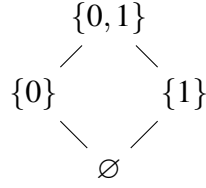


Figure 1.3: Hasse diagram of the subset-inclusion relation  $\subseteq$  on  $\mathcal{P}(\{0,1\})$

**Proposition 1.2.3** ([13]). *Every nonempty, finite lattice is complete.*

Therefore, the poset depicted in Figure 1.3 is in fact a lattice. Since it is finite, it is also complete. We now proceed to discuss lattices from an algebraic viewpoint and eventually connect the two definitions.

## 1.2.2 Lattice as an algebra

**Definition 1.2.4** ([2]). *An algebraic semilattice  $\langle S, \diamond \rangle$  is an algebraic structure with a set  $S$  and a binary operator  $\diamond$ , satisfying the following properties for all  $x, y, z \in S$ .*

- *Associativity:*  $(x \diamond y) \diamond z = x \diamond (y \diamond z)$ .
- *Commutativity:*  $x \diamond y = y \diamond x$ .
- *Idempotence:*  $x \diamond x = x$ .

**Definition 1.2.5.**

- $\langle S, \diamond, \perp \rangle$  *is a semilattice with bottom* if  $\langle S, \diamond \rangle$  is an algebraic semilattice and  $\perp$  is a bottom element, i.e.,  $\perp \in S$  and  $\forall s \in S. s \diamond \perp = s$ .
- $\langle S, \diamond, \top \rangle$  *is a semilattice with top* if  $\langle S, \diamond \rangle$  is an algebraic semilattice and  $\top$  is a top element, i.e.,  $\top \in S$  and  $\forall s \in S. s \diamond \top = \top$ .

**Definition 1.2.6** ([2]). *An algebraic lattice  $\langle S, \sqcap, \sqcup \rangle$  is an algebraic structure with a set  $S$  and two binary operators  $\sqcap$  and  $\sqcup$  satisfying commutativity, associativity, idempotence, and the absorption laws, i.e, for all  $x, y \in S$ :*

1.  $x \sqcup y = y \sqcup x$ ,  
 $x \sqcap y = y \sqcap x$ ,
2.  $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$ ,  
 $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$ ,
3.  $x \sqcup x = x$ ,  
 $x \sqcap x = x$ ,
4.  $x \sqcap (x \sqcup y) = x$ , and  
 $x \sqcup (x \sqcap y) = x$ .

The absorption laws ensure the right interaction between the two semilattices formed by  $\sqcup$  and  $\sqcap$ , namely that they are the dual of each other. Moreover one may note that in this definition, idempotence is actually not required. First consider the meet operation. Using the second absorption law, we substitute for  $x$ . By considering the term  $(x \sqcap y)$  as a value  $z \in S$  we can make use of the first absorption law to get  $x$ . For arbitrary  $x, y \in S$

$$x \sqcap x = x \sqcap (x \sqcup \overbrace{(x \sqcap y)}^z) = x \sqcap (x \sqcup z) = x.$$

We can apply the same idea to show idempotence of the join operation, by using the two absorption laws in reverse order. For arbitrary  $x, y \in S$

$$x \sqcup x = x \sqcup (x \sqcap \overbrace{(x \sqcup y)}^z) = x \sqcup (x \sqcap z) = x.$$

**Proposition 1.2.7** ([2]). *Let  $(S, \leq)$  be a join- (or meet-) semilattice. Then  $\langle S, \sqcup \rangle$  (or  $\langle S, \sqcap \rangle$ ) is an algebraic semilattice.*

**Proposition 1.2.8** ([13]). *Let  $\langle S, \diamond \rangle$  be an algebraic semilattice.*

*Define an order  $\leq$  on  $S$  by  $a \leq b \Leftrightarrow a \diamond b = b$ . Then  $(S, \leq)$  is a join-semilattice.*

*Define an order  $\leq$  on  $S$  by  $a \leq b \Leftrightarrow a \diamond b = a$ . Then  $(S, \leq)$  is a meet-semilattice.*

**Proposition 1.2.9** ([2]). *Let  $\langle S, \sqcap, \sqcup \rangle$  be an algebraic lattice. Then  $(S, \leq)$  is a lattice with:*

$$a \leq b \Leftrightarrow a \sqcup b = b$$

*or equivalently*

$$a \leq b \Leftrightarrow a \sqcap b = a.$$

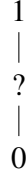
Hence, (semi) lattices as posets and algebraic (semi) lattices coincide and we no longer distinguish them and omit the prefix "algebraic" when we talk about them in their algebraic sense.

### 1.2.3 Three element semilattices

Lastly, we shall look at three-element semilattices. There exist exactly two three-element semilattices up to duality and isomorphism. Only one of them is bounded. We focus on this semilattice, depicted in Figure 1.4, and call it  $\mathfrak{3}$ . We will use  $\mathfrak{3}$  to define our new class of automata.

In fact  $\mathfrak{3}$  is not only a semilattice. From the order in Figure 1.4 we can derive a join  $\sqcup$  and a meet  $\sqcap$  such that it becomes a lattice. Moreover, as it is finite, we get a complete lattice  $\langle \{0, ?, 1\}, \sqcap, \sqcup, 1, 0 \rangle$  with bottom element  $\perp = 0$  and top element  $\top = 1$ .

In the upcoming section we will discuss why a join semilattice with bottom is of interest to us, but we can first show that only  $\mathfrak{3}$  fits for our new class of automata.

Figure 1.4: Hasse Diagram of  $\mathfrak{3}$ 

**Lemma 1.2.10.**  $\mathfrak{3}$  is, up to isomorphism, the only three element join-semilattice with bottom.

*Proof.* Let  $\langle S, \sqcup, a \rangle$  be a join-semilattice with bottom. Let  $S = \{a, b, c\}$  and w.l.o.g.  $a$  be the bottom element. Then we know  $a \sqcup b = b$  and  $a \sqcup c = c$ . Due to idempotence and commutativity we can derive the join for almost all pairs of elements, depicted in Table 1.3, with  $x \in S$ .

$\sqcup$	$a$	$b$	$c$
$a$	$a$	$b$	$c$
$b$	$b$	$b$	$x$
$c$	$c$	$x$	$c$

Table 1.3: Join of a three element semilattice with bottom  $a$ 

First assume  $x = a$ , then associativity no longer holds, since

$$\begin{aligned}
 (b \sqcup c) \sqcup c &= a \sqcup c = c, \text{ and} \\
 b \sqcup (c \sqcup c) &= b \sqcup c = a.
 \end{aligned}$$

Hence, the missing join cannot be the bottom element. Therefore, we get the two options  $x = b$  and  $x = c$ . Let us now consider the two resulting join-semilattices  $\langle S, \sqcup_1 \rangle$  and  $\langle S, \sqcup_2 \rangle$  with their joins defined in Tables 1.4 and 1.5.

$\sqcup_1$	$a$	$b$	$c$
$a$	$a$	$b$	$c$
$b$	$b$	$b$	$b$
$c$	$c$	$b$	$c$

Table 1.4: Definition of  $\sqcup_1$ 

$\sqcup_2$	$a$	$b$	$c$
$a$	$a$	$b$	$c$
$b$	$b$	$b$	$c$
$c$	$c$	$c$	$c$

Table 1.5: Definition of  $\sqcup_2$ 

Now consider  $f : S \rightarrow S$  with  $f(a) = a$ ,  $f(b) = c$ ,  $f(c) = b$ . One can then show that  $f$  is an isomorphism by showing  $f(x \sqcup_1 y) = f(x) \sqcup_2 f(y)$  for arbitrary  $x, y \in S$ . The resulting mappings are equal and depicted in Table 1.6.

$\sqcup_2 \circ (f \times f) = f \circ \sqcup_1$	$a$	$b$	$c$
$a$	$a$	$c$	$b$
$b$	$c$	$c$	$c$
$c$	$b$	$c$	$b$

Table 1.6: The resulting mapping of  $f(x \sqcup_1 y)$  and  $f(x) \sqcup_2 f(y)$ 

□

### 1.3 Automata from a coalgebraic perspective

In this section we briefly discuss the generalization of systems, in particular automata, from the perspective of coalgebra. We can then see how to naturally adapt the semantics of finite automata to support three output values. The notions are mainly drawn from [6].

We now describe a deterministic automaton as a map. It has an output function  $o$ , assigning an output to each state and a transition function  $t$ , describing the transitions in our automaton. We write  $Q^\Sigma$  to denote the set of all functions  $f : \Sigma \rightarrow Q$ . The set of states is implicitly given by the domain of the two functions. Note, that in this definition there is no initial state and neither the set of states  $Q$ , nor the set of symbols  $\Sigma$  is necessarily finite. If we restrict both sets to be finite, we can speak of deterministic finite automata.

**Definition 1.3.1.** *A coalgebraic deterministic automaton is a map  $\langle o, t \rangle : Q \rightarrow O \times Q^\Sigma$  with:*

- $o : Q \rightarrow O$  the output function mapping states to outputs in a set of observations  $O$ ,
- $t : Q \rightarrow Q^\Sigma$  the transition function mapping a state  $q$  to a function  $t(q) : \Sigma \rightarrow Q$ .

Until now, in the coalgebraic sense, we spoke of deterministic automata with an arbitrary set of observations  $O$ . To represent deterministic finite automata, we can simply fix the set of observations to  $2 = \{0, 1\}$ . Then our example DFA in Figure 1.1, omitting the definition of its starting state, can be represented by an output function  $o : Q \rightarrow 2$  and the transition function  $\delta : Q \rightarrow Q^\Sigma$ . Note that, we can equivalently express the transition function as a function  $\delta : Q \times \Sigma \rightarrow Q$ , just as for the standard definition of DFA. Therefore, for the transition function, we can reuse the definition given in Table 1.1 and we give the output function  $o$  in Table 1.7. Moreover, one can interpret  $F \subseteq Q$ , the set of accepting states, simply as a function mapping states to 1 if they are in the set and 0 otherwise.

	$q_0$	$q_1$	$q_2$
$o$	0	0	1

Table 1.7: Output function  $o$  for the coalgebraic definition of the DFA in Figure 1.1

For coalgebraic automata, we can just reuse the definitions of an extended transition function and the language of a state from Section 1.1. To introduce nondeterminism just the transition function has to be adapted. As the set of states is not necessarily finite, we write  $\mathcal{P}_{fin}(Q) = \{X \mid X \subseteq Q, X \text{ is finite}\}$  to denote the finite powerset.

**Definition 1.3.2.** *A coalgebraic nondeterministic automaton is a map  $\langle o, t \rangle : Q \rightarrow O \times \mathcal{P}_{fin}(Q)^\Sigma$  with:*

- $o : Q \rightarrow O$  the output function mapping states to outputs in a set of observations  $O$ ,
- $t : Q \rightarrow \mathcal{P}_{fin}(Q)^\Sigma$  the transition function mapping a state  $q$  to a function  $t(x) : \Sigma \rightarrow \mathcal{P}_{fin}(Q)$ .

In [11] a generalized powerset construction is introduced to determinize not only automata but also several other kinds of coalgebraic systems. To apply the algorithm to nondeterministic automata, the set of observations  $\{0, 1\}$  needs to be a join-semilattice with bottom, which it can be regarded as, with the natural order  $0 \leq 1$ . This is a necessity, due to the induced algebra by the powerset monad. We omit giving any details and point to the literature, e.g. [6, 11]. Now, we take this idea to define our  $\mathfrak{J}$ -valued automata. We use  $\mathfrak{J}$  as a semilattice in the same manner, allowing us to have additional semantics on our set of outputs. In the following sections, although originally inspired by the coalgebraic approach, we will define our automata similar to the original definition of DFA and only sometimes draw notions from the coalgebraic definitions.

## 2 | $\mathfrak{Z}$ -valued automata and $\mathfrak{Z}$ -regular languages

### 2.1 $\mathfrak{Z}$ -valued Automata

With the underlying theory out of the way, we can finally introduce our new class of automata and their languages. The key difference is in the output of a state. Now our output function maps from the set of states to  $\mathfrak{Z}$  rather than  $\sigma : Q \rightarrow 2$ , as it is the case for finite automata. We define our automata similarly to the standard definition of finite automata and therefore denote our output function  $F$ . We call them  $\mathfrak{Z}$ -valued automata and their class of recognized languages, the  $\mathfrak{Z}$ -regular languages. The new output symbol  $?$ , is to be interpreted as an uncertain answer. Therefore, an input can always be accepted or rejected as usual, but one can also receive an undetermined answer.  $\mathfrak{Z}$ -valued automata are fairly similar to finite automata, as no other part changes.

#### 2.1.1 Deterministic $\mathfrak{Z}$ -valued Automata

We can first look at an example in Figure 2.1, where accepting states are doubly lined and uncertain states are drawn with a dashed line.

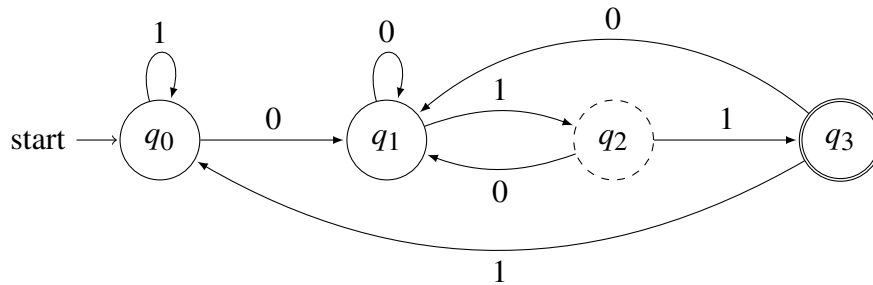


Figure 2.1: Transition diagram of a D $\mathfrak{Z}$ A

Note that this automaton is similar to the example automaton in Figure 1.1. There is an additional state  $q_3$ , which is an accepting state. The former accepting state  $q_2$  is now an uncertain state. Therefore, all previously accepted words, namely those ending in 01, are now subject to uncertainty. Words ending in 011 are being accepted, all others rejected. We can now give a formal definition.

**Definition 2.1.1.** A deterministic  $\mathfrak{Z}$ -valued automaton (D $\mathfrak{Z}$ A) is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  with:

- $Q$  a finite set of states,

- $\Sigma$  a finite set of input symbols,
- $\delta : Q \times \Sigma \rightarrow Q$  the transition function,
- $q_0 \in Q$  denoting the initial or starting state, and
- $F : Q \rightarrow \mathfrak{Z}$  the output function with the intended meaning:

$$F(q) = \begin{cases} 1 & \text{if } q \text{ is accepting,} \\ ? & \text{if } q \text{ is uncertain,} \\ 0 & \text{if } q \text{ is rejecting.} \end{cases}$$

Since the key difference is only in the output function, there is no need to redefine the concepts of a path or the extended transition function, as the given definitions in Section 1.1 apply. We can therefore get back to our example in Figure 2.1 and give it's formal definition  $M = (Q, \Sigma, \delta, q_0, F)$  with:

- $Q = \{q_0, q_1, q_2, q_3\}$ ,
- $\Sigma = \{0, 1\}$ , and
- $\delta$  as well as  $F$  defined in Table 2.1 and Table 2.2.

$\delta$	0	1
$q_0$	$q_1$	$q_0$
$q_1$	$q_1$	$q_2$
$q_2$	$q_1$	$q_3$
$q_3$	$q_1$	$q_0$

Table 2.1: Transition table for  $\delta$  of the D3A in Figure 2.1

	$q_0$	$q_1$	$q_2$	$q_3$
$F$	0	0	?	1

Table 2.2: Definition of the output function  $F$  for the D3A in Figure 2.1

## Language of a D3A

**Definition 2.1.2.** The language of a state  $q$  in a D3A  $M$  is a map  $\mathfrak{L}(q) : \Sigma^* \rightarrow \mathfrak{Z}$  with

$$\mathfrak{L}(q)(w) = F(\delta^*(q, w))$$

or equivalently a triple  $L(q) = (L_0, L_?, L_1)$  with:

$$\begin{aligned} L_1 &= \{w \in \Sigma^* \mid F(\delta^*(q, w)) = 1\}, \\ L_? &= \{w \in \Sigma^* \mid F(\delta^*(q, w)) = ?\}, \\ L_0 &= \{w \in \Sigma^* \mid F(\delta^*(q, w)) = 0\}. \end{aligned}$$

This gives us a language map  $L : Q \rightarrow \mathfrak{Z}^{\Sigma^*}$ .

Note that  $L_0 \cup L_? \cup L_1 = \Sigma^*$  and they are pairwise disjoint. We say a language  $L$  accepts a word  $w$  if  $\mathfrak{L}(w) = 1$  or equivalently  $w \in L_1$ ,  $L$  is uncertain about  $w$  if  $\mathfrak{L}(w) = ?$  or  $w \in L_?$  and  $L$  rejects  $w$  if  $\mathfrak{L}(w) = 0$  or  $w \in L_0$ .

**Remark 2.1.3.** Note that the language map for a finite automaton  $L : Q \rightarrow \mathcal{P}(\Sigma^*)$  can also be written as  $L : Q \rightarrow 2^{\Sigma^*}$  where  $2 = \{0, 1\}$ . Every set  $X \in \mathcal{P}(\Sigma^*)$ , can be uniquely identified by a function  $f : \Sigma^* \rightarrow 2$ , such that  $f(w) = 1 \Leftrightarrow w \in X$ .

**Definition 2.1.4.** The recognized language  $L$  of a deterministic 3-valued automaton  $M$ , denoted  $L(M)$  is the language of its starting state, i.e.,  $L(M) = L(q_0)$ .

We can now speak of the class of 3-regular languages, namely those recognized by deterministic 3-valued automata.

**Definition 2.1.5.** A language  $L$  is a 3-regular language if there exists a D3A  $M$  where  $L(M) = L$ .

Furthermore, we can give the formal definition for the language of our example automaton in Figure 2.1. We will give both the definition as a triple, as well as the definition as a map  $\mathfrak{L} : \Sigma^* \rightarrow \mathfrak{Z}$ .  $L(M) = (L_0, L_?, L_1)$  with:

- $L_1 = \{w \in \Sigma^* \mid w \text{ ends with } 011\}$ ,
- $L_? = \{w \in \Sigma^* \mid w \text{ ends with } 01\}$ , and
- $L_0 = \{w \in \Sigma^* \mid w \text{ neither ends with } 011 \text{ nor with } 01\}$

or equivalently the language as a map  $\mathfrak{L}$  with

$$\mathfrak{L}(w) = \begin{cases} 1 & \text{if } w \text{ ends with } 011, \\ ? & \text{if } w \text{ ends with } 01, \\ 0 & \text{if } w \text{ neither ends with } 011 \text{ nor with } 01. \end{cases}$$

We will now go on to show that every part of our language is a regular language, therefore also justifying the name 3-regular languages.

**Theorem 2.1.6.** For every language  $L = (L_0, L_?, L_1)$  of a D3A  $M = (Q, \Sigma, \delta, q_0, F)$  the languages  $L_0, L_?, L_1$  are regular languages.

*Proof.* For each part of the language we construct a DFA  $M_\circ$  recognizing  $L_\circ$  with  $\circ \in \{0, ?, 1\}$ . We thereby show that all 3 components of a 3-regular language are regular languages.

- $M_\circ = (Q, \Sigma, \delta, q_0, F_\circ)$  with:
- $$F_\circ = \{q \in Q \mid F(q) = \circ\}.$$

We have

$$\begin{aligned} L(M_\circ) &= \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F_\circ\} \\ &= \{w \in \Sigma^* \mid F(\delta^*(q_0, w)) = \circ\} \\ &= L_\circ \end{aligned}$$

showing that  $L_\circ$  is a regular language. □

### 2.1.2 Nondeterministic 3-valued Automata

The changes when introducing nondeterminism, are similar to those for finite automata. We therefore omit to provide another example and give its formal definition right away.

**Definition 2.1.7.** A nondeterministic 3-valued automaton (N3A) is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  with:

- $Q$  a finite set of states,
- $\Sigma$  a finite set of input symbols,
- $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$  the transition function,
- $q_0 \in Q$  denoting the initial or starting state, and
- $F : Q \rightarrow \mathfrak{Z}$  the output function, similar to that of a D3A.

For N3A, similar to NFA, we want to accept a word if there exists a path ending in an accepting state. Moreover, words should only be rejected if all paths end in rejecting states. All other words, those where we can reach an uncertain state but no accepting state, are subject to uncertainty. This notion of acceptance is achieved simply by taking the join of the outputs of all reachable states.

**Definition 2.1.8.** The language  $L$  of a state  $q$  in an N3A  $M$  is given by  $L(q) = (L_0, L_?, L_1)$  with:

$$\begin{aligned} L_1 &= \{w \in \Sigma^* \mid \bigsqcup F(\delta^*(q, w)) = 1\}, \\ L_? &= \{w \in \Sigma^* \mid \bigsqcup F(\delta^*(q, w)) = ?\}, \\ L_0 &= \{w \in \Sigma^* \mid \bigsqcup F(\delta^*(q, w)) = 0\}. \end{aligned}$$

Note that all these joins exist as  $\mathfrak{Z}$  is a complete lattice.

**Definition 2.1.9.** The recognized language  $L$  of a nondeterministic 3-valued automaton  $M$ , denoted  $L(M)$ , is the language of its starting state, i.e.,  $L(M) = L(q_0)$ .

### 2.1.3 Determinization

We will now show that the class of languages recognized by N3A are the 3-regular languages. For this, we need to show that for every N3A, there exists a D3A recognizing the same language, and conversely for every D3A  $D$ , there exists an N3A  $N$  with  $L(N) = L(D)$ .

For the one direction we adapt the determinization algorithm for finite automata given in [5]. First we define a procedure to convert an arbitrary N3A into a D3A and then prove that they recognize the same language.

From an N3A  $N = (Q_N, \Sigma, \delta_N, n_0, F_N)$  construct D3A  $D = (Q_D, \Sigma, \delta_D, d_0, F_D)$  with:

- $Q_D = \mathcal{P}(Q_N)$ ,
- $\delta_D(q, a) = \bigcup_{q_i \in q} E(\delta_N(q_i, a))$ ,
- $d_0 = E(n_0)$ , and
- $F_D(q) = \bigcup_{q_i \in q} F_N(q_i)$ .

Note that this procedure is almost identical to that for finite automata. We only change the definition of the output function  $F$ , therefore resulting in the same construction in the view of  $F : Q \rightarrow 2$  for NFA and DFA.

**Theorem 2.1.10.**  $L(M_N) = L(M_D)$  for an N3A  $M_N = (Q_N, \Sigma, \delta_N, n_0, F_N)$  and its matching D3A  $M_D = (Q_D, \Sigma, \delta_D, d_0, F_D)$ , constructed by the determinization procedure.

*Proof.* Let  $N = L(M_N) = (N_0, N_?, N_1)$  be the language of an N3A, then for  $\circ \in \{0, ?, 1\}$

$$N_\circ = \{w \mid \bigcup F_N(\delta_N^*(n_0, w)) = \circ\}.$$

Similarly for the D3A  $M_D$ , the language  $D = L(M_D) = (D_0, D_?, D_1)$  and

$$D_\circ = \{w \mid F_D(\delta_D^*(d_0, w)) = \circ\}.$$

Therefore, for  $N_\circ = D_\circ$  it suffices to show that  $\delta_N^*(n_0, w) = \delta_D^*(d_0, w)$  since

$$F_D(\delta_D^*(d_0, w)) = \bigcup_{q_i \in \delta_D^*(d_0, w)} F_N(q_i) = \bigcup F_N(\delta_N^*(d_0, w)).$$

We show this by induction on word length  $|w|$ .

IB: If  $|w| = 0$ , the equality follows immediately from the definitions of  $\delta_N^*$  and  $\delta_D^*$ :

$$\delta_N^*(n_0, \varepsilon) = E(n_0) = d_0 = \delta_D^*(d_0, \varepsilon).$$

IH: Let  $k \in \mathbb{N}$  be arbitrary and  $\delta_N^*(n_0, w) = \delta_D^*(d_0, w)$  for all  $w \in \Sigma^*$  with  $|w| = k$ .

IS: Let  $w \in \Sigma^*$  be an arbitrary word of length  $|w| = k + 1$ . We separate  $w$  into  $w = xa$ , a word

$x \in \Sigma^*$  with  $|x| = k$  and a single symbol  $a \in \Sigma$ . From (IH) we know  $\delta_N^*(n_0, x) = \delta_D^*(d_0, x)$ . Let the result be  $\{r_1, \dots, r_m\}$ . Therefore, by substituting the resulting set of states for  $\delta_N^*(n_0, x)$ , we get

$$\delta_N^*(n_0, w) = E\left(\bigcup_{r \in \delta_N^*(n_0, x)} \delta_N(r, a)\right) = E\left(\bigcup_{r_i \in \{r_1, \dots, r_m\}} \delta_N(r_i, a)\right).$$

and from the definition of  $\delta_D$  we see that

$$\delta_D^*(d_0, w) = \delta_D(\delta_D^*(d_0, x), a) = \delta_D(\{r_1, \dots, r_m\}, a) = E\left(\bigcup_{r_i \in \{r_1, \dots, r_m\}} \delta_N(r_i, a)\right).$$

□

**Corollary 2.1.11.** *A Language  $L$  is recognized by some deterministic  $\mathfrak{Z}$ -valued automaton if and only if it is recognized by some nondeterministic  $\mathfrak{Z}$ -valued automaton.*

### 2.1.4 Language equivalence of automata

We defined two finite automata to be equivalent, if and only if their recognized languages are the same. To show language equivalence, two common approaches can be taken. One can use a minimization algorithm, given in [3], where states are partitioned into equivalence classes. If the two starting states are in the same equivalence class, the automata are equivalent. Another algorithm, given in [4], builds a relation starting from the initial states. For all already related pairs, the states reachable by the same symbol are considered. If they are either both accepting or rejecting, they are added as well. This is repeated until either two states differ in their acceptance or no new pairs can be added. If the relation can be successfully built, the two starting states are equivalent.

We omit to give any of the algorithms, but do show that bisimilarity and language equivalence coincide in DFA. This result we will use for our  $\mathfrak{Z}$ -valued automata, but first, we introduce the notion of a bisimulation and bisimilarity on finite automata.

**Definition 2.1.12.** *A symmetric relation  $R \subseteq Q \times Q$  on a finite automaton  $M = (Q, \Sigma, \delta, q_0, F)$  is a bisimulation iff for all  $(p, q) \in R$*

- $p \in F \Leftrightarrow q \in F$ , and
- $\forall a \in \Sigma. \forall p' \in Q. p \xrightarrow{a} p' \Rightarrow \exists q'. q \xrightarrow{a} q' \wedge (p', q') \in R$ .

*We then say two states  $p, q$  are bisimilar, denoted  $p \sim q$ , iff there exists a bisimulation relation  $R$  with  $(p, q) \in R$ .*

In Figure 2.2 we have two automata recognizing the same language. When we try to build a bisimulation, starting at their respective starting state, we quickly come to the conclusion that the two automata are not bisimilar. In  $p_1$  we can make transitions both for  $b$  and  $c$ , but neither in  $q_1$ , nor in  $q_2$  this is possible. Therefore there cannot exist a bisimulation relating  $p_0$  and  $q_0$ .

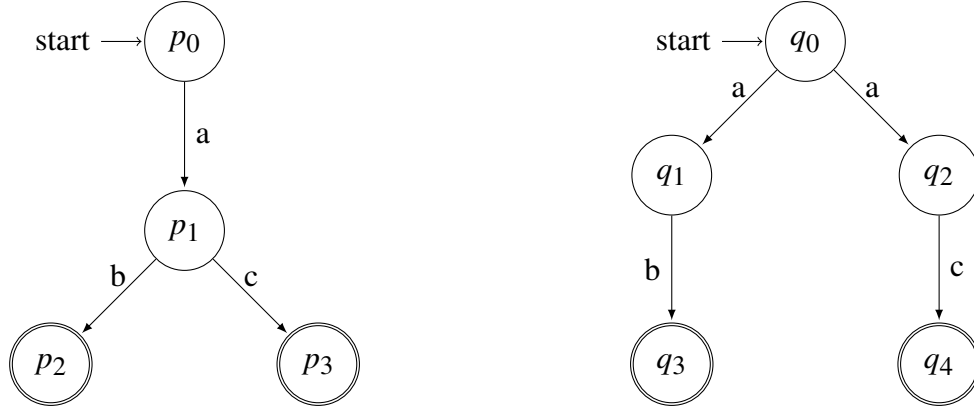


Figure 2.2: Two non-bisimilar automata

This example shows, that two language equivalent automata are not necessarily bisimilar. This is the case in the presence of nondeterminism. In Figure 2.3 we determinized the NFA of Figure 2.2 and we can now give a bisimulation  $R$  with

$$R = \{(p_0, \{q_0\}), (p_1, \{q_1, q_2\}), (p_2, \{q_3\}), (p_3, \{q_4\})\}.$$

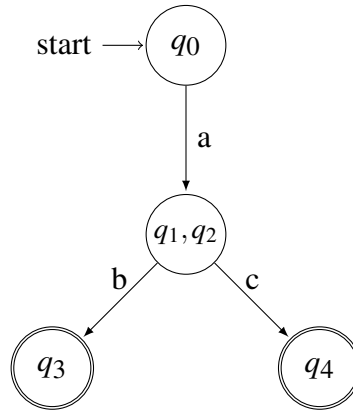


Figure 2.3: Determinized automaton of NFA in Figure 2.2

Before we go on to prove the property for DFA we first give the definition of the language equivalence relation on automata. This and the following Lemma 2.1.14 will help us in the proof of Theorem 2.1.15. Definition 2.1.13 below applies to both finite automata and 3-valued automata using the respective definition for the language of a state.

**Definition 2.1.13.** The language equivalence relation  $\equiv_L \subseteq Q \times Q$  on an automaton  $M = (Q, \Sigma, \delta, q_0, F)$  is the kernel of the language map, i.e.,  $p \equiv_L q \Leftrightarrow L(p) = L(q)$ .

**Lemma 2.1.14.** If two states  $p, q$  in a DFA  $M = (Q, \Sigma, \delta, q_0, F)$  are bisimilar, then also  $\delta^*(p, w)$  and  $\delta^*(q, w)$  are for every word  $w \in \Sigma^*$ .

*Proof.* Since  $M$  is deterministic, for a given symbol there is exactly one possible transition to a state. Therefore, for two states  $p, q$  with  $p \sim q$ , we can simplify the second condition of

bisimulation to

$$\forall a \in \Sigma. p \xrightarrow{a} p' \wedge q \xrightarrow{a} q' \Rightarrow p'Rq'. \quad (*)$$

Now assume  $p \sim q$ , we prove  $\forall w \in \Sigma^*. \delta^*(p, w) \sim \delta^*(q, w)$  by induction on word length.

IB: For  $|w| = 0$ , we get

$$\delta^*(p, \varepsilon) = p \sim q = \delta^*(q, \varepsilon).$$

IH: Let  $n \in \mathbb{N}$  be arbitrary and  $\delta^*(p, w) \sim \delta^*(q, w)$  for all  $w \in \Sigma^*$  with  $|w| = n$ .

IS: Let  $w \in \Sigma^*$  be an arbitrary word of length  $|w| = n + 1$ . We separate  $w$  into  $w = xa$ , a word  $x$  with  $|x| = n$  and a single symbol  $a$ . We get

$$\begin{aligned} \delta^*(p, w) &= \delta(\delta^*(p, x), a), \\ \delta^*(q, w) &= \delta(\delta^*(q, x), a). \end{aligned}$$

From (IH) we already know  $\delta^*(p, x) \sim \delta^*(q, x)$ . Now, since  $M$  is deterministic and  $\delta^*(p, x) \xrightarrow{a} \delta(\delta^*(p, x), a)$  and  $\delta^*(q, x) \xrightarrow{a} \delta(\delta^*(q, x), a)$ , we conclude by (\*) that also

$$\delta(\delta^*(p, x), a) \sim \delta(\delta^*(q, x), a).$$

□

**Theorem 2.1.15.** *The languages of two states  $p, q$  in a DFA  $M$ , are equal iff  $p$  and  $q$  are bisimilar, i.e.,  $L(p) = L(q) \Leftrightarrow p \sim q$ .*

*Proof.* For the one direction, let  $p, q \in Q$  and assume  $p \sim q$ .

To prove  $L(p) = L(q)$ , we show that  $\forall w \in \Sigma^*. \delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F$

From Lemma 2.1.14 we know, since  $p \sim q$ , for arbitrary  $w \in \Sigma^*$

$$\delta^*(p, w) \sim \delta^*(q, w).$$

Therefore  $\delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F$ . Hence  $L(p) = L(q)$ .

For the second direction, we will prove that, for deterministic automata,  $\equiv_L$  is a bisimulation. Let  $p, q \in Q$  and assume  $p \equiv_L q$ . Since  $r \in F \Leftrightarrow \varepsilon \in L(r)$  for any  $r \in Q$  and  $L(p) = L(q)$ , we conclude  $p \in F \Leftrightarrow q \in F$ . We now look at  $p', q'$ , with  $p \xrightarrow{a} p', q \xrightarrow{a} q'$  for a given  $a \in \Sigma$  and show  $p' \equiv_L q'$ . Assume, towards a contradiction,  $p' \not\equiv_L q'$ . Therefore  $L(p') \neq L(q')$  and w.l.o.g.  $\exists w \in \Sigma^*. w \in L(p') \wedge w \notin L(q')$ . Then, since  $w \in L(p')$ ,  $aw \in L(p)$ . But, since  $w \notin L(q')$  and we can make a transition  $q \xrightarrow{a} q'$ ,  $aw \notin L(q)$ , contradicting our original assumption  $L(p) = L(q)$ .

Hence for  $p, q \in Q$ , if  $p \equiv_L q$  then  $p \sim q$ . □

We can now go on to define bisimulation and bisimilarity for 3-valued automata.

**Definition 2.1.16.** *A symmetric relation  $R \subseteq Q \times Q$  on a 3-valued automaton  $M = (Q, \Sigma, \delta, q_0, F)$  is a bisimulation iff for all  $(p, q) \in R$*

- $F(p) = F(q)$ , and
- $\forall a \in \Sigma. \forall p' \in Q. p \xrightarrow{a} p' \Rightarrow \exists q'. q \xrightarrow{a} q' \wedge (p', q') \in R$

We then say two states  $p, q$  are bisimilar, denoted  $p \sim q$ , iff there exists a bisimulation relation  $R$  with  $(p, q) \in R$ .

Since there is no difference in the transitions of finite automata and 3-valued automata, we will lift Lemma 2.1.14 to 3-valued automata without giving an additional proof.

**Lemma 2.1.17.** *If two states  $p, q$  in a 3-valued automaton  $M = (Q, \Sigma, \delta, q_0, F)$  are bisimilar, then also  $\delta^*(p, w)$  and  $\delta^*(q, w)$  are for every word  $w \in \Sigma^*$ .*

We can already observe from the definition of a bisimulation on 3-valued automata, that very little changed. Therefore it should be no surprise that, also for D3A, language equivalence and bisimilarity coincide.

**Theorem 2.1.18.** *The languages of two states  $p, q$  in a D3A  $M$  are equal iff  $p$  and  $q$  are bisimilar, i.e.,  $L(p) = L(q) \Leftrightarrow p \sim q$ .*

*Proof.* For the "if" direction, let  $p, q \in Q$  and assume  $p \sim q$ .

To show  $L(p) = L(q)$  we show  $F(\delta^*(p, w)) = F(\delta^*(q, w))$  for all  $w \in \Sigma^*$ . From Lemma 2.1.17 we know that  $\delta^*(p, w) \sim \delta^*(q, w)$ . Therefore  $F(\delta^*(p, w)) = F(\delta^*(q, w))$ .

For the "only if" direction, we show that  $\equiv_L$  is a bisimulation. Let  $p, q \in Q$  and assume  $p \equiv_L q$ . Once again we look at  $p', q'$ , with  $p \xrightarrow{a} p', q \xrightarrow{a} q'$  for arbitrary  $a \in \Sigma$  and show  $p' \equiv_L q'$ . Assume, towards a contradiction, that  $p' \not\equiv_L q'$ . Therefore w.l.o.g.  $\exists w \in \Sigma^*. F(\delta^*(p', w)) \neq F(\delta^*(q', w))$ . Then

$$F(\delta^*(p, aw)) = F(\delta^*(p', w)) \neq F(\delta^*(q', w)) = F(\delta^*(q, aw))$$

contradicting our original assumption  $p \equiv_L q$ . Further, if  $F(p) = 1$ , then  $\varepsilon \in L(p)_1 = L(q)_1$ , so  $F(q) = 1$ . We can argue similarly for  $F(p) = ?$  and  $F(p) = 0$  as  $F(p) = ? \Leftrightarrow \varepsilon \in L(p)_? = L(q)_?$  and  $F(p) = 0 \Leftrightarrow \varepsilon \in L(p)_0 = L(q)_0$ . Therefore, we conclude  $F(p) = F(q)$ . Hence, for  $p, q \in Q$ , if  $p \equiv_L q$  then  $p \sim q$ .  $\square$

**Corollary 2.1.19.** *The languages  $L(M)$  and  $L(M')$  of two 3-regular automata  $M, M'$  are equivalent if and only if their starting states are bisimilar.*

## 2.2 3-regular Languages

In this section we will define some operations on 3-regular languages and give a simple way to lift the definition of regular expressions to 3-regular expressions to fit for our new class of languages.

### 2.2.1 Operators and Properties of 3-regular languages

We will first define the 3-regular operators union, concatenation, and Kleene star. We will also define the intersection of two languages and the complement of a language. Note that for any 3-regular language  $L$ , we know  $L_0 \cup L_? \cup L_1 = \Sigma^*$  and they are disjoint. All operations are defined in such a way, that these properties are still satisfied.

#### Union

The union of two 3-regular languages accepts all words that either one of the two accepts and reject all that are being rejected by both. For the remaining words, i.e., those for which one language is uncertain but neither accepts, we get an uncertain answer.

**Definition 2.2.1.** For two 3-regular languages  $L = (L_0, L_?, L_1)$ ,  $L' = (L'_0, L'_?, L'_1)$  the union  $L \cup L'$  is given by

$$L \cup L' = (L_0 \cap L'_0, (L_? \cap L'_?) \cup (L_0 \cap L'_?) \cup (L_? \cap L'_0), L_1 \cup L'_1).$$

#### Concatenation

The concatenation  $L \cdot L'$  or  $LL'$  of two 3-regular languages  $L$  and  $L'$  is defined in such a way, that if a word  $w$  can be separated into two words  $w_1 w_2$  and  $L$  accepts  $w_1$  while  $L'$  accepts  $w_2$ ,  $w$  is accepted by the concatenated language. If in turn, this is not possible, but there are  $w_1, w_2$  such that none of them is rejected, we get an uncertain answer. Otherwise  $w$  is being rejected.

**Definition 2.2.2.** For two 3-regular languages  $L = (L_0, L_?, L_1)$ ,  $L' = (L'_0, L'_?, L'_1)$  the concatenation  $LL'$  is given by

$$LL' = ((L_? L'_? \cup L_1 L'_? \cup L_? L'_1 \cup L_1 L'_1)^c, (L_? L'_? \cup L_1 L'_? \cup L_? L'_1) \setminus (L_1 L'_1), L_1 L'_1).$$

#### Kleene Star

The Kleene star of a language  $L$  accepts all words  $w$  that can be separated into words  $w = w_1 \dots w_n$ , such that all  $w_i$ , with  $i \leq n$ , are accepted by  $L$ . If no such separation exists, but one where each  $w_i$  is either accepted or is subject to uncertainty,  $L^*$  will give an uncertain answer. All others are rejected.

**Definition 2.2.3.** For a 3-regular language  $L = (L_0, L_?, L_1)$  the Kleene star  $L^*$  of  $L$  is given by

$$L^* = (((L_? \cup L_1)^*)^c, (L_? \cup L_1)^* \setminus L_1^*, L_1^*).$$

**Theorem 2.2.4.** For an arbitrary 3-regular language  $L$ ,

$$L^* = \bigcup_{i \in \mathbb{N}} L^i,$$

where  $L^0 = (\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\})$  and  $L^n = L^{n-1} \cdot L$ .

Before we state the proof of the theorem we first consider a lemma dealing with the notion of raising a 3-regular language  $L$  to the power of some  $n \in \mathbb{N}$ .

**Lemma 2.2.5.** *For an arbitrary 3-regular language  $L$  and  $n \in \mathbb{N}$ ,*

$$L^n = (((L_? \cup L_1)^n)^c, (L_? \cup L_1)^n \setminus L_1^n, L_1^n).$$

*Proof.* Since, all three parts of the languages are disjoint and their union is  $\Sigma^*$  it suffices to show the equality of two parts of the languages. The equality of the accepting parts follows from a simple inductive argument. For an arbitrary 3-regular language  $L$  and  $n = 0$  we get

$$(L^0)_1 = \{\varepsilon\} = L_1^0,$$

as  $L^0 = \{\varepsilon\}$  for any regular language. Now assuming the equality to be true for arbitrary  $n \in \mathbb{N}$ , for  $n + 1$  we get

$$(L^{n+1})_1 = (L^n L)_1 = L_1^n L_1 = L_1^{n+1}.$$

We can now go on to show that  $(L^n)_? = (L_? \cup L_1)^n \setminus L_1^n$  by induction on  $n$ .

IB: For  $n = 0$ , the equality follows from the definition of  $L^0$ .

$$(L^0)_? = \emptyset = \{\varepsilon\} \setminus \{\varepsilon\} = (L_? \cup L_1)^0 \setminus L_1^0.$$

IH: Let  $n \in \mathbb{N}$  be arbitrary and  $(L^n)_? = (L_? \cup L_1)^n \setminus L_1^n$

IS: We begin by substituting from the hypothesis and then apply distributive laws several times.

We also make use of the result of the previous induction.

$$\begin{aligned} (L^{n+1})_? &= (L^n \cdot L)_? = ((L^n)_? L_? \cup (L^n)_? L_1 \cup (L^n)_1 L_?) \setminus L_1^n L_1 \\ &\stackrel{IH}{=} (((L_? \cup L_1)^n \setminus L_1^n) \cdot L_? \cup ((L_? \cup L_1)^n \setminus L_1^n) \cdot L_1 \cup L_1^n L_?) \setminus L_1^n L_1 \\ &= (((L_? \cup L_1)^n \setminus L_1^n) \cdot (L_? \cup L_1) \cup L_1^n L_?) \setminus L_1^{n+1} \\ &= (((L_? \cup L_1)^{n+1} \setminus (L_1^n \cdot (L_? \cup L_1))) \cup L_1^n L_?) \setminus L_1^{n+1} \\ &= (((L_? \cup L_1)^{n+1} \setminus (L_1^{n+1} \cup L_1^n L_?)) \cup L_1^n L_?) \setminus L_1^{n+1} \\ &= (((L_? \cup L_1)^{n+1} \setminus L_1^{n+1}) \setminus L_1^n L_?) \cup L_1^n L_?) \setminus L_1^{n+1} \\ &\stackrel{(*)}{=} (L_? \cup L_1)^{n+1} \setminus L_1^{n+1} \end{aligned}$$

Now, at  $(*)$ , we can see that both inner set differences are redundant. Moreover,  $L_1^n L_? \subseteq (L_? \cup L_1)^{n+1}$ . Since all three parts are disjoint and their union is  $\Sigma^*$ , it follows that  $(L^n)_0 = ((L_? \cup L_1)^n)^c$ .  $\square$

With this intermediate result at hand, we can now continue to proof Theorem 2.2.4.

*Proof of Theorem 2.2.4.* Since all three parts of the languages are disjoint and their union is  $\Sigma^*$ , it suffices to show the equalities of two parts. For the accepting part, from the definition of union and Lemma 2.2.5, it follows immediately that

$$\left(\bigcup_{i \in \mathbb{N}} L^i\right)_1 = \bigcup_{i \in \mathbb{N}} (L^i)_1 = \bigcup_{i \in \mathbb{N}} (L_1^i) = L_1^* = (L^*)_1.$$

Similarly, for the rejecting part we get

$$\left(\bigcup_{i \in \mathbb{N}} L^i\right)_0 = \bigcap_{i \in \mathbb{N}} (L^i)_0 = \bigcap_{i \in \mathbb{N}} ((L_? \cup L_1)^i)^c = \left(\bigcup_{i \in \mathbb{N}} (L_? \cup L_1)^i\right)^c = ((L_? \cup L_1)^*)^c = (L^*)_0$$

and therefore  $L^* = \bigcup_{i \in \mathbb{N}} L^i$ . □

### Complement

The complement of a language  $L$  is simply accepting all previously rejected words and conversely rejects all words that  $L$  accepts. For the uncertain part of a language, nothing changes.

**Definition 2.2.6.** For a 3-regular language  $L = (L_0, L_?, L_1)$  the complement  $L^c$  of  $L$  is given by

$$L^c = (L_1, L_?, L_0).$$

### Intersection

We define the intersection of two 3-regular languages such that words are only accepted if both languages accept, rejected if either rejects and subject to uncertainty otherwise.

**Definition 2.2.7.** For two 3-regular Languages  $L = (L_0, L_?, L_1), L' = (L'_0, L'_?, L'_1)$  the intersection  $L \cap L'$  is given by

$$L \cap L' = (L_0 \cup L'_0, (L_? \cap L'_?) \cup (L_1 \cap L'_1) \cup (L_? \cap L'_1), L_1 \cap L'_1).$$

### Precedence of operators

For 3-regular languages we use the same precedence as for regular languages given in [5]. Additionally, we'll include complement and the intersection. Operations of equal precedence, must be grouped together by parentheses. The following list, gives the operators in decreasing precedence.

1. Kleene star  $^*$ , Complement  $^c$ .
2. Concatenation  $\cdot$ .
3. Union  $\cup$ , Intersection  $\cap$ .

### De Morgan's Laws

Since we not only redefined the regular operators to fit for 3-regular languages, but also the set operations intersection and complement, we will prove that De Morgan's laws hold.

**Lemma 2.2.8.** *De Morgan's laws hold for 3-regular languages, i.e., for arbitrary  $L, L'$*

$$\begin{aligned}(L \cup L')^c &= L^c \cap L'^c, \\ (L \cap L')^c &= L^c \cup L'^c.\end{aligned}$$

*Proof.* Let  $L = (L_0, L_?, L_1), L' = (L'_0, L'_?, L'_1)$  be arbitrary 3-regular languages. For the first law, by applying the complement we immediately get the intersection of the complemented languages.

$$\begin{aligned}(L \cup L')^c &= (L_0 \cap L'_0, (L_? \cap L'_?) \cup (L_0 \cap L'_?) \cup (L_? \cap L'_0), L_1 \cup L'_1)^c \\ &= (L_1 \cup L'_1, (L_? \cap L'_?) \cup (L_0 \cap L'_?) \cup (L_? \cap L'_0), L_0 \cap L'_0) \\ &= (L_1, L_?, L_0) \cap (L'_1, L'_?, L'_0) \\ &= L^c \cap L'^c.\end{aligned}$$

For the second law, similar to the first one, we immediately get the union of the complemented languages.

$$\begin{aligned}(L \cap L')^c &= (L_0 \cup L'_0, (L_? \cap L'_?) \cup (L_1 \cap L'_?) \cup (L_? \cap L'_1), L_1 \cap L'_1)^c \\ &= (L_1 \cap L'_1, (L_? \cap L'_?) \cup (L_1 \cap L'_?) \cup (L_? \cap L'_1), L_0 \cup L'_0) \\ &= (L_1, L_?, L_0) \cup (L'_1, L'_?, L'_0) \\ &= L^c \cup L'^c.\end{aligned}$$

□

### 2.2.2 3-regular expressions

We introduce 3-regular expressions which, similar to regular expressions, are a representation of languages.

**Definition 2.2.9.** *For a given alphabet  $\Sigma$  we call  $R$  a 3-regular expression if it is either:*

1.  $(R_0, R_?, R_1)$ , for regular expressions  $R_0, R_?, R_1$  over  $\Sigma$ , where  $L(R_0 \cup R_? \cup R_1) = \Sigma^*$  and they are pairwise disjoint,
2.  $(R' \cup R'')$ , for 3-regular expressions  $R', R''$ ,
3.  $(R' \cdot R'')$ , for 3-regular expressions  $R', R''$ ,
4.  $(R')^*$ , for a 3-regular expression  $R'$ ,

and its language is given by:

1.  $L((R_0, R_?, R_1)) = (L(R_0), L(R_?), L(R_1))$ ,
2.  $L(R' \cup R'') = L(R') \cup L(R'')$ ,
3.  $L(R' \cdot R'') = L(R') \cdot L(R'')$ ,
4.  $L(R'^*) = L(R')^*$ .

One may notice, that in this definition we have only 4 cases of valid 3-regular expressions, rather than 6, as for regular expressions. Our simplest form already is a triple of regular languages and not just a symbol. The empty language  $(\Sigma^*, \emptyset, \emptyset)$ , is simply rejecting all words. The language only containing the empty word  $\varepsilon$  is  $(\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\})$ . We will show in Section 3.2, that the class of languages represented by 3-regular expressions is the class of 3-regular languages.

## 3 | Results on automata and language

### 3.1 Closure Properties

We will now prove that  $\mathfrak{Z}$ -regular languages are closed under the operators we introduced. For each operator we give a procedure to construct an automaton accepting the resulting language, proving the language to be  $\mathfrak{Z}$ -regular. We will give illustrations for the more complex constructions for concatenation and Kleene star. To explain the illustrations, we also depict the simpler construction of the automaton accepting the union of two languages. For all operators we will argue about the correctness of the resulting automata, but omit giving a formal proof. The constructed automata, resemble those for regular languages given in [12, 5].

#### 3.1.1 Union

**Theorem 3.1.1.** *The class of  $\mathfrak{Z}$ -regular languages is closed under union.*

We construct a nondeterministic automaton  $\hat{M}$  accepting the union of two  $\mathfrak{Z}$ -regular languages. The idea is to simulate two automata and accept if and only if either one of the two accepts and reject if and only if both of them reject. The construction is identical to that for regular languages.

Consider two  $\mathfrak{Z}$ -regular languages  $L, L'$  and their corresponding D $\mathfrak{Z}$ A  $M = (Q, \Sigma, \delta, q_0, F)$ ,  $M' = (Q', \Sigma', \delta', q'_0, F')$  such that  $L = L(M)$  and  $L' = L(M')$ . We construct the N $\mathfrak{Z}$ A  $\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \hat{F})$  with:

$$\begin{aligned} & \bullet \hat{Q} = Q \cup Q' \cup \{\hat{q}_0\}, \\ & \bullet \hat{\delta}(q, a) = \begin{cases} \delta(q, a) & \text{if } q \in Q, a \in \Sigma, \\ \delta'(q, a) & \text{if } q \in Q', a \in \Sigma, \\ \{q_0, q'_0\} & \text{if } q = \hat{q}_0, a = \varepsilon, \\ \emptyset & \text{otherwise.} \end{cases} \\ & \bullet \hat{F}(q) = \begin{cases} F(q) & \text{if } q \in Q, \\ F'(q) & \text{if } q \in Q', \\ 0 & \text{if } q = \hat{q}_0. \end{cases} \end{aligned}$$

In Figure 3.1 we can observe the resulting automaton, where each box represents one of the two automata  $M, M'$ . The circle intersecting the boxes, represents the respective starting state. The dashed and doubly-lined states are representing all uncertain and rejecting states, respectively. Note that these boxes are representing an arbitrary  $\mathfrak{Z}$ -regular automaton. Therefore, there are not necessarily any accepting or uncertain states and not all depicted transition out of

and into the box must be present. Moreover, the starting state can of course also be one of the accepting or uncertain states.

Then, from the defined construction, our new starting state  $\hat{q}_0$  has one  $\varepsilon$ -transitions to each of the starting states of  $M$  and  $M'$ . Otherwise, the new automaton simulates the behavior of the two original ones.

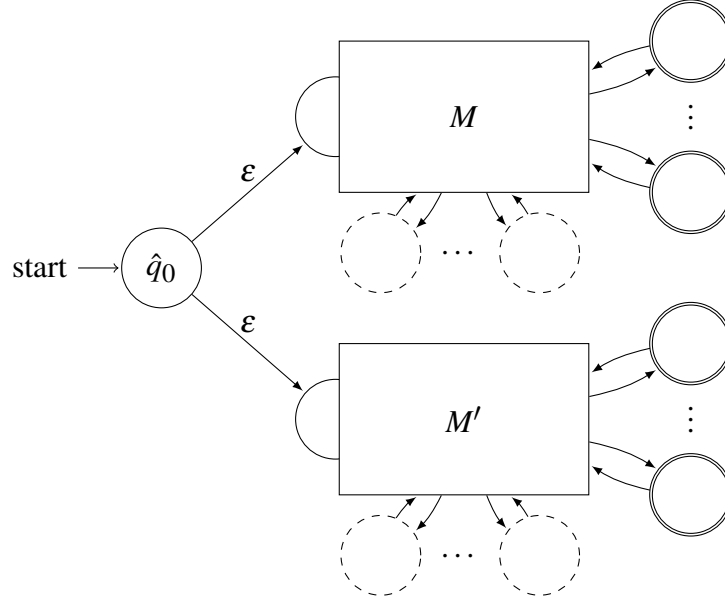


Figure 3.1: N3A  $\hat{M}$  recognizing the union of two languages

*Proof Idea.* From a simple inductive argument it follows that for arbitrary  $w \in \Sigma^*$ , we get  $\hat{\delta}^*(\hat{q}_0, w) = \{\delta^*(q_0, w), \delta'^*(q'_0, w)\}$ , since in the first step, without reading a symbol, we either go to  $q_0$  or  $q'_0$  and then simulate the behavior of the original automata. Therefore we also get

$$\sqcup F(\hat{\delta}^*(q_0, w)) = F(\delta^*(q_0, w)) \sqcup F(\delta'^*(q'_0, w)).$$

Hence,  $\sqcup F(\delta^*(q_0, w)) = 1$  if and only if one of the two automata accepts and  $\sqcup F(\delta^*(q_0, w)) = 0$  if and only if both reject. Therefore  $L(M)_0 = L_0 \cap L'_0$  and  $L(M)_1 = L_1 \cup L'_1$ . Since, all three parts of  $L \cup L'$  are disjoint and their union is  $\Sigma^*$ ,  $L(M) = L \cup L'$ .

### 3.1.2 Concatenation

**Theorem 3.1.2.** *The class of 3-regular languages is closed under concatenation.*

We now construct an N3A  $\hat{M}$  accepting the concatenation of the languages  $L(M)$  and  $L(M')$  of two 3-valued automata  $M, M'$ . We run the two automata in succession, but need to consider the output of the first automaton. If it is uncertain, no accepting state should be reachable.

Consider two 3-regular languages  $L, L'$  and their corresponding D3A  $M = (Q, \Sigma, \delta, q_0, F), M' = (Q', \Sigma, \delta', q'_0, F')$  with  $L = L(M)$  and  $L' = L(M')$ . We first construct an automaton, similar to  $M'$ , but change the output of all accepting states, such that they give an uncertain answer. We call the resulting automaton  $M'' = (Q'', \Sigma, \delta'', q''_0, F'')$  with:

- $Q'' = \{(q)' \mid q \in Q'\}$ ,
- $\delta''(q', a) = (\delta'(q, a))'$ , and
- $F''(q') = F'(q) \sqcap ?$ .

Now construct the N3A  $\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, q_0, \hat{F})$  with:

- $\hat{Q} = Q \cup Q' \cup Q''$ ,
- $\hat{\delta}(q, a) = \begin{cases} \delta(q, a) & \text{if } q \in Q, a \neq \varepsilon, \\ q'_0 & \text{if } q \in Q, F(q) = 1, a = \varepsilon, \\ q''_0 & \text{if } q \in Q, F(q) = ?, a = \varepsilon, \\ \delta'(q, a) & \text{if } q \in Q', a \in \Sigma, \\ \delta''(q, a) & \text{if } q \in Q'', a \in \Sigma, \\ \emptyset & \text{otherwise,} \end{cases}$
- $\hat{F}(q) = \begin{cases} 0 & \text{if } q \in Q, \\ F'(q) & \text{if } q \in Q', \\ F''(q) & \text{if } q \in Q''. \end{cases}$

Note that the starting state of the new automaton  $\hat{M}$  is the starting state of  $M$ .

The resulting automaton is depicted in Figure 3.2. Once again the boxes represent the behaviour of  $M$ ,  $M'$ , and  $M''$ . The gray text, describes the previous behavior of all states which output changed.

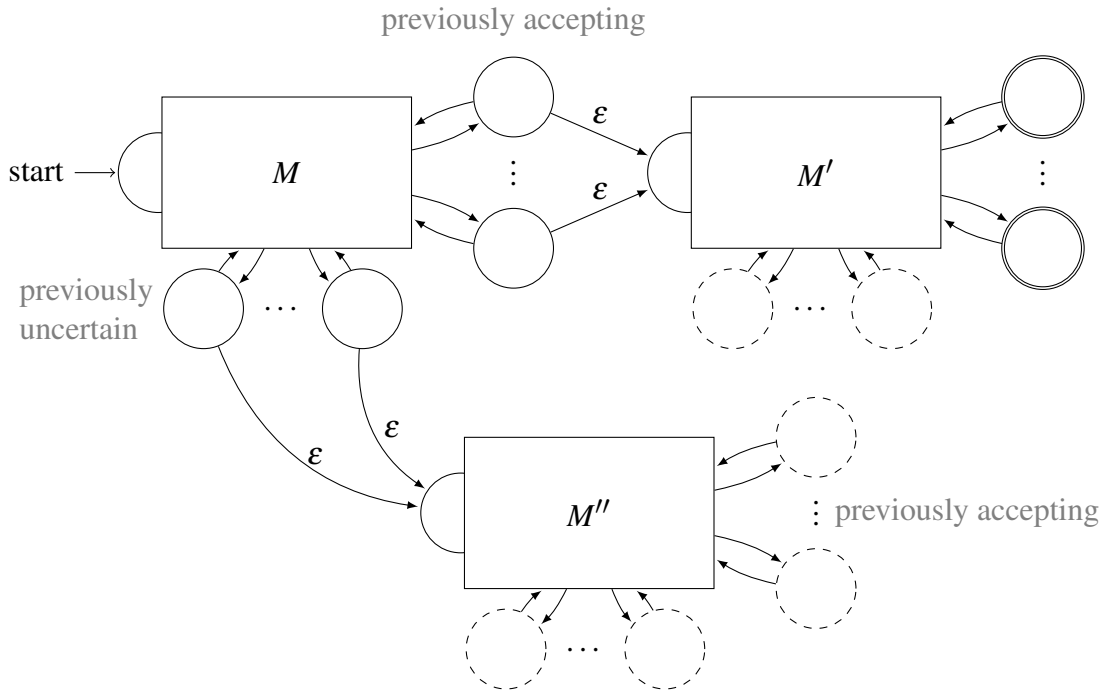


Figure 3.2: N3A  $\hat{M}$  recognizing the concatenation of two languages

*Proof Idea.* Let us first look at the accepting part. Assume  $w \in (LL')_1$  and let  $w_1 \in L_1$  and  $w_2 \in L'_1$ , such that  $w = w_1 w_2$ . Then,  $F(\delta^*(q_0, w_1)) = 1$  and  $F'(\delta'^*(q'_0, w_2)) = 1$ . Therefore, in our new automaton, from  $\hat{\delta}^*(q_0, w_1)$  we can reach  $q'_0$ , from which in turn we can reach an accepting state. Hence,  $\hat{F}(\hat{\delta}^*(q_0, w_1 w_2)) = 1$  and  $(LL')_1 \subseteq L(\hat{M})_1$ .

Now assume a word  $w \in L(\hat{M})_1$ . Then from the structure of our automaton we conclude, that we can split  $w$  into two words  $w_1, w_2$  with  $w_1 w_2 = w$ , such that, with  $\hat{\delta}^*(q_0, w_1)$  we reach a previously accepting state of the simulated automaton  $M$ . Therefore  $w_1 \in L_1$ . Moreover, with the remaining part  $w_2$  we reach an accepting state in  $M'$  and  $w_2 \in L'_1$ . Hence,  $L(M)_1 \subseteq (LL')_1$  and  $L(M)_1 = (LL')_1$ .

To argue for the equality of the rejecting parts we first show  $L(\hat{M})_0 \subseteq (LL')_0$ . Let  $w \in L(\hat{M})_0$  and assume towards a contradiction, that there exists a separation  $w = w_1 w_2$  with  $w_1 \notin L_0$  and  $w_2 \notin L'_0$ . Then there is a path such that, with  $w_1$  we reach a non-rejecting state in  $M$  and with  $w_2$  a non-rejecting state in  $M'$ . Therefore also reaching a non-rejecting state in  $\hat{M}$  contradicting our original assumption.

Let  $w \in (LL')_0$ , then  $\nexists w_1, w_2 \in L_0^c \times L_0'^c$ .  $w = w_1 w_2$ . Now assume towards a contradiction that  $w \notin L(\hat{M})_0$ . Hence, from the structure of our automaton, we know we can split our word into  $w_1 \notin L_0$  and  $w_2 \notin L'_0$ , as otherwise no non-rejecting state could be reached. This contradicts our original assumption and we conclude that  $(LL')_0 \subseteq L(\hat{M})_0$  and therefore  $L(\hat{M})_0 = (LL')_0$ . Moreover, as all three parts of the language are disjoint and their union is  $\Sigma^*$ , we conclude that  $L(\hat{M}) = LL'$  and therefore  $LL'$  is a  $\mathfrak{Z}$ -regular language.

### 3.1.3 Kleene star

**Theorem 3.1.3.** *The class of  $\mathfrak{Z}$ -regular languages is closed under Kleene star.*

For a D $\mathfrak{Z}$ A  $M$  we construct an N $\mathfrak{Z}$ A  $\hat{M}$ . We create a new starting state, to accept the empty word and then simulate the original automaton. By adding  $\varepsilon$ -transitions from all accepting states back to the starting state, we accept those words, which are a concatenation of words in  $L(M)_1$ . Similar to the automaton for concatenation, we need a copy of the automaton, switching all accepting states to uncertain states.

Consider a  $\mathfrak{Z}$ -regular language  $L$  with corresponding D $\mathfrak{Z}$ A  $M = (Q, \Sigma, \delta, q_0, F)$  where  $L = L(M)$ . First construct an automaton  $M'(Q', \Sigma, \delta', q'_0, F') =$  with:

- $Q' = \{q' \mid q \in Q\}$ ,
- $\delta(q', a) = (\delta(q, a))'$ , and
- $F'(q') = F(q) \sqcap ?$ .

Now construct the N $\mathfrak{Z}$ A  $\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \hat{F})$  with:

- $\hat{Q} = Q \cup Q' \cup \{\hat{q}_0\}$ ,

$$\begin{aligned}
\bullet \hat{\delta}(q, a) &= \begin{cases} q_0 & \text{if } q = \hat{q}_0 \vee (q \in Q \wedge F(q) = 1), a = \varepsilon, \\ \delta(q, a) & \text{if } q \in Q, a \in \Sigma, \\ q'_0 & \text{if } (q \in Q \wedge F(q) = ?) \vee (q \in Q' \wedge F'(q) = ?), a = \varepsilon, \\ \delta'(q, a) & \text{if } q \in Q', a \in \Sigma, \\ \emptyset & \text{otherwise,} \end{cases} \\
\bullet \hat{F}(q) &= \begin{cases} F(q) & \text{if } q \in Q, \\ F'(q) & \text{if } q \in Q', \\ 1 & \text{if } q = \hat{q}_0. \end{cases}
\end{aligned}$$

In Figure 3.3 the resulting automaton is depicted. Note that for illustrative purposes, the uncertain states and the accepting states have switched position. Once again, the boxes simulate the behavior of the original automaton  $M$ , where for the box labeled  $M'$  the accepting states are replaced with uncertain states, as it is stated in the gray text.

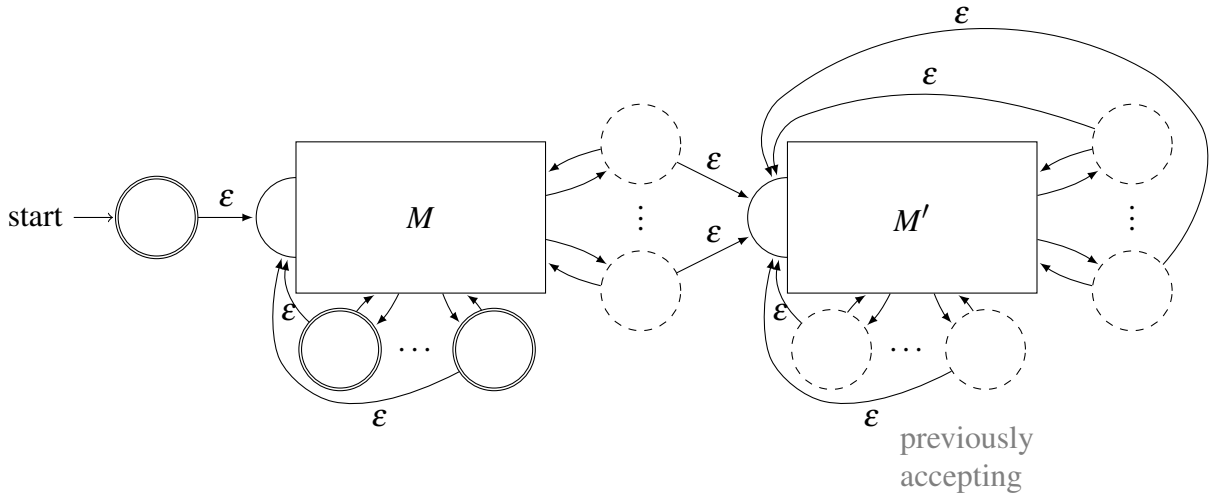


Figure 3.3: NFA  $\hat{M}$  recognizing the Kleene star of a language

*Proof Idea.* For the accepting part, it is clear that for a word  $w \in L(\hat{M})_1$  there can't be a transition leading to  $M'$ . As we only consider  $M$  and can ignore uncertain states in our argumentation, in particular the transitions to  $M'$ , the idea is the same as in the construction for finite automata. Therefore,  $w \in \hat{M}_1$  if and only if we can separate  $w$  into a finite concatenation of words  $w_1 w_2 \cdots w_n$ , such that  $w = w_1 w_2 \cdots w_n$  and each  $w_i \in L(M)_1$  for  $i \leq n$ . Hence  $L(\hat{M})_1 = (L^*)_1$ .

For the rejecting part, first assume  $w \in L(\hat{M})_0$  and, towards a contradiction, that there is a separation  $w = w_1 w_2 \cdots w_n$  such that  $F(\delta^*(q_0, w_i)) \neq 0$  for all  $i \leq n$ . As in our automaton, from any non-rejecting state, we can take an  $\varepsilon$ -transition back to either  $q_0$  or  $q'_0$ , and therefore  $\hat{F}(\hat{\delta}^*(\hat{q}_0, w)) \neq 0$ , contradicting our assumption. Hence,  $L(\hat{M})_0 \subseteq (L^*)_0$ .

Now, assume that  $w \in (L^*)_0$  and, towards a contradiction, that  $w \notin L(\hat{M})_0$ . From the structure of our automaton we therefore know that there must be some separation  $w = w_1 w_2 \cdots w_n$

such that  $F(\delta^*(q_0, w_i)) \neq 0$  for all  $i \leq n$ . This in turn contradicts our original assumption and we conclude that  $(L^*)_0 \subseteq L(\hat{M})_0$  and  $L(\hat{M})_0 \subseteq (L^*)_0$ . Now, once again, since all three parts of  $L^*$  are disjoint and their union is  $\Sigma^*$ ,  $L(\hat{M}) = L^*$  and  $L^*$  is a  $\mathfrak{Z}$ -regular language.

### 3.1.4 Complement

**Theorem 3.1.4.** *The class of  $\mathfrak{Z}$ -regular languages is closed under complement.*

The automaton accepting the complement of a  $\mathfrak{Z}$ -regular language is rather simple. For a D $\mathfrak{Z}$ A  $M$  we can switch previously accepting states to rejecting ones and previously rejecting ones to accepting states. This already results in the desired behavior. We therefore omit illustrating the automaton due to its simplicity.

For a  $\mathfrak{Z}$ -regular language  $L$  with corresponding D $\mathfrak{Z}$ A  $M = (Q, \Sigma, \delta, q_0, F)$ , such that  $L = L(M)$ , construct the D $\mathfrak{Z}$ A  $\hat{M} = (Q, \Sigma, \delta, q_0, \hat{F})$  with:

$$\bullet \hat{F}(q) = \begin{cases} 1 & \text{if } F(q) = 0, \\ ? & \text{if } F(q) = ?, \\ 0 & \text{if } F(q) = 1. \end{cases}$$

Note that no components of  $\hat{M}$  change, except for the output function  $\hat{F}$ .

*Proof Idea.* Since no component other than the output function changed, since both  $M$  and  $\hat{M}$  are deterministic, the execution with word  $w$  results in the same state. We get  $w \in L(M)_0$  if and only if  $w \in L(\hat{M})_1$  and  $w \in L(M)_1$  iff  $w \in L(\hat{M})_0$ . Therefore,  $L(\hat{M}) = L^c$  and  $L^c$  is a  $\mathfrak{Z}$ -regular language.

### 3.1.5 Intersection

**Theorem 3.1.5.** *The class of  $\mathfrak{Z}$ -regular languages is closed under intersection.*

For the intersection, we use the so-called product-construction. The idea is to imitate a simultaneous execution of both automata and accept if and only if both accept and reject if and only if either of the two rejects. We omit giving an illustration as it is similar to that for regular languages.

Consider two  $\mathfrak{Z}$ -regular languages  $L, L'$  and their corresponding D $\mathfrak{Z}$ A  $M = (Q, \Sigma, \delta, q_0, F), M' = (Q', \Sigma, \delta', q'_0, F')$  with  $L = L(M)$  and  $L' = L(M')$ . We construct a D $\mathfrak{Z}$ A  $\hat{M} = (Q \times Q', \Sigma, \hat{\delta}, \hat{q}_0, \hat{F})$  with:

- $\hat{\delta}((q, q'), a) = (\delta(q, a), \delta'(q', a)),$
- $\hat{q}_0 = (q_0, q'_0),$  and
- $\hat{F}(q, q') = F(q) \sqcap F'(q').$

*Proof Idea.* By a simple inductive argument one can show that for any word  $w$ ,  $\hat{\delta}^*(\hat{q}_0, w) = (\delta^*(q_0, w), \delta'^*(q'_0, w))$ . Then

$$\hat{F}(\hat{\delta}^*(\hat{q}_0, w)) = F(\delta^*(q_0, w)) \sqcap F'(\delta'^*(q'_0, w)).$$

Well, from this we already conclude that  $w \in L(\hat{M})_1$  iff  $w \in L_1 \wedge w \in L'_1$  and  $w \in L(\hat{M})_0$  iff  $w \in L_0 \vee w \in L'_0$ . Therefore  $L(M) = L \cap L'$ , and  $L \cap L'$  is  $\exists$ -regular.

### 3.2 Kleene Theorem for $\exists$ -regular languages

With the definition of  $\exists$ -regular expressions in place, we can now show that  $\exists$ -regular expressions and the  $\exists$ -regular languages represent the same class of languages. For this we formulate a theorem similar to the Kleene's Theorem for regular languages.

**Theorem 3.2.1.** *A language  $L$  over an alphabet  $\Sigma$  is a  $\exists$ -regular language iff it is the language of a  $\exists$ -regular expression.*

We will state the two directions in two lemmas and prove them separately.

**Lemma 3.2.2.** *The language of a  $\exists$ -regular expression is a  $\exists$ -regular language.*

*Proof.* Let  $L$  be the language of a  $\exists$ -regular expression of the form  $R = (R_0, R_?, R_1)$  over an alphabet  $\Sigma$ , such that  $L = L(R) = (L(R_0), L(R_?), L(R_1))$ . Then all three  $L_0, L_?$ , and  $L_1$  are pairwise disjoint regular languages, with  $L_0 \cup L_? \cup L_1 = \Sigma^*$  and there exist DFA  $M_? = (Q_?, \Sigma, \delta_?, q_{0?}, F_?)$  and  $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$  such that  $L(M_?) = L_?$  and  $L(M_1) = L_1$ .

From these we can construct a  $\exists$ -valued automaton  $\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \hat{F})$  with:

- $\hat{Q} = Q_? \cup Q_1 \cup \{\hat{q}_0\}$ ,
- $\hat{\delta}(q, a) = \begin{cases} \delta_?(q, a) & \text{if } q \in Q_?, a \in \Sigma, \\ \delta_1(q, a) & \text{if } q \in Q_1, a \in \Sigma, \\ \{q_{0?}, q_{01}\} & \text{if } q = q_0, a = \varepsilon, \\ \emptyset & \text{otherwise.} \end{cases}$
- $\hat{F}(q) = \begin{cases} 1 & \text{if } q \in F_1, \\ ? & \text{if } q \in F_?, \\ 0 & \text{otherwise.} \end{cases}$

We now go on to show that  $L_? = L(\hat{M})_?$  and  $L_1 = L(\hat{M})_1$ , which suffices to show  $L(R) = L(\hat{M})$ , since  $L_0 = (L_? \cup L_1)^c$  and  $L(\hat{M})_0 = (L(\hat{M})_? \cup L(\hat{M})_1)^c$ . The way we constructed our automaton allows for a simple inductive argument showing that

$$\hat{\delta}^*(\hat{q}_0, w) = \{\delta_?^*(q_{0?}, w), \delta_1^*(q_{01}, w)\}.$$

From our starting state  $\hat{q}_0$ , we have to take an  $\varepsilon$ -transition to either  $q_{0?}$  or  $q_{01}$ . From there, the behavior of each original DFA is simulated accordingly. Moreover we get

$$\bigsqcup \hat{F}(\hat{\delta}^*(\hat{q}_0, w)) = F_?( \delta_?^*(q_{0?}, w) ) \sqcup F_1( \delta_1^*(q_{01}, w) ).$$

Now assume  $w \in L_?$  and therefore  $w \notin L_1$ . We get  $w \in L(\hat{M})_?$  since

$$\bigsqcup \hat{F}(\delta^*(\hat{q}_0, w)) = ? \sqcup 0 = ?.$$

Conversely, if  $w \in L(\hat{M})_?$  and therefore  $\bigsqcup \hat{F}(\delta^*(\hat{q}_0, w)) = ?$ , it must be true that  $F_?( \delta_?^*(q_{0?}, w) ) = ?$ , as we can not reach an uncertain state, starting in  $q_{01}$ . Hence,  $w \in L_?$ .

Now assume  $w \in L_1$  and therefore  $w \notin L_?$ . Then,  $w \in L(\hat{M})_1$  since

$$\bigsqcup \hat{F}(\delta^*(\hat{q}_0, w)) = 0 \sqcup 1 = 1.$$

If  $w \in L(\hat{M})_1$ , since starting in  $q_{0?}$  no accepting state is reachable, it follows that  $F_1( \delta_1^*(q_{01}, w) ) = 1$  and therefore  $w \in L_1$ .  $\square$

**Lemma 3.2.3.** *If a language  $L$  is  $\mathfrak{Z}$ -regular, then there exists a  $\mathfrak{Z}$ -regular expression  $R$  such that  $L = L(R)$ .*

*Proof.* Since  $L$  is  $\mathfrak{Z}$ -regular, there exists a  $\mathfrak{Z}$ -valued automaton  $M$  such that  $L = L(M)$ . From Theorem 2.1.6 we know, that there are DFA  $M_0, M_?, M_1$  such that  $L(M) = (L(M_0), L(M_?), L(M_1))$ . Then  $L(M_0) \cup L(M_?) \cup L(M_1) = \Sigma^*$  and they are pairwise disjoint. Let  $R_0, R_?, R_1$  be regular expressions, such that  $L(R_0) = L(M_0)$ ,  $L(R_?) = L(M_?)$  and  $L(R_1) = L(M_1)$ . Then  $(R_0, R_?, R_1)$  is a  $\mathfrak{Z}$ -regular expression and  $L(R) = L(M)$ .  $\square$

### 3.3 Kleene Algebra

In this section we prove several algebraic properties of the operators on  $\mathfrak{Z}$ -regular languages. We first give the definition of a Kleene Algebra and then prove that, just as the regular languages, the  $\mathfrak{Z}$ -regular languages, together with the  $\mathfrak{Z}$ -regular operators and distinguished identity elements satisfy all properties of a Kleene algebra.

**Definition 3.3.1** ([8]). *A Kleene algebra is an algebraic structure  $\langle K, +, \cdot, *, 0, 1 \rangle$  satisfying the following axioms for all  $a, b, c \in K$ :*

1.  $a + (b + c) = (a + b) + c$ ,  
 $a(bc) = (ab)c$ ,
2.  $a + b = b + a$ ,
3.  $a + a = a$ ,

$$4. a(b+c) = ab+ac,$$

$$(a+b)c = ac+bc,$$

$$5. a+0 = a,$$

$$1a = a1 = a,$$

$$6. 0a = a0 = 0,$$

$$7. 1+aa^* = a^*,$$

$$8. 1+a^*a = a^*,$$

$$9. ab \leq b \Rightarrow a^*b \leq b,$$

$$10. ba \leq b \Rightarrow ba^* \leq b$$

where  $a \leq b \Leftrightarrow a+b = b$ .

Note that therefore, from axioms 1 – 6,  $\langle K, +, \cdot, 0, 1 \rangle$  forms an idempotent semiring.

**Theorem 3.3.2** ([8]). *The Set  $\Sigma_{Reg}^*$  containing all regular languages over a finite alphabet  $\Sigma$ , together with the three regular operators, forms a Kleene algebra  $\langle \Sigma_{Reg}^*, \cup, \cdot, *, \emptyset, \epsilon \rangle$ .*

We now go on to show that  $\langle \Sigma_{\mathfrak{Z}}^*, \cup, \cdot, *, 0, 1 \rangle$  is a Kleene algebra, with  $\Sigma_{\mathfrak{Z}}^*$  as the set of all  $\mathfrak{Z}$ -regular languages,  $0 = (\Sigma^*, \emptyset, \emptyset)$  and  $1 = (\Sigma^* \setminus \{\epsilon\}, \emptyset, \{\epsilon\})$ . We first show that all properties of an idempotent semiring are satisfied and then continue to prove the remaining axioms involving the Kleene star.

### 3.3.1 Idempotent Semiring

We state all properties in separate lemmas and in the proofs we rely on the fact that all 3 parts of a  $\mathfrak{Z}$ -regular language are disjoint and their union is  $\Sigma^*$ . It therefore always suffices to show the equality of languages by only considering two parts.

**Lemma 3.3.3.** *For  $\mathfrak{Z}$ -regular languages, union is associative, i.e.,  $(A \cup B) \cup C = A \cup (B \cup C)$  for arbitrary  $\mathfrak{Z}$ -regular languages  $A, B, C$ .*

*Proof.* We show that the accepting and rejecting parts satisfy associativity, i.e.,  $((A \cup B) \cup C)_0 = (A \cup (B \cup C))_0$  and  $((A \cup B) \cup C)_1 = (A \cup (B \cup C))_1$ . We omit writing the lengthy uncertain part and simply write  $(*)$  instead.

$$\begin{aligned} (A \cup B) \cup C &= (A_0 \cap B_0, (*), A_1 \cup B_1) \cup C \\ &= ((A_0 \cap B_0) \cap C_0, (*), (A_1 \cup B_1) \cup C_1) \\ &= (A_0 \cap (B_0 \cap C_0), (*), A_1 \cup (B_1 \cup C_1)) \\ &= A \cup (B \cup C). \end{aligned}$$

□

**Lemma 3.3.4.** *For  $\mathfrak{J}$ -regular languages, concatenation is associative, i.e.,  $(A \cdot B) \cdot C = A \cdot (B \cdot C)$  for arbitrary  $\mathfrak{J}$ -regular languages  $A, B, C$ .*

*Proof.* We first show that the accepting part of a language is associative, which follows directly from the associativity of regular languages.

$$((A \cdot B) \cdot C)_1 = (A_1 \cdot B_1) \cdot C_1 = A_1 \cdot (B_1 \cdot C_1) = (A \cdot (B \cdot C))_1.$$

To show the equality of the two rejecting parts, more work needs to be done. We first consider the resulting terms.

$$\begin{aligned} ((AB)C)_0 &= \left( ((A_?B_? \cup A_1B_? \cup A_?B_1) \setminus A_1B_1)C_? \right. \\ &\quad \left. \cup (A_1B_1)C_? \cup ((A_?B_? \cup A_1B_? \cup A_?B_1) \setminus A_1B_1)C_1 \cup (A_1B_1)C_1 \right)^c \\ (A(BC))_0 &= \left( A_?((B_?C_? \cup B_1C_? \cup B_?C_1) \setminus B_1C_1) \cup A_?(B_1C_1) \right. \\ &\quad \left. \cup A_1((B_?C_? \cup B_1C_? \cup B_?C_1) \setminus B_1C_1) \cup A_1(B_1C_1) \right)^c \end{aligned}$$

We will now start with  $((AB)C)_0$ . By applying the distributivity and associativity laws for regular languages, we get

$$\begin{aligned} ((AB)C)_0 &= \left( ((A_?B_? \cup A_1B_? \cup A_?B_1)C_? \setminus A_1B_1C_?) \right. \\ &\quad \left. \cup A_1B_1C_? \cup ((A_?B_? \cup A_1B_? \cup A_?B_1)C_1 \setminus A_1B_1C_1) \cup A_1B_1C_1 \right)^c. \end{aligned}$$

We can now remove the redundant set differences, as each of the terms  $A_1B_1C_?$  and  $A_1B_1C_1$  also appears in the unions.

$$((AB)C)_0 = (A_?B_?C_? \cup A_1B_?C_? \cup A_?B_1C_? \cup A_1B_1C_? \cup A_?B_?C_1 \cup A_1B_?C_1 \cup A_?B_1C_1 \cup A_1B_1C_1)^c.$$

In order to see the similarity to  $(A(BC))_0$  already, we first reorder the terms and apply distributivity.

$$((AB)C)_0 = (A_?(B_?C_? \cup B_1C_? \cup B_?C_1) \cup A_?(B_1C_1) \cup A_1(B_?C_? \cup B_1C_? \cup B_?C_1) \cup A_1(B_1C_1))^c.$$

We now can add back redundant set differences and we get

$$\begin{aligned} ((AB)C)_0 &= \left( A_?((B_?C_? \cup B_1C_? \cup B_?C_1) \setminus B_1C_1) \cup A_?(B_1C_1) \right. \\ &\quad \left. \cup A_1((B_?C_? \cup B_1C_? \cup B_?C_1) \setminus B_1C_1) \cup A_1(B_1C_1) \right)^c \\ &= (A(BC))_0. \end{aligned}$$

□

**Lemma 3.3.5.** *For  $\mathfrak{J}$ -regular languages, union is commutative, i.e.,  $A \cup B = B \cup A$  for arbitrary*

$\mathfrak{Z}$ -regular languages  $A, B$ .

*Proof.* Commutativity follows directly from the commutativity of  $\cup$  and  $\cap$  for regular languages.

$$\begin{aligned} A \cup B &= (A_0 \cap B_0, (A_? \cap B_?) \cup (A_0 \cap B_?) \cup (A_? \cap B_0), A_1 \cup B_1) \\ &= (B_0 \cap A_0, (B_? \cap A_?) \cup (B_0 \cap A_?) \cup (B_? \cap A_0), B_1 \cup A_1) \\ &= B \cup A. \end{aligned}$$

□

**Lemma 3.3.6.** *For  $\mathfrak{Z}$ -regular languages, union is idempotent, i.e.,  $A \cup A = A$  for an arbitrary  $\mathfrak{Z}$ -regular language  $A$ .*

*Proof.* To show idempotence we make use of the fact that all three parts  $A_0, A_?, A_1$  of a language  $A$  are disjoint and therefore their intersection empty.

$$\begin{aligned} A \cup A &= (A_0 \cap A_0, (A_? \cap A_?) \cup (A_0 \cap A_?) \cup (A_? \cap A_0), A_1 \cup A_1) \\ &= (A_0, A_? \cup \emptyset \cup \emptyset, A_1) \\ &= (A_0, A_?, A_1) \\ &= A. \end{aligned}$$

□

**Lemma 3.3.7.** *For  $\mathfrak{Z}$ -regular languages, concatenation distributes over union, i.e.,*

$$\begin{aligned} (A \cup B) \cdot C &= AC \cup BC, \text{ and} \\ A \cdot (B \cup C) &= AB \cup AC \end{aligned}$$

for arbitrary  $\mathfrak{Z}$ -regular languages  $A, B, C$ .

*Proof.* We first consider all the rejecting and accepting parts of  $(A \cup B) \cdot C$  separately.

$$\begin{aligned} ((A \cup B)C)_1 &= (A_1 \cup B_1)C_1, \text{ and} \\ ((A \cup B)C)_0 &= \left( ((A_? \cap B_?) \cup (A_0 \cap B_?) \cup (A_? \cap B_0))C_? \cup (A_1 \cup B_1)C_? \right. \\ &\quad \left. \cup ((A_? \cap B_?) \cup (A_0 \cap B_?) \cup (A_? \cap B_0))C_1 \cup (A_1 \cup B_1)C_1 \right)^c. \end{aligned}$$

For  $AC \cup BC$  we get the following.

$$\begin{aligned} (AC \cup BC)_1 &= A_1C_1 \cup B_1C_1, \text{ and} \\ (AC \cup BC)_0 &= (A_?C_? \cup A_1C_? \cup A_?C_1 \cup A_1C_1)^c \cap (B_?C_? \cup B_1C_? \cup B_?C_1 \cup B_1C_1)^c. \end{aligned}$$

We now go on to show the equality of the accepting and rejecting parts. For the accepting part the equality follows directly from distributivity of regular languages.

$$((A \cup B) \cdot C)_1 = (A_1 \cup B_1) \cdot C_1 = A_1 C_1 \cup B_1 C_1 = (AC \cup BC)_1.$$

Once again, for the rejecting parts, a bit more work needs to be done. We first apply De Morgan's laws.

$$(AC \cup BC)_0 = (A_? C_? \cup A_1 C_? \cup A_? C_1 \cup A_1 C_1 \cup B_? C_? \cup B_1 C_? \cup B_? C_1 \cup B_1 C_1)^c.$$

We now transform  $((A \cup B)C)_0$  into the same form by first applying distributivity resulting in

$$\begin{aligned} ((A \cup B)C)_0 = & ((A_? \cap B_?)C_? \cup (A_0 \cap B_?)C_? \cup (A_? \cap B_0)C_? \cup A_1 C_? \cup B_1 C_? \\ & \cup (A_? \cap B_?)C_1 \cup (A_0 \cap B_?)C_1 \cup (A_? \cap B_0)C_1 \cup A_1 C_1 \cup B_1 C_1)^c. \end{aligned}$$

We can now add  $(A_? \cap B_1)C_?$  to the union, since  $(A_? \cap B_1)C_? \subseteq B_1 C_?$ . Then, since  $B_0 \cup B_? \cup B_1 = \Sigma^*$  the following equality holds.

$$(A_? \cap B_0)C_? \cup (A_? \cap B_?)C_? \cup (A_? \cap B_1)C_? = A_? C_?.$$

We can apply the same procedure to retrieve  $A_? C_1, B_? C_?$ , and  $B_? C_1$ . If we now substitute all those terms and reorder we get

$$\begin{aligned} ((A \cup B)C)_0 &= (A_? C_? \cup A_1 C_? \cup A_? C_1 \cup A_1 C_1 \cup B_? C_? \cup B_1 C_? \cup B_? C_1 \cup B_1 C_1)^c \\ &= (AC \cup BC)_0. \end{aligned}$$

We omit giving the similar proof for the second law. □

**Lemma 3.3.8.** *For  $\exists$ -regular languages,  $0 = (\Sigma^*, \emptyset, \emptyset)$  is the identity element for union, i.e.,  $A \cup 0 = 0$  for an arbitrary  $\exists$ -regular language  $A$ .*

*Proof.*

$$\begin{aligned} 0 \cup A &= (\Sigma^*, \emptyset, \emptyset) \cup (A_0, A_?, A_1) \\ &= (\Sigma^* \cap A_0, (A_? \cap \emptyset) \cup (A_0 \cap \emptyset) \cup (A_? \cap \Sigma^*), A_1 \cup \emptyset) \\ &= (A_0, \emptyset \cup \emptyset \cup A_?, A_1) \\ &= A. \end{aligned}$$

□

**Lemma 3.3.9.** *For  $\exists$ -regular languages,  $1 = (\Sigma^* \setminus \{\epsilon\}, \emptyset, \{\epsilon\})$  is the identity element for concatenation, i.e.,  $A \cdot 1 = 1 \cdot A = A$  for an arbitrary  $\exists$ -regular language  $A$ .*

*Proof.* Let  $A$  be arbitrary, then

$$\begin{aligned}
 1 \cdot A &= (\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\}) \cdot (A_0, A_?, A_1) \\
 &= ((\emptyset A_? \cup \{\varepsilon\} A_? \cup \emptyset A_1 \cup \{\varepsilon\} A_1)^c, (\emptyset A_? \cup \{\varepsilon\} A_? \cup \emptyset A_1) \setminus \{\varepsilon\} A_1, \{\varepsilon\} A_1) \\
 &= ((A_? \cup A_1)^c, A_?, A_1) \\
 &= A.
 \end{aligned}$$

$$\begin{aligned}
 A \cdot 1 &= (A_0, A_?, A_1) \cdot (\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\}) \\
 &= ((A_? \emptyset \cup A_1 \emptyset \cup A_? \{\varepsilon\} \cup A_1 \{\varepsilon\})^c, (A_? \emptyset \cup A_1 \emptyset \cup A_? \{\varepsilon\}) \setminus A_1 \{\varepsilon\}, A_1 \{\varepsilon\}) \\
 &= ((A_? \cup A_1)^c, A_?, A_1) \\
 &= A.
 \end{aligned}$$

□

**Lemma 3.3.10.** *For  $\mathfrak{Z}$ -regular languages, the annihilator for concatenation is  $0 = (\Sigma^*, \emptyset, \emptyset)$ , i.e.,  $A \cdot 0 = 0 \cdot A = 0$  for an arbitrary  $\mathfrak{Z}$ -regular language  $A$ .*

*Proof.* Let  $A$  be arbitrary, then

$$\begin{aligned}
 0 \cdot A &= (\Sigma^*, \emptyset, \emptyset) \cdot (A_0, A_?, A_1) \\
 &= ((\emptyset A_? \cup \emptyset A_? \cup \emptyset A_1 \cup \emptyset A_1)^c, (\emptyset A_? \cup \emptyset A_? \cup \emptyset A_1) \setminus \emptyset A_1, \emptyset A_1) \\
 &= (\emptyset^c, \emptyset, \emptyset) \\
 &= (\Sigma^*, \emptyset, \emptyset) = 0.
 \end{aligned}$$

$$\begin{aligned}
 A \cdot 0 &= (A_0, A_?, A_1) \cdot (\Sigma^*, \emptyset, \emptyset) \\
 &= ((A_? \emptyset \cup A_1 \emptyset \cup A_? \emptyset \cup A_1 \emptyset)^c, (A_? \emptyset \cup A_1 \emptyset \cup A_? \emptyset)^c, A_1 \emptyset) \\
 &= ((A_? \cup A_1)^c, \emptyset, \emptyset) \\
 &= (\Sigma^*, \emptyset, \emptyset) = 0.
 \end{aligned}$$

□

Finally, we can now, from the preceding lemmas state the following theorem.

**Theorem 3.3.11.**  *$\langle \Sigma_3^*, \cup, \cdot, 0, 1 \rangle$  is an idempotent semiring where  $\Sigma_3^*$  is the set of all  $\mathfrak{Z}$ -regular languages,  $0 = (\Sigma^*, \emptyset, \emptyset)$  and  $1 = (\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\})$ .* □

### 3.3.2 Axioms involving Kleene Star

We now proceed to show that  $\langle \Sigma_3^*, \cup, \cdot, *, 0, 1 \rangle$  satisfies the four remaining axioms involving the Kleene star.

**Lemma 3.3.12.** *For an arbitrary  $\mathfrak{J}$ -regular language  $A$  the following properties hold:*

$$1 \cup AA^* = A^*, \text{ and}$$

$$1 \cup A^*A = A^*,$$

where  $1 = (\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\})$ .

*Proof.* Let  $A$  be an arbitrary  $\mathfrak{J}$ -regular language, then from Theorem 2.2.4 we know  $A^* = \bigcup_{i \in \mathbb{N}} A^i$ . Due to the previously proven associativity and the distributive laws we can now show the first equality.

$$1 \cup A(A^*) = (\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\}) \cup A \cdot \left( \bigcup_{i \in \mathbb{N}} A^i \right) = A^0 \cup \left( \bigcup_{i \in \mathbb{N}} (A \cdot A^i) \right) = A^0 \cup \bigcup_{i \in \mathbb{N}, i \geq 1} A^i = \bigcup_{i \in \mathbb{N}} A^i = A^*.$$

The proof for  $1 \cup A^*A = A^*$  is analogous as  $A \cdot \left( \bigcup_{i \in \mathbb{N}} A^i \right) = \left( \bigcup_{i \in \mathbb{N}} A^i \right) \cdot A$ .  $\square$

**Lemma 3.3.13.** *For arbitrary  $\mathfrak{J}$ -regular languages  $A, B$  the following properties hold:*

$$AB \cup B = B \Rightarrow A^*B \cup B = B, \text{ and}$$

$$BA \cup B = B \Rightarrow BA^* \cup B = B.$$

*Proof.* Consider  $AB \cup B = B \Rightarrow A^*B \cup B = B$ . Let  $A, B$  be arbitrary  $\mathfrak{J}$ -regular languages and  $AB \cup B = B$ . First, we prove  $\forall n \in \mathbb{N}. A^n B \cup B = B$ , by induction on  $n$ .

IB: For  $n = 0$ , the equality follows immediately, since  $A^0 = (\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\})$  is the identity of multiplication.

$$A^0 B \cup B = (\Sigma^* \setminus \{\varepsilon\}, \emptyset, \{\varepsilon\}) B \cup B = B \cup B = B.$$

IH: Let  $n \in \mathbb{N}$  be arbitrary and  $A^n B \cup B = B$  hold.

IS: By substituting from our original assumption, marked with  $(*)$ , we complete the inductive proof.

$$A^{n+1} B \cup B \stackrel{(*)}{=} A^{n+1} B \cup (AB \cup B) = AA^n B \cup AB \cup B = A(A^n B \cup B) \cup B \stackrel{(IH)}{=} AB \cup B \stackrel{(*)}{=} B.$$

We can now use this equality and from idempotence and associativity of  $\cup$ , as well as distributivity we get

$$A^* B \cup B = \left( \bigcup_{i \in \mathbb{N}} A^i \right) B \cup B = \bigcup_{i \in \mathbb{N}} (A^i B \cup B) = \bigcup_{i \in \mathbb{N}} B = B,$$

concluding the proof of the first implication.

We omit a detailed proof for the second implication as it is similar to the first one. With a similar inductive argument as for the first implication we can show  $BA^n \cup B = B$  for  $n \in \mathbb{N}$ .

Therefore, we also get

$$BA^* \cup B = B \left( \bigcup_{i \in \mathbb{N}} A^i \right) \cup B = \bigcup_{i \in \mathbb{N}} (BA^i \cup B) = \bigcup_{i \in \mathbb{N}} B = B.$$

□

**Theorem 3.3.14.**  $\langle \Sigma_3^*, \cup, \cdot, *, 0, 1 \rangle$  is a Kleene Algebra, where  $\Sigma_3^*$  is the set of all 3-regular languages,  $0 = (\Sigma^*, \emptyset, \emptyset)$  and  $1 = (\Sigma^* \setminus \{\epsilon\}, \emptyset, \{\epsilon\})$ . □

# Conclusion

$\mathfrak{J}$ -valued automata are a simple extension of finite automata, or rather, a special case of a Moore Machine with a fixed set of outputs. Drawing inspiration from the coalgebraic perspective on automata, we used a three-element join-semilattice with bottom. This structure imposed on the set of outputs allowed for a concise definition of a nondeterministic  $\mathfrak{J}$ -valued automaton. We adapted the determinization algorithm and showed that this procedure coincides with the classic definition for finite automata. Moreover, we gave a direct proof for a known result, namely that bisimilarity and language equivalence coincide for deterministic finite automata, and extended it to deterministic  $\mathfrak{J}$ -valued automata.

Similar to the regular languages, we defined several operations for our new class, the  $\mathfrak{J}$ -regular languages. We gave constructions to show closure under union, concatenation, Kleene star, intersection, and complement. With these definitions in place,  $\mathfrak{J}$ -regular languages together with union, concatenation, and Kleene star satisfy all properties of a Kleene Algebra. We also defined  $\mathfrak{J}$ -regular expressions, for which we showed the equivalence to  $\mathfrak{J}$ -regular languages via a theorem similar to the Kleene theorem for regular languages.

The intent of this work is to serve as a first step in the direction of defining a more general class of lattice automata. Still, most of the work was carried out with only three output values in mind. This impacted the choice of representation for the languages as the intended semantics of our output values are reflected in the structure of  $\mathfrak{J}$  and also the operations were defined accordingly.

When building on these results by extending the number of elements in the set of outputs, one needs to keep in mind that in fact  $\mathfrak{J}$  is also a totally ordered set. Extending the work to such a subclass of lattice automata should be quite natural, but possibly not of interest. We showed that  $\mathfrak{J}$  is the only three-element join-semilattice with bottom, but for larger output sets, several orders can be defined. The challenge will be to provide a more general approach that allows the introduction of more interesting semantics, such as incomparable outputs.

# Bibliography

- [1] Andreas Bauer, Martin Leucker, and Christian Schallhart. “Monitoring of Real-Time Properties”. In: *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*. Springer Berlin Heidelberg, 2006, pp. 260–272.
- [2] Stanley Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer, 1981.
- [3] John Hopcroft. “An  $n \log n$  algorithm for minimizing states in a finite automaton”. In: *Theory of Machines and Computations*. Academic Press, 1971, pp. 189–196.
- [4] John E. Hopcroft and R. M. Karp. *A linear algorithm for testing equivalence of finite automata*. Technical Report. Cornell University, Dec. 1971.
- [5] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 2nd Edition*. Addison-Wesley-Longman, 2001.
- [6] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge University Press, 2016.
- [7] S. C. Kleene. “Representation of events in nerve nets and finite automata”. In: *Automata Studies*. Princeton, NJ: Princeton University Press, 1956, pp. 3–41.
- [8] Dexter C. Kozen. *Automata and Computability*. 1st. Springer-Verlag, 1997.
- [9] Edward F. Moore. “Gedanken-Experiments on Sequential Machines:” in: *Automata Studies. (AM-34), Volume 34*. Princeton University Press, 2016, pp. 129–154.
- [10] Alexandra Silva, Marcello Bonsangue, and Jan Rutten. “Non-Deterministic Kleene Coalgebras”. In: *Logical Methods in Computer Science* 6 (July 2010).
- [11] Alexandra Silva et al. “Generalizing the powerset construction, coalgebraically”. In: vol. 8. Jan. 2010, pp. 272–283.
- [12] Michael Sipser. *Introduction to the Theory of Computation, Second Edition*. 2005.
- [13] Wikipedia. *Lattice (order)* — Wikipedia, The Free Encyclopedia. [http://en.wikipedia.org/w/index.php?title=Lattice%20\(order\)&oldid=1017201389](http://en.wikipedia.org/w/index.php?title=Lattice%20(order)&oldid=1017201389). [Online; accessed 25-May-2021]. 2021.