UNIVERSITÀ DEGLI STUDI DI UDINE
HIC SUNT FUTURA

DEPARTMENT OF MATHEMATICS, COMPUTER SCIENCE AND PHYSICS

BACHELOR THESIS IN
COMPUTER SCIENCE

Co-lexicographically Ordering Automata

The Hasse Automaton

CANDIDATE
Alessandro Minisini

SUPERVISOR
Prof. Alberto Policriti

Academic Year 2023-2024

Institute Contacts
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

© 2025 Alessandro Minisini
This work is shared under the Creative Commons 4.0 License Attribution-NonCommercial-ShareAlike.

Abstract

Finite automata are fundamental models of computation that recognize regular languages. Despite their simplicity, automata and regular languages remain active areas of research due to their wide applicability in fields like text processing, pattern matching, and formal verification. A key challenge in automata theory is developing efficient representations and algorithms for tasks like searching and compression.

This thesis describes an approach to represent and analyze finite automata using co-lexicographic orderings of their states, focusing on the concept of co-lex width of automata and regular languages, which quantifies an automaton's tendency to be sorted. Moreover, we describe the Hasse automaton as a canonical representation that achieves optimal width for a given language, though not necessarily minimal size. The contribution of this thesis lies in characterizing equivalence relations that induce minimum-state, fixed-width deterministic finite automata (DFAs) for regular languages. We prove that it suffices to consider $\mathcal{L}$-convex equivalences, which respect both the Myhill-Nerode equivalence and the convexity imposed by the co-lex order. These results provide some new insights for identifying "good" equivalences that may lead to minimal-size, minimum-width DFAs.

Contents

Introduction **1**

1 Background **3**

1.1 Symbols, Strings, and Languages . . . 3

1.2 Automata and Regular Languages . . . 4

1.2.1 Automata . . . 4

1.2.2 Regular languages . . . 6

1.3 Relations and DFA minimization . . . 7

1.3.1 Equivalence relations . . . 7

1.3.2 Myhill-Nerode Theorem . . . 8

1.3.3 Minimizing DFAs . . . 9

1.4 Partial orders . . . 9

2 The Co-lexicographic Order **13**

2.1 Co-lexicographic order of strings . . . 13

2.2 Co-lexicographic order of Automata . . . 14

2.3 The co-lex Width . . . 16

2.3.1 Co-lex width on automata . . . 16

2.3.2 Co-lex width of Regular Languages . . . 18

3 The Hasse Automaton **24**

3.1 High-level intuition . . . 24

3.2 Entangled convex decompositions . . . 25

3.3 The Hasse equivalence . . . 26

3.4 The Hasse automaton . . . 30

3.5 $\mathcal{L}$-convex equivalences . . . 34

4 Barrows-Wheeler transform for Automata **40**

4.1 BWT on strings 40

4.2 BWT on DFAs 42

4.2.1 Wheeler DFAs 42

4.2.2 BWT on generic DFAs 44

5 Conclusion **47**

Bibliography **48**

Introduction

Finite automata are fundamental models of computation that recognize regular languages. Despite their simplicity, automata and regular languages remain active research areas due to their wide applicability in fields like text processing[CLRS09], compilers[ALSU06], and speech recognition[MPR05]. In text processing, specifically in pattern-matching tasks, finite automata are the foundation of many efficient algorithms. They provide a representation that allows for fast searching within large texts. This capability is crucial in applications ranging from text editors[Gus97] and search engines[MA03] to intrusion detection systems in cybersecurity[RMV14]. For instance, in bioinformatics, automata-based algorithms are employed to rapidly search for specific DNA sequences within vast genomic databases[MA03]. However, as patterns become more complex and data volumes continue to grow, a key challenge that automata theory has to face is developing more efficient representations while maintaining fast searching capabilities.

This thesis explores an approach to represent and analyze finite automata based on the co-lexicographic orderings of their states. By imposing a partial order on automata states that is compatible with the co-lexicographic order of strings, we obtain useful properties that enable more efficient algorithms and representations. This work, developed in [CDPP23], was presented as an extension to the theory of Wheeler automata and languages [GMS17, ADPP21], which showed that totally ordered automata admit an efficient index data structure for searching for subpaths with a given path label.

The co-lexicographic (co-lex) order on strings provides a natural way to induce partial orders on automata states. We define co-lex orders on automata through two axioms relating the order of states to their incoming transitions. This allows us to associate a new parameter to automata and languages, the width of the co-lex order, which somehow represents the "propensity" of the automaton to be totally ordered.

We describe the theory of co-lex width for both automata and languages, proving fundamental properties and exploring connections to classical concepts in automata theory. A key construct in this theory is the Hasse automaton of a language - a canonical automaton that realizes the language's co-lex width. We show that the Hasse automaton provides an optimal representation in terms of width, though not necessarily in the number of states.

The main contributions of this thesis, presented in Section 3.5, focus on characterizing equivalence relations that induce minimum-state, fixed-width deterministic finite automata (DFAs) for a given regular language. We prove that it suffices to consider $\mathcal{L}$-convex equivalences - those that respect both the Myhill-Nerode equivalence of the language and the convexity imposed by the co-lex order (implicitly defined in [ADPP21]). Our key result shows that, for every DFA $\mathcal{D}$, there exists a $\mathcal{L}$-convex relation that

induces an automaton equivalent to the first, with the same width, and (possibly) less states (Theorem 3.27). This reduces the space of "good" equivalences that may lead to minimal-size, minimum-width DFAs.

The thesis is organized as follows: Chapter 1 provides relevant background on automata theory, regular languages, and partial orders. Chapter 2 introduces co-lexicographic orders on automata and defines the notion of co-lex width for both automata and languages. Chapter 3 describes the theory of the Hasse automaton and presents the results on $\mathcal{L}$-convex equivalences. Finally, Chapter 4 explores applications of this theory to automata compression and indexing, showing how co-lex orders enable an efficient Burrows-Wheeler Transform representation of automata.

1

Background

1.1 Symbols, Strings, and Languages

The smallest element on which the whole thesis is based is the symbol. The *symbol* is a primitive concept that cannot be defined, like the *point* and the *line* in geometry. It intuitively represents the smallest unit of information: generally, letters and digits fulfill this role.

An *alphabet* is a nonempty finite set of symbols, generally denoted by Σ; its cardinality is usually denoted by σ.

A *word* (or *string*) is a finite sequence of symbols over the alphabet Σ. If the sequence is empty, we denoted the word by ε, the *empty string*. Otherwise, the finite sequence is given by $w \stackrel{\text{def}}{=} a_1 a_2 \dots a_n$ where $a_i \in \Sigma$, for each $i \in \{1, \dots, n\}$ and $n \geq 1$. We say that n is the *length* of w, denoted by $|w| = n$. The empty string ε has length 0.

In this thesis, we assume that there is a fixed total order $\preceq$ on Σ (in the examples, the standard alphabetical order is used). We consider that there is a special character, i.e. $\# \notin \Sigma$, which is smaller than any element in Σ.

A *prefix* of a string is any number of leading symbols of that string, and a suffix is any number of trailing symbols. For example, string *abc* has prefixes ε, *a*, *ab*, and *abc*; its suffixes are ε, *c*, *bc*, and *abc*. A prefix or suffix of a string, other than the string itself, is called a *proper* prefix or suffix.

The *concatenation* of two strings is the string formed by writing the first, followed by the second, with no intervening space. For example, the concatenation of *abbra* and *cadraba* is *abbacadabra*. Formally, if w and x are strings, then wx is the concatenation of the two. The empty string is the identity of the concatenation operator, that is, $\varepsilon w = w\varepsilon = w$ for each string w.

A *language* is a set of strings of symbols from one alphabet. The empty set, $\emptyset$, and the set consisting of the empty string $\{\varepsilon\}$ are languages. Another example of language is the set of *palindromes* (strings that read the same forward and backward) over the alphabet $\{0, 1\}$ is an infinite language. Some members of this language are ε, 0, 1, 00, 11, 010 and 1101011. Concatenation can be naturally extended to

languages as follows:

$$w \cdot \mathcal{L} = \{w \cdot v \mid v \in \mathcal{L}\}, \qquad \text{for every string } w \text{ and language } \mathcal{L},$$
$$\mathcal{L}_1 \cdot \mathcal{L}_2 = \{w \cdot v \mid w \in \mathcal{L}_1, v \in \mathcal{L}_2\}, \qquad \text{for every pair of languages } \mathcal{L}_1, \mathcal{L}_2.$$

Another standard operation that can be defined for languages, is the *Kleene star closure*. Let $\mathcal{L}$ be a language and

$$\mathcal{L}^0 = \{\varepsilon\}$$
$$\mathcal{L}^{n+1} = \mathcal{L} \cdot \mathcal{L}^n$$

Then, the Klene star closure of $\mathcal{L}$, denoted by $\mathcal{L}^*$ is the set

$$\mathcal{L}^* = \bigcup_{n \geq 0} \mathcal{L}^n$$

Sometimes the *positive closure* of $\mathcal{L}$ is used, which is defined as $\mathcal{L}^+ = \bigcup_{n \geq 1} \mathcal{L}^n = \mathcal{L} \cdot \mathcal{L}^*$. The operation let us formally define Σ^*, the set of all possible strings over an alphabet Σ.

1.2 Automata and Regular Languages

As we have seen in the previous section, the definition of language is very general. Due to this, much effort was made to characterize and classify various types of languages. One of the most famous and comprehensive works was done by Noam Chomsky. In his work, *Three models for the description of language* [Cho56], he defined a hierarchy of languages, based on their syntax.

The first level, the *Type-0* level, describes the most general languages, those that can be characterized by the Turing Machine.

The second level, *Type-1* level, contains all the languages that can be *decided*, that is, we can establish if a string belongs or not to a specific language in a finite amount of time.

In the third level, *Type-2* level, we find languages that are called *context-free* languages. These are less expressive than the previous ones but can be recognized by simpler machines (i.e. pushdown automata, which we will not talk about in this thesis).

The last level, *Type-3* level, contains the simplest languages, which are called *regular languages*. Finite State Automata can characterize these types of languages, so, in this section, we will first introduce this model of computation and then we will describe the properties of regular languages.

1.2.1 Automata

The finite automaton is a mathematical model of a system, with discrete inputs and outputs. The system can be in any one of a finite number of internal configurations or *states*.

To move between states, the system also defines a set of transitions that are triggered by an input symbol from the alphabet Σ. For each input symbol, there can be one or more transitions out of each state (possibly back to the state itself). One state, usually denoted by q_0 or s, is the initial state, in which the automaton starts. Some states are considered final or accepting states. Before defining finite automata formally, we must distinguish between *deterministic* and *non-deterministic* finite automata. The difference is inside the transition function: in the deterministic case, in each state, there must be at most one transition per symbol. Conversely, in the non-deterministic case, for each state, there can be an arbitrary number of transitions labeled with the same symbol.

Definition 1.1. (Non-deterministic finite state automata)

A *non-deterministic finite state automaton* (NFA) is a 5-tuple $\mathcal{N} = (Q, \Sigma, \delta, s, F)$ where:

- Q is a finite set of states;
- Σ is an alphabet;
- $\delta : Q \times \Sigma \to \mathscr{P}(Q)$ is the transition function;
- s is the initial state;
- $F \subseteq Q$ is the set of final states.

We say that $\mathcal{N}$ is *complete* if $|\delta(q, a)| \geq 1$ for every state and symbol. Usually, this distinction is not necessary, since it is always possible to complete an automaton by adding one state - the sink - and suitable transitions. However, when order comes into play, this trick cannot be done (as shown in [ADPP21]).

The transition function can be inductively extended to strings, that is, we define $\hat{\delta}\, Q \times \Sigma^* \to \mathscr{P}(Q)$ as follows:

$$\begin{cases} \hat{\delta}(q, \varepsilon) = q, \\ \hat{\delta}(q, a \cdot w) = \bigcup_{p \in \delta(q,a)} \hat{\delta}(p, w), \quad \text{if } a \in \Sigma \text{ and } w \in \Sigma^*. \end{cases}$$

Since there will be no ambiguity, $\hat{\delta}$ is usually denoted by δ.

We say that state $q \in Q$ *accepts* a string $w \in \Sigma^*$ if $\delta(q, w) \cap F \neq \emptyset$. The language accepted by state q, that is, the set of strings accepted by q, is denoted by $\mathcal{L}(q)$. The language accepted by an automaton is $\mathcal{L}(\mathcal{N}) = \mathcal{L}(s)$. We say that two automata are *equivalent* if they accept the same language.

In further analysis, it will be useful to consider that all states of automata are *useful*, that is, from each state one can reach a final state (possibly, the state itself). We assume also that every state is reachable from the initial state. In addition, it will be useful to define:

- $\lambda(u)$, the set of all symbols that label an incoming transition of $u \in Q$;
- $\mathrm{Pref}(\mathcal{L}(\mathcal{N}))$, the set of strings that can be reached on $\mathcal{N}$ from the initial state;
- I_u, the set of strings *reaching* a state $u \in Q$ from the initial state:

$$I_u = \{\alpha \in \mathrm{Pref}(\mathcal{L}(\mathcal{N})) \mid u \in \delta(s, \alpha)\};$$

- I_α, the set $\delta(s, \alpha)$ of all states *reached* from the initial state by $\alpha \in \text{Pref}(\mathcal{L}(\mathcal{N}))$.

A *deterministic finite state automaton* (DFA) is an NFA $\mathcal{D}$ with the added condition that, for each symbol a and each state q, $|\delta(q, a)| \leq 1$. If the automaton is deterministic we write $\delta(q, a) = p$ for the unique p such that $\delta(q, a) = \{p\}$ (if defined, since we are not assuming a DFA to be complete).

1.2.2 Regular languages

Regular languages are defined recursively and are easily described by simple expressions called regular expressions.

Definition 1.2 (Regular expressions)**.** Let Σ be an alphabet. The *regular expressions* over Σ and the sets that they denote are defined recursively as follows:

1. $\emptyset$ is a regular expression and denotes the empty set;
2. ε is a regular expression and denotes the set $\{\varepsilon\}$;
3. for each $a \in \Sigma$, a is a regular expression and denotes the set $\{a\}$.
4. if r and s are regular expressions denoting the languages R and S, respectively, then $(r + s)$, rs, r^* are regular expressions that denote the sets $R \cup S$, $R \cdot S$ and R^*, respectively.

Moreover, the language represented by a regular expression r is denoted by $\mathcal{L}(r)$.

The following are some examples of regular languages, accompanied by their respective regular expressions. It can be proven that Regular Expressions and Finite State Automata recognize the same set of languages, named *Regular Languages*.

Theorem 1.3. *Let $\mathcal{L} \subseteq \Sigma^*$ be a language. Then there exists a regular expression r such that $\mathcal{L} = \mathcal{L}(r)$ if and only if there exists a DFA $\mathcal{D}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{D})$.*

Closure properties

Since languages are sets of strings, we can talk about union, intersection, and complementation. In this paragraph, we will see that these properties are preserved for regular languages.

Theorem 1.4. *Regular languages are closed under union, concatenation, and Kleene star closure.*

Proof. Immediate from the definition of regular expressions. □

Theorem 1.5. *Regular languages are closed under complementation. That is, if $\mathcal{L}$ is a regular language and $\mathcal{L} \subseteq \Sigma^*$, then $\overline{\mathcal{L}} = \Sigma^* \setminus \mathcal{L}$ is a regular language.*

Proof. Let L be $\mathcal{L}(\mathcal{D})$ for DFA $\mathcal{D} = \{Q, \Sigma', \delta, s, F\}$ and let $\mathcal{L} \subseteq \Sigma^*$. First, we may assume $\Sigma' = \Sigma$: if Σ' has some symbols that do not belong to Σ, we may delete all transitions of M on symbols not in Σ. If there are symbols in Σ not in Σ', then none of these symbols appear in strings of $\mathcal{L}$. We may therefore introduce a "dead state" that collects all the unuseful transitions.
Then, the automaton $\mathcal{D} = \{Q, \Sigma', \delta, s, Q \setminus F\}$ recognizes $\Sigma^* \setminus \mathcal{L}$. □

Theorem 1.6. *Regular languages are closed under intersection.*

Proof. The thesis follows immediately from the fact that $\mathcal{L}_1 \cap \mathcal{L}_2 = \overline{\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}}$. □

It is worth noting that a direct construction of a DFA for the intersection of two regular languages exists. The construction involves taking the Cartesian product of states and is defined as follows. Let $\mathcal{D}_1 = \{Q_1, \Sigma, \delta_1, s_1, F_1\}$ and $\mathcal{D}_2 = \{Q_2, \Sigma, \delta_2, s_2, F_2\}$ be two DFAs. Let

$$\mathcal{D} = \{Q_1 \times Q_2, \Sigma, \delta, \{s_1, s_2\}, F_1 \times F_2\},$$

where for all $q_1 \in Q_1$, $q_2 \in Q_2$, and a in Σ,

$$\delta(\{q1, q2\}, a) = \{\delta_1(q_1, a), \delta_2(q_2, a)\}.$$

It can be shown that $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{D}_1) \cap \mathcal{L}(\mathcal{D}_2)$.

1.3 Relations and DFA minimization

In this section, we will talk about relations, in particular equivalence relations. In addition, we will see two important equivalences that appear in Automata theory.

A *binary relation* is a set of pairs. The first component of a relation is chosen from a set called *domain*, and the second component is chosen from a (possibly different) set called *range*. Usually, if R is a relation and (a, b) is a pair in R, then we can write aRb and say that aRb is true (conversely, if (a, b) is not in R, we say that aRb is false).

The following properties usually characterize relations. We can say that a relation R on set S is

- *reflexive* if aRa for all a in S;
- *irreflexive* if aRa is false for all a in S;
- *transitive* if aRb and bRc implies aRc;
- *symmetric* if aRb implies bRa;
- *antisymmetric* if aRb implies that bRa is false.

Note that any symmetric relation must be irreflexive.

1.3.1 Equivalence relations

Given a nonempty set S, an *equivalence relation* $\approx$ on S is a binary relation that is reflexive, symmetric, and transitive.

Every equivalence relation $\approx$ induces a *partition* $P_{\approx}$ of S, i.e. a family of subsets $S_1, S_2, \ldots$ such that:

1. $S_i \neq \emptyset$ for all i;

2. $S_i \cap S_j = \emptyset$ for all $i \neq j$;
3. $S = S_1 \cup S_2 \cup \ldots$;
4. for each a and b in S_i, $a \approx b$ is true;
5. for each $a \in S_i$ and $b \in S_j$, $a \approx b$ is false.

The S_i are called *equivalence classes.*

We say that a partition is $finite$ if there is a finite number of S_i. Then, if $P_\approx$ is a finite partition, $\approx$ is an equivalence relation of *finite index*, i.e. $\approx$ defines a finite number of equivalence classes. Given two equivalence relations $\approx_1$ and $\approx_2$, we say that $\approx_1$ is *finer or equal* to $\approx_2$ when $\approx_1 \subseteq \approx_2$. Sometimes, we also say that $\approx_2$ is *coarser or equal* to $\approx_1$.

In the next Subsection, we will use the notion of *right-invariance*: Let S be a set of elements that you can concatenate (usually, strings). Then, the equivalence $\approx$ is right invariant if and only if $x \approx y$ implies $xz \approx yz$ for all $z \in S$.

1.3.2 Myhill-Nerode Theorem

In *Myhill-Nerode theorem*, two different equivalence relations are presented. The first is between strings of an arbitrary language: $x \approx_{\mathcal{L}} y$ if and only if for each z, either both or neither of xz and yz is in $\mathcal{L}$. The second is between strings associated with a finite automaton. Let $\mathcal{D} = \{Q, \Sigma, \delta, s, F\}$ be a DFA. For x and y in Σ^*, let $x \approx_{\mathcal{D}} y$ if and only if $\delta(s, x) = \delta(s, y)$. We will see that, in regular languages, these two relations are related, in particular, the second is a refinement of the first.

Theorem 1.7 (Myhill-Nerode, [HMU01])**.** *The following three statements are equivalent:*

1. *The set $\mathcal{L} \subseteq \Sigma^*$ is accepted by some finite automaton.*
2. *$\mathcal{L}$ is the union of some of the equivalence classes of a right invariant equivalence relation of finite index.*
3. *Let the equivalence relation $\approx_{\mathcal{L}}$ be defined by: $x \approx_{\mathcal{L}} y$ if and only if for all $z \in \Sigma^*$, xz is in $\mathcal{L}$ exactly when yz is in $\mathcal{L}$. Then $\approx_{\mathcal{L}}$ is of finite index.*

Proof.

(1) $\Rightarrow$ (2) Assume that L is accepted by some DFA $\mathcal{D} = \{Q, \Sigma, \delta, s, F\}$. Let $\approx_{\mathcal{D}}$ be the equivalence relation $x \approx_{\mathcal{D}}$ if and only if $\delta(s, x) = \delta(s, y)$. $\approx_{\mathcal{D}}$ is right invariant since, for any z, if $\delta(s, x) = \delta(s, y)$, then $\delta(s, xz) = \delta(s, yz)$. The index of $\approx_{\mathcal{D}}$ is finite since the index is at most the number of states in Q. Furthermore, $\mathcal{L}$ is the union of the equivalence classes that include a string x such that $\delta(s, x)$ is in F, that is, the equivalence classes corresponding to final states.

(2) ⇒ (3) We show that any equivalence relation $\approx$ satisfying (2) is a refinement of $\approx_{\mathcal{L}}$; that is, every equivalence class of $\approx$ is entirely contained in some equivalence class of $\approx_{\mathcal{L}}$. Thus the index of $\approx_{\mathcal{L}}$ cannot be greater than the index of $\approx$ and so is finite. Assume that $x \approx y$. Then since $\approx$ is right invariant, for each z in Σ^*, $xz \approx yz$, and thus ys is in $\mathcal{L}$ if and only if xz is in $\mathcal{L}$. Thus $x \approx_{\mathcal{L}} y$, and hence the equivalence class of x in $\approx$ is contained in the equivalence of x in $\approx_{\mathcal{L}}$. We conclude that each equivalence class o $\approx$ is contained within some equivalence class of $\approx_{\mathcal{L}}$.

(3) ⇒ (1) We must first show that $\approx_{\mathcal{L}}$ is right invariant. Suppose $x \approx_{\mathcal{L}} y$ and let $w \in \Sigma^*$. We must prove that $xw \approx_{\mathcal{L}} yw$; that is, for any z, xwz is in $\mathcal{L}$ exactly when ywz is in $\mathcal{L}$. But since $x \approx_{\mathcal{L}} y$, we now by definition of $\approx_{\mathcal{L}}$ that $x(wz)$ is in $\mathcal{L}$ exactly when $y(wz)$ is in $\mathcal{L}$. So $\approx_{\mathcal{L}}$ is right-invariant. Now let Q' be the finite set of equivalence classes of $\approx_{\mathcal{L}}$ and $[x]$ the element of Q' containing x. Define $\delta'([x], a) = [xa]$. The definition is consistent since $\approx_{\mathcal{L}}$ is right invariant. Let $s' = [\varepsilon]$ and let $F' = \{x \mid x \in \mathcal{L}\}$. The finite automaton $\mathcal{D}' = \{Q', \Sigma, \delta', s', F'\}$ accepts $\mathcal{L}$, since $\delta'(s', x) = [x]$, and thus x is in $\mathcal{L}(\mathcal{D}')$ if and only if $[x]$ is in F'.

□

1.3.3 Minimizing DFAs

As a consequence of the Myhill-Nerode theorem, there is an essentially unique minimum-size DFA for every regular language.

Theorem 1.8. *The minimum-size automaton accepting a set $\mathcal{L}$ is unique up to isomorphism and is given by $\mathcal{D}'$ in the proof of Theorem 1.7.*

Proof. In the proof of Theorem 1.7 we saw that any DFA $\mathcal{D} = \{Q, \Sigma, \delta, s, F\}$ accepting $\mathcal{L}$ defines an equivalence relation that is a refinement of $\approx_{\mathcal{L}}$. Thus the number of states of $\mathcal{D}$ is greater than or equal to the number of states of $\mathcal{D}'$ of Theorem 1.7. If equality holds, then each of the states of $\mathcal{D}$ can be identified with one of the states of $\mathcal{D}'$. That is, every state q of $\mathcal{D}$ can be mapped to $\delta'(s', x)$, where x is any string such that $\delta(s, x) = q$. It is easy to see that this map is consistent, so the theorem follows. □

1.4 Partial orders

The co-lexicographic order defined in Chapter 2 is a partial order. Thus, in this section, we will introduce partial orders and their main properties. Further, we will define chains, antichains, and the width. Finally, we will state an important theorem of partial orders, Dilworth's theorem.

Definition 1.9 (Partial order)**.** A relation $\leq$ on a set A is a *partial order* if it is transitive, antisymmetric, and reflexive. The relation is said to be a *strong partial order* if it is irreflexive instead of reflexive.

Examples of partial orders include "$\leq$" on $\mathbb{R}$, "$\subseteq$" on the set of subsets of A, and the "divides" relation on $\mathbb{N}^+$. Examples of strong partial orders include "$<$" and $\subset$.

Analogously to equivalences, we say that $\leq'$ is *finer* than $\leq$ if $\leq' \subseteq \leq$; otherwise, if $\leq \subseteq \leq'$, $\leq'$ is *coarser* than $\leq$.

A partial order is "partial" because there can be two elements with no relation between them. In general, we say that two elements a and b are *incomparable* if neither $a \leq b$ nor $b \leq a$ holds; in this case, we write $a||b$. Conversely, we say that a and b are *comparable*.

If there is no pair a, b of incomparable elements, we say that the order is *total* (or *linear*).

Definition 1.10 (Total order)**.** Let $\leq$ be a partial order on set A. if, for every pair a, b, either $a \leq b$ or $b \leq a$ holds, then the order is called *total order*.

Among the previous examples, $\leq$ on $\mathbb{R}$ is a total order because we can always compare two real numbers. Conversely, $\subseteq$ and "divides" are not total orders.

The definition of partially ordered sets comes naturally from the definition of partial order.

Definition 1.11. Given a partial order $\leq$ on a set A, the pair $(A, \leq)$ is called a *partially ordered set* or *poset*.

In terms of graph theory, a poset is a directed graph $G = (A, \leq)$ with vertex set A and edge set $\leq$. We can notice that every graph induced by a poset, without considering self-loops, is acyclic (i.e. it is a DAG).

Theorem 1.12. *A poset has no directed cycles other than self-loops.*

Proof. We use proof by contradiction. Let $(A, \leq)$ be a poset. Suppose that there exists $n \leq 2$ distinct elements $a_1, a_2, a_3, \ldots, a_n$ such that

$$a_1 \leq a_2 \leq a_3 \leq \cdots \leq a_{n-1} \leq a_n \leq a_1.$$

Since $a_1 \leq a_2$ and $a_2 \leq a_3$, transitivity implies $a_1 \leq a_3$. If we iterate this argument, we obtain that $a_1 \leq a_n$. Since we know that $a_n \leq a_1$, asymmetry implies $a_1 = a_n$, contradicting the supposition that $a_1, \ldots, a_n$ are distinct. Thus, there is no such directed cycle. □

One problem with viewing a poset as a directed graph is that there tend to be lots of edges due to the transitivity property. To avoid it, we can omit self-loops, implied by the reflexive property, and all the edges implied by the transitivity property. This type of representation is called *Hasse diagram*.

Definition 1.13. Given a poset $(A, \leq)$, a *Hasse diagram* is a directed graph with vertex set A and edge set $\leq$, minus all self-loops and edges implied by transitivity.

The following Hasse diagram represents the poset $(\{1, \ldots, 12\}$, "*divides*"$)$.

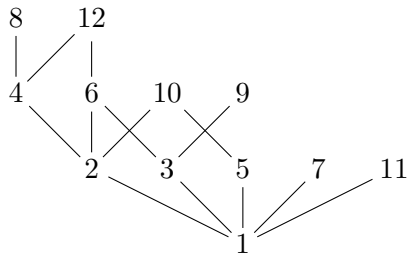


Figure 1.1: Hasse diagram of $(\{1,\ldots,12\}, divides)$.

In posets, the definition of minimum and maximum is not always well-defined, since not all elements may not be comparable with the hypothetical minimum or maximum. We can define minimal and maximal elements instead of the minimum and the maximum. In a poset $(A, \leq)$, an element $x \in A$ is *minimal* if there is no other element $y \in A$ such that $y \leq x$. Similarly, an element z is *maximal* if there is no other element $w \in A$ such that $x \leq w$.

The next theorem proves the existence of minimal and maximal elements in finite posets.

Theorem 1.14. *Every finite poset has a minimal (maximal) element.*

Proof. Let $(A, \leq)$ be a poset. Let $a_1, a_2, \ldots, a_n$ be a maximum-length sequence of distinct elements in A such that

$$a_1 \leq a_2 \leq \cdots \leq a_n.$$

The existence of such a maximum-length sequence follows from the Well Ordering Principle and the fact that A is finite. Now $a \leq a_1$ cannot hold for any $a \in A$ not in the sequence, since it already has maximum length. Since $a_i \not\leq a_1$ for $2 \leq i \leq n$ follows from Theorem 1.12, a_1 is a minimal element.

The proof of the existence of the maximal element is analogous. □

A sequence of elements that can be totally ordered, as the one used in the Theorem 1.14, is called *chain*.

Definition 1.15. A chain is a sequence $a_1 \leq a_2 \leq \cdots \leq a_h$, where $a_i \leq a_j$ for all $i \leq j$, such that each element is comparable to the next in the chain, and it is smaller with respect to $\leq$. The length of the chain is h, the number of elements in the chain.

When the chain cannot be extended in any direction, it is called a *maximum-length chain*. The cardinality of a maximum-length chain is called the *height* of the poset (denoted by height(A)).

The "dual" version of the chain is the *antichain*, that is, a subset of pairwise incomparable elements.

Definition 1.16. An antichain is a sequence $a_1, a_2, \ldots, a_w$ where, for every pair of distinct elements a_i, a_j, we have that $a_i || a_j$.

Similar to the height, we have the *width*: the largest cardinality of an antichain (denoted by width(A)). The width will be the crucial concept of this thesis since it will be used to classify different types of languages and automata.

The following results talk about the ways to partition a finite poset $(A, \leq)$ into chains or antichains. Thus, it is useful to define formally these concepts.

Definition 1.17. A *chain (or antichain) decomposition* of a poset $(A, \leq)$ is a partition $A_1, A_2, \ldots, A_n$ such that every block A_i is a chain (or antichain).

In a finite poset, it is easy to see that a chain C and an antichain A on a poset $(X, \leq)$ have at most one element in common. Hence the least number of antichains whose union is X is not less than the size of the height. It can be shown that an antichain partition of such length always exists.

There is a kind of dual statement, known as *Dilworth theorem.*

Theorem 1.18 (Dilworth, [Dil50])**.** *Let $(A, \leq)$ be a finite poset. Then there is a chain decomposition of A of size* width(A).

Proof (F. W. Galvin, [Gal94]). We will prove the theorem by induction. Firstly, if A is empty, then the theorem holds. So, let's assume that A is nonempty, and let a be a maximal element of A.

By induction, we assume that for some integer k the poset $A' = A \setminus \{a\}$ can be decomposed into k chains $C_1, C_2, \ldots, C_k$. Let N_0 be an antichain of size k. We can see that $N_0 \cap C_i \neq \emptyset$ for every i. Let x_i be the maximal element in C_i that belongs to an antichain of size k in A', and set $N = \{x_1, x2, \ldots, x_k\}$.

> **Claim.** *N is an antichain.*
>
> *Proof.* Let N_i be an antichain of size k that contains x_i. We can see, again, that $N_i \cap C_j \neq \emptyset$ for every j. Let's fix indices i and j. Let $y \in N_i \cap C_j$. Then $y \leq x_j$, by definition of x_j. This implies that $x_i \not\geq x_j$, since $x_i \not\geq y$. Let's swap the indices i and j: we obtain that $y \leq x_i$, so $x_i \not\geq y$ and $x_j \not\geq x_i$. So N is an antichain. □

The end of the proof is divided into two cases.

1. If $a \geq x_i$ for some $i \in \{1, 2, \ldots, k\}$, then let K be the chain $\{a\} \cup \{z \in C_i \mid z \leq x_i\}$. By the choice of x_i, $A \setminus K$ does not have an antichain of size k. Induction then implies that $A \setminus K$ can be covered by $k-1$ disjoint chains since $N \setminus \{x_i\}$ is an antichain of size $k-1$ in $A \setminus K$. Thus, A can be decomposed into k chains.

2. If $a \not\geq x_i$ for each $i \in \{1, 2, \ldots, k\}$, then $N \cup \{a\}$ is an antichain of size $k+1$ in A (since a is maximal in A). Now A can be covered by the $k+1$ chains $\{a\}, C_1, C_2, \ldots, C_k$, completing the proof.

It is worth noting that in the first case, the size of the antichain does not change between A' and A (in particular, width(A') = width(A)), while in the second increases by 1. □

2

The Co-lexicographic Order

The co-lexicographic order, often abbreviated as co-lex-order, is a fundamental concept in this thesis. It provides an alternative way to order strings compared to the more commonly known lexicographic order: preferring the first instead of the second will simplify its generalization to the states of automata. In this chapter, we will define the co-lexicographic order of strings and immediately adapt this concept to automata. We will highlight that this order is a partial order over the set of states and we will introduce the meaning of the width of this poset. Then we will generalize this definition to regular languages: after describing its closure properties, we will introduce the concept of entanglement, a measure related to the width.

2.1 Co-lexicographic order of strings

Let Σ be a set of symbols, namely, an alphabet. As said in Chapter 1, it is usual to fix an order on Σ: for the English alphabet, we have that $a \preceq b \preceq c \preceq \ldots \preceq z$ and for digits we have $0 \preceq 1 \preceq 2 \preceq \ldots \preceq 9$.

$\preceq$ can be easily extended to strings: we start comparing strings from left to right and, when we find that the strings differ, we assign them the order of the two different characters we encountered. This order is called *lexicographic order* and is the standard order that vocabularies use.

Definition 2.1 (Lexicographic order)**.** Let α and β be two strings on the alphabet Σ. We say that $\alpha \preceq \beta$ if and only if α is a prefix of β, or there exists $\alpha', \beta', \gamma \in \Sigma^*$ and $a, b \in \Sigma$, such that $\alpha = \gamma a \alpha'$ and $\beta = \gamma b \beta'$ and $a \prec b$.

The lexicographic order is the base of many algorithms and data structures used for string processing. For example, suffix arrays store all suffixes of a string in sorted lexicographic order, enabling efficient string matching and pattern searching. Another more sophisticated example is the Barrows-Wheeler transform. It can be obtained by sorting all its cyclic rotations of a string lexicographically and extracting the last column, creating a reversible permutation that groups similar characters together. This lexicographic sorting in BWT facilitates effective data compression.

The dual version of the lexicographic order is the *co-lexicographic order*: instead of comparing the string

from left to right, we do it in reverse order.

Definition 2.2 (Co-lexicographic order)**.** Let α and β be two strings on the alphabet Σ. We say that $\alpha \preceq \beta$ if and only if α is a suffix of β, or there exists $\alpha', \beta', \gamma \in \Sigma^*$ and $a, b \in \Sigma$, such that $\alpha = \alpha' a \gamma$ and $\beta = \beta' b \gamma$ and $a \prec b$.

Although they are practically equivalent on strings, the latter can be extended easily to automata: each state has an incoming transition labeled with the final character of every string that reaches it.

The co-lex order is a total order over strings: since all the symbols are comparable, it is always possible to establish which string comes first in the order. Another important property is the monotonicity: if $\alpha \preceq \beta$, then for all $\gamma \in \Sigma^*$ $\alpha\gamma \leq \beta\gamma$.

2.2 Co-lexicographic order of Automata

This section will extend the co-lexicographic order of strings to automata. Although notions defined here can be easily adapted to NFAs, we will see only the deterministic case, since in the following sections we will not deal with nondeterminism.

First, as usual for string processing algorithms, let's add a symbol to the alphabet Σ, namely #, smaller than all the other symbols. Then let's add an *ad-hoc* transition labeled with # to the initial state. This will avoid some annoying corner cases, making the theory more clear.

If we see the states as sets of strings - the set I_u of all strings reaching a state u - then a natural way to define the order is the following:

$$u \leq v \Rightarrow (\forall \alpha \in I_u)\,(\forall \beta \in I_v)\ (\alpha \prec \beta) \tag{2.1}$$

Where $\preceq$ is the co-lexicographic order on strings.

Roughly speaking, if we want that if $u \leq v$, all the strings in I_u must be smaller than those in I_v.

The above property characterizes the order globally: to compare two states, we must consider paths that end in those states. The definition that we will use, conversely, is local.

Definition 2.3. Let $\mathcal{D} = \{Q, s, \delta, F\}$ be an DFA. A co-lexicographic order $<$ on $\mathcal{D}$ is a partial order on Q that satisfies the following two axioms:

1. (Axiom 1) For every $u, v \in Q$, if $u < v$, then $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$;
2. (Axiom 2) For every $a \in \Sigma$ and $u, v, u', v' \in Q$, if $u \in \delta(u', a)$, $v \in \delta(v', a)$ and $u < v$, then $u' < v'$.

Axiom 1 implies that $<$ must respect the order of the incoming characters (i.e. strings of length 1) and Axiom 2 propagates this property backward. When $\leq$ is a total order, we say it is a *Wheeler order*. *Wheeler automata* recognizes only a limited class of languages (called Wheeler languages). In contrast, any automaton admits a co-lex order: for example, the order $\{(u, u) \mid u \in Q\}$ and the order $\{(u, v) \mid \max_{\lambda(u)} \prec \min_{\lambda(v)}\} \cup \{(u, u) \mid u \in Q\}$ are co-lex orders.

Remark 2.4. Since $\# \in \lambda(s)$ and $\# \in \lambda(u)$ for $u \neq s$, then from Axiom 1 it follows that for every $u \in Q$ it holds $u \not\preceq s$.

This implies that s is always the minimum element of the partial order.

Let's note that Definition 2.3 implies Property 2.1.

Theorem 2.5. *If $\leq$ is a co-lex order over a DFA and $u \leq v$ then $(\forall \alpha \in I_u)\,(\forall \beta \in I_v)\ (\alpha \prec \beta)$.*

Proof. Suppose, by contradiction, that for some distinct $u, v \in Q$, $u < v$ and there exists $\alpha \in I_u$, $\beta \in I_v$ with $\alpha \prec \beta$. Let us fix u, v, α, and β with the above properties such that β has the minimum possible length. Notice that β cannot be the empty string, otherwise v would be the initial state s, while $z \not\preceq s$ for all $z \in Q$ (see Remark 2.4). Hence, $\beta = \beta' e$ for some $e \in \Sigma$ and $\beta \prec \alpha$ implies $\alpha = \alpha' f$ for some $f \in \Sigma$. We then have $e \in \lambda(v)$, $f \in \lambda(u)$, and for Axiom 1 of co-lex orders we get $f \preceq e$. From $\beta = \beta' e \prec \alpha = \alpha' f$ we conclude $f = e$ and $\beta' \prec \alpha'$. If u', v' are such that $\delta(u', e) = u$, $\delta(v', e) = v$ and $\alpha' \in I_{u'}$, $\beta \in I_{v'}$, then by Axiom 2 of co-lex orders we get $u' \leq v'$; however, u', v', α' and β' has the same properties that we stated at the beginning, so β is not minimal. □

The converse does not always hold. Indeed, consider the DFA in Figure 2.1 and the co-lex order $u \leq v \iff u = v$. Since $I_1 = (a+b)^*a$ and $I_2 = (a+b)^*b$, all the strings in I_1 are smaller than the strings in I_2 (strings that ends with character a are smaller than strings that ends with b), but state 1 is not $\leq$-comparable with state 2 because they are different.

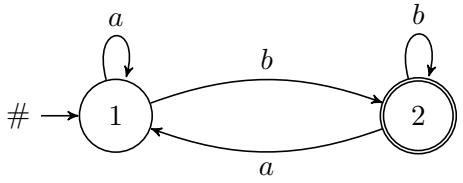


Figure 2.1: A DFA that recognizes the language $\mathcal{L} = (a+b)^*b$.

As we will see, turning the right implication of Theorem 2.5 into an *if-and-only-if* will make the order *maximal*, that is, the order that refines all the other valid co-lex orders on $\mathcal{D}$.

Definition 2.6. Let $\mathcal{D}$ be a DFA. The relation $<_{\mathcal{D}}$ over Q is defined by:

$$u \leq_{\mathcal{D}} v \iff (\forall \alpha \in I_u)\,(\forall \beta \in I_v)\,(\alpha \prec \beta)\,.$$

Theorem 2.7. *Let $\leq_{\mathcal{D}}$ be $<_{\mathcal{D}} \cup \{(u,u) \mid u \in Q\}$. Then, if $\mathcal{D}$ is a DFA, $(Q, \leq_{\mathcal{D}})$ is the maximum co-lex order on $\mathcal{D}$.*

Proof. First, let us prove that $\leq_{\mathcal{D}}$ is a co-lex order on $\mathcal{D}$.

To see that Axiom 1 holds assume that $u <_{\mathcal{D}} v$: we must prove that $e = \max_{\lambda(u)} \preceq e' = \min_{\lambda(v)}$. Notice that it must be $v \neq s$ because the empty string ε is in I_s and ε is co-lexicographically smaller than any other string. Hence, $e' \succ \#$ and if $e = \#$ we are done. Otherwise, there are $\alpha e \in I_u$ and $\alpha' e' \in I_v$, so that $u <_{\mathcal{D}} v$ implies $\alpha e \prec \alpha' e'$ and therefore $e \preceq e'$. As for Axiom 2, assume that $u \in \delta(u', a)$,

$v \in \delta(v', a)$, and $u <_{\mathcal{D}} v$. We must prove that $u' <_{\mathcal{D}} v'$. Fixing $\alpha \in I_{u'}$ and $\beta \in I_{v'}$, we must prove that $\alpha \prec \beta$. We have $\alpha a \in I_u$ and $\beta a \in I_v$, hence from $u <_{\mathcal{D}} v$ it follows $\alpha a \prec \beta a$, and therefore $\alpha \prec \beta$.

The fact that $\leq_{\mathcal{D}}$ is the maximum co-lex order follows from Theorem 2.5:

$$u \leq v \xRightarrow{\text{Thm. 2.5}} (\forall \alpha \in I_u)\,(\forall \beta \in I_v)\ (\alpha \prec \beta) \xRightarrow{\text{Def. 2.6}} u \leq_{\mathcal{D}} v.$$

□

2.3 The co-lex Width

In Chapter 4 we will see that a co-lex order over an automaton enables compression and indexing mechanisms whose efficiency is parametrized by the width of the co-lex order (the smaller, the better): this justifies introducing the *co-lex width* of a DFA as a meaningful measure for compression and indexing. The co-lex width also has lots of language-theoretic consequences, which will be described in the next chapter. Thus, first, we will introduce this concept and see an algorithm that computes it. Then, we will extend the width to languages, and give a general measure related to it: the entanglement.

2.3.1 Co-lex width on automata

Definition 2.8. The *co-lex width* of a DFA $\mathcal{D}$ is the minimum width of a co-lex order on $\mathcal{D}$:

$$\text{width}(\mathcal{D}) = \min\{\text{width}(\leq) \mid \leq \text{ is a co-lex order on } \mathcal{D}\}.$$

In Section 2.2, we defined the maximum co-lex order over a DFA $\leq_{\mathcal{D}}$. Since $\leq_{\mathcal{D}}$ refines any possible co-lex order on Q, it realizes the width of the automaton $\mathcal{D}$, as stated by the following lemma.

Lemma 2.9. *If $\mathcal{D}$ is a DFA then* $\text{width}(\leq_{\mathcal{D}}) = \text{width}(\mathcal{D})$.

This simplifies the search for a co-lex order realizing the width of the automaton and, as we will see, it can be determined in polynomial time.

To do that, we start characterizing $\leq_{\mathcal{D}}$ in terms of graph reachability.

Definition 2.10. We say that a pair $(u', v') \in Q \times Q$ precedes a pair $(u, v) \in Q \times Q$ if $u' \neq v', u \neq v$ and there exists $\alpha \in \Sigma^*$ such that $\delta(u', \alpha) = u, \delta(v', \alpha) = v$.

Lemma 2.11. *Let $\mathcal{D}$ be a DFA and let $u, v \in Q$, with $u \neq v$. Then:*

$$u <_{\mathcal{D}} v \ \Leftrightarrow \textit{for all pairs } (u', v') \textit{ preceding } (u, v) \textit{ it holds } \max_{\lambda(u')} \preceq \min_{\lambda(v')}.$$

Proof.

($\Rightarrow$) Suppose $u <_{\mathcal{D}} v$. Let (u', v') be a pair preceding (u, v) and let $\gamma \in \Sigma^*$ be such that $\delta(u', \gamma) = u$ and $\delta(v', \gamma) = v$. We must prove that $\max_{\lambda(u')} \preceq \min_{\lambda(v')}$. First, notice that it cannot be $v' = s$, otherwise, given $\alpha \in I_{u'}$, we would have $\alpha\gamma \in I_u$ and $\gamma \in I_v$, which contradicts $u <_{\mathcal{D}} v$.

So we are only left with proving that if $u' = \delta(u'', e)$ and $v' = \delta(v'', e')$, with $e, e' \in \Sigma$, then $e \preceq e'$. Let $\alpha'' \in I_{u''}$ and $\beta'' \in I_{v''}$. Then $\alpha''e\gamma \in I_u$, $\beta''e'\gamma \in I_v$ and from $u <_{\mathcal{D}} v$ it follows that $\alpha''e\gamma \prec \beta''e'\gamma$, which implies $e \preceq e'$.

($\Leftarrow$) Suppose that for all pairs (u', v') preceding (u, v) it holds $\max_{\lambda(u')} \preceq \min_{\lambda(v')}$. Let $\alpha \in I_u$ and $\beta \in I_v$. We must prove that $\alpha \prec \beta$. Since $u \neq v$, then $\alpha \neq \beta$. Write $\alpha = \alpha'\gamma$ and $\beta = \beta'\gamma$, where α' and β' end with a distinct letter (or, possibly, exactly one of them is equal to the empty string). Let $u', v' \in Q$ be such that $\alpha' \in I_{u'}$ and $\beta' \in I_{v'}$. Then, (u', v') precedes (u, v), so it must be $\max_{\lambda(u')} \preceq \min_{\lambda(v')}$. This implies that $v' \neq s$, so β is not a suffix of α. If α is a suffix of β, we are done. Otherwise, it must be $\alpha' = \alpha''a$ and $\beta' = \beta''b$, with $a, b \in \Sigma$, $a \neq b$; from $\max_{\lambda(u')} \preceq \min_{\lambda(v')}$ it then follows $a \prec b$, which implies $\alpha \prec \beta$.

□

Thanks to this result, we can provide an $O(|\delta|^2)$ algorithm for computing $\leq_{\mathcal{D}}$. We will build a graph $G = (V, E)$ whose nodes are state pairs, $V = \{(u, v) \in Q \times Q \mid u \neq v\}$, and we will mark all incomparable pairs. To do that, we need to encode the "preceding" definition: put edges between pairs that are connected by the same character transitions, $E = \{((u', v'), (u, v)) \in V \times V \mid (\exists e \in \Sigma)(\delta(u', e) = u \wedge \delta(v', e) = v)\}$. Note that $|E| \leq |\delta|^2$.

Then, the algorithm is composed of these two points:

1. mark all nodes (u, v) in G for which $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$ does not hold.
2. mark all nodes reachable on G from a marked node (found in 1).

Point 1 requires $O(|Q|^2)$ time: we pre-process every state and find the minimum and the maximum of $\lambda(u)$ by scanning the transitions of the automaton; then we can decide in constant time if Axiom 1 hold. Point 2 can be done in $O(|\delta|^2)$ time: we can use a DFS visit on G, initializing the stack with all marked nodes. Finally, the set of unmarked pairs forms $<_{\mathcal{D}}$. Hence, we proved:

Lemma 2.12. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. We can find the order $\leq_{\mathcal{D}}$ in $O(|\delta|^2)$ time.*

This bound has been improved by two other works: [Cot23] showed how to find $\leq_{\mathcal{D}}$ in $O(|Q|^2)$, while [BCC+23] found an algorithm that solves the same task in time $O(|\delta| \log |Q|)$.

It follows also that the width of a DFA is computable in polynomial time: by the following lemma, the smallest chain partition can be found in $\tilde{O}(|\delta|^2)$.

Lemma 2.13. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. A co-lex order $\leq$ of width equal to* $\mathrm{width}(\mathcal{D})$ *and a corresponding chain partition of cardinality equals to* $\mathrm{width}(\mathcal{D})$ *can be computed in $\tilde{O}(|\delta|^2)$ time with high probability*[1].

Proof. A partial order $(A, \leq)$ can be seen as a DAG with $|A|$ nodes and $O(|A|^2)$ edges. Since a co-lex order is a partial order on Q, the co-lex order can be seen as a DAG with Q nodes and $O\left(|Q|^2\right)$ edges. In [KP22], it is shown how to compute a minimum chain partition of a DAG with m edges and n nodes

[1] *with high probability* means with success probability at least $1 - N^{-c}$, where N is the size of the input and c is a user-defined constant.

in $\tilde{O}(m + n^{3/2})$ time with high probability. So, plugging in the co-lex order, we obtain a running time of $\tilde{O}(|Q|^2)$.

So, the whole process takes time $\tilde{O}(|Q|^2) + O(|\delta|^2) = \tilde{O}(|\delta|^2)$. □

If we consider the improvements cited above, the total running time reduces to $\tilde{O}(|Q|^2)$.

2.3.2 Co-lex width of Regular Languages

On the grounds of the definition of automata's width, we also study some implications on the theory of regular languages. We start by defining the width of a regular language based on the co-lex orders of the automata recognizing it.

Definition 2.14. Let $\mathcal{L}$ be a regular language. The *deterministic co-lex width* of $\mathcal{L}$, denoted by $\text{width}^D(\mathcal{L})$, is the smallest integer p for which there exists a DFA $\mathcal{D}$ such that $\mathcal{L}(\mathcal{D}) = \mathcal{L}$ and $\text{width}(\mathcal{D}) = p$.

Regular languages are closed under standard set operations. It can be interesting to see how the width behaves with respect to these operations.

Theorem 2.15 (Complementation)**.** *Let $\mathcal{L}$ be a regular language. Then*

$$\text{width}^D(\Sigma^* \setminus \mathcal{L}) \leq \text{width}^D(\mathcal{L}) + 1.$$

Proof. Let $\mathcal{D}$ be a DFA such that $\mathcal{L}(\mathcal{D}) = \mathcal{L}$ and $\text{width}(\mathcal{D}) = \text{width}^D(\mathcal{L})$. We must prove that there exists a DFA $\mathcal{D}'$ such that $\mathcal{L}(\mathcal{D}') = \Sigma^* \setminus \mathcal{L}$ and $\text{width}(\mathcal{D}') \leq \text{width}(\mathcal{D}) + 1$. The DFA $\mathcal{D}'$ is obtained by first transforming $\mathcal{D}$ into a complete DFA by adding a non-final sink state that is reached by all transitions not defined in $\mathcal{D}$ (including the ones leaving the sink itself), then switching final and non-final states, and finally removing all states that neither are final, nor allow to reach a final state. In the worst case, the new sink state is not comparable with any other state in $\mathcal{D}'$, and therefore the width cannot increase by more than one. □

Theorem 2.16 (Intersection)**.** *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two regular languages. Then*

$$\text{width}^D(\mathcal{L}_1 \cap \mathcal{L}_2) \leq \text{width}^D(\mathcal{L}_1) \cdot \text{width}^D(\mathcal{L}_2)$$

Proof. Let $\mathcal{D}_1 = (Q_1, s_1, \delta_1, F_1), \mathcal{D}_2 = (Q_2, s_2, \delta_2, F_2)$ be DFAs such that $\mathcal{L}(\mathcal{D}_1) = \mathcal{L}_1$, $\mathcal{L}(\mathcal{D}_2) = \mathcal{L}_2$, $\text{width}(\mathcal{D}_1) = \text{width}^D(\mathcal{L}_1) = p_1$ and $\text{width}(\mathcal{D}_2) = \text{width}^D(\mathcal{L}_2) = p_2$. We have to prove that there exists a DFA $\mathcal{D}$ such that $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{D}_1) \cap \mathcal{L}(\mathcal{D}_2)$ and $\text{width}(\mathcal{D}) \leq p_1 \cdot p_2$. In the following, we will implicitly use Lemma 2.9. Let $\{Q_1, \ldots, Q_{p_1}\}$ and $\{Q_1^*, \ldots, Q_{p_2}^*\}$ be a chain partition of Q_1 and a chain partition of Q_2, respectively. To build both $\mathcal{D}$, we first turn $\mathcal{D}_1$ and $\mathcal{D}_2$ into complete DFAs by adding non-final sinks $\star_1, \star_2$ to Q_1 and Q_2, respectively (like in Theorem 2.15), then we build the standard product automaton: the set of states is $(Q_1 \cup \{\star_1\}) \times (Q_2 \cup \{\star_2\})$, the initial state is (s_1, s_2), and the transition function is defined by $\delta((u, v), a) = (\delta_1(u, a), \delta_2(v, a))$, for every state (u, v) and for every $a \in \Sigma$.

Finally, let $F = F_1 \times F_2$ be the set of all final states, and then remove

- all states that are not reachable from the initial state and
- all states that neither are final, nor allow to reach a final state.

Let $\mathcal{D} = (Q, s, \delta, F)$ the resulting DFA at the end of the construction. Notice that $Q \subseteq Q_1 \times Q_2$ (that is, the sinks play no role) because in $\mathcal{D}_1$ and $\mathcal{D}_2$ the sinks are not final and do not allow to reach a final state. Moreover we have $I_{(u,v)} = I_u \cap I_v$ for every $(u, v) \in Q$, because for every $\alpha \in \Sigma^*$ there exists a path labeled α from (s_1, s_2) to (u, v) on $\mathcal{D}$ if and only there exist a path labeled α from s_1 to u on $\mathcal{D}_1$ and a path labeled α from s_2 to v on $\mathcal{D}_2$. As a consequence, $\mathcal{L}(\mathcal{D}) = \bigcup_{(u,v)\in F} I_{(u,v)} = \bigcup_{u\in F_1, v\in F_2} I_u \cap I_v = \mathcal{L}(\mathcal{D}_1) \cap \mathcal{L}(\mathcal{D}_2)$. Now, let us prove that $\text{width}(\mathcal{D}) \leq p_1 \cdot p_2$. For every $i = 1, \ldots, p_1$ and for every $j = 1, \ldots, p_2$, define:

$$Q_{i,j} = \{(u, v) \in Q \mid u \in Q_i, v \in Q_j^*\}.$$

Since $\{Q_{i,j} \mid 1 \leq i \leq p_1, 1 \leq j \leq p_2\}$ is a partition of Q, we only have to show that every $Q_{i,j}$ is a chain. Let $(u_1, v_1), (u_2, v_2)$ be distinct elements in $Q_{i,j}$. Hence, at least one between $u_1 \neq u_2$ and $v_1 \neq v_2$ holds true. Assume that $u_1 \neq u_2$ (the other case is analogous). Since $u_1, u_2 \in Q_i$, then $u_1 <_{\mathcal{D}_1} u_2$ or $u_2 <_{\mathcal{D}_1} u_1$. Assuming without loss of generality that $u_1 <_{\mathcal{D}_1} u_2$, then from $I_{(u_1,v_1)} \subseteq I_{u_1}$ and $I_{(u_2,v_2)} \subseteq I_{u_2}$ we conclude $(u_1, v_1) <_{\mathcal{D}} (u_2, v_2)$. □

Theorem 2.17 (Union)**.** *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two regular languages. Then*

$$\text{width}^D(\mathcal{L}_1 \cup \mathcal{L}_2) \leq \text{width}^D(\mathcal{L}_1) \cdot \text{width}^D(\mathcal{L}_2) + \text{width}^D(\mathcal{L}_1) + \text{width}^D(\mathcal{L}_2).$$

Proof. Let $\mathcal{D}_1 = (Q_1, s_1, \delta_1, F_1), \mathcal{D}_2 = (Q_2, s_2, \delta_2, F_2)$ be DFAs such that $\mathcal{L}(\mathcal{D}_1) = \mathcal{L}_1$, $\mathcal{L}(\mathcal{D}_2) = \mathcal{L}_2$, $\text{width}(\mathcal{D}_1) = \text{width}^D(\mathcal{L}_1) = p_1$ and $\text{width}(\mathcal{D}_2) = \text{width}^D(\mathcal{L}_2) = p_2$. The construction of the automaton $\mathcal{D}$ that recognizes the union of the two languages is the same as the one in Theorem 2.16. What changes is the set of final states: in this case, $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. Again, we have $I_{(u,v)} = I_u \cap I_v$ for every $(u, v) \in Q'$ such that $u \in Q_1$ and $v \in Q_2$, so $\mathcal{L}(\mathcal{D}) = \bigcup_{(u,v)\in F} I_{(u,v)} = \bigcup_{u\in F_1 \vee v\in F_2} I_u \cap I_v = \mathcal{L}(\mathcal{D}_1) \cup \mathcal{L}(\mathcal{D}_2)$. Let us prove that $\text{width}(\mathcal{D}) \leq p_1 \cdot p_2 + p_1 + p_2$. Notice that this time if $(u, v) \in Q$, then it may happen that $u = \star_1$ or $v = \star_2$ (but not both). Moreover, if $(u, \star_2) \in Q$, then $I_{(u,\star_2)} = I_u \setminus \text{Pref}(\mathcal{L}_2)$, and if $(\star_1, v) \in Q$, then $I_{(\star_1,v)} = I_v \setminus \text{Pref}(\mathcal{L}_1)$. For every $i = 1, \ldots, p_1$ and for every $j = 1, \ldots, p_2$, define:

$$\begin{aligned} Q_{i,j} &= \{(u, v) \in Q \mid u \in Q_i, v \in Q_j^*\} \\ Q_{i,\star_2} &= \{(u, \star_2) \in Q \mid u \in Q_i\} \\ Q_{\star_1,j} &= \{(\star_1, v) \in Q \mid v \in Q_j^*\}. \end{aligned}$$

These sets identify a partition of Q, and like before one can show that each set is a chain. We conclude that $\text{width}(\mathcal{D}') \leq p_1 \cdot p_2 + p_1 + p_2$. □

As we have seen in the previous subsection, it is possible to determine the width of a DFA in polynomial time. A natural question that emerges from that result is whether it is possible to determine the width of a regular language.

One way to characterize regular languages is through the minimum DFA that recognizes it. Due to

its uniqueness and since it can be built in polynomial time, it is a good candidate for this purpose. Unfortunately, as we will see in the following example, the minimum-size accepting automaton generally does not realize the minimum width of a language. Moreover, there is no unique (up to isomorphism) minimum DFA of minimum width recognizing a given regular language.

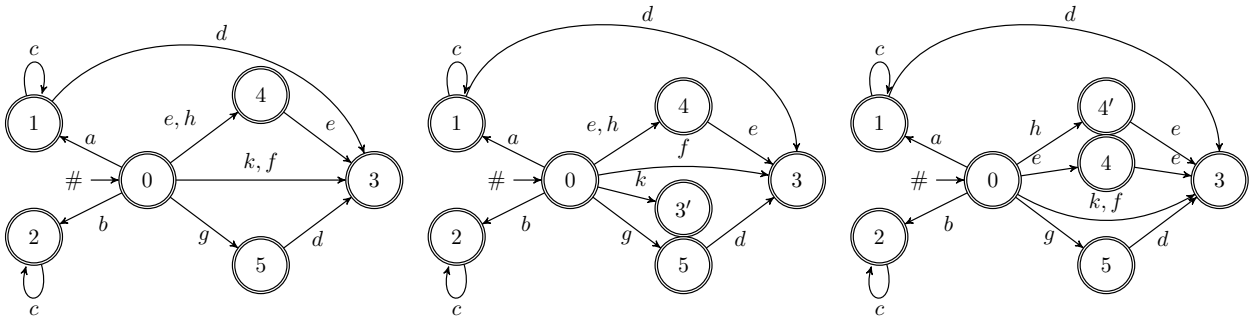


Figure 2.2: Three DFAs recognizing the same language.

Example 2.18 ([CDPP23])**.** In Figure 2.2, there are three DFAs recognizing the same language $\mathcal{L}$. We prove that $\text{width}^D(\mathcal{L}) = 2$, the width of the minimum DFA for $\mathcal{L}$ is 3, and there is not a unique minimum automaton among all DFAs recognizing $\mathcal{L}$ of width 2. Consider the DFA $\mathcal{D}_1$ on the left of Figure 2.2 and let $\mathcal{L} = \mathcal{L}(\mathcal{D}_1)$. The automaton $\mathcal{D}_1$ is a minimum DFA for $\mathcal{L}$. The states $0, \ldots, 5$ are such that: $I_0 = \{\#\}$, $I_1 = ac^*$, $I_2 = bc^*$, $I_3 = ac^*d \cup \{gd, ee, he, f, k\}$, $I_4 = \{e, h\}$, $I_5 = \{g\}$. States 1 and 2 are $\leq_{\mathcal{D}_1}$-incomparable because $a \in I_1, b \in I_2, ac \in I_1$ and $a \prec b \prec ac$. Similarly one checks that states $3, 4, 5$ are pairwise $\leq_{\mathcal{D}_1}$-incomparable. On the other hand, 0 is the minimum and states $1, 2$ precede states $3, 4, 5$ in the order $\leq_{\mathcal{D}_1}$. We conclude that the Hasse diagram of the partial order $\leq_{\mathcal{D}_1}$ is the one depicted in Figure 2.3.

The width of the DFA $\mathcal{D}_1$ is 3 because $\{3, 4, 5\}$ is a largest $\leq_{\mathcal{D}_1}$-antichain. A $\leq_{\mathcal{D}_1}$-chain partition of cardinality 3 is, for example, $\{\{0, 1, 3\}, \{2, 4\}, \{5\}\}$.

Let us prove that $\text{width}^D(\mathcal{L}) \geq 2$. Suppose by contradiction that there exists a DFA $\mathcal{D}$ of width 1 recognizing $\mathcal{L}$. Then, the order $\leq_{\mathcal{D}}$ is total. Moreover, there exists a state u such that two words of the infinite set $ac^* \in \text{Pref}(\mathcal{L})$, say ac^i, ac^j with $i < j$, belong to I_u. Consider the word $bc^i \in \text{Pref}(\mathcal{L})$. Since $bc^i \not\equiv_{\mathcal{L}} ac^i$, it follows that $bc^i \notin I_u$. If u' is such that $bc^i \in I_{q'}$, from $ac^i \prec bc^i \prec ac^j$ we have that u and u' are $\leq_{\mathcal{D}}$-incomparable, a contradiction.

Finally, let $\mathcal{D}_2$ be the DFA in the center of Figure 2.2 and let $\mathcal{D}_3$ be the DFA on the right of Figure

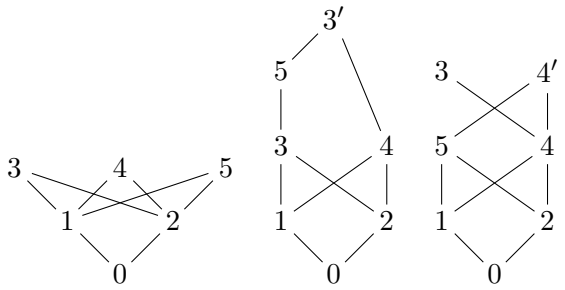


Figure 2.3: The Hasse diagram of, respectively, $\leq_{\mathcal{D}_1}$, $\leq_{\mathcal{D}_2}$, $\leq_{\mathcal{D}_3}$.

2.2. Notice that $\mathcal{L}(\mathcal{D}_2) = \mathcal{L}(\mathcal{D}_3) = \mathcal{L}$, and $\mathcal{D}_2$ and $\mathcal{D}_3$ have just one more state than $\mathcal{D}_1$ and are non-isomorphic. We know that $\mathcal{D}_2$ and $\mathcal{D}_3$ cannot have width equal to 1. On the other hand, they both have width 2, as witnessed by the chains $\{\{0,1,4\},\{2,3,5,3'\}\}$ (for $\mathcal{D}_2$) and $\{\{0,1,3\},\{2,4,5,4'\}\}$ (for $\mathcal{D}_3$). For completeness, in Figure 2.3 we provide the partial order of $\mathcal{D}_2$ and $\mathcal{D}_3$.

So, we must look for another property that captures the width of the language accepted by an automaton. In order to solve our problem, let's analyze a simpler case, the Wheelerness test for a regular language (i.e. $\text{width}^D(L) = 1$).

In Section 3 of [ADPP21], the authors found the following necessary condition to decide whether or not a language is Wheeler. $\approx_{\mathcal{L}}$ represents the Myhill-Nerode equivalence between strings.

Lemma 2.19 ([ADPP21])**.** *A regular language $\mathcal{L}$ is Wheeler if all monotone sequences in* $(\text{Pref}(\mathcal{L}), \prec)$ *become eventually constant modulo $\approx_{\mathcal{L}}$. In other words, for all sequences $(\alpha_i)_{i\geq 1}$ in* $\text{Pref}(\mathcal{L})$ *with*

$$\alpha_1 \preceq \alpha_2 \preceq \cdots \preceq \alpha_i \preceq \ldots \text{ or } \alpha_1 \succeq \alpha_2 \succeq \cdots \succeq \alpha_i \succeq \ldots$$

there exists an n such that $\alpha_h \approx_{\mathcal{L}} \alpha_k$, for all $h, k \geq n$.

Proof. Let $\mathcal{D}$ a Wheeler DFA that recognizes $\mathcal{L}$. In this case, $\leq_{\mathcal{D}}$ is a total order. Let $\alpha \in \text{Pref}(\mathcal{L}(\mathcal{D}))$. Then for all $u \in Q$, it holds

$$\alpha \in I_u \iff u = \delta(s, \alpha),$$

and

$$\alpha \prec \beta \Rightarrow \delta(s,\alpha) \leq_{\mathcal{D}} \delta(s,\beta), \text{ and } \delta(s,\alpha) <_{\mathcal{D}} \delta(s,\beta) \Rightarrow \alpha \prec \beta$$

by the definition of $\leq_{\mathcal{D}}$. If $(\alpha_i)_{i\geq 1}$ is a monotone sequence in $\text{Pref}(\mathcal{L}(\mathcal{D}))$, then the first implication above implies that $(\delta(s,\alpha_i))_{i\in\mathbb{N}}$ is a monotone sequence in $(Q, <_{\mathcal{D}})$. Since Q is a finite set, the theorem follows. □

This Theorem implies that, if there exists (at least) two states that are infinitely visited by a monotone sequence of strings in $\text{Pref}(\mathcal{L})$, then the language is not Wheeler. On automaton $\mathcal{D}_1$ in Example 2.18, we can observe this phenomenon: the sequence

a	$\prec$	b	$\prec$	ac	$\prec$	bc	$\prec$	ac^2	$\prec$	bc^2	$\prec$	$\ldots$	$\prec$	ac^i	$\prec$	bc^i	$\prec$	$\ldots$
1		2		1		2		1		2		$\ldots$		1		2		$\ldots$

alternates between states 1 and 2, so the language cannot be Wheeler (as shown in the example). Moreover, this phenomenon makes 1 and 2 incomparable: indeed, $ac, acc \in I_1$, $bc \in I_2$ and $ac \prec bc \prec acc$. In general, if the states that are visited by the monotone sequence are n, they are always incomparable: for each pair of nodes (u, v) there will be three strings, $\alpha, \alpha' \in I_u$ and $\beta \in I_v$ such that $\alpha \prec \beta \prec \alpha'$. Thus, the width of the automaton will be at least n (the n states are an antichain in the order).

This motivates the fact that if we want to characterize the width of a language, we must look for these alternating sequences of states over a monotone set of strings. This characteristic will be defined *entanglement*.

Definition 2.20. Let $\mathcal{D}$ be a DFA with a set of states Q.

1. A subset $Q' \subseteq Q$ is *entangled* if there exists a monotone sequence $(\alpha_i)_{i\in\mathbb{N}}$ in $\mathrm{Pref}(\mathcal{L}(\mathcal{D}))$ such that for all $u' \in Q'$ it holds $\delta(s, \alpha_i) = u'$ for infinitely many i's. In this case the sequence $(\alpha_i)_{i\in\mathbb{N}}$ is said to be a *witness* for (the entanglement of) Q'.
2. A set $V \subseteq \mathrm{Pref}(\mathcal{L}(\mathcal{D}))$ is *entangled in* $\mathcal{D}$ if there exists a monotone sequence $(\alpha_i)_{i\in\mathbb{N}}$, with $\alpha_i \in V$ for every i, witnessing that the set $\{\delta(s, \alpha) \mid \alpha \in V\}$, consisting of all states occurring in V, is entangled.

Moreover, define :

$$\mathrm{ent}(\mathcal{D}) = \max\{|Q'| \mid Q' \subseteq Q \text{ and } Q' \text{ is entangled }\}$$
$$\mathrm{ent}(\mathcal{L}) = \min\{\mathrm{ent}(\mathcal{D}) \mid \mathcal{D} \text{ is a DFA } \wedge\ \mathcal{L}(\mathcal{D}) = \mathcal{L}\}.$$

In Example 2.18, in all the three automata, the set of states $\{1, 2\}$ is an entangled set. Moreover, the set of strings $(a + b)c^*$ is entangled in $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$.

Notice that any singleton $\{u\} \subseteq Q$ is entangled, as witnessed by the trivially monotone sequence $(\alpha_i)_{i\in\mathbb{N}}$ where all α_i's are equal and $\delta(s, \alpha) = u$.

When two states $u \neq u'$ of a DFA $\mathcal{D}$ belong to an entangled set, there are words $\alpha \prec \beta \prec \alpha'$ such that $\alpha, \alpha' \in I_u$, $\beta \in I_{u'}$ so that neither $u <_{\mathcal{D}} u'$ nor $u' <_{\mathcal{D}} u$ can hold (indeed, in Example 2.18 state 1 and 2 are incomparable). However, being entangled is not a necessary condition for incomparability: in Example 2.18, state $3, 4, 5$ in $\mathcal{D}_1$ are incomparable although they are not entangled.

Since by Lemma 2.9 we have $\mathrm{width}(\leq_{\mathcal{D}}) = \mathrm{width}(\mathcal{D})$, it follows that the entanglement of a DFA is always smaller than or equal to its width.

Lemma 2.21. *Let $\mathcal{D}$ be a DFA. Then* $\mathrm{ent}(\mathcal{D}) \leq \mathrm{width}(\mathcal{D})$.

As you can see from Example 2.18, the converse is not always true: the minimum automata $\mathcal{D}_1$ has $\mathrm{width}(\mathcal{D}_1) = 3$, while $\mathrm{ent}(\mathcal{D}) = 2$.

Conversely to the width, the entanglement of a regular language is equal to the entanglement of its minimum-size automaton.

Lemma 2.22. *If $\mathcal{D}_{\mathcal{L}}$ is the minimum-size DFA recognizing $\mathcal{L}$, then* $\mathrm{ent}(\mathcal{L}) = \mathrm{ent}(\mathcal{D}_{\mathcal{L}})$.

Proof. It is enough to prove that $\mathrm{ent}(\mathcal{D}_{\mathcal{L}}) \leq \mathrm{ent}(\mathcal{D})$, for any DFA $\mathcal{D}$ such that $\mathcal{L}(\mathcal{D}_{\mathcal{L}}) = \mathcal{L}(\mathcal{D})$. Suppose $u_1, \ldots, u_k$ are pairwise distinct states which are entangled in $\mathcal{D}_{\mathcal{L}}$, witnessed by the monotone sequence $(\alpha_i)_{i\in\mathbb{N}}$. Since $\mathcal{D}_{\mathcal{L}}$ is minimum, each I_{u_j} is a union of a finite number of I_v, with $v \in Q_{\mathcal{D}}$. The monotone sequence $(\alpha_i)_{i\in\mathbb{N}}$ goes through u_j infinitely often, so there must be a state $v_j \in Q_{\mathcal{D}}$ such that $I_{v_j} \subseteq I_{u_j}$ and $(\alpha_i)_{i\in\mathbb{N}}$ goes through v_j infinitely often. Then $(\alpha_i)_{i\in\mathbb{N}}$ goes through the pairwise distinct states $v_1, \ldots, v_k$ infinitely often and $v_1, \ldots, v_k$ are entangled in $\mathcal{D}$.

□

For now, we know that $\text{ent}(\mathcal{D}_{\mathcal{L}}) = \text{ent}(\mathcal{L}) \leq \text{width}^D(\mathcal{L})$. It would be nice if the last inequality becomes equality: in this case, to calculate $\text{width}^D(\mathcal{L})$, it would be sufficient to build the minimum-size automata that recognizes $\mathcal{L}$ and calculate $\text{ent}(\mathcal{D}_{\mathcal{L}})$ (we will not discuss how to do that, but, as shown in [CDPP23], it is fixed-parameter tractable).

It turns out that the above request can be satisfied: in the next chapter, we will focus on building an automata $\mathcal{H}_{\mathcal{L}}$, called Hasse Automaton, that recognizes $\mathcal{L}$ and have $\text{width}(\mathcal{H}_{\mathcal{L}}) = \text{ent}(\mathcal{H}_{\mathcal{L}})$.

3

The Hasse Automaton

In this chapter, we will focus on constructing an automaton whose width is equal to the entanglement of the minimum-size automaton. This result will allow us to prove that:

$$\text{width}(\mathcal{L}) = \text{ent}(D_{\mathcal{L}})$$

In other words, it will allow us to prove that the entanglement measure captures the width of regular languages. From this, it follows that:

- calculating the width of a language is equivalent to calculating the entanglement of the minimal automaton (that it is fixed-parameter tractable, as shown in [CDPP23]);
- we can give an automaton-free description of languages with width $= p$.

The automaton we will obtain is called the *Hasse automaton*. First, we will provide a high-level intuition of the construction, and then we will describe all the technical details of the case. Finally, through some examples, we will demonstrate that while the Hasse automaton achieves minimum width, it may not be optimal in terms of the number of states. This observation leads us to explore basic characteristics of DFAs that simultaneously minimize both width and number of states.

3.1 High-level intuition

As a first step, given a DFA $\mathcal{D}$ we prove that there exists an equivalent DFA $\mathcal{D}'$ that realizes the entanglement of $\mathcal{D}$ as its width: $\text{ent}(\mathcal{D}) = \text{width}(\mathcal{D}')$ (Theorem 3.10). Then, it will be sufficient to apply the whole construction to the minimum-size DFA, and we will obtain the desired automaton.

Consider a hypothetical $(\text{Pref}(\mathcal{L}), \preceq)$ (represented below), where each string is labeled with the state u_i it reaches. The DFA we are dealing with is composed by states $\{u_1, u_2, u_3\}$.

$$\begin{array}{ccccccccccccccccccccccccccc}
\alpha_1 & \prec & \beta_1 & \prec & \alpha_2 & \prec & \beta_2 & \prec & \cdots & \prec & \alpha_i & \prec & \beta_i & \prec & \cdots & \prec & \beta'_1 & \prec & \gamma_1 & \prec & \beta'_2 & \prec & \gamma_2 & \prec & \cdots & \prec & \beta'_i & \prec & \gamma_i & \prec & \cdots & \prec & \alpha' \\
u_1 & & u_2 & & u_1 & & u_2 & & \ldots & & u_1 & & u_2 & & \ldots & & u_2 & & u_3 & & u_2 & & u_3 & & \ldots & & u_2 & & u_3 & & \ldots & & u_1
\end{array}$$

As you can see, the sets $\{u_1, u_2\}$ and $\{u_2, u_3\}$ are entangled, since are witnessed by the sequences

$\alpha_1 \prec \beta_1 \prec \cdots \prec \alpha_i \prec \beta_i \prec \dots$ and $\beta'_1 \prec \gamma_1 \prec \cdots \prec \beta'_i \prec \gamma_i \prec \dots$. Conversely, the set $\{u_1, u_3\}$ is not entangled and therefore the set $\{u_1, u_2, u_3\}$ is not entangled. However, $\{u_1, u_2, u_3\}$ contains pairwise incomparable states. Hence the whole triplet $\{u_1, u_2, u_3\}$ does not contribute to the entanglement but does contribute to the width so that $\mathrm{ent}(\mathcal{D}) = 2 < \mathrm{width}(\mathcal{D}) = 3$.
In general, an automaton where incomparability and entanglement coincide would have $\mathrm{ent}(\mathcal{D}) = \mathrm{width}(\mathcal{D})$. Hence, we would like to force *all* sets of incomparable states in the new automaton $\mathcal{D}'$ to be entangled. To this end, we will first prove that there always exists a finite, ordered partition $\mathcal{V} = \{V_1, \dots, V_r\}$ of $\mathrm{Pref}(\mathcal{L})$ composed of convex sets which are entangled in $\mathcal{D}$. In the example above we can write $\mathrm{Pref}(\mathcal{L}) = V_1 \cup V_2 \cup V_3$, where:

$$V_1 = \{\alpha_1, \beta_1, \dots, \alpha_i, \beta_i, \dots\}, V_2 = \{\beta'_1, \gamma_1, \dots, \beta'_i, \gamma_i \dots\}, V_3 = \{\alpha'\}$$

and the states occurring (and entangled) in V_1, V_2, V_3 are, respectively: $\{u_1, u_2\}$, $\{u_2, u_3\}$, and $\{u_1\}$. To construct an equivalent automaton $\mathcal{D}'$ in which the pairwise incomparability of the three states u_1, u_2, u_3 is eliminated and $\mathrm{width}(\mathcal{D}') = 2$, we could try to *duplicate* some of the original states, as it would be the case if the states occurring in V_1, V_2, V_3 where, respectively, $\{u_1, u_2\}$, $\{u_2, u_3\}$, and $\{u'_1\}$. To this end, we will consider a refinement $\sim$ of the Myhill-Nerode equivalence on $\mathrm{Pref}(\mathcal{L})$ stating that two strings are equivalent if and only if they are in the same I_u and all $V \in \mathcal{V}$ laying between the two strings intersect I_u. In the above example we have $\beta_i \sim \beta'_j$ for all integers i, j, because no $V \in \mathcal{V}$ is contained in $[\beta_i, \beta'_j]$, while $\alpha_1 \not\sim \alpha'$, since $V_2 \subseteq [\alpha_1, \alpha']$ but $V_2 \cap I_{u_1} = \emptyset$.

We will prove that the equivalence $\sim$ decomposes the set of words reaching u into a *finite* number of $\sim$-classes and induces a well-defined quotient automaton $\mathcal{D}'$ equivalent to $\mathcal{D}$.

By construction, in the automaton $\mathcal{D}'$ any set of $<_{\mathcal{D}'}$-incomparable states $\{u_1, \dots, u_k\}$ will occur in at least an element $V \in \mathcal{V}$, so that, V being entangled, they will contribute to the entanglement number of $\mathcal{D}$ and $\mathrm{width}(\mathcal{D}') = \mathrm{ent}(\mathcal{D})$ will follow.

In our example, the new automaton $\mathcal{D}'$ will leave the following trace:

$$\begin{array}{ccccccccccccccccccccccccccccc}
\alpha_1 & \prec & \beta_1 & \prec & \alpha_2 & \prec & \beta_2 & \prec & \cdots & \prec & \alpha_i & \prec & \beta_i & \prec & \cdots & \prec & \beta'_1 & \prec & \gamma_1 & \prec & \beta'_2 & \prec & \gamma_2 & \prec & \cdots & \prec & \beta'_i & \prec & \gamma_i & \prec & \cdots & \prec & \alpha' \\
u_1 & & u_2 & & u_1 & & u_2 & & \dots & & u_1 & & u_2 & & \dots & & u_2 & & u_3 & & u_2 & & u_3 & & \dots & & u_2 & & u_3 & & \dots & & u'_1
\end{array}$$

and $\mathrm{ent}(\mathcal{D}) = \mathrm{width}(\mathcal{D}') = 2$.

3.2 Entangled convex decompositions

In order to give a formal definition of $\mathcal{D}'$ we need some properties of entangled sets. These properties are collected in a more general setting in Appendix A of [CDPP23], while here are stated without proof. In particular, most of the properties of $\mathcal{D}'$ will be proved using a finite partition of $(\mathrm{Pref}(\mathcal{L}(\mathcal{D})), \preceq)$, composed by entangled, convex sets.

Definition 3.1. If $\mathcal{D}$ is a DFA, we say that a partition $\mathcal{V}$ of $\mathrm{Pref}(\mathcal{L}(\mathcal{D}))$ is an *entangled, convex decomposition of* $\mathcal{D}$ (*e.c. decomposition*, for short) if all the elements of $\mathcal{V}$ are convex in $(\mathrm{Pref}(\mathcal{L}(\mathcal{D})), \preceq)$

and entangled in $\mathcal{D}$.

Theorem 3.2. *If $\mathcal{D}$ is a DFA, then there exists a finite partition $\mathcal{V}$ of Pref$(\mathcal{L}(\mathcal{D}))$ which is an e.c. decomposition of $\mathcal{D}$.*

Proof. The existence of such a decomposition is guaranteed by Theorem A.5 of Appendix A in [CDPP23], applied to $(Z, \leq) = (\mathrm{Pref}(\mathcal{L}(\mathcal{D})), \preceq)$ and $\mathcal{P} = \{I_u \mid u \in Q\}$. □

Using an e.c. decomposition of an automaton $\mathcal{D}$ we can express a condition implying the equality $\mathrm{width}(\mathcal{D}) = \mathrm{ent}(\mathcal{D})$.

Lemma 3.3. *Let $\mathcal{D}$ be a DFA. Suppose $V_1 \prec V_2 \prec \cdots \prec V_m$ is an e.c. decomposition of $\mathcal{D}$ such that for every $u \in Q$ there are $1 \leq i \leq j \leq m$ with:*

$$I_u \subseteq V_i \cup V_{i+1} \cup \cdots \cup V_j, \quad \textit{and} \quad V_h \cap I_u \neq \emptyset, \quad \textit{for all} \quad i \leq h \leq j.$$

Then, $\mathrm{width}(\mathcal{D}) = \mathrm{ent}(\mathcal{D})$.

Proof. We already proved that $\mathrm{ent}(\mathcal{D}) \leq \mathrm{width}(\mathcal{D})$ in Lemma 2.21. In order to prove the reverse inequality, let $\mathrm{width}(\mathcal{D}) = p$ and let $u_1, \ldots, u_p$ be p $\leq_{\mathcal{D}}$-incomparable states. If we prove that $u_1, \ldots, u_p$ are entangled, we are done. Fix $i \in \{1, \ldots, p\}$ and denote by W_i the convex set $V_h \cup V_{h+1} \cup \cdots \cup V_k$ where $I_{u_i} \subseteq V_h \cup V_{h+1} \cup \cdots \cup V_k$ and $V_j \cap I_{u_i} \neq \emptyset$, for all $h \leq j \leq k$. From the incomparability of the u_i's it follows that $W_i \cap W_j \neq \emptyset$, for all pairs i, j, so that $\bigcap_i W_i \neq \emptyset$ by Lemma A.13 in Appendix A of [CDPP23]. Since all W_i's are unions of consecutive elements in the partition $\mathcal{V}$, there must be an element $V \in \mathcal{V}$ with $V \subseteq \bigcap_i W_i$. Such a V must contain an occurrence of every u_i, and since V is an entangled set, we conclude that $u_1, \ldots, u_p$ are entangled. □

This confirms what we said in Section 3.1: if we are able to duplicate states such that the new automaton $\mathcal{D}'$ satisfies the condition of the previous Lemma, we will have that $\mathrm{width}(\mathcal{D}') = \mathrm{ent}(\mathcal{D}') = \mathrm{ent}(\mathcal{D})$.

3.3 The Hasse equivalence

Similarly to Myhill-Nerode theorem, an equivalence relation $\sim_{\mathcal{D}}$ over $\mathrm{Pref}(\mathcal{L})$ is used to introduce the new states of the automaton $\mathcal{D}'$. In order to maintain the definition of $\mathcal{D}'$ independent from any particular e.c. decomposition $\mathcal{V}$, in the definition below we will use generic entangled convex sets instead of elements of an e.c. decomposition. In Lemma 3.8 is proved that this is equivalent to using elements of an e.c. decomposition of minimum cardinality.

Definition 3.4. Let $\mathcal{D}$ be a DFA and let $\sim_{\mathcal{D}}$ be the equivalence relation on $\mathrm{Pref}(\mathcal{L}(\mathcal{D}))$ defined as follows: $\alpha \sim_{\mathcal{D}} \alpha'$ if and only if:

- $\delta(s, \alpha) = \delta(s, \alpha')$ and
- there are entangled convex sets $C_1, \ldots, C_n \subseteq \mathrm{Pref}(\mathcal{L}(\mathcal{D}))$ such that:
  - $[\alpha, \alpha']^{\pm} \subseteq \bigcup_{i=1}^{n} C_i$;

- $C_i \cap I_{\delta(s,\alpha)} \neq \emptyset$, for all $i \in \{1, \ldots, n\}$.

Note that $\sim_{\mathcal{D}}$ is indeed an equivalence relation (in particular, it is transitive). When the DFA $\mathcal{D}$ is clear from the context, we shall drop the subscript $\mathcal{D}$ in $\sim_{\mathcal{D}}$.

The following Lemma states that the equivalence $\sim$ has finite index and $\mathcal{L}(\mathcal{D})$ is equal to the union of some of its classes.

Lemma 3.5. *Let $\mathcal{D}$ be a DFA. Then, $\sim$ has a finite number of classes on Pref$(\mathcal{L}(\mathcal{D}))$ and $\mathcal{L}(\mathcal{D})$ is equal to the union of some $\sim$-classes.*

Proof. Consider an e.c. decomposition $\mathcal{V} = \{V_1, \ldots, V_m\}$ of $\mathcal{D}$, whose existence is guaranteed by Theorem 3.2. Since all V_i's are entangled convex sets in Pref$(\mathcal{L}(\mathcal{D}))$, two words belonging to the same V_i and ending in the same state belong to the same $\sim$-class. Hence, the number of $\sim$-classes is at most $m \times |Q|$. Moreover, $\alpha \sim \beta$ implies $\delta(s, \alpha) = \delta(s, \beta)$. Hence, $\alpha \in \mathcal{L}(\mathcal{D}))$ and $\alpha \sim \beta$ imply $\beta \in \mathcal{L}(\mathcal{D})$, proving that $\mathcal{L}(\mathcal{D})$ is equal to the union of some $\sim$-classes. □

Lemma 3.6. *Let $\mathcal{D}$ be a DFA. Then, the equivalence relation $\sim$ is right-invariant.*

Proof. Let $\mathcal{L} = \mathcal{L}(\mathcal{D})$, assume $\alpha \sim \alpha'$ and let $a \in \Sigma$ be such that $\alpha a \in \text{Pref}(\mathcal{L})$. We must prove that $\alpha' a \in \text{Pref}(\mathcal{L})$ and $\alpha a \sim \alpha' a$. Since $\alpha \sim \alpha'$ we know that $\delta(s, \alpha) = \delta(s, \alpha')$ and there exist entangled convex sets $C_1, \ldots, C_n$ such that $[\alpha, \alpha']^{\pm} \subseteq C_1 \cup \cdots \cup C_n$ and $C_i \cap I_{\delta(s,\alpha)} \neq \emptyset$ for all $i = 1, \ldots, n$. We must prove that $\alpha' a \in \text{Pref}(\mathcal{L})$, $\delta(s, \alpha a) = \delta(s, \alpha' a)$, and there exist entangled convex sets $C'_1, \ldots, C'_{n'}$ such that $[\alpha a, \alpha' a]^{\pm} \subseteq C'_1 \cup \cdots \cup C'_{n'}$ and $C'_i \cap I_{\delta(s,\alpha a)} \neq \emptyset$ for all $i = 1, \ldots, n'$.

From $\delta(s, \alpha) = \delta(s, \alpha')$ and $\alpha a \in \text{Pref}(\mathcal{L})$ we immediately obtain $\alpha' a \in \text{Pref}(\mathcal{L})$ and $\delta(s, \alpha a) = \delta(s, \alpha' a)$. Moreover, from $[\alpha, \alpha']^{\pm} \subseteq C_1 \cup \cdots \cup C_n$ we obtain $[\alpha a, \alpha' a]^{\pm} = [\alpha, \alpha']^{\pm} a \subseteq C_1 a \cup \cdots \cup C_n a$, and from $C_i \cap I_{\delta(s,\alpha)} \neq \emptyset$ we obtain $C_i a \cap I_{\delta(s,\alpha a)} \neq \emptyset$. We are only left with showing that every $C_i a = \{\gamma a \mid \gamma \in C_i\}$ is an entangled convex set. The fact that they are convex follows directly from the definition of colex ordering. Let us prove that the $C_i a$'s are entangled. Fix i and consider a monotone sequence $(\alpha_j)_{j \in \mathbb{N}}$ witnessing that C_i is entangled. Then $(\alpha_j a)_{j \in \mathbb{N}}$ is a monotone sequence witnessing that $C_i a$ is entangled. □

We are now ready to complete the construction of the automaton $\mathcal{D}'$ using $\sim$.

Definition 3.7. Let $\mathcal{D}$ be a DFA and let $\mathcal{L} = \mathcal{L}(\mathcal{D})$. Define $\mathcal{D}' = (Q', s', \delta', F')$ by:

- $Q' = \{[\alpha]_{\sim} : \alpha \in \text{Pref}(\mathcal{L})\}$;
- $\delta'([\alpha]_{\sim}, a) = [\alpha a]_{\sim}$ for every $\alpha \in \text{Pref}(\mathcal{L})$ and for every $a \in \Sigma$ such that $\alpha a \in \text{Pref}(\mathcal{L})$;
- $s' = [\varepsilon]_{\sim}$;
- $F' = \{[\alpha]_{\sim} : \alpha \in \mathcal{L}\}$.

The equivalence relation $\sim$ is right-invariant (Lemma 3.6), has finite index, and $\mathcal{L}$ is the union of some $\sim$-classes (Lemma 3.5). Hence, $\mathcal{D}'$ is a well-defined DFA, and $\alpha \in [\beta]_{\sim} \iff \delta'(s', \alpha) = [\beta]_{\sim}$, which

implies that for every $\alpha \in \mathrm{Pref}(\mathcal{L})$ it holds:

$$I_{[\alpha]_\sim} = [\alpha]_\sim \tag{3.1}$$

and so $\mathcal{L}(\mathcal{D}') = \mathcal{L}$.

The following Lemma shows that it is safe to replace $C_1, \ldots, C_n$ in Definition 3.4 by the elements of a *minimum-size* e.c. decomposition, that is, an e.c. decomposition with minimum cardinality.

Lemma 3.8. *Let $\mathcal{D}$ be a DFA and let $\mathcal{V} = \{V_1, \ldots, V_r\}$, with $V_1 \prec \cdots \prec V_r$, be a minimum-size e.c. decomposition of $\mathcal{D}$. Then, $\alpha \sim \alpha'$ holds if and only if:*

- *$\delta(s, \alpha) = \delta(s, \alpha')$ and*
- *there exist integers $i \leq j$ such that:*
  - *$[\alpha, \alpha']^{\pm} \subseteq \bigcup_{h=i}^{j} V_h$;*
  - *$V_h \cap I_{\delta(s,\alpha)} \neq \emptyset$, for all $h \in \{i, \ldots, j\}$.*

Proof. To prove that $\alpha \sim \alpha'$ holds under the above hypotheses, it is sufficient to recall that $V_i, \ldots, V_j$ are entangled convex sets and apply Definition 3.4.

Let us prove the reverse implication. Pick $\alpha \sim \alpha' \in \mathrm{Pref}(\mathcal{L}(\mathcal{D}))$ and let $C_1, \ldots, C_n$ be entangled convex sets such that $[\alpha, \alpha']^{\pm} \subseteq \bigcup_{i=1}^{n} C_i$ and $C_i \cap I_u \neq \emptyset$, for every $i = 1, \ldots, n$, where $u = \delta(s, \alpha) = \delta(s, \alpha')$. By Lemma A.12 of Appendix A in [CDPP23] we can assume that $C_1 \prec C_2 \prec \cdots \prec C_n$. Let $i \leq j$ be such that $[\alpha, \alpha']^{\pm} \subseteq \bigcup_{h=i}^{j} V_h$ and $V_h \cap [\alpha, \alpha']^{\pm} \neq \emptyset$ for all $h \in \{i, \ldots, j\}$. We just have to prove that $V_h \cap I_u \neq \emptyset$, for all $h \in \{i, \ldots, j\}$. From $V_h \cap [\alpha, \alpha']^{\pm} \neq \emptyset$ it follows that either V_h contains α or α', or $V_h \subseteq [\alpha, \alpha']^{\pm} \subseteq \bigcup_{i=1}^{n} C_i$. In the first case we have $V_h \cap I_u \neq \emptyset$ because $\alpha, \alpha' \in I_u$, while in the second case $V_h \cap I_u \neq \emptyset$ follows from Lemma A.9 of Appendix A in [CDPP23]. □

Let $\mathcal{D}'$ be the automaton of Definition 3.7. The following corollary allows us to consider an e.c. decomposition of $\mathcal{D}$ of minimum size as an e.c. decomposition of $\mathcal{D}'$.

Corollary 3.9. *Any e.c. decomposition of minimum size $V_1 \prec \cdots \prec V_r$ of $\mathcal{D}$ is also an e.c. decomposition of $\mathcal{D}'$. Moreover, for all $u' \in Q'$, there exist $i \leq j$ such that $I_{u'} \subseteq \bigcup_{h=i}^{j} V_h$ and $V_h \cap I_{u'} \neq \emptyset$, for $h = i, \ldots, j$.*

Proof. In order to prove that an e.c. decomposition of minimum size $V_1 \prec \cdots \prec V_r$ of $\mathcal{D}$ is also an e.c. decomposition of $\mathcal{D}'$ we just have to check that V_h is entangled in $\mathcal{D}'$, for all $h = 1, \ldots, r$. Let $u'_1, \ldots, u'_k$ be the pairwise distinct $\mathcal{D}'$-states occurring in V_h. Notice that by the definition of δ' we have $\{u'_1, \ldots, u'_k\} = \{\delta'(s', \alpha) \mid \alpha \in V_h\} = \{[\alpha]_\sim \mid \alpha \in V_h\}$. Hence, for every $j = 1, \ldots, k$ there exists $\alpha_j \in V_h$ such that $u'_j = [\alpha_j]_\sim$. Then the $\mathcal{D}$-states $u_j = \delta(s, \alpha_j)$, for $j = 1, \ldots, k$, occur in V_h. Notice that $u_1, \ldots, u_k$ are pairwise distinct as well: if u_i were equal to u_j for $i \neq j$, then from Lemma 3.8 we would have $\alpha_i \sim \alpha_j$ and $u'_i = [\alpha_i]_\sim = [\alpha_j]_\sim = u'_j$ would follow. Since V_h is entangled in $\mathcal{D}$, there exists a monotone sequence $(\beta_i)_{i \in \mathbb{N}}$ in V_h reaching each u_j infinitely many times. Fix $j \in \{1, \ldots, k\}$. If $\delta(s, \beta_i) = u_j$ then from $\delta(s, \alpha_j) = u_j$ and $\beta_i, \alpha_j \in V_h$ it follows $\alpha_j \sim \beta_i$ again by Lemma 3.8, so that

$\delta'(s', \beta_i) = [\beta_i]_\sim = [\alpha_j]_\sim = u'_j$ in $\mathcal{D}'$. It follows that the sequence $(\beta_i)_{i\in\mathbb{N}}$ reaches u'_j infinitely many times. Hence, V_h is entangled in $\mathcal{D}'$.

As for the second part of the Corollary, if $u' = [\alpha]_\sim \in Q'$ then $I_{u'} = [\alpha]_\sim$ (see Equation 3.1 above). Let i (j, respectively) be the minimum (maximum) index h with $[\alpha]_\sim \cap V_h \neq \emptyset$; then $[\alpha]_\sim \subseteq V_i \cup \cdots \cup V_j$. Fix $h \in \{i, \ldots, j\}$ and consider $u = \delta(s, \alpha)$. Since there exist $\alpha', \alpha'' \in \mathrm{Pref}(\mathcal{L}(\mathcal{D}))$ such that $\alpha' \sim \alpha \sim \alpha''$, $\alpha' \in V_i$ and $\alpha'' \in V_j$, then, Lemma 3.8 implies $\delta(s, \alpha') = \delta(s, \alpha'') = \delta(s, \alpha) = u$ and $V_h \cap I_u \neq \emptyset$. Pick $\beta \in V_h \cap I_u$. Then, the same lemma implies $\beta \sim \alpha$ so that $\beta \in I_{u'}$ and $V_h \cap I_{u'} \neq \emptyset$ as well. □

We can now prove that $\mathcal{D}'$ has width equal (to its entanglement and) to the entanglement of $\mathcal{D}$.

Theorem 3.10. *If $\mathcal{D}$ is a DFA, then* $\mathrm{ent}(\mathcal{D}) = \mathrm{ent}(\mathcal{D}') = \mathrm{width}(\mathcal{D}')$.

Proof. Let us prove that $\mathrm{ent}(\mathcal{D}') \leq \mathrm{ent}(\mathcal{D})$. Consider an entangled collection $\{[\alpha_1]_\sim, \ldots, [\alpha_h]_\sim\}$ of h states in $\mathcal{D}'$. Then, there is a monotone sequence $(\gamma_i)_{i\in\mathbb{N}}$ such that, for each $j \in \{1, \ldots, h\}$ we have $\delta'(s', \gamma_i) = [\alpha_j]_\sim$ for infinitely many i's. Let $\mathcal{V}$ be a minimum-size finite e.c. decomposition of $\mathcal{D}$. Since $\mathcal{V}$ is a finite partition and all the elements of $\mathcal{V}$ are convex, there exists $V \in \mathcal{V}$ and n_0 such that $\gamma_i \in V$ for all $i \geq n_0$. In particular, there are words $\beta_1, \ldots, \beta_h$ in V such that $\delta'(s', \beta_k) = [\alpha_k]_\sim$, for every $k = 1, \ldots, h$. Define $u_k = \delta(s, \beta_k)$ and notice that the states $u_1, \ldots, u_h$ are pairwise distinct. In fact, if $u_r = u_s$ for $r \neq s$, then by Lemma 3.8 we would have $\beta_r \sim \beta_s$ and $[\alpha_r]_\sim = \delta'(s', \beta_r) = [\beta_r]_\sim = [\beta_s]_\sim = \delta'(s', \beta_s) = [\alpha_s]_\sim$, a contradiction. Moreover, $\{u_1, \ldots, u_h\}$ is an entangled set in $\mathcal{D}$, because all these states occur in V (as witnessed by $\beta_1, \ldots, \beta_h$) and V is an element of an e.c. decomposition. Since this holds for any collection of entangled states in $\mathcal{D}'$, it follows that $\mathrm{ent}(\mathcal{D}') \leq \mathrm{ent}(\mathcal{D})$.

Let us now prove that $\mathrm{ent}(\mathcal{D}) \leq \mathrm{ent}(\mathcal{D}')$. Let $\{u_1, \ldots, u_h\}$ be an entangled set of h states in $\mathcal{D}$, witnessed by some monotone sequence $(\alpha_i)_{i\in\mathbb{N}}$. Every I_{u_k} is equal to a finite union of some $I_{u'}$'s, with $u' \in Q'$, and $(\alpha_i)_{i\in\mathbb{N}}$ goes through any u_k infinitely many times. Therefore, for all $k = 1, \ldots h$ there exist $u'_k \in Q'$ such that $I_{u'_k} \subseteq I_{u_k}$ and $(\alpha_i)_{i\in\mathbb{N}}$ goes through u'_k infinitely many times. We conclude that $u'_1, \ldots, u'_h$ are pairwise distinct and $\{u'_1, \ldots, u'_h\}$ is an entangled set of states in $\mathcal{D}'$, which implies $\mathrm{ent}(\mathcal{D}) \leq \mathrm{ent}(\mathcal{D}')$.

Finally, we prove that $\mathrm{width}(\mathcal{D}') = \mathrm{ent}(\mathcal{D}')$. If $\mathcal{V}$ is an e.c. decomposition of $\mathcal{D}$ of minimum size, then Corollary 3.9 implies that $\mathcal{V}$ is an e.c. decomposition of $\mathcal{D}'$ satisfying the hypothesis of Lemma 3.3 so that $\mathrm{width}(\mathcal{D}') = \mathrm{ent}(\mathcal{D}')$ follows from this lemma. □

If we start from the minimum DFA $\mathcal{D}_\mathcal{L}$ of a regular language $\mathcal{L}$, then, the equivalence relation $\sim_{\mathcal{D}_\mathcal{L}}$ allows us to define an automaton that realizes the deterministic width of $\mathcal{L}$ (the one defined in the next Section).

Definition 3.11 (Hasse equivalence)**.** If $\mathcal{D}_\mathcal{L}$ is the minimum DFA of a regular language $\mathcal{L}$, then $\sim_{\mathcal{D}_\mathcal{L}}$ is called the *Hasse equivalence* of the language $\mathcal{L}$, and is denoted by $\sim_{\mathcal{H}_\mathcal{L}}$.

This relation will be used in further considerations in Section 3.5.

3.4 The Hasse automaton

As we just said, applying the above construction to the minimum-size automaton of a regular language, we obtain a DFA $\mathcal{D}'_{\mathcal{L}}$ that realizes the deterministic width of the language $\mathcal{L}$.

Definition 3.12. If $\mathcal{D}_{\mathcal{L}}$ is the minimum DFA of a regular language $\mathcal{L}$, the DFA $\mathcal{D}'_{\mathcal{L}}$ is called the *Hasse automaton* for $\mathcal{L}$ and it is denoted by $\mathcal{H}_{\mathcal{L}}$.

The authors motivates this name with the fact that the width of the language can be "visualized" by the Hasse diagram of the partial order $\leq_{\mathcal{H}_{\mathcal{L}}}$.

Theorem 3.13. *If $\mathcal{D}_{\mathcal{L}}$ is the minimum DFA of the regular language $\mathcal{L}$, then:*

$$\text{width}^D(\mathcal{L}) = \text{width}(\mathcal{H}_{\mathcal{L}}) = \text{ent}(\mathcal{D}_{\mathcal{L}}) = \text{ent}(\mathcal{L}).$$

Proof. By Lemma 2.22 we have $\text{ent}(\mathcal{D}_{\mathcal{L}}) = \text{ent}(\mathcal{L})$. Since $\text{ent}(\mathcal{D}) \leq \text{width}(\mathcal{D})$ for all DFAs (Lemma 2.21), we obtain $\text{ent}(\mathcal{L}) \leq \text{width}^D(\mathcal{L})$, while from Theorem 3.10 we know that $\text{width}(\mathcal{H}_{\mathcal{L}}) = \text{ent}(\mathcal{D}_{\mathcal{L}})$. Hence, we have:

$$\text{width}(\mathcal{H}_{\mathcal{L}}) = \text{ent}(\mathcal{D}_{\mathcal{L}}) = \text{ent}(\mathcal{L}) \leq \text{width}^D(\mathcal{L}) \leq \text{width}(\mathcal{H}_{\mathcal{L}})$$

and the conclusion follows. □

In order to better understand the construction of $\mathcal{H}_{\mathcal{L}}$, let's make some examples.

Example 3.14. Consider the language $\mathcal{L} = \varepsilon + (a+b)c^* + ac^*d + gd + ee + he + f + k + e + h + g$. The minimum DFA that recognizes $\mathcal{L}$ is the DFA $\mathcal{D}_1$ depicted in Figure 2.2. Figure 3.1 represents the sets I_u as intervals in $(\text{Pref}(\mathcal{L}), \preceq)$, for every state $u \in \mathcal{D}_1$: thicker lines represents "dense" zones of I_u, that is, zones with at least one string in I_u.

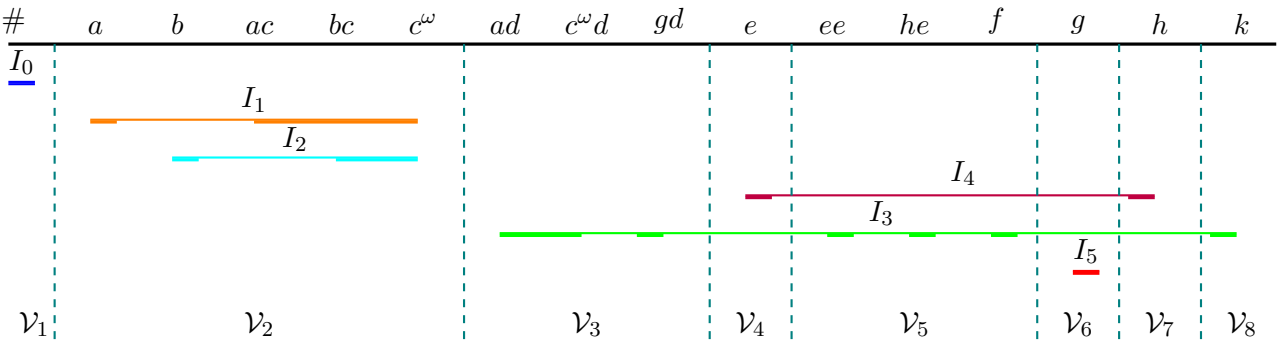


Figure 3.1: Interval representation of I_u, $u \in \mathcal{D}_1$

.

The first step is to find a minimum-size entangled convex decomposition of $\text{Pref}(\mathcal{L})$.

Claim. *The decomposition $\mathcal{V}_1 = \{\#\}, \mathcal{V}_2 = ac^* + bc^*, \mathcal{V}_3 = dc^*a + gd, \mathcal{V}_4 = \{e\}, \mathcal{V}_5 = \{ee, he, f\}, \mathcal{V}_6 = \{g\}, \mathcal{V}_7 = \{h\}$ and $\mathcal{V}_8 = \{k\}$ is a minimum-size e.c. decomposition of* $\text{Pref}(\mathcal{L})$.

Proof. To see this, we first prove that this is a valid e.c. decomposition, and then we will prove that it has the minimum size. First, we note that any one of $\mathcal{V}_1, \mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5, \mathcal{V}_6, \mathcal{V}_7, \mathcal{V}_8$ contains strings reaching

the same state. Thus, they are entangled sets (and convex, as the Figure shows). The set $\mathcal{V}_2$ contains all the strings reaching states 1 and 2: since the set $\{1, 2\}$ is entangled (as implicitly proven in Example 2.18), $\mathcal{V}_2$ is entangled.

Now, we prove that cannot exist a decomposition with 7 blocks. Consider strings $\#, a, ad, e, ee, g, h, k$, and suppose, by contradiction, that a decomposition with 7 blocks exists. By the Pigeonhole principle, one of those blocks must contain 2 consequent strings from the ones listed above. However, no pair of consequent strings in $\{\#, a, ad, e, ee, g, h, k\}$ reaches two entangled states, so no pair can belong to the same entangled block. Thus, a decomposition with 7 blocks doesn't exist. □

Now, let's see, for every state u, which strings reaching u are equivalent with respect to $\sim_{\mathcal{D}_1}$.

1. All the strings reaching 1 are contained in block $\mathcal{V}_2$, so all of them must be equivalent.
2. All the strings reaching 2 are contained in block $\mathcal{V}_2$, so all of them must be equivalent.
3. In state 3, string ee, he, f belong to the same e.c. block, so they are equivalent. Analogously, all the strings in $ac^*d + gd$ are equivalent.

   Note that $ad \not\sim_{\mathcal{D}_1} ee$, since $[ad, ee] \subseteq \mathcal{V}_3 \cup \mathcal{V}_4 \cup \mathcal{V}_5$ and $V_4 \cap I_3 = \emptyset$. Analogously, $ad \not\sim_{\mathcal{D}_1} k$ and $ee \not\sim_{\mathcal{D}_1} k$. Thus, state 3 will be split in 3 states, 3, $3'$ and $3''$.
4. Note that $e \not\sim_{\mathcal{D}_1} h$, since $[e, h] \subseteq \mathcal{V}_4 \cup \mathcal{V}_5 \cup \mathcal{V}_6 \cup \mathcal{V}_7$ and $V_5 \cap I_4 = \emptyset$. So, state 4 will be split in 2 states, 4 and $4'$.
5. State 5 is reached by only one string, so it won't be split.

Taking into account all the considerations above, the resulting Hasse automaton for $\mathcal{L}$ is the one in Figure 3.2.

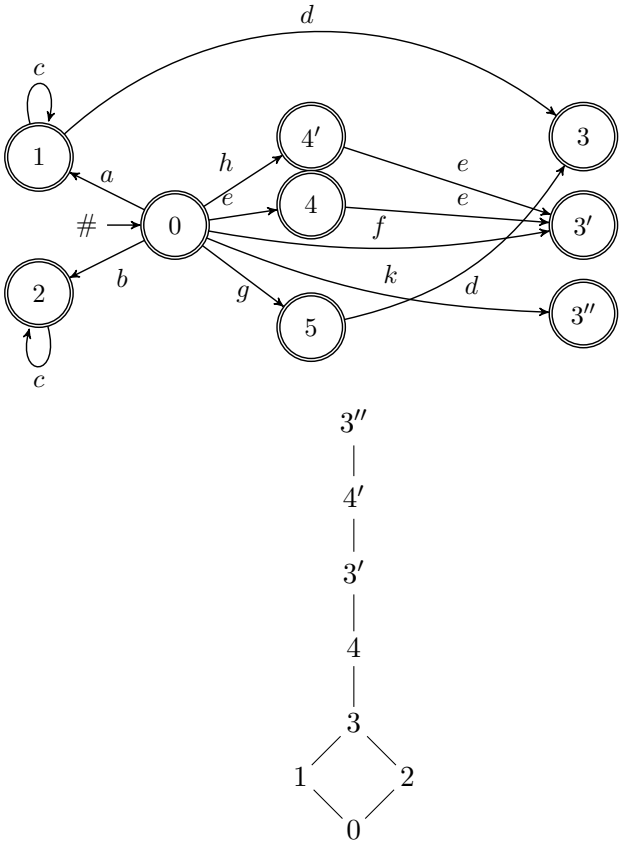


Figure 3.2: The Hasse automaton of the language $\mathcal{L} = \varepsilon + (a+b)c^* + ac^*d + gd + ee + he + f + k + e + h + g$ and the corresponding Hasse diagram.

The Hasse automaton in the previous example has 9 states, 2 more than DFAs $\mathcal{D}_2, \mathcal{D}_3$ in Example 2.18: states 3 and 4 of $\mathcal{D}_1$, since they are not entangled, were forced to be split into totally ordered states. However, this is unnecessary: for example, if state 4 of D_1 were not split, the width of the resulting DFA would still be 2. This motivates the fact that, in general, the Hasse automaton has more state than necessary.

In the previous example, only non-entangled states were split. In the following one, we will see that splits can happen also to entangled states.

Example 3.15. Let's consider the language $\mathcal{L} = (a + d)c^* + df^* + b(c + f)^*$. The minimum-size DFA is depicted in Figure 3.3. As before, Figure 3.4 represents the various I_u over the total order $(\mathrm{Pref}(\mathcal{L}), \preceq)$. In contrast to Example 3.14, there are two entangled sets of states: $a, b, ac, bc, \ldots, ac^i, bc^i, \ldots$ witnesses the set $\{1, 2\}$ and $bf, df, bff, dff, \ldots, bf^i, df^i, \ldots$ witnesses the set $\{2, 3\}$. As you can note, state 2 is entangled with two distinct states, 1 and 3, that are not entangled with each other. This fact will cause the splitting of state 2.

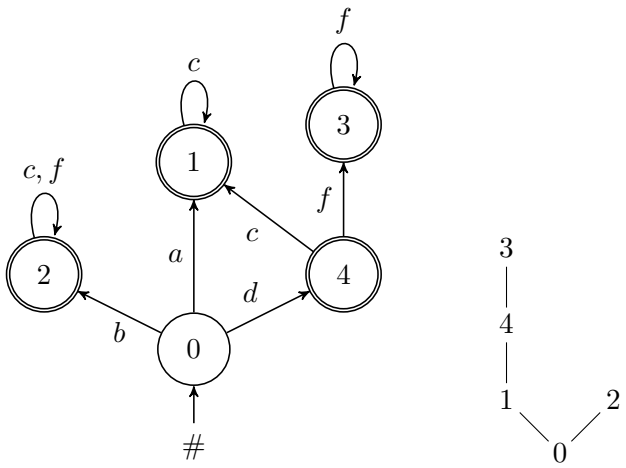


Figure 3.3: Minimum-size automaton of the language $\mathcal{L} = (a+d)c^*+df^*+b(c+f)^*$ and the corresponding Hasse diagram.

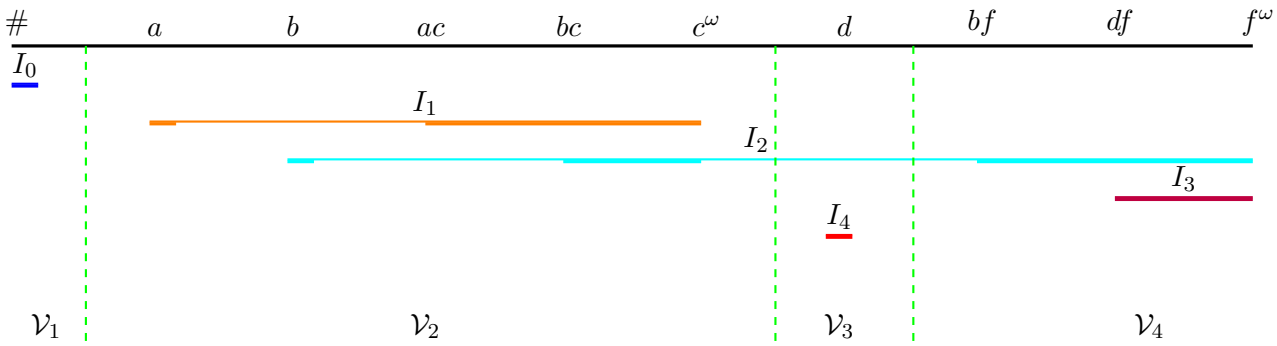


Figure 3.4: Interval representation of I_u, $u \in \mathcal{D}_\mathcal{L}$.

In this case, the e.c. decomposition is $\mathcal{V}_1 = \{\varepsilon\}, \mathcal{V}_2 = ac^* + dc^+ + b + b(c+f)^*c, \mathcal{V}_3 = \{d\}, \mathcal{V}_4 = b(c+f)^*f + df^+$. Analogously to Example 3.1, it can be proven that this is a minium-size e.c. decomposition.

Finally, let's see which strings are $\sim_{\mathcal{D}_\mathcal{L}}$-equivalent.

1. All the strings reaching state 1 belong to $\mathcal{V}_2$, so they are all equivalent.
2. The strings reaching state 2 are split in two entangled blocks: strings in $A_1 = b+b(c+f)^*c$ belong to $\mathcal{V}_2$, while strings in $A_2 = b(c+f)^*f$ belong to $\mathcal{V}_4$. So, both sets contain strings $\sim_{\mathcal{D}_\mathcal{L}}$-equivalent. Now, consider the strings b and bf: the interval $[b, bf]$ is totally included in $\mathcal{V}_2 \cup \mathcal{V}_3 \cup \mathcal{V}_4$, but $I_2 \cap V_3 = \emptyset$. Thus, $b \not\sim_{\mathcal{D}_\mathcal{L}} bf$ and by transitivity all the strings in A_1 are not equivalent to the strings in A_2. This leads to a split of state 2.

The resulting Hasse automaton is the DFA in Figure 3.5.

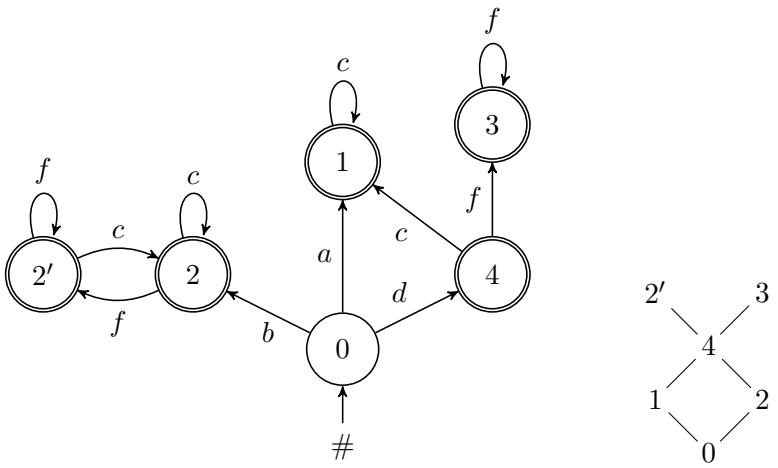


Figure 3.5: Hasse automaton of the language $\mathcal{L} = (a + d)c^* + df^* + b(c + f)^*$ and the corresponding Hasse diagram.

In the last example, splitting does not reduce the width: automaton $\mathcal{D}_\mathcal{L}$ already realizes the minimum width of the language it recognizes. Moreover, splitting state 2 does not involve other states: the transition function changed only for state 2 itself.

Although in these two examples, the number of states increases by 1 or 2, unuseful splitting may cause an exponential blowup in the number of states. Indeed, using Theorem 29 in [MPPR24], it can be shown that the ratio between the size of the Hasse automaton and the minimum-size DFA recognizing $\mathcal{L}$ grows exponentially with respect to the size of the last.

Therefore, it is necessary to examine approaches that reduce the number of states while maintaining the width constant. In the next Section, we will see that the minimum-index equivalences that induce a DFA of width p are convex refinements of $\approx_\mathcal{L}$.

3.5 $\mathcal{L}$-convex equivalences

In order to talk about the width of a DFA obtained by an equivalence relation on strings, it can be useful to give a formal definition of *width of an equivalence relation*: considering the equivalence classes as sets of strings, we can define a partial order on the classes coherent with Definition 2.6.

Definition 3.16. For some $\mathcal{L}$, let $(\mathrm{Pref}(\mathcal{L}), \preceq)$ be a totally co-lex ordered set of strings and $\approx$ an equivalence relation on $\mathrm{Pref}(\mathcal{L})$. Then, the order $<_\approx$ is defined as follows:

$$[\alpha]_\approx <_\approx [\beta]_\approx \iff \big(\forall \alpha' \in [\alpha]_\approx\big)\big(\forall \beta' \in [\beta]_\approx\big)\, \alpha' \prec \beta'.$$

It is easy to see that $\leq_\approx$ is a partial order over the $\approx$-classes and that, interpreting $\alpha \approx \beta$ as $\delta(s, \alpha) = \delta(s, \beta)$ on a DFA, the above definition is equivalent to Definition 2.6.

The definition of width of an equivalence relation comes straightforwardly.

Definition 3.17. The *width of an equivalence relation* $\approx$ over a set of strings Z, denoted by width($\approx$), is the width of the partial order $\leq_\approx$ over the $\approx$-classes.

We can note that, instead of considering all the strings in the $\approx$-classes, we can define the partial order using only the minimum and the maximum string in each equivalence class.

Theorem 3.18. *Let $(Z, \preceq)$ be a totally co-lex ordered set of strings and $\approx$ an equivalence relation on Pref$(\mathcal{L})$. Let $\sup([\alpha]_\approx)$ the superior limit string $[\alpha]_\approx$, and $\inf([\alpha]_\approx)$ the inferior limit. Then,*

$$[\alpha] <_\approx [\beta] \iff \sup([\alpha]_\approx) \preceq \inf([\beta]_\approx).$$

This allows us to view the order of classes as the typical partial order on the set of intervals $\{[\inf([\alpha]_\approx), \sup([\alpha]_\approx)], \alpha \in \mathrm{Pref}(\mathcal{L})\}$. This will help to simplify further proofs of the section.

Given a language $\mathcal{L}$, our goal is to find an automaton $\mathcal{H}'_\mathcal{L}$ that realizes the width of $\mathcal{L}$ and that has the minimum number of states. Thus, we want to find a relation $\approx$ over strings that is (generally) finer than the Myhill-Nerode equivalence $\approx_\mathcal{L}$ and (generally) coarser than Hasse equivalence $\approx_{\mathcal{H}_\mathcal{L}}$, and such that it is right-invariant and its width is equal to $\mathrm{width}^D(\mathcal{L})$.

Instead of considering all these properties at once, we can divide the problem into two parts: first, we can characterize equivalences that have the same width, and then we consider right-invariant refinements of those equivalences: we will see that, in some cases, the refinement will not make the width increase.

First, let's define the family of equivalences that share the same width.

Definition 3.19. Let $\approx$ be an equivalence relation over a co-lex ordered set of strings Pref$(\mathcal{L})$. Then, we say that $\approx$ *belongs to the family of equivalences* $\mathcal{W}(p)$ if the width of $\approx$ is at most p.

If we fix $\mathcal{W}(p)$, we want to better characterize the set of equivalences with the minimum number of classes. In addition, since our final goal is to define an automaton that recognizes language $\mathcal{L}$, we only consider the relations in $\mathcal{W}(p)$ with finite index which refines $\approx_\mathcal{L}$.

In [ADPP21], it is shown that Wheeler languages admit a unique minimum-size Wheeler DFA. This result is obtained by defining a refinement of $\approx_\mathcal{L}$, where each class in the new equivalence is convex relatively to Pref$(\mathcal{L})$. This suggests that convex equivalences are good candidates in characterizing the set of equivalences in each $\mathcal{W}(p)$ with the minimum number of classes (minimum-index equivalences).

Indeed, we will prove that every equivalence in $\mathcal{W}(p)$ has a corresponding convex equivalence in $\mathcal{W}(p)$.

Definition 3.20. Let $\approx$ be a refinement of $\approx_\mathcal{L}$. Then $\approx$ is $\mathcal{L}$*-convex* if and only if:

$$\alpha \approx \beta \quad \Rightarrow \quad \forall \gamma \in \mathrm{Pref}(\mathcal{L})\ (\alpha \approx_\mathcal{L} \gamma \approx_\mathcal{L} \beta \wedge \alpha \prec \gamma \prec \beta) \Rightarrow \alpha \approx \gamma.$$

Note that the previous definition is equivalent to Definition 10 in [ADPP21], without the input-consistency requirement.

Definition 3.21. Let $\approx$ be an equivalence relation with finite index that refines $\approx_\mathcal{L}$. Then, the $\mathcal{L}$*-convex closure* of $\approx$, denoted by $\approx^\circ$, is the smallest $\mathcal{L}$-convex refinement of $\approx_\mathcal{L}$ that contains $\approx$.

Note that, if $\approx$ is already $\mathcal{L}$-convex, then $\approx^\circ$ concides with $\approx$. Moreover, since $\approx^\circ$ is coarser than $\approx$, the number of $\approx^\circ$-classes is smaller than the number of $\approx$-classes (in particular, its index is finite).

The next theorem will prove that the width of the $\mathcal{L}$-convex closure of $\approx$ is always smaller or equal to the width of $\approx$.

Theorem 3.22. *Let $\approx \in \mathcal{W}(p)$ be an equivalence relation with finite index that refines $\approx_{\mathcal{L}}$. Then the $\mathcal{L}$-convex closure $\approx^{\circ}$ is in $\mathcal{W}(p)$.*

Proof. The proof is divided into two parts: first, we provide an algorithm that, starting from $\approx$, builds the relation $\approx^{\circ}$ in a finite number of steps; then, we will prove that after every step of the algorithm, the width of the equivalence does not change.

Consider the following procedure:

1: $\mathcal{C} \leftarrow \{[\alpha_1], [\alpha_2], \ldots, [\alpha_n]\}$; {the set of classes of $\approx$.}
2: **while** exists $C_1, C_2 \in \mathcal{C}$ such that $C_1 \approx_{\mathcal{L}} C_2$ and $C_1 || C_2$ **do**
3: $\quad \mathcal{C} \leftarrow \mathcal{C} \setminus \{C_1, C_2\}$;
4: $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{C_1 \cup C_2\}$;
5: **end while**
6: **return** $\mathcal{C}$; {the $\mathcal{L}$-convex relation $\approx^*$ expressed as the set of classes $\mathcal{C}$.}

Since the number of elements in $\mathcal{C}$ strictly decreases at every step of the while-loop, then the algorithm must halt in a finite number of steps.

Let $\approx^*$ be the equivalence relation defined by the above algorithm, expressed as equivalence classes contained in $\mathcal{C}$. We will prove that $\approx^* = \approx^{\circ}$.

- Since $\mathcal{C}$ is initialized with the classes of $\approx$, and the algorithm just merges these classes, then $\approx \subseteq \approx^*$.
- Since the procedure merges only classes that are $\approx_{\mathcal{L}}$ equivalent, $\approx^*$ is a refinement of $\approx_{\mathcal{L}}$.
- Suppose by contradiction that $\approx^*$ is not $\mathcal{L}$-convex. Then there exists $\alpha \approx_{\mathcal{L}} \beta \approx_{\mathcal{L}} \gamma$ such that $\alpha, \gamma \in C_i$, $\beta \in C_j$ $(i \neq j)$ and $\alpha \prec \beta \prec \gamma$. Since $\alpha \prec \beta \prec \gamma$, the classes C_i and C_j are incomparable. Since C_i and C_j fulfill all the conditions of the while in line 2, the algorithm would have merged C_i and C_j, and, thus, such scenario cannot happen. Hence, the equivalence $\approx^*$ is $\mathcal{L}$-convex.
- Let $\approx^+$ be a $\mathcal{L}$-convex refinement of $\approx_{\mathcal{L}}$ that contains $\approx$. We will prove by induction on the steps of the while-loop, that each class of $\approx^*$ is contained in exactly one class of $\approx^+$, which means that $\approx^* \subseteq \approx^+$.

  At step 0, the set $\mathcal{C}$ contains the classes of $\approx$, that by assumption are fully contained in one class of $\approx^+$ each. Suppose now that, after step i, each class in $\mathcal{C}$ is fully contained in one class of $\approx^+$. We want the same property hold after step $i+1$. Let $\mathcal{C}_1, \mathcal{C}_2$ the two classes identified in line 2 at step $i+1$. Suppose w.l.o.g. that $\inf(C_1) \preceq \inf(C_2)$. Since $C_1 || C_2$, we have that $\inf(C_1) \preceq \inf(C_2) \prec \sup(C_1)$ and, by definition of $\mathcal{L}$-convexity, these three strings must be $\approx^+$-equivalent. Hence, by transitivity, all the strings in C_1 must be $\approx^+$-equivalent to the strings in C_2. Thus, C_1 and C_2 must be contained in the same $\approx^+$-class, making $C_1 \cup C_2$ fully contained in only one $\approx^+$-class.

Now, we will prove that after every step of the while-loop, the width of the intermediate equivalence $\approx^{\sim}$ - expressed by $\mathcal{C}$ - does not increase. Suppose that, before step i, the width of $\approx^{\sim}$ is p: this means that there exists a $\leq_{\approx^{\sim}}$-chain decomposition D of $\mathcal{C}$ whose size is p. Let C_1 and C_2 be the two classes identified in line 2 at step i and suppose w.l.o.g. that $\inf(C_1) \preceq \inf(C_2)$. Let D_1 and D_2 be the chains in D that contains, respectively, C_1 and C_2 . Note that these two chains must be different, since $C_1||C_2$. Let $(D_1)_{<C_1} = \{C_i \in D_1 \mid C_i <_{\approx^{\sim}} C_1\}$ the set of classes in the chain that are $<_{\approx^{\sim}}$-smaller than C_1. Similarly, let's define $(D_1)_{>C_1}, (D_2)_{<C_2}$, and $(D_2)_{>C_2}$. With these sets, we will prove that, after step i, there exists a $<_{\approx^{\sim}}$-chain decomposition of $\mathcal{C}$ whose size is p. The proof will be divided into two cases:

Case 1: $\sup(C_1) \preceq \inf(C_2)$**.**

Let

$$
\begin{aligned}
D_1' &= (D_1)_{<C_1} \cup \{C_1 \cup C_2\} \cup (D_2)_{>C_2} \\
D_2' &= (D_2)_{<C_2} \cup (D_1)_{>C_1} \\
D' &= (D \setminus \{D_1 \cup D_2\}) \cup \{D_1', D_2'\}
\end{aligned}
$$

If we show that D_1' and D_2' are $<_{\approx^{\sim}}$-chains, then D' will be a $<_{\approx^{\sim}}$-chain decomposition of size p (the other chains are not influenced by the changes at step i). Let's prove this fact:

- Since $\inf(C_1) \preceq \inf(C_2)$, all the elements in $(D_1)_{<C_1}$ are smaller than $C_1 \cup C_2$. Analogously, since $\sup(C_1) \preceq \sup(C_2)$, all the elements in $(D_2)_{>C_2}$ are grater than $C_1 \cup C_2$. So, D_1' is a $<_{\approx^{\sim}}$-chain.
- Since incomparability implies $\inf(C_2) \preceq \sup(C_1)$, all the classes in $(D_2)_{<C_2}$ are smaller than the classes in $(D_1)_{>C_1}$.

Case 2: $\sup(C_2) \preceq \sup(C_1)$**.**

Let

$$
\begin{aligned}
D_1' &= (D_1)_{<C_1} \cup \{C_1 \cup C_2\} \cup (D_1)_{>C_1} \\
D_2' &= D_2 \setminus C_2 \\
D' &= (D \setminus \{D_1 \cup D_2\}) \cup \{D_1', D_2'\}
\end{aligned}
$$

If we show that D_1' and D_2' are $<_{\approx^{\sim}}$-chains, then D' will be a $<_{\approx^{\sim}}$-chain decomposition of size p (the changes at step i do not influence the other chains). Let's prove this fact:

- Since $\inf(C_1) \preceq \inf(C_2)$ and $\sup(C_2) \preceq \sup(C_1)$, all the elements in $(D_1)_{<C_1}$ are smaller than $C_1 \cup C_2$, and all the elements in $(D_1)_{>C_1}$ are grater than $C_1 \cup C_2$. So, D_1' is a $<_{\approx^{\sim}}$-chain.
- Since D_2' is a subset of D_2, it is a $<_{\approx^{\sim}}$-chain.

Thus, since after every step of the while-loop there exists a valid $<_{\approx^{\sim}}$-chain decomposition of size p, then the width of the resulting equivalence relation is at most p. So $\approx^{\circ}$ has width at most p and hence belongs to $\mathcal{W}(p)$. □

Since $\approx^{\circ}$ is coarser than $\approx$, all the refinements of $\approx$ are refinements of $\approx^{\circ}$. Also, as stated by Theorem

3.24, the right-invariant refinement of a $\mathcal{L}$-convex equivalence is still $\mathcal{L}$-convex.

Definition 3.23. Let $\approx$ be a refinement of $\approx_{\mathcal{L}}$. Then, for every pair of strings α, β we define:

$$\alpha \approx^r \beta \iff (\forall \phi \in \Sigma^*)(\alpha\phi \in \mathrm{Pref}(\mathcal{L}) \Rightarrow \alpha\phi \approx \beta\phi).$$

We say that $\approx^r$ is the *right-invariant refinement* of $\approx$.

Theorem 3.24. *Let $\approx$ be a $\mathcal{L}$-convex refinement of $\approx_{\mathcal{L}}$. Then, the right-invariant refinement of $\approx$, $\approx^r$, is $\mathcal{L}$-convex.*

Proof. Assume that $\alpha, \beta, \gamma \in \mathrm{Pref}(\mathcal{L})$ are such that $\alpha \approx^r \gamma$, $\alpha \prec \beta \prec \gamma$ and $\alpha \approx_{\mathcal{L}} \beta$. Being $\approx^r$ a refinement of $\approx_{\mathcal{L}}$, we have $\alpha \approx_{\mathcal{L}} \beta \approx_{\mathcal{L}} \gamma$. We must prove that $\alpha \approx^r \beta$. Fix $\phi \in \Sigma^*$ such that $\alpha\phi \in \mathrm{Pref}(\mathcal{L})$. We must prove that $\alpha\phi \approx \beta\phi$. Since $\alpha \approx_{\mathcal{L}} \beta$ and $\approx_{\mathcal{L}}$ is right-invariant, we have $\alpha\phi \approx_{\mathcal{L}} \beta\phi$. Moreover, $\alpha \approx^r \gamma$ implies $\alpha\phi \approx^r \beta\phi$ by right-invariance, so $\alpha\phi \approx \gamma\phi$. By $\mathcal{L}$-convexity, from $\alpha\phi \approx \gamma\phi$, $\alpha\phi \approx_{\mathcal{L}} \alpha\phi$ and $\alpha\phi \prec \beta\phi \prec \gamma\phi$ (since $\alpha \prec \beta \prec \gamma$) we conclude $\alpha\phi \approx \beta\phi$. □

DFAs are induced by right-invariant equivalences, so we focus on the number of classes these equivalences generate. We will demonstrate that among right-invariant equivalences, the $\mathcal{L}$-convex equivalences have the smallest number of classes (i.e., the minimum index).

Theorem 3.25. *Let $\approx$ be a refinement of $\approx_{\mathcal{L}}$. Then, $\approx^r$ is a refinement of $(\approx^\circ)^r$.*

Proof. Since $\approx^\circ$ is coarser than $\approx$ and $\approx^r$ is a right-invariant refinement of $\approx$, then $\approx^r$ is a right-invariant refinement of $\approx^\circ$. Thus, since $(\approx^\circ)^r$ is the coarest right-invariant refinement of $\approx^\circ$, $\approx^r$ is a refinement $(\approx^\circ)^r$. □

In addition, right-invariant refinements of $\mathcal{L}$-convex equivalences maintain the same width as the original equivalences they refine.

Theorem 3.26. *Let $\approx \in \mathcal{W}(p)$ be a $\mathcal{L}$-convex relation that refines $\approx_{\mathcal{L}}$. Then, $\approx^r$ is in $\mathcal{W}(p)$.*

Proof. Let $D = \{D_1, D_2, \ldots, D_p\}$ be a $\leq_{\approx}$-chain decomposition over the classes of $\approx$ of size p. Since $\approx$ is $\mathcal{L}$-convex, then also $\approx^r$ is L-convex, so every class C_i of $\approx$ is decomposed into classes $C_{i,1} < C_{i,2} < \cdots < C_{i,k}$ that are all pair-wise $\leq_{\approx^r}$-comparable. Thus, the sets

$$\begin{aligned}
D'_1 &= \{C' \text{ class of } \approx^r \mid C' \subseteq C \wedge C \text{ is a class of } \approx \wedge C \in D_1\} \\
D'_2 &= \{C' \text{ class of } \approx^r \mid C' \subseteq C \wedge C \text{ is a class of } \approx \wedge C \in D_2\} \\
&\vdots \\
D'_p &= \{C' \text{ class of } \approx^r \mid C' \subseteq C \wedge C \text{ is a class of } \approx \wedge C \in D_p\}
\end{aligned}$$

are $\leq_{\approx^r}$-chains. Hence, $D' = \{D'_1, D'_2, \ldots, D'_p\}$ is a $\leq_{\approx^r}$-chain decomposition of the classes of $\approx^r$. □

All the theorems above contribute to proving the next theorem, which states that for any DFA whose width is p, there exists a (possibly) smaller automaton such that its width is at most p.

Theorem 3.27. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA such that* $\text{width}(\mathcal{D}) = p$*. Then, there exists an automaton $\mathcal{D}' = (Q', s', \delta', F')$ such that $\mathcal{L}(\mathcal{D}') = \mathcal{L}$,* $\text{width}(\mathcal{D}') \leq p$ *and $|Q'| \leq |Q|$.*

Proof. Let $\mathcal{L} = \mathcal{L}(\mathcal{D})$ and $\approx_{\mathcal{D}}$ be the equivalence relation on $\text{Pref}(\mathcal{L})$ induced by the states of $\mathcal{D}$

$$\alpha \approx \beta \iff \delta(s, \alpha) = \delta(s, \beta).$$

The relation $\approx_{\mathcal{D}}$ is right-invariant, and $\text{width}(\approx_{\mathcal{D}}) = p$.

Let $\approx = ((\approx_{\mathcal{D}})^{\circ})^{r}$. This relation is right-invariant, $\mathcal{L}$-convex (by Theorem 3.24) and it is coarser than $\approx_{\mathcal{D}}$ (by Theorem 3.25, noting that $(\approx_{\mathcal{D}})^{r} = \approx_{\mathcal{D}}$). Thus, its index is at most $|Q|$ (in particular, it is finite) and $\mathcal{L}$ is the union of some $\approx$-classes (since it refines $\approx_{\mathcal{L}}$). Hence, the automaton $\mathcal{D}'$

- $Q' = \{[\alpha]_{\approx} : \alpha \in \text{Pref}(\mathcal{L})\}$;
- $\delta'([\alpha]_{\approx}, a) = [\alpha a]_{\approx}$ for every $\alpha \in \text{Pref}(\mathcal{L})$ and for every $a \in \Sigma$ such that $\alpha a \in \text{Pref}(\mathcal{L})$;
- $s' = [\varepsilon]_{\approx}$;
- $F' = \{[\alpha]_{\approx} : \alpha \in \mathcal{L}\}$.

is a well-defined DFA, and $\alpha \in [\beta]_{\approx} \iff \delta'(s', \alpha) = [\beta]_{\approx}$, which implies that for every $\alpha \in \text{Pref}(\mathcal{L})$ it holds $I_{[\alpha]_{\approx}} = [\alpha]_{\approx}$ and so $\mathcal{L}(\mathcal{D}') = \mathcal{L}$.

Moreover, by Theorem 3.22 applied to $\approx_{\mathcal{D}}$ and Theorem 3.26 applied to $(\approx_{\mathcal{D}})^{\circ}$, we have that $\approx$ is in $\mathcal{W}(p)$ and thus the automaton $\mathcal{D}'$ has width at most p. □

Summarizing, the existence of an $\mathcal{L}$-convex closure that preserves the width allows us to coarsen equivalence relations without increasing their complexity in terms of width. In addition, the fact that $\mathcal{L}$-convex equivalences have the minimum index among right-invariant equivalences suggests that these equivalences are the right candidates for constructing minimal DFAs of minimum width.

Results in this section are far from solving this problem: Hasse equivalence is already $\mathcal{L}$-convex, so the convex closure cannot be used to construct a smaller automaton; in addition, these ideas do not contribute to provide an alternative construction of minimum-size Wheeler DFAs, since every totally co-lex ordered automaton induces a relation that is already $\mathcal{L}$-convex. Still, I hope this content might somehow be useful in further developments in this research field.

4

Barrows-Wheeler transform for Automata

This Chapter is dedicated to an application of the theory described so far. Indeed, augmenting a structure with sorting allows better searching and compressing capabilities: for instance, a sorted list of integers supports fast membership queries and is more compressible than any of its permutations. This fact represents the core of some algorithms in string processing: the Barrows-Wheeler transform and the FM-index - which will be introduced later - are some examples of structures supporting fast pattern-matching queries in compressed space. All these techniques can be generalized to automata.

4.1 BWT on strings

The Barrows-Wheeler transform (shortly, BWT) is an invertible transformation of a given string with good compression properties. In the original definition of the BWT, the lexicographic order was used however, to be coherent with the rest of the thesis, we will adapt it to work with the co-lexicographic order.

As usual in sorting-based data structures in string processing, it is useful to mark the beginning of the string we want to process with a special character #, different from all the characters in the string and smaller than them.

Before introducing the BWT, it is necessary to define the notion of circular permutation and BW-Matrix.

Definition 4.1. Let $\alpha = a_1 a_2 \dots a_n \in \Sigma^*$ be a string. Then, for every $i \in \{1, \dots, n\}$, the *i-th circular permutation* of α is defined as follows

$$\alpha_i^{cyc} = a_{i+1} a_{i+2} \dots a_n a_1 a_2 \dots a_i$$

that is, the concatenation of the suffix of length $n - i$ with the prefix of length i.

Definition 4.2. The BW-Matrix of a string $\alpha = \#a_1 a_2 \dots a_n$ is the matrix $B \in \mathbb{M}^{(n+1)\times(n+1)}$ consisting of the co-lexicographically ordered circular permutations of the string α.

An example of the BW-Matrix, applied to the string *#abacadabra*, can be found in Figure 4.1.

F											L
a	b	r	a	c	a	d	a	b	r	a	$\#$
b	r	a	c	a	d	a	b	r	a	$\#$	a
d	a	b	r	a	$\#$	a	b	r	a	c	a
b	r	a	$\#$	a	b	r	a	c	a	d	a
c	a	d	a	b	r	a	$\#$	a	b	r	a
$\#$	a	b	r	a	c	a	d	a	b	r	a
r	a	c	a	d	a	b	r	a	$\#$	a	b
r	a	$\#$	a	b	r	a	c	a	d	a	b
a	d	a	b	r	a	$\#$	a	b	r	a	c
a	b	r	a	$\#$	a	b	r	a	c	a	d
a	c	a	d	a	b	r	a	$\#$	a	b	r
a	$\#$	a	b	r	a	c	a	d	a	b	r

Figure 4.1: The BW-Matrix of the string *#abracadabra*. The first and the last column are marked by letters F and L.

Now, we are ready to define the Barrow-Wheeler transform.

Definition 4.3 ([BW94])**.** The Barrows-Wheeler transform of a string $\alpha = \#a_1a_2\dots a_n$, denoted by BWT($\alpha$) is the string obtained by concatenating the characters in the first column of the BW-Matrix calculated on α, from top to bottom.

The main property of the BWT, which makes it invertible, is the so-called First-Last property (FL-property in short). It states that, for every character $a \in \Sigma$, the i-th occurrence of a in the first column of the BW-Matrix corresponds to the i-th occurrence of a in the last one: this means that they represent the same position of the occurrence of a in α.

Now we will see that this property suffices to reconstruct the string with just these two columns. Consider the BW-Matrix in Figure 4.1, built on the string α = *#abracadabra* (the letters F and L denote, respectively, the first and the last column of the BW-Matrix).

1. We begin with the first character of α, which is #. This # can be found at position 6 in F and position 1 in L.
2. The # is in the first row of L, so we move to the same row in F. Due to the circular permutation property, we know the letter a in this position is the a that comes after # in α.
3. This a is in the first row of F. According to the LF-property, it corresponds to the first occurrence of a in L, which is at position 2.
4. We then move to the second row of F. The b we find here is the one that follows the a we just visited in α.
5. We continue this process, repeating steps 3 and 4 for each subsequent character, until we return

to #.

We can note that, when we return to #, the path followed encodes the string α itself. Moreover, the last column of the BW-Matrix can be extracted directly from the BWT itself: it is sufficient to sort all its characters alphabetically. So, to retrieve the original string, the last column is not needed. Hence, we can say that the BWT(α) is a one-to-one encoding of the string α.

it is worth noting that, for every string α, the BWT(α) consist only of a permutation of the characters in α and, thus, it can be represented with just $n+1$ characters. For this reason, the BWT is the basis of a family of tools that can answer pattern-matching queries efficiently, needing very little extra data structures [NM07]. One of them, probably the most famous and widely used, is the FM-Index.

Theorem 4.4 ([FM00])**.** *The FM-index data structure, built on a given string $\alpha \in \Sigma^*$, is able to answer the following queries:*

- *Count the number occ of occurrences of any string β in α.*
- *Locate the occurrences of β in α.*
- *Extract any substring $\alpha[i, \ldots, i+l-1]$.*

All these queries run in $O(\log \sigma)$ per matched character. Moreover, this data structure occupies $n \log \sigma + o(n \log \sigma)$ bits in the worst case.

4.2 BWT on DFAs

Passing to finite state automata, things get more though: instead of characters, we have to encode states and transitions. First, we will see how to encode Wheeler DFAs, i.e. finite state machines whose co-lex width is 1, and then we will give an idea of how to extend it to a generic DFA. Consistently to the rest of the thesis, we will limit ourselves to deterministic automata, but all the theory can be adapted to non-deterministic machines.

4.2.1 Wheeler DFAs

In Wheeler DFAs, states can be totally ordered. In addition, Wheeler automata are input-consistent, that is, all the ingoing edges of one state are labeled with the same symbol. In order to define an invertible encoding of Wheeler DFAs, we need something similar to the LF-property, that is, something that guarantees ordering properties when moving on transitions labeled with the same symbol. Such a property is called *path-coherence*.

Theorem 4.5 (Path coherence, [GMS17])**.** *Let $\mathcal{D}$ be a Wheeler DFA and $\leq_{\mathcal{D}}$ be the (total) co-lex order on the states. In addition, let $\alpha \in \Sigma^*$ and U be an $\leq_{\mathcal{D}}$-interval of states. Then, the set U' of all states in Q that can be reached from U by following edges whose labels, when concatenated, yield to α, is still a $\leq_{\mathcal{D}}$-interval.*

As we will see, this property - along with the existence of a total order - is sufficient to extend the BWT to Wheeler DFAs.

Let $s = u_1 <_{\mathcal{D}} u_2 <_{\mathcal{D}} u_3 <_{\mathcal{D}} \cdots <_{\mathcal{D}} u_n$ the the co-lex ordered states. For every $i \in \{1, \ldots, n\}$, let l_i and k_i be, respectively, the number of incoming and outgoing edges of state u_i. Let `OUT_DEG` and `IN_DEG` be two bit-vectors of length $|Q| + |\delta|$ defined as follows:

$$\texttt{IN_DEG} = 0^{l_1}10^{l_2}1 \ldots 0^{l_n}1, \quad \texttt{OUT_DEG} = 0^{k_1}10^{k_2}1 \ldots 0^{k_n}1.$$

These sets represent the integers l_i, k_i in unary, for every i. Let $\texttt{OUT}_i$ denote the set of labels on the outgoing edges of u_i arranged in an arbitrary order, and let `OUT` denote the concatenation $\texttt{OUT} = \texttt{OUT}_1\texttt{OUT}_2\texttt{OUT}_3 \ldots \texttt{OUT}_n$. By construction, $|\texttt{OUT}_i| = k_i$ and there is a natural one-to-one correspondence between the 0's in `OUT_DEG` and the characters in `OUT`. As an example, for a hypothetical Wheeler DFA, the two arrays may look like the followings

`OUT_DEG` =	001	001	01	1	001	001	001	01
`OUT` =	ab	ac	b		ac	bc	bc	a

In addition, let $C[1, \ldots, \sigma]$ be the vector that stores, for every character $a \in \Sigma$, the number of edges whose label is smaller than a. Note that for every character $a \in \Sigma$, $C[a] \leq C[a+1]$ and $\lambda(u_j) = a$ if and only if $C[a] < \sum_{i \leq j} k_i \leq C[a+1]$.

Finally, let `FINAL` be a bit-vector of length n such that

$$\texttt{FINAL}[i] = \begin{cases} 1 & \text{if } u_i \in F \\ 0 & \text{otherwise.} \end{cases}.$$

These 4 arrays are sufficient to recover the transition function δ and, thus, represent a one-to-one encoding f DFA. To prove that, we need to use standard bit-vector queries `rank` and `select`:

- $\texttt{rank}_a(A, i)$ denotes the number of occurrences of a in the array $A[1 \ldots i]$.
- $\texttt{select}_a(A, i)$ denotes position of the i-th occurrence of a in A.

Now, we are ready to show that:

1. the out-degree k_i of state u_i is $\texttt{select}_1(\texttt{OUT_DEG}, i) - \texttt{select}_1(\texttt{OUT_DEG}, i-1)$. The labels in the edges leaving u_i are $\texttt{OUT}[w_i - k_i + 1, w_i]$, where $w_i = \texttt{select}_1(\texttt{OUT_DEG}, i) - i$ is the number of edges exiting the first i states;
2. ff state u_i has one outgoing edge labeled a, then the index j such that $\delta(u_i, a) = u_j$ can be computed as follows. Let w_i be defined as above and let $h_i = \texttt{rank}_a(\texttt{OUT}, w_i)$. Then the edge $(u_i, u_j, a) \in \delta$ is the h_i-th edge labeled a. Since there are $C[a]$ edges with a label smaller than a, we have $j = 1 + \texttt{rank}_1(\texttt{IN_DEG}, \texttt{select}_0(\texttt{IN_DEG}, h_i + C[a]))$.

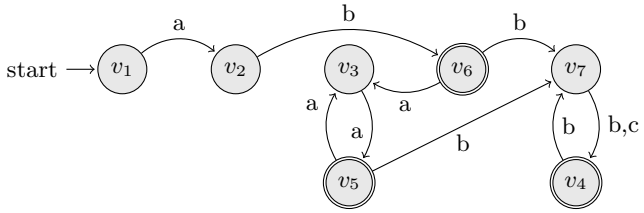
Thus, we can reconstruct δ by iterating the procedure in point 2 over all the states and all the labels.

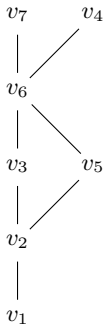
The space required to store the 4 arrays is $2(|Q| + |\delta|) + |\delta| \log \sigma + \sigma \log |\delta| + |Q|$ bits. Analogously to the string-BWT, we can augment this representation with lightweight data structures to support forward and backward traversing of the edges in $O(\log \sigma)$ time.

4.2.2 BWT on generic DFAs

We can extend the path coherence property to a generic DFA. However, contrarily to Wheeler DFAs, it will not regard the whole set of states Q, but only totally ordered subsets of Q. As we will show, this fact allows us to define the BWT on a non-Wheeler DFA $\mathcal{D}$ paying just an extra of $\log(\text{width}(\mathcal{D}))$ bits per transition.

Theorem 4.6 (Extended path coherence [CDPP23])**.** *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA, $\leq_{\mathcal{D}}$ be the co-lex order on $\mathcal{D}$, $\{Q_i\}_{i=1}^{p}$ be a $\leq_{\mathcal{D}}$-chain partition of $\mathcal{D}$, $\alpha \in \Sigma^*$, and U be a $\leq$-convex set of states. Then, the set U' of all states in Q that can be reached from U by following edges whose labels, when concatenated, yield α, is the disjoint union of p (possibly empty) sets $U'_1, \ldots, U'_p$, where $U'_i = U' \cap Q_i$ is $\leq_{Q_i}$-convex, for $i = 1, \ldots, p$.*





		CHAIN	1	0	0	0	1	0	0
		FINAL	0	0	0	1	1	1	0
		IN_DEG	1	01	001	001	01	01	0001
OUT_DEG	OUT		1	2	3	4	5	6	7
01	(1,a)	1		(1,a)					
01	(2,b)	2						(2,b)	
01	(2,a)	3					(2,a)		
01	(2,b)	4							(2,b)
001	(1,a),(2,b)	5			(1,a)				(2,b)
001	(1,a),(2,b)	6			(1,a)				(2,b)
001	(1,b),(1,c)	7				(1,b) (1,c)			

Figure 4.2: [CDPP23] A DFA $\mathcal{D}$ accepting $\mathcal{L} = ab(aa)^*(b(b+c))^*$, together with the Hasse diagram of its maximum co-lex order $\leq$ and the adjacency matrix of $\mathcal{D}$. In the following examples we consider the $\leq$-chain partition given by $Q_1 = \{v_1, v_2, v_3, v_4\}$, $Q_2 = \{v_5, v_6, v_7\}$. The adjacency matrix is sorted according to the total order $Q = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$. The two different shades of gray divide the edges by destination chain (either 1 or 2). Each edge is represented in this matrix as the pair (i, c), where i is the destination chain and $c \in \Sigma$ is the edge's label. This way of visualizing the adjacency matrix can be viewed as a two-dimensional representation of the automaton Burrows-Wheeler transform (aBWT, Definition 4.7). The aBWT can be linearized in five sequences, as shown here and in Example 4.8.

So, the BWT of a generic DFA $\mathcal{D}$ will depend on a $\leq_{\mathcal{D}}$-chain partition of Q. Since each chain in the partition contains totally ordered states, we can treat each of them similarly to the Wheeler case. To manage transitions between states in different chains, the OUT array needs to store both the character and the destination chain for each edge.

Definition 4.7 (automaton BWT [CDPP23])**.** Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. Let $\leq_{\mathcal{D}}$ be a co-lex order

on $\mathcal{D}$, and let $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq_{\mathcal{D}}$-chain partition of Q, where w.l.o.g. $s \in Q_1$. Let $\pi(v)$ be the unique map such that $v \in Q_{\pi(v)}$ and consider the total state order $Q = \{v_1, \ldots, v_n\}$ such that, for every $1 \leq i < j \leq n$, it holds [1] $\pi(v_i) < \pi(v_j) \vee (\pi(v_i) = \pi(v_j) \wedge v_i < v_j)$. The *automaton Burrows-Wheeler transform* $\texttt{aBWT}(\mathcal{D}, \leq, \{Q_i \mid 1 \leq i \leq p\})$ of $(\mathcal{D}, \leq, \{Q_i \mid 1 \leq i \leq p\})$ consists of the following sequences.

- $\texttt{CHAIN} \in \{0,1\}^n$ is such that the i-th bit is equal to 1 if and only if v_i is the first state of some chain Q_j.
- $\texttt{FINAL} \in \{0,1\}^n$ is such that the i-th bit is equal to 1 if and only if $v_i \in F$.
- $\texttt{IN_DEG} \in \{0,1\}^{e+n}$ stores the nodes' in-degrees in unary. More precisely, (1) IN_DEG contains exactly n characters equal to 1, (2) IN_DEG contains exactly e characters equal to 0, and (3) the number of zeros between the $(i-1)$-th character equal to one (or the beginning of the sequence if $i = 1$) and the i-th character equal to 1 yields the in-degree of v_i.
- $\texttt{OUT_DEG} \in \{0,1\}^{e+n}$ stores the nodes' out-degrees in unary. More precisely, (1) OUT_DEG contains exactly n characters equal to 1, (2) OUT_DEG contains exactly e characters equal to 0, and (3) the number of zeros between the $(i-1)$-th character equal to one (or the beginning of the sequence if $i = 1$) and the i-th character equal to 1 yields the out-degree of v_i.
- OUT stores the edges' labels and destination chains, as follows. Sort all edges (v_j, v_i, c) by their starting state v_j according to their index j. Edges originating from the same state are further sorted by their label c. Edges sharing the starting state and label are further sorted by destination node v_i. Then, OUT is obtained by concatenating the pairs $(\pi(v_i), c)$ for all edges (v_j, v_i, c) sorted in this order.

An example of the aBWT is provided in Figure 4.2: the states are sorted in a total order which agrees with the co-lex order $\leq_{\mathcal{D}}$ on pairs whose elements belong to the same class of the partition $\{Q_i \mid 1 \leq i \leq p\}$ and the transition function is represented as adjacency matrix. Additionally to the arrays IN_DEG, OUT_DEG, FINAL, and the augmented OUT (already used in the Wheeler case), there is an array that encodes chain borders (CHAIN). The following Example shows the resulting aBWT of the automaton in Figure 4.2.

Example 4.8. The aBWT of $(\mathcal{D}, \leq, \{Q_i \mid 1 \leq i \leq 2\})$ in Figure 4.2 consists of the following sequences:

- $\texttt{CHAIN} = 1000100$.
- $\texttt{FINAL} = 0001110$.
- $\texttt{IN_DEG} = 10100100101010001$.
- $\texttt{OUT_DEG} = 01010101001001001$.
- $\texttt{OUT} = (1,a)(2,b)(2,a)(2,b)(1,a)(2,b)(1,a)(2,b)(1,b)(1,c)$.

It can be proven that the aBWT is a one-to-one encoding of DFAs, that is, we can retrieve the whole transition function from the arrays we defined. Since the algorithm involves various intermediate theo-

[1] Notice the overload on symbol $\leq$, also used to indicate the co-lex order among states.

rems and more sophisticated data structures, we do not describe it here. If interested, you can find it in [CDPP23].

The whole `aBWT`, that consists of the 5 arrays described above, can be stored in $2|Q| + 2(|Q| + |\delta|) + |\delta|(\log p + \log \sigma)$ bits. Analogously to the Wheeler case, we can augment this representation with some extra data structures to support forward and backward traversing of the edges in $O(p^2 (\log p + \log \sigma))$ time.

5

Conclusion

In this thesis, we described the theory of co-lexicographically ordered automata and explored its implications on regular languages. We have established that the co-lex width of an automaton/language is a fundamental complexity measure for both theoretical and practical aspects.

Although the Hasse automaton represents an important step towards understanding the relation between language structure and automata orderability, it cannot be used to efficiently encode a regular language, due to the possible exponential number of states it might have. Thus, it is worth exploring new ways to find the right trade-off between minimizing the width and the number of states. The little contribution that this thesis provides may help in this direction. Given a regular language $\mathcal{L}$, we can represent the states of the minimum-size automaton as non-dense intervals of $(\mathrm{Pref}(\mathcal{L}), \preceq)$. Refining the relation $\approx_{\mathcal{L}}$ with a $\mathcal{L}$-convex equivalence leads to cut those intervals into smaller ones. This suggests that every $\mathcal{L}$-convex relation refining $\approx_{\mathcal{L}}$ can be seen as a wisely chosen sequence of cuts of the intervals representing the states of the minimum-size automaton.

Deciding the right points where to cut is not a trivial task. A possible approach, that has not been explored yet, involves reducing the aforementioned problem to an instance of min-cost flow. It's known that max-flow algorithms can be adapted to calculate a minimum chain partition of a partial order [FF62] and, in our case, the same strategies can be adapted to choose the intervals to cut: if we are able to create the right graph, the amount of flow we require through it will bound the final width, while the cost of the edges will represent the number of states the cuts will produce. Minimizing the total cost, that is, minimizing the number of states would lead to one width-p equivalence with the minimum index.

Bibliography

[ADPP21] Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Wheeler languages. *Information and Computation*, 2021.

[ALSU06] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

[BCC+23] Ruben Becker, Manuel Cáceres, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Francisco Olivares, and Nicola Prezza. Sorting Finite Automata via Partition Refinement. In *31st Annual European Symposium on Algorithms (ESA 2023)*, 2023.

[BW94] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. In *Technical report, DIGITAL SRC RESEARCH REPORT*, 1994.

[CDPP23] Nicola Cotumaccio, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Colexicographically ordering automata and regular languages - part i. *J. ACM*, 2023.

[Cho56] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 1956.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Cot23] Nicola Cotumaccio. Prefix Sorting DFAs: A Recursive Algorithm. In *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, 2023.

[Dil50] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 1950.

[FF62] L.R. Ford and D.R. Fulkerson. Flows in networks. *Princeton University Press*, 1962.

[FM00] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, 2000.

[Gal94] Fred Galvin. A proof of dilworth's chain decomposition theorem. *The American Mathematical Monthly*, 1994.

[GMS17] Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theoretical Computer Science*, 2017.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 2nd edition.* ACM Press, 2001.

[KP22] Shimon Kogan and Merav Parter. Beating Matrix Multiplication for $n^{1/3}$-Directed Shortcuts. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, 2022.

[MA03] S. Mitra and T. Acharya. *Data Mining: Multimedia, Soft Computing and Bioinformatics.* Wiley- Interscience, 2003.

[MPPR24] Giovanni Manzini, Alberto Policriti, Nicola Prezza, and Brian Riccardi. The Rational Construction of a Wheeler DFA. In *35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024)*, Leibniz International Proceedings in Informatics (LIPIcs), 2024.

[MPR05] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted automata in text and speech processing. *arXiv, abs/cs/0503077*, 2005.

[NM07] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 2007.

[RMV14] Prashantkumar M. Rathod, Nilesh Marathe, and Amarsinh V. Vidhate. A survey on finite automata-based pattern matching techniques for network intrusion detection systems (nids). In *2014 International Conference on Advances in Electronics Computers and Communications*, 2014.