DATA COMPRESSION MEETS AUTOMATA THEORY

by

Nicola Cotumaccio

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
January 2024

Contents

Abstract

We introduce a new paradigm in graph compression and formal language theory. We show that the ideas behind some of the most important data structures for compressing and indexing strings — such as the suffix array, the Burrows-Wheeler Transform and the FM-index — are much more general and provide a new approach to studying automata and regular languages, which retrospectively explains the impact of these data structures. We classify *all* automata and *all* regular languages by their *propensity to be sorted*. Our classification represents a useful parameterization *simultaneously* for diverse automata-related measures: (i) the encoding bit-complexity of automata/labeled graphs, (ii) the complexity of operations on regular languages (e.g. membership) and on labeled graphs (e.g. pattern matching), (iii) the complexity of NFA determinization by the powerset-construction algorithm. To the best of our knowledge, ours is the only parameterization of automata/labeled graphs capturing *simultaneously* all these aspects. We show that our parameterization has deep and unexpected consequences both in data compression (encoding, pattern matching) and in automata theory (nondeterminism, entanglement, minimization).

Acknowledgements

My PhD started shortly before the outbreak of the COVID-19 pandemic. Nonetheless, I had the opportunity to conduct research activity both in Italy and Canada. This was possible thanks to my supervisors Travis Gagie, Nicola Prezza and Catia Trubiani. They supported me in many different ways: they proved to be consistent and reliable people, who quickly introduced me to high-level research and helped me in multiple situations, from bureaucratic issues to housing. Over the years, I developed a solid and personal relationship with all of them.

I am grateful to Dante Labs for funding my scholarship at GSSI.

I would like to thank GSSI for providing a trustworthy and collaborative environment. In particular, I am grateful for the administrative support received for the Joint PhD arrangements with Dalhousie University. Analogously, I would like to thank Dalhousie University for accepting the logistic challenge of handling a Joint PhD student in Computer Science for the first time.

I would like to thank Giovanna D'Agostino and Alberto Policriti for guiding me during my bachelor's degree and my master's degree at University of Udine, and I am happy that we have had a chance to collaborate during my PhD.

Many thanks to my external reviewers (Kunihiko Sadakane and Alexandru Tomescu) and the other members of my PhD thesis committee (Pierluigi Crescenzi, Roberto Grossi and Meng He) for their availability and flexibility. Thanks to Meng He and Norbert Zeh for their support in the thesis proposal defense at Dalhousie. Thanks to Pierluigi Crescenzi, Michele Flammini, Michael McAllister and Patrizio Pelliccione for their help at GSSI and Dalhousie University.

Lastly, I hope that over the years I have suitably expressed my gratitude to all the people who directly or indirectly contributed to my PhD (many of whom are not mentioned here).

Chapter 1

Introduction

One of the biggest challenges of computer science is handling the explosive growth of data. While technology is improving our ability to store data, it is paramount to rely on compression algorithms. But compression is not sufficient: we also need to efficiently retrieve and query data. The classical data structures required to query data impose a significant space overhead. A typical field in which we need to both compress and process data is bioinformatics. The DNA of a human genome consists of about 3.3 billion bases. Each base can be of one of the following four types: adenine (A), cytosine (C), guanine (G) and thymine (T). If we store each base using 2 bits, we need almost 800 megabytes. In bioinformatics applications — such as sequence assembly — we also need to efficiently process and query genomic data. To this end, a typical solution is to build the suffix tree of the genome. However, storing the suffix tree requires at least 10 bytes per base, and so we need more than 30 gigabytes [96].

The field of *compressed data structures* aims to *compress* data in such a way that it is possible to *efficiently* solve queries *without decompressing the data.* The nineties marked the beginning of a golden age for compressed data structures, a time of prosperity that went beyond any reasonable expectation and keeps having a strong impact on current research. Some new conferences on the topic (DCC, CPM, SPIRE) were established. New ideas for storing fundamental data structures — such as trees [105] — proved that space-time tradeoffs in computer science can often be overcome: it is possible to design procedures that are both time-efficient and space-efficient. More generally, the new techniques inspired by compressed data structures have influenced the field of algorithms as a whole.

One of the striking features of compressed data structures is that they combine elegance and practicality, thus removing the gap between the beauty of theory and the pervasiveness of applications. In 1990 Udi Manber and Gene Myers invented the suffix array [88], in 1994 Michael Burrows and David Wheeler invented the Burrows-Wheeler

Transform [27], and in 2000 Paolo Ferragina and Giovanni Manzini invented the FM-index [56]. These data structures solve theoretical problems which lead to surprising applications in bioinformatics, as witnessed by the 2022 ACM Paris Kanellakis Theory and Practice Award awarded to Micheal Burrows, Paolo Ferragina and Giovanni Manzini.

The suffix array, the Burrows-Wheeler Transform and the FM-index can handle unstructured data, that is, strings and texts. However, the World Wide Web relies on *hypertext*, in which there are relationships between data. These relationships are modeled through *graphs*: for example, graphs can be used to describe the behavior of social networks. Graphs can also be used to model the idea of computation itself: the field of *automata theory* can be traced back to the fifties and it is related to Turing machines. The classical approach to automata theory — which establishes a deep connection between automata, regular expressions and monoid theory — focuses on the study of *regular languages*. Most classical results that are presented in an introductory course on automata theory (such as Kleene's theorem, the Myhill-Nerode theorem and Hopcroft's algorithm for minimizing a DFA [74]) date back to more than fifty years ago.

In this thesis, we present a new paradigm in automata theory, based on the same flavor of the classical results in the field. The suffix array, the Burrows-Wheeler Transform and the FM-index are explicitly or implicitly based on the idea of sorting. We show that the idea behind these data structures is much more general and provides a new approach to studying automata and regular languages, which retrospectively explains the impact of these data structures. We will classify automata and regular languages by their *propensity to be sorted*. Our classification represents a useful parameterization *simultaneously* for diverse automata-related measures:

1. the encoding bit-complexity of automata/labeled graphs.

2. the complexity of operations on regular languages (e.g. membership) and on labeled graphs (e.g. pattern matching).

3. the complexity of NFA determinization by the powerset-construction algorithm.

To the best of our knowledge, ours is the only parameterization of automata/labeled graphs capturing *simultaneously* all these aspects. We will show that our parameterization has unexpected consequences both in data compression and in formal language theory.

1.1 Graph Compression

Graph compression is a vast topic that has been extensively studied in the literature (see for example the survey [16]). Most solutions discussed below consider the unlabeled case; a compressor for a labeled graph can be obtained by compressing the unlabeled version of the given labeled graph and storing the labels separately using $\lceil \log \sigma \rceil$ additional bits per edge (σ is the alphabet's size).

Existing results studying worst-case information-theoretic lower bounds of graph encodings can be used as the reference base for the compression methods discussed below. First of all, note that the worst-case information-theoretic number of bits needed to represent a directed graph with m edges and n vertices is $\log \binom{n^2}{m} = m \log(n^2/m) + \Theta(m)$, that is, $\log(n^2/m) + \Theta(1)$ bits per edge [96]. The same lower bound holds on undirected graphs up to a constant additive number of bits per edge.

Other useful bounds (on automata) are studied in the recent work of Chakraborty et al. [29]. In that paper, the authors present a succinct encoding for DFAs using $\log \sigma + \log n + 1.45$ bits per transition (n is the number of states) and provide worst-case lower bounds as a function of the number of states: in the worst case, DFAs cannot be encoded using less than $(\sigma - 1) \log n + O(1)$ bits per state and NFAs cannot be encoded in less than $\sigma n + 1$ bits per state. The same paper provides encodings matching these lower bounds up to low-order terms.

For our purposes, it is useful to divide graph compression strategies into *general graph compressors* and *compact encodings* for particular graph classes. The former compressors work on arbitrary graphs and exploit sources of redundancy in the graph's topology in order to achieve a compact representation. Compressors falling into this category include (this list is by no means complete, see [16] for further references) K^2 trees on the graph's adjacency matrix [24], straight-line programs on the graph's adjacency list representation [33], and context-free graph grammars [48]. A shared feature of these compressors is that, in general, they do not provide guarantees on the number of bits per edge that will be used to encode an arbitrary graph (for example, a guarantee linked with a particular topology or graph parameter such as the ones discussed below); the compression parameter associated with the graph is simply the size of the compressed representation itself. This makes these techniques not directly comparable with our approach (if not experimentally).

Techniques exploiting particular graph topologies or structural parameters of the graph to achieve more compact encodings are closer to our parameterized approach, bearing in mind that also in this case a direct comparison is not always possible in the absence of known relations between our parameter and the graph parameters mentioned below. A first example of such a parameter (on undirected graphs) is represented by *boxicity* [107], that is, the minimum number b of dimensions such that the graph's edges correspond to the intersections of b-dimensional axes-parallel boxes (the case $b = 1$ corresponds to interval graphs). Any graph with boxicity b can be represented naively using $O(b)$ words per vertex (that is, storing each vertex as a b-dimensional box), regardless of the fact that its number of edges could be quadratic in the number of vertices (even in the interval graph case). Similar results are known for graphs of small clique-width/bandwidth/treedepth/treewidth [78, 79, 52] and bounded genus [43]; any graph from these graph families can be encoded in $O(k)$ bits per vertex, where k is the graph parameter under consideration. Similarly, posets (transitively-closed DAGs) of width w can be encoded succinctly using $2w + o(w)$ bits per vertex [120]. While the above-mentioned methods focus on particular graph parameters, another popular approach is to develop ad-hoc compact encodings for particular graph topologies. Separable graphs (graphs admitting a separator of size $O(n^c)$ breaking the graph into components of size αn for some $c < 1$ and $\alpha < 1$) allow for an encoding using $O(1)$ bits per vertex [18]. This class includes planar graphs — admitting also an encoding of 4 bits per vertex [58] — and trees — admitting an encoding of 2 bits per vertex (e.g. a simple balanced-parenthesis representation) and an encoding of $1 + o(1)$ bits per vertex when every internal node has exactly two children [77]. Circular-arc graphs (a class including interval graphs) of maximum degree Δ can be encoded in $\log \Delta + O(1)$ bits per vertex, and this bound is asymptotically tight [30]; in the same paper, the authors show that circular-arc graphs with chromatic number χ admit an encoding using $\chi + o(\chi)$ bits per vertex.

1.2 Regular Expression Matching and String Matching on Labeled Graphs

"Regular expression matching" (REM) refers to the problem of determining whether there exist substrings of an input string that can be derived from an input regular expression. This problem generalizes that of determining membership of a string to a regular language, and it finds important applications which include text processing utilities (where regular expressions are used to define search patterns), computer networks (see [119]), and databases (see [41]). A closely-related problem is that of "exact string matching on labeled graphs" (SMLG): find which paths of an edge-labeled graph match (without edits) a given string (see [49]). This problem arises naturally in several fields, such as bioinformatics [10, 114], where the *pan-genome* is a labeled graph capturing the genetic variation within a species (and pattern matching queries are used to match an individual's genome on this graph), and graph databases [7]. Since NFAs can be viewed as labeled graphs, it is not surprising that existing lower and upper bounds for both problems have been derived using the same set of techniques.

Backurs and Indyk in [9] carry out a detailed study of the complexity of the REM problem as a function of the expression's structure for all regular expressions of depth up to 3. For each case (there are in total 36 ways of combining the regular operators "|", Kleene plus, Kleene star, and concatenation up to depth 3), they either derive a sub-quadratic upper bound (where *quadratic* means the string's length multiplied by the regular expression's size) or a quadratic lower bound conditioned on the Strong Exponential Time Hypothesis [76] or on the Orthogonal Vectors conjecture [23]. Note that this classification does not capture regular expressions of arbitrary depth. Similarly, Equi et al. [49] establish lower and upper bounds for the SMLG problem, even in the scenario where one is allowed to pre-process the graph in polynomial time [50] (that is, building a graph index); their work represents a *complete* classification of the graph topologies admitting either sub-quadratic pattern matching algorithms or quadratic lower bounds (obtained assuming the Orthogonal Vectors conjecture); in this context, *quadratic* means proportional to the string's length times the graph's size. In all these works, as well as in further papers refining these analyses by providing finer lower bounds or better upper bounds for particular cases [22, 67, 65, 68, 15], the problem's complexity is studied by cases and does not depend on a parameter of the language or a parameter of the graph.

Techniques parameterizing the problem's complexity on a graph parameter do exist in the literature, and are closer to the spirit of our work. These parameters include (these works consider the SMLG problem) the size of the labeled direct product [106], the output size of powerset construction [97], and a generalization of DAGs called k-funnels [28]. Like in our setting, in all these cases quadratic query complexity is obtained in the worst case (on graphs maximizing the parameter under consideration)

1.3 NFA Determinization and Existing Subregular Classifications

An extensive and detailed classification of the complexity of the powerset construction algorithm on families of subregular languages is carried out in [19]. That study proves that for the most popular and studied classes of subregular languages — including (but not limited to) star-free [92], ordered [112], comet [25] and suffix/prefix/infix-closed languages — the output of the powerset construction is exponential in the size of the input NFA: for all the mentioned families, the resulting DFA may have at least 2^{n-1} states in the worst case, where n is the number of states of the input NFA. Previously-known families with a sub-exponential upper bound include unary regular languages, with a bound of $e^{\Theta(\sqrt{n \ln n})}$ states [32] and the family of finite languages over alphabet of size σ, with a bound of $O(\sigma^{\frac{n}{\log_2 \sigma + 1}})$ states [109]. In this context, our nondeterministic hierarchy of subregular languages represents a more complete classification than the above-mentioned classes (since it captures all regular languages).

Interestingly, [19] shows that the class of ordered automata [112] — automata admitting a total states' order that must propagate through pairs of equally-labeled edges — does have a worst-case exponential-output powerset construction. Since in our work we show that the powerset construction builds a small-size DFA on bounded-width automata, this fact shows that the small difference between simply imposing an order on the states which is consistent with the transition relation (ordered automata) and linking this property with a fixed order of the underlying alphabet (our framework), does have significant practical consequences in terms of deterministic state complexity.

As far as other parameterizations of powerset construction are involved, we are aware of only one previous attempt in the literature: the notion of *automata width* introduced in [86]. Intuitively, given an NFA $\mathcal{N}$ the width of $\mathcal{N}$ as defined in [86] is the maximum number of $\mathcal{N}$'s states one needs to keep track of simultaneously while looking for an accepting path for some input word (for the word maximizing such quantity). By its very definition, this quantity is directly linked to the output's size of powerset construction.

Further notable classifications of subregular languages include the star-height hierarchy [46] (capturing all regular languages) and the Straubing-Thérien hierarchy [116, 117] (capturing the star-free languages). To the best of our knowledge, these classifications do not lead to useful parameterizations for the automata/graph problems considered in this thesis.

1.4 Our Approach: The Power of Sorting

Equipping the domain of a structure with some kind of order is often a fruitful move performed in both computer science and mathematics. An order provides direct access to data or domain elements and sometimes allows tackling problems otherwise too computationally difficult to cope with. For example, in descriptive complexity it is not known how to logically capture the class P in general, while this can be done on ordered structures [85]. In general, the price to be paid when requiring/imposing an order, is a — sometimes significant — restriction of the class of structures to which subsequent results refer. If we do not wish to pay such a price, a *partial* order can be a natural alternative. Then, the "farther" the partial order is from a total order, the less powerful will be the applications of the established results. In other words, the "distance" from a total order of the partial order at hand becomes a *measure* of the extent to which we have been able to "tame" the class of structures under consideration.

Partial orders and automata have already met and attracted attention because of their relation with logical, combinatorial, and algebraic characterizations of languages. In the literature (see, among many others, [26, 110, 90]) a partially-ordered NFA is an automaton where the transition relation induces a partial order on the set of its states. Here we pursue a different approach, closer to the one given in [112].

Our starting point is a work by Gagie et al. [61], presenting a simple and unified perspective on several algorithmic techniques related to *suffix sorting* (in particular, to the Burrows-Wheeler transform). The general idea is to enforce and exploit a total order among the states of a given automaton, induced by an *a priori* fixed order of its underlying alphabet which propagates through the automaton's transition relation. The resulting automata, called *Wheeler automata*, admit efficient data structures for solving string matching on the automaton's paths and enable a representation of the automaton in space proportional to that of the edges' labels — as well as enabling more advanced compression mechanisms, see [6, 103]. This is in contrast with the fact that general graphs require a logarithmic (in the graph's size) number of bits per edge to be represented, as well as with recent results showing that in general, regular expression matching and string matching on labeled graphs can not be solved in subquadratic time, unless the strong exponential time hypothesis is false [9, 50, 49, 66, 102]. Wheeler languages — i.e. languages accepted by Wheeler automata — form an interesting class of subregular languages, where determinization becomes computationally easy (polynomial). As was to be expected, however, requiring the existence of a *total* Wheeler order over an automaton comes with a price. Not all automata enjoy the Wheeler property, and languages recognized by Wheeler automata constitute a relatively small class: a subclass of star-free languages [5].

1.5 Our Contribution

In this thesis, we present a new paradigm, based on (partially) sorting automata. We lay out a theory that encompasses *all* automata and *all* regular languages, with applications to both data compression and formal language theory. This thesis is the first work in which the results obtained in the last years are presented in a systematic and comprehensive fashion (see Figure 1.1). All results in this thesis appeared in journals or conference proceedings; some results have been slightly modified or rearranged for the purpose of designing a consistent and coherent presentation. Chapter 4 is based on [40, 38]. Chapter 5 is based on [40, 38, 3]. Chapter 6 is based on [35]. Chapter 7 is based on [36]. Chapter 8 is based on [34, 39]. In addition, before the PhD defence the results on generalized automata sketched in Section 9.1 were also accepted for publication [37].

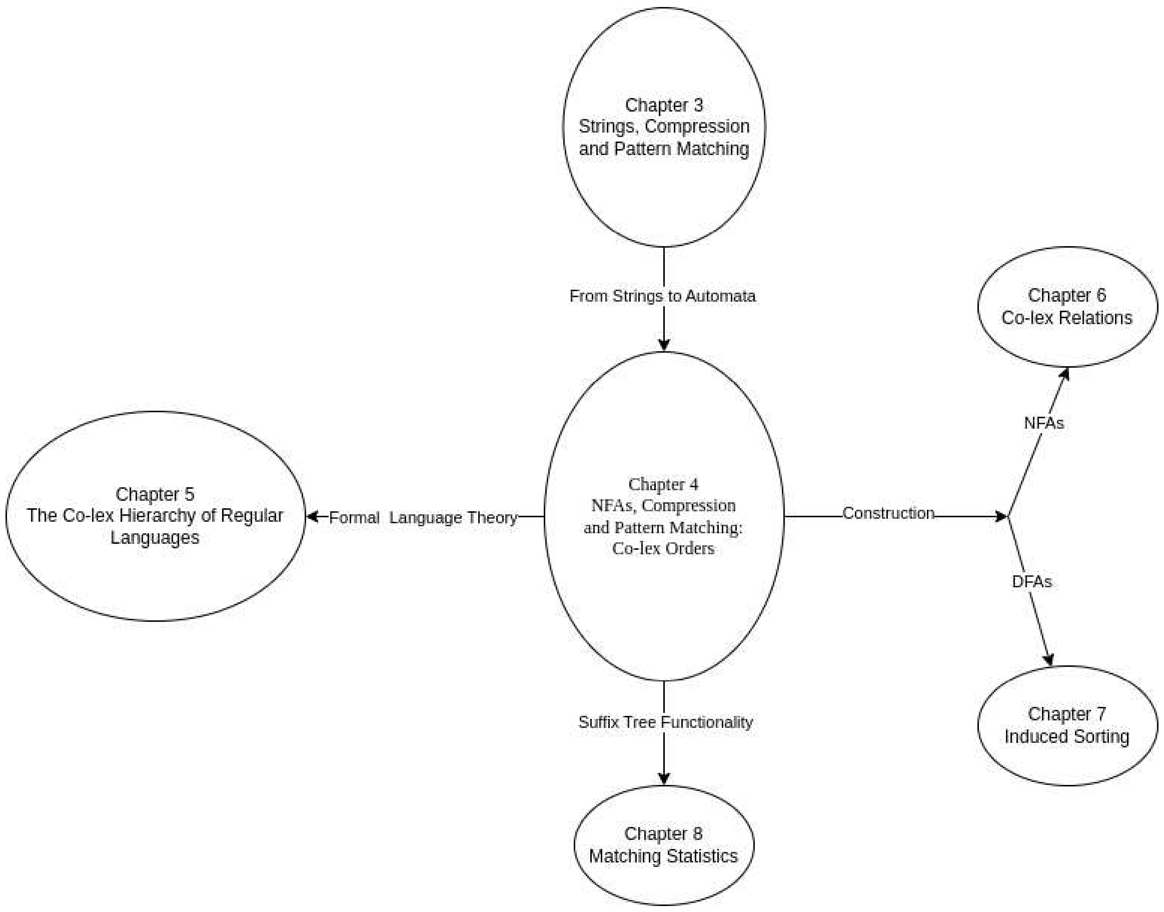


Figure 1.1: Organization of the thesis

- In Chapter 2, we introduce our notation and we present some classical results on compressed data structures that will be used in the subsequent chapters.

- In Chapter 3, we describe the suffix array, the Burrows-Wheeler Transform and the FM-index of a string and we explain how they can be used to compress a string while supporting efficient pattern matching (Problem 1). We do not aim to provide a comprehensive discussion of the literature on the topic. Our presentation deviates from the textbook approach (see for example the definition of the Burrows-Wheeler Transform, Definition 3.4), because we want to outline the properties of these data structures that can be extended to automata.

- In Chapter 4, we implement our approach by defining *co-lex orders*, which partially sort the states of an automaton (Definition 4.1). The key parameter is the *width* of a partial order, which measures how far the partial order is from being a total order. Co-lex orders can be seen as an extension of suffix arrays to automata. The setting is more complicated, though: an automaton may admit several co-lex orders, so we need a measure of optimality (Definition 4.6). We will see that, if we consider *deterministic* automata, then there exists a single optimal co-lex order (Lemma 4.12), the *maximum co-lex order*, that can be determined in polynomial time (Lemma 4.15); on arbitrary non-deterministic automata, the problem is NP-hard. Next, we extend the Burrows-Wheeler Transform (Definition 4.28 and Lemma 4.50) and the FM-index (Theorem 4.47) to arbitrary automata, thus showing how to compress automata while supporting efficient pattern matching queries.

- In Chapter 5, we explore the applications of co-lex orders to formal language theory, thus defining the deterministic and non-deterministic widths of a regular language (Definition 5.1). We show that co-lex orders lead to a surprising parametrization of the powerset construction (Theorem 5.4), implying that problems that are computationally difficult on non-deterministic automata but tractable on deterministic automata are fixed-parameter tractable with respect to the widths of the considered automata. In the remainder of the chapter, we study the deterministic width. We prove that the deterministic width can be obtained by studying the *entanglement* of the minimum DFA recognizing the language (Theorem 5.24) and thus can be effectively computed (Theorem 5.30). To this end, we introduce the so-called *Hasse automaton* (Definition 5.23), which requires a deep understanding of the structure of convex partition (see Appendix A). Next, we study the classical problem of minimization in the context of co-lexicographically ordered language and we prove a full Myhill-Nerode theorem (Theorem 5.40). In particular, the convex structure of Wheeler languages implies that Wheeler DFAs can be minimized in linear time, without resorting to Hopcroft's algorithm (Theorem 5.42). In Section 5.6 we investigate the relationship between our hierarchy and star-free languages.

- In Chapter 6, we overcome the hardness of determining an optimal co-lex order on arbitrary non-deterministic automata (see Chapter 4). Intuitively, we prove that the hardness is only due to some states that are inherently equivalent from a pattern matching perspective, which cannot be captured by partial orders. As a consequence, we switch from partial orders to relations: we define *co-lex relations* (Definition 6.1), we prove that on arbitrary NFAs there is always one single optimal co-lex relation (Lemma 6.8), the *maximum co-lex relation*, which can be computed in polynomial time (Theorem 6.17). The maximum co-lex relation allows building a quotient automaton (Definition 6.22) which can be used to improve the bounds of the FM-index that we present in Chapter 4 (Theorem 6.32).

- In Chapter 7, we consider the problem of building the maximum co-lex order of a deterministic finite automata. In Chapter 4, we show that the maximum co-lex order of a DFA can be determined in polynomial time, namely in $O(|\delta|^2 + |Q|^{5/2})$ time (Lemma 4.15), where $|Q|$ is the number of states and $|\delta|$ is the number of transitions. We significantly improve this bound by proposing an $O(|\delta| + |Q|^2)$ time algorithm inspired by Farach's algorithm for building the suffix tree of a string (Theorem 7.1). To this end, we show how to build the *min/max-partition* of a DFA efficiently; intuitively, we show how to quickly sort the minimum and the maximum string reaching each state. We design a rather intricate algorithm that recursively builds the min/max partition of a smaller DFA.

- In Chapter 8 we show how to compute matching statistics on Wheeler DFAs (Theorem 8.1); intuitively, we show how to solve a popular variant of the pattern matching problem on Wheeler DFAs. This is possible by extending the notion of *longest common prefix array* from strings to Wheeler DFAs (Definition 8.3). If the suffix array of a string is augmented with the longest common prefix array of the string and some additional data structures, we can retrieve the functionality of a *suffix tree*, a powerful and versatile data structure in computer science. As a consequence, we managed to make remarkable advancements toward building the suffix tree of an automaton. We also show how to sample the longest common prefix array of a DFA so that it can it can be accessed efficiently (Theorem 8.5), and we present an application to variable-order de Bruijn graphs, which are popular in bioinformatics to perform Eulerian sequence assembly (Theorem 8.8).

- In Chapter 9, we present some additional partial results that are still unpublished. We also outline open problems and future research directions.

Chapter 2

Notation and Preliminaries

In this chapter we first present all basic definitions required in order to follow the thesis. We also recall and adapt some classical results that we will use in the next chapters (see Section 2.5).

2.1 Sequences and Basics

Let Σ be a finite alphabet and let Σ^* be the set of all finite sequences (also called *words* or *strings*) on Σ, with ε being the empty sequence.

As customary, we denote by Σ^* the set of all finite strings over the alphabet Σ, and we denote by Σ^ω the set of all coutably infinite strings over the alphabet Σ. If $n \geq 0$, let $\Sigma^n \subseteq \Sigma^*$ be the set of all strings of length n. If $\alpha \in \Sigma^* \cup \Sigma^\omega$, we denote by $\alpha[i]$ the i-th character of α; moreover, $\alpha[i,j] = \alpha[i]\alpha[i+1]\dots\alpha[j-1]\alpha[j]$ if $j \geq i$, and $\alpha[i,j] = \varepsilon$ if $j < i$. We write $\beta \dashv \alpha$ if $\alpha, \beta \in \Sigma^*$ and β is a suffix of α.

Throughout our thesis, we assume that there is a fixed total order $\preceq$ on Σ (in our examples, the alphabetical order). The special symbol $\# \notin \Sigma$ is considered smaller than any element in Σ. We extend $\preceq$ to words in Σ^* either *lexicographically* or *co-lexicographically*. The lexicographic order on Σ^* is the standard dictionary order; the co-lexicographic order on Σ^* is the order in which a string α is co-lexicographically smaller than a string β if and only if the reverse string α^R is lexicographically smaller than the reverse string β^R. In other words, up to reversing the strings, the lexicographic order and the co-lexicographic order are the same order. For notational convenience, it will be expedient to use both orders. In Chapter 3 we will outline the interplay between these two orders (see in particular Section 3.4). In Chapters 4, 5 abd 6 we will use the co-lexicographic order, and in Chapters 7 and 8 we will use the lexicographic order.

If a, b are natural numbers, with $a \leq b$, we write $[a,b]$ for the integer set $\{a, a+1, \dots, b\}$. If $b < a$, then $[a,b] = \emptyset$. All logarithms are in base 2.

2.2 Finite Automata

A *non-deterministic* finite automaton (an NFA) accepting strings in Σ^* is a tuple $\mathcal{N} = (Q, s, \delta, F)$ where Q is a finite set of states, s is a the initial state, $\delta(\cdot,\cdot) : Q \times \Sigma \to \mathcal{P}ow(Q)$ is the transition function (where $\mathcal{P}ow(Q)$ is the set of all subsets of Q), and $F \subseteq Q$ is the set of final states. We write $Q_{\mathcal{N}}, s_{\mathcal{N}}, \delta_{\mathcal{N}}, F_{\mathcal{N}}$ when the automaton $\mathcal{N}$ is not clear from the context and, conversely, if the automaton is clear from the context we will not explicitly say that s is the initial state, δ is the transition function and F is the set of final states.

With $|\delta|$ we denote the cardinality of δ when seen as a set of triples over $Q \times Q \times \Sigma$. In other words, $|\delta| = |\{(u, v, a) \mid v \in \delta(u, a),\ u, v \in Q, a \in \Sigma\}$. In fact, in our results (especially the data-structure related ones) we will often treat NFAs as edge-labeled graphs having as set of nodes Q and set of edges $E = \{(u, v, a) \mid v \in \delta(u, a),\ u, v \in Q, a \in \Sigma\}$.

As customary, we extend δ to operate on strings as follows: for all $u \in Q, a \in \Sigma$, and $\alpha \in \Sigma^*$:

$$\delta(u, \varepsilon) = \{u\}, \qquad \delta(u, \alpha a) = \bigcup_{v \in \delta(u,\alpha)} \delta(v, a).$$

We denote by $\mathcal{L}(\mathcal{N}) = \{\alpha \in \Sigma^* : \delta(s, \alpha) \cap F \neq \emptyset\}$ the language accepted by the automaton $\mathcal{N}$. We say that two automata are *equivalent* if they accept the same language.

We assume, without loss of generality, that all states in our 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 (unique) initial state. Hence, the collection of prefixes of words accepted by $\mathcal{N}$, $\text{Pref}(\mathcal{L}(\mathcal{N}))$, will consist of the set of words that can be read on $\mathcal{N}$ starting from the initial state.

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

A *deterministic* finite automaton (a DFA), is an NFA $\mathcal{D}$ where $|\delta(u, a)| \leq 1$, for any $u \in Q$ and $a \in \Sigma$. If the automaton is deterministic we write $\delta(u, \alpha) = v$ for the unique v such that $\delta(u, \alpha) = \{v\}$ (if defined: we are not assuming a DFA to be complete).

Let $\mathcal{L} \subseteq \Sigma^*$ be a language. An equivalence relation $\sim$ on $\text{Pref}(\mathcal{L})$ is *right-invariant* if for every $\alpha, \beta \in \text{Pref}(\mathcal{L})$ such that $\alpha \sim \beta$ and for every $a \in \Sigma$ it holds $\alpha a \in \text{Pref}(\mathcal{L})$ iff $\beta a \in \text{Pref}(\mathcal{L})$ and, if so, $\alpha a \sim \beta a$. We will extensively use the *Myhill-Nerode equivalence* induced by $\mathcal{L}$, namely, the right-invariant equivalence relation $\equiv_\mathcal{L}$ on $\text{Pref}(\mathcal{L})$ such that for every $\alpha, \beta \in \text{Pref}(\mathcal{L})$ it holds:

$$\alpha \equiv_\mathcal{L} \beta \iff \{\gamma \in \Sigma^* \mid \alpha\gamma \in \mathcal{L}\} = \{\gamma \in \Sigma^* \mid \beta\gamma \in \mathcal{L}\}.$$

We denote by $\mathcal{D}_\mathcal{L}$ the minimum (with respect to state-cardinality) deterministic automaton recognizing a regular language $\mathcal{L}$.

If $u \in Q$, then $\lambda(u)$ denotes the set of labels of edges entering u, except when $u = s$ when we also add $\# \notin \Sigma$ to $\lambda(s)$, with $\# \prec e$ for all $e \in \Sigma$ (see e.g. Figure 4.1 where $s = 0$, $\lambda(s) = \{\#, a\}, \lambda(1) = \{a\}$, and $\lambda(5) = \{b, c\}$). If $u \in Q$, by $\min_{\lambda(u)}$, $\max_{\lambda(u)}$ we denote the minimum and the maximum, with respect to the order $\preceq$, among the elements in $\lambda(u)$.

If $u \in Q$, we will use the symbol I_u with two distinct, yet related, meanings. In Chapters 4, 5 and 6, where we use the co-lexicographic order (see Section 2.1), we denote by I_u the set of all finite words reaching u from the initial state:

$$I_u = \{\alpha \in \text{Pref}(\mathcal{L}(\mathcal{N})) : u \in \delta(s, \alpha)\}.$$

In Chapters 7 and 8, where we use the lexicographic order, the set of I_u is defined starting from the notion of occurrence. An *occurrence* of $\alpha \in \Sigma^*$ starting at $u \in Q$ and ending at $u' \in Q$ is a sequence of states $u_1, u_2, \dots, u_{|\alpha|+1}$ of Q such that (i) $u_1 = u$, (ii) $u_{|\alpha|+1} = u'$ and (iii) $u_i \in \delta(u_{i+1}, \alpha[i])$ for every $1 \leq i \leq |\alpha|$. An *occurrence* of $\alpha \in \Sigma^\omega$ starting at $u \in Q$ is a sequence of states $(u_i)_{i \geq 1}$ of Q such that (i) $u_1 = u$ and (ii) $u_i \in \delta(u_{i+1}, \alpha[i])$ for every $i \geq 1$. Intuitively, a string $\alpha \in \Sigma^* \cup \Sigma^\omega$ has an occurrence starting at $u \in Q$ if we can read α on $\mathcal{N}$ starting from u and following edges *in a backward fashion*. In Chapters 7 and 8, if the initial state has no incoming edge, we add a self-loop labeled #; in this way, every state u has at least one incoming edge, so $I_u \neq \emptyset$. We denote by $\min_u$ and the $\max_u$ the (lexicographically) smallest and largest string in I_u.

2.3 Relations and Orders

If V is a set, a *(binary) relation* R on V is a subset of $V \times V$. We say that $u, v \in V$ are *R-comparable* if $(u,v) \in R \vee (v,u) \in R$ (note that $(u,v) \in R$ and $(v,u) \in R$ may be *both* true). We write $u \parallel v$ if u and v are not R-comparable. We denote by $\mathrm{Trans}(R)$ the transitive closure of R. If R and R' are binary relations on V, we say that R *refines* R' if $(u,v) \in R' \Rightarrow (u,v) \in R$. If R is a binary relation on V and $U \subseteq V$, we say that U is *R-convex* if:

$$(\forall u, v, z \in V)((u, z \in U \wedge (u,v) \in R \wedge (v,z) \in R) \implies v \in U).$$

A *preorder* $\leq$ on V is a binary relation being reflexive and transitive. We write $u < v$ if $u \leq v$ and $u \neq v$. Moreover, the preorder $\leq$ is a *partial order* if it antisymmetric, and it is a *total order* if it is a partial order and every pair of elements are $\leq$-comparable. If $(V, \leq)$ is a partial order and $V' \subseteq V$, we denote by $(V', \leq_{V'})$ the restriction of the partial order $\leq$ to the set V'. To simplify notation, we will also use $(V', \leq)$ when clear from the context.

We introduce some notation typical of partial order, and we naturally extend it to preorders. Let $(V, \leq)$ be a preorder. A set $V' \subseteq V$ is a *$\leq$-chain* if every $u, v \in V$ are $\leq$-comparable. A set $V' \subseteq V$ is a *$\leq$-antichain* if every distinct $u, v \in V$ are not $\leq$-comparable. A partition $\{V_i \mid 1 \leq i \leq p\}$ of V is a *$\leq$-chain partition* if every V_i is a $\leq$-chain. The *width* of $(V, \leq)$, denoted by $\mathrm{width}(\leq)$, is the minimum size of a $\leq$-chain partition. Note that if $\leq$ and $\leq'$ are preorders on V, and $\leq$ refines $\leq'$, then the width of $\leq$ is smaller than or equal to the width of $\leq'$ (because every $\leq'$-chain partition is also a $\leq$-chain partition). If $(V, \leq)$ is a partial order, then *Dilworth's theorem* [45] states that the width of $(V, \leq)$ is equal to the maximum size of a $\leq$-antichain.

Let us recall a standard method for obtaining a partially-ordered quotient set from a preorder. Let $(V, \leq)$ be a preorder. For every $u, v \in V$, let $u \sim_{\leq} v$ if and only if $(u \leq v) \wedge (v \leq u)$. It is immediate to check that $\sim_{\leq}$ is an equivalence relation. Now, let $[v]_{\leq}$ be the quotient class of v, and consider the quotient set $V/_{\leq} = \{[v]_{\leq} | v \in V\}$. Define $\leq^{\sim}$ on $V/_{\leq}$ by letting $[u]_{\leq} \leq^{\sim} [v]_{\leq}$ if and only if $u \leq v$. The definition of $\sim_{\leq}$ implies that $\leq^{\sim}$ is well-defined (that is, the definition does not depend on the choice of representatives), because if $u \sim_{\leq} u'$, $v \sim_{\leq} v'$ and $u \leq v$, then $u' \leq u \leq v \leq v'$. Moreover $(V/_{\sim}, \leq^{\sim})$ is a *partial* order. Indeed, if $[u]_{\leq} \leq^{\sim} [v]_{\leq}$ and $[v]_{\leq} \leq^{\sim} [u]_{\leq}$, then

$u \leq v$ and $v \leq u$, so $[u]_{\leq} = [v]_{\leq}$.

If A, B are *disjoint* subsets of a partial order $(V, \leq)$, then $A < B$ denotes:

$$(\forall a \in A)(\forall b \in B)(a < b).$$

A *monotone sequence* in (a partial order) $(V, \leq)$ is a sequence $(v_n)_{n\in\mathbb{N}}$ with $v_n \in V$ and either $v_i \leq v_{i+1}$, for all $i \in \mathbb{N}$, or $v_i \geq v_{i+1}$, for all $i \in \mathbb{N}$.

If $\alpha \preceq \alpha' \in \Sigma^*$, we define $[\alpha, \alpha'] = \{\beta : \alpha \preceq \beta \preceq \alpha'\}$; if the relative order between α, α' is not known, we set $[\alpha, \alpha']^{\pm} = [\alpha, \alpha']$, if $\alpha \preceq \alpha'$, while $[\alpha, \alpha']^{\pm} = [\alpha', \alpha]$, if $\alpha' \preceq \alpha$.

2.4 Data Structures

When presenting our data structures in detail, we will assume to be working with *integer* alphabets of the form $\Sigma = [0, \sigma - 1]$, that is, alphabets formed by all integers $\{0, 1, \ldots, \sigma - 1\}$. Our data structure results hold in the word RAM model with words of size $w \in \Theta(\log u)$ bits, where u is the size of the input under consideration (for example, u may be the size of an automaton or the length of a string, depending on the input of the algorithm under consideration). When not specified otherwise, the space of our data structures is measured in words.

Recall that the zero-order entropy of a sequence $S \in \Sigma^n$ of length n over alphabet Σ is $H_0(S) = \sum_{c\in\Sigma} \frac{|S|_c}{n} \log_2 \frac{n}{|S|_c}$, where $|S|_c$ denotes the number of occurrences of character c in S. We will use some well-known properties of $H_0(S)$: the quantity $nH_0(S)$ is a lower bound to the length of any encoding of S that encodes each character independently from the others via a prefix code of the alphabet Σ, and in particular $H_0(S) \leq \log_2 |\Sigma|$.

2.5 Rank and Select

Let $S \in \Sigma^n$ be a string. We define the following operation on S:

- *Access:* compute $S[i]$, for any $1 \leq i \leq n$.
- *Rank:* compute $S.rank(i, c) = |\{j \in \{1, \ldots, i\} \mid S[j] = c\}|$, for any $1 \leq i \leq n$ and $c \in \Sigma$.

- *Select:* compute $S.select(i, c)$ equals the integer j such that $S[j] = c$ and $S.rank(j, c) = i$, for any $1 \leq i \leq S.rank(n, c)$ and $c \in \Sigma$.

In other words, the operation $S[i]$ simply returns the i-th character appearing in S, the operation $S.rank(i, c)$ returns the number of occurrences of character c among the first i characters of S, and the operation $S.select(i, c)$ returns the position of the i-th occurrences of character c in S (if it exists). It will be expedient to assume $S.rank(0, c) = S.select(0, c) = 0$, for $c \in \Sigma$, and $S.select(i, c) = n + 1$, for $c \in \Sigma$ and $i > S.rank(n, c)$.

We report a few results on data structures that will be helpful in the following. Recall that $H_0(S)$ is the zero-order entropy of $S \in \Sigma^*$.

Lemma 2.1 (Succinct string [13], Thm 5.2 and [96], Sec. 6.3)**.** *Let $S \in \Sigma^n$ be a string over an integer alphabet $\Sigma = [0, \sigma - 1]$ of size $\sigma \leq n$. Then, there exists a data structure of $nH_0(S)(1 + o(1)) + O(n)$ bits supporting the following operations in time $O(\log \log \sigma)$:*

- *compute $S[i]$, for any $1 \leq i \leq n$.*
- *compute $S.rank(i, c)$, for any $1 \leq i \leq n$ and $c \in \Sigma$.*
- *compute $S.select(i, c)$ equals the integer j such that $S[j] = c$ and $S.rank(j, c) = i$, for any $1 \leq i \leq S.rank(n, c)$ and $c \in \Sigma$.*

Given S, the data structure can be built in $O(n \log \log \sigma)$ worst-case time.

Note that Lemma 2.1 requires the cardinality σ of the alphabet to be no larger than the length of the string. However, this will turn out to be too restrictive, for two reasons: (1) we would like to be able to handle also automata labeled with larger alphabets and, most importantly, (2) in our data structures (see the proof of Theorem 4.47) we will also need to manage *rank* and *select* queries over strings defined not on Σ, but $[1, p] \times \Sigma$ (where $p \leq n$ is an integer will be the *width* of the underlying automaton), and even if $\sigma \leq n$ it may still be $p \cdot \sigma > n$. With the following lemma we cover this more general case. The requirement $|\Sigma| \leq n^{O(1)}$ ensures that characters fit in a constant number of computer memory words and thus they can be manipulated in constant time. Note that we lose fast access functionality (which however will not be required in our application of this data structure).

Lemma 2.2 (Succinct string over large alphabet)**.** *Let $S \in \Sigma^n$ be a string over an integer alphabet $\Sigma = [0, \sigma - 1]$ of size $\sigma = |\Sigma| \le n^{O(1)}$. Then, there exists a data structure of $nH_0(S)(1 + o(1)) + O(n)$ bits, where $H_0(S)$ is the zero-order entropy of S, supporting the following operations in time $O(\log\log\sigma)$:*

- Rank: *$S.rank(i, c) = |\{j \in \{1, \ldots, i\} \mid S[j] = c\}|$, for $1 \le i \le n$ and $c \in \Sigma$ that occurs in S.*
- Select: *$S.select(i, c)$ equals the integer j such that $S[j] = c$ and $S.rank(j, c) = i$, for any $1 \le i \le S.rank(n, c)$ and for any character $c \in \Sigma$ that occurs in S.*

Given S, the data structure can be built in expected $O(n \log\log\sigma)$ time.

Proof. If $\sigma \le n$, then we simply use the structure of Lemma 2.1. Otherwise ($\sigma > n$), let $\Sigma' = \{S[i] \mid 1 \le i \le n\}$ be the *effective alphabet* of S. We build a minimal perfect hash function $h : \Sigma \to [0, |\Sigma'| - 1]$ mapping (injectively) Σ' to the numbers in the range $[0, |\Sigma'| - 1]$ and mapping arbitrarily $\Sigma \setminus \Sigma'$ to the range $[0, |\Sigma'| - 1]$. We store h using the structure described in [72]. This structure can be built in $O(n)$ expected time, uses $O(n)$ bits of space, and answers queries of the form $h(x)$ in $O(1)$ worst-case time. Note that $|\Sigma'| \le n$, so we can build the structure of Lemma 2.1 starting from the string $S' \in [0, |\Sigma'| - 1]^n$ defined as $S'[i] = h(S[i])$. Then, *rank* and *select* operations on S can be answered as $S.rank(i, c) = S'.rank(i, h(c))$ and $S.select(i, c) = S'.select(i, h(c))$, provided that $c \in \Sigma'$. Notice that the zero-order entropies of S and S' coincide, since the character's frequencies remain the same after applying h to the characters of S. We conclude that the overall space of the data structure is at most $nH_0(S)(1 + o(1)) + O(n)$ bits. $\square$

Finally, we need a fully-indexable dictionary data structure. Such a data structure encodes a set of integers and supports efficiently a variant of *rank* and *select* queries as defined below:

Lemma 2.3 (Fully-indexable dictionary [53], Theorem 4.1)**.** *A set $A = \{x_1, \ldots, x_n\} \subseteq [1, u]$ of cardinality n can be represented with a data structure of $n \log(u/n) + O(n)$ bits so that the following operations can be implemented in $O(\log\log(u/n))$ time:*

- Rank: *$A.rank(x) = |\{y \in A \mid y \le x\}|$, for any $1 \le x \le u$.*

- Select: *$A.select(i) = x$ such that $x \in A$ and $A.rank(x) = i$, for any $1 \leq i \leq |A|$.*

Given A as input, the data structure can be built in $O(n)$ worst-case time.

Remark 2.4. The queries of Lemma 2.3 can be used to solve in $O(\log\log(u/n))$ time also:

- *Predecessor:* the largest element of A smaller than or equal to x, if it exists. For any $1 \leq x \leq u$, $A.pred(x) = A.select(A.rank(x))$ if $A.rank(x) > 0$, and $A.pred(x) = \perp$ otherwise.

- *Strict-Successor:* the smallest element of A strictly greater than x, if it exists. For any $1 \leq x \leq u$, $A.succ(x) = A.select(A.rank(x) + 1)$ if $A.rank(x) < |A|$, and $A.succ(x) = \perp$ otherwise.

- *Membership:* For any $1 \leq x \leq u$, $x \in A$ if and only if $x = A.pred(x)$.

Chapter 3

Compressing and Indexing Strings

In this chapter, we outline some classical results in the data compression field. The literature has constantly grown in the past 30 years, so we do not aim to provide a comprehensive discussion. We believe that a good starting point for the reader interested in the field is Navarro's book [96], which can be complemented by additional resources depending on the specific interests of the reader (bioinformatics applications [93], the Burrows-Wheeler Transform [2], suffix trees [71], and so on). In this thesis, we extend several important results in data compression from string to automata, so the aim of this chapter is to present some classical results on string from a perspective that better encompasses the ideas that we will develop in the subsequent chapters.

3.1 Pattern Matching in Compressed Space

Given a text, we want to compress the text in such a way that we can efficiently solve pattern matching queries without decompressing the text. We can describe our problem as follows.

Problem 1. *Let $S \in \Sigma^n$. Build a data structure S' such that:*

1. *S' is small.*

2. *S' is an encoding of S.*

3. *S' can be used to solve the following problem efficiently: given $\pi \in \Sigma^*$, count the number of occurrences of π as a substring of S by only using S'.*

Requirement 2 means that, given S', it is possible to retrieve S. Requirement 3 implies that we can solve *pattern matching queries* on S by using S'. In addition, we want S' to be a *space-efficient* encoding (Requirement 1) that supports a *time-efficient* pattern matching algorithm (Requirement 3).

i	Sorted suffixes	SA_S	SA_S^{-1}	BWT_S
1	\$	12	6	i
2	i\$	11	5	p
3	ippi\$	8	12	s
4	issippi\$	5	10	s
5	ississippi\$	2	4	m
6	mississippi\$	1	11	\$
7	pi\$	10	9	p
8	ppi\$	9	3	i
9	sippi\$	7	8	s
10	sissippi\$	4	7	s
11	ssippi\$	6	2	i
12	ssissippi\$	3	1	i

Figure 3.1: The sorted suffixes of "mississippi\$" and the arrays SA_S, SA_S^{-1}, BWT_S. We assume that \$ is the smallest character.

In the following, we consider an alphabet Σ of size $\sigma = |\Sigma|$, and we assume that there is a fixed total order $\preceq$ on Σ. We denote by Σ^* the set of all finite strings on the alphabet Σ, and we extend Σ to Σ^* *lexicographically*, that is, we consider the standard dictionary order. For example, if Σ is the English alphabet and $\preceq$ is the order in which $a \prec b \prec c \prec \dots$, then it holds *cost* $\prec$ *cottage*.

3.2 The Suffix Array

A first solution to Problem 1 is based on suffix arrays (see Figure 3.1 for an example).

Definition 3.1. Let $S \in \Sigma^n$. The *suffix array* SA_S of S is the sequence of length n such that, for every $1 \leq i \leq n$, it holds $\mathsf{SA}_S[i] = j$ if and only if $S[j, n]$ is the i-th lexicographically smallest suffix of S.

The *inverse suffix array* SA_S^{-1} is the inverse permutation of SA_S, that is, $\mathsf{SA}_S^{-1}[j] = i$ if and only if $S[j, n]$ is the i-th lexicographically smallest suffix.

Now, let S' be the data structure consisting of the original string S and the suffix array SA_S. Let us show how to use S' for solving pattern matching queries (as defined in Requirement 3 of Problem 1). Consider a pattern string $\pi \in \Sigma^m$. Note that every substring of S is a prefix of some suffix of S. As a consequence, we only have to count the number of suffixes of S starting with π. The definition of suffix array implies that, if π occurs as a substring of S, then there exist unique $1 \leq l \leq r \leq n$ such that for every $1 \leq i \leq n$ we have that suffix $S[\mathsf{SA}_S[i], n]$ starts with π if and only if $l \leq i \leq r$. For example, in Figure 3.1, if $\pi = issi$, then $l = 4$ and $r = 5$. We conclude that we only have to explain how to determine l and r (or establish that π does not occur as a substring of S). We can determine l, if it exists, by means of a binary search. In our example, since $n = 12$, we perform a binary search on $[1, 12]$. We start from $i = 12/2 = 6$, and we compare $\psi = issi$ and $S[\mathsf{SA}_S[i], n] = mississippi\$$ to determine which one is smallest. To this end, it is sufficient to compare at most $|\psi| = m$ characters. Since $\psi \preceq S[\mathsf{SA}_S[i], n]$, we conclude that it must be $l \leq 6$, if l exists, so we only have to search the interval $[1, 6]$ recursively. The binary search for l requires $O(\log n)$ steps, and each time we compare at most m characters, so the running time is $O(m \log n)$. Analogously, we can compute r in $O(m \log n)$ if it exists. Then, the number of occurrences of ψ as a substring of S is simply $r - l + 1$.

To sum up, if S' is the data structure consisting of the original string S and the suffix array SA_S, then we can solve Problem 1, since (i) S' is trivially an encoding of S (because S' contains S) and we can efficiently solve pattern matching queries. More precisely, S' requires $n(\log n + \log \sigma)$ bis (because $n \log n$ bits are required for storing SA_S and $n \log \sigma$ bits are required for storing S) and allows solving pattern matching queries in $O(m \log n)$, where m is the length of the pattern.

The suffix array can be compressed by means of the techniques introduced by Grossi and Vitter [70], but we will not further pursue this direction here.

3.3 The Burrows-Wheeler Transform

Let us present a second solution to Problem 1 based on the Burrows-Wheeler Transform. In this section, we assume that a string $S \in \Sigma^n$ ends with a special character $\$ \notin \Sigma$ that is assumed to be smaller than all characters in Σ.

It is expedient to see a string as a graph (see Figure 3.2) because (i) it will help to develop our intuition on the problem and (ii) it is the approach that will be generalized in the next chapters. Intuitively, a node is labeled j if starting from node j one can read the j-th lexicographically smallest string.

Definition 3.2. Let $S \in \Sigma^n$. We define the graph $G_S = (V_S, E_S)$ as follows:

1. $V_S = \{1, 2, \dots, n\}$.

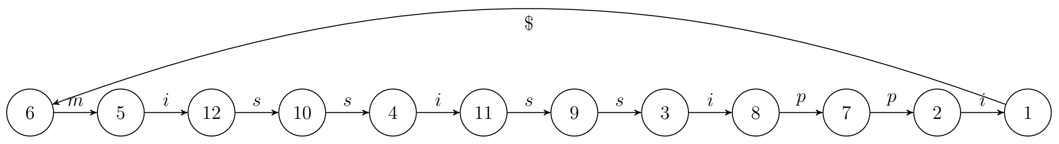


Figure 3.2: The graph $G_S = (V_S, E_S)$, for S = "$mississippi\$$".

2. $E_S = \{(\mathsf{SA}^{-1}[i], \mathsf{SA}^{-1}[i+1], S[i]) \mid 1 \leq i \leq n\} \cup \{(1, \mathsf{SA}^{-1}[1], \$)\}$.

The next lemma establishes some simple but crucial properties that will be at the core of this thesis (see Definition 4.1).

Lemma 3.3. *Let $S \in \Sigma^n$ and consider the graph $G_S = (V_S, E_S)$.*

1. For every $(i, i', a), (j, j', b) \in E_S$, if $i < j$, then $a \preceq b$.

2. For every $(i, i', a), (j, j', a) \in E_S$, if $i < j$, then $i' < j'$.

Proof. 1. The property is trivially true if $i = 1$, so we can assume $1 < i < j$. Define $i^* = \mathsf{SA}_S[i]$ and $j^* = \mathsf{SA}_S[j]$; then $(i, i', a) = (\mathsf{SA}_S^{-1}[i^*], \mathsf{SA}_S^{-1}[i^*+1], S[i^*])$ and $(j, j', b) = (\mathsf{SA}_S^{-1}[j^*], \mathsf{SA}_S^{-1}[j^*+1], S[j^*])$. We must prove that $S[i^*] \preceq S[j^*]$. Since $i < j$, then $S[\mathsf{SA}_S[i], n] \prec S[\mathsf{SA}_S[j], n]$, or equivalently, $S[i^*, n] \prec S[j^*, n]$, which implies $S[i^*] \preceq S[j^*]$.

2. Only one edge is labeled with \$, so we can assume $1 < i < j$. Define $i^* = \mathsf{SA}_S[i]$ and $j^* = \mathsf{SA}_S[j]$; then $(i, i', a) = (\mathsf{SA}_S^{-1}[i^*], \mathsf{SA}_S^{-1}[i^*+1], S[i^*])$ and $(j, j', a) = (\mathsf{SA}_S^{-1}[j^*], \mathsf{SA}_S^{-1}[j^*+1], S[j^*])$, with $S[i^*] = S[j^*]$. We must prove that $\mathsf{SA}_S^{-1}[i^*+1] < \mathsf{SA}_S^{-1}[j^*+1]$. Since $i < j$, then $S[\mathsf{SA}_S[i], n] \prec S[\mathsf{SA}_S[j], n]$, or equivalently, $S[i^*, n] \prec S[j^*, n]$. From $S[i^*] = S[j^*]$ we obtain $S[i^*+1, n] \prec S[j^*+1, n]$, so $\mathsf{SA}_S^{-1}[i^*+1] < \mathsf{SA}_S^{-1}[j^*+1]$.

□

We can now definite the Burrows-Wheeler Transform of a string.

Definition 3.4. Let $S \in \Sigma^n$. The *Burrows-Wheeler Transform* BWT_S of S is the array of length n such that $\mathsf{BWT}_S[i]$ is the label of the edge entering node i in $G_S = (V_S, E_S)$.

Let us give an explicit characterization of the Burrows-Wheeler Transform, which corresponds to the usual definition of the Burrows-Wheeler Transform in the literature.

Lemma 3.5. *Let $S \in \Sigma^n$. Then:*

$$\mathsf{BWT}_S[i] = \begin{cases} S[\mathsf{SA}_S[i]-1] & \text{if } \mathsf{SA}_S[i] \neq 1 \\ \$ & \text{if } \mathsf{SA}_S[i] = 1. \end{cases}$$

Proof. $\mathsf{BWT}_S[i]$ is the label of the edge entering node i in $G_S = (V_S, E_S)$. If $\mathsf{SA}_S[i] \neq 1$, then the definition of E_S implies that this label is $S[\mathsf{SA}_S[i]-1]$; if $\mathsf{SA}_S[i] = 1$, then the definition of E_S implies that this label is \$. □

Let us prove that BWT_S satisfies Requirement 2 of Problem 1.

Lemma 3.6. *Let $S \in \Sigma^n$. Then, BWT_S is an encoding of S.*

Proof. In order to retrieve S, we only have to show how to retrieve then G_S, because then S is the string that can be read starting from the node whose incoming edge is labeled with \$. If BWT_S has length n, then $V_S = \{1, 2, \dots, n\}$, so we only have to show how to retrieve E_S. In order words, we must show how to retrieve the edge leaving each node $1 \leq i \leq n$. The definition of BWT_S implies the BWT_S contains all characters of S with the same multiplicities as in S. Fix any character a in S; we are only left with the problem of determining the start node and the end node of each edge labeled a. By the second property of Lemma 3.3, we only have to determine the set B_a of all nodes whose outgoing edge is labeled a and the set C_a of all nodes whose incoming edge is labeled a. For example, in Figure 3.2, we have $B_i = \{2, 3, 4, 5\}$ and $C_i = \{1, 8, 11, 12\}$, and by the second property of Lemma 3.3 the edges labeled i are $(2, 1, i), (3, 8, i), (4, 11, i), (5, 12, i)$.

First, we can retrieve each set C_a by the definition of BWT_S. For example, if $S = \mathit{mississippi}\$$, we know that $\mathsf{BWT}_S = \mathit{ipssm}\$\mathit{pissii}$, which implies $C_\$ = \{6\}$, $C_i = \{1, 8, 11, 12\}$, $C_m = \{5\}$, $C_p = \{2, 7\}$, $C_s = \{3, 4, 9, 10\}$.

Second, we can retrieve each set B_a by the first property of Lemma 3.3. For example, if $S = \mathit{mississippi}\$$, we know $\mathsf{BWT}_S = \mathit{ipssm}\$\mathit{pissii}$; by sorting the characters in BWT_S we obtain $\$\mathit{iiiimppssss}$, so $B_\$ = \{1\}$, $B_i = \{2, 3, 4, 5\}$, $B_m = \{6\}$, $B_p = \{7, 8\}$, $B_s = \{9, 10, 11, 12\}$. □

Now, let $D_S[1, n]$ a bit array such that $D_S[i] = 1$ if and only $i = 1$ or the i-th smallest character of S is distinct from the $(i-1)$-th smallest character of S. For example, if $S = mississippi\$$, we consider $\$iiiimppssss$ and so $D_S[1, 12] =$ 110001101000. Let us see how to search for the pattern $\psi = issi$. We will start from the end of ψ (*backward search*). Since i is the second smallest character (including \$), we consider the positions between the second 1 in C and the third 1 in C, that it, $2, 3, 4, 5$ (we include the position of the second i and we exclude the position of the third i). The definition of C implies that the suffixes starting with i are the second, the third, the fourth and the fifth one. Now we want to determine which suffixes start with si. The edges labeled s in Figure 3.2 are $(9, 3, s)$, $(10, 4, s)$, $(11, 9, s)$ $(12, 10, s)$; the set of all end nodes is $C_s = \{3, 4, 9, 10\}$ and the set of all start nodes is $B_s = \{9, 10, 11, 12\}$.

The number of edges labeled s reaching node 1 is zero, and the number of edges labeled s reaching nodes $1, 2, 3, 4, 5$ is two. By the second property of Lemma 3.3, from B_s we conclude the suffixes starting with si are the ninth and the tenth suffix. Let us determine the suffixes starting with ssi. The number of edges labeled s reaching node $1, 2, 3, 4, 5, 6, 7, 8$ is two, and then number of edges labeled s reaching node $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ is four, By the second property of Lemma 3.3, from B_s we conclude the suffixes starting with ssi are the eleventh and the twelfth suffix. Analogously, we finally obtain all suffixes starting with $\psi = issi$.

Notice that, when computing the interval starting with ssi from the interval starting with si, we only need to solve rank and select queries (Section 2.5) on B_s and C_s. The set B_s is implicitly stored in D_S (since s is the fifth smallest character, we must consider the fifth i in D_S), while the set C_s is implicitly stored in BWT_S by the definition of the Burrows-Wheeler Transform of S. We conclude that, in order to perform the backward search, we only need to store BWT_S and D_S in such a way that they efficiently support rank and select operations.

Let S' be the data structure consisting of the strings BWT_S and D_S stored as in Lemma 2.1. Then S' solves Problem 1, because it requires $n(H_0(S) + H_0(D_S))(1 + o(1)) + O(n) \leq n \log \sigma(1 + o(1)) + O(n)$ bits (Requirement 1), it is an encoding of S (by Lemma 3.6) and allows solving pattern matching queries in $O(m \log \log \sigma)$ time, where m in the length of the pattern (Requirement 3).

The Burrows-Wheeler Transform is a compressible encoding of a string because it can be stored using a number of bits equal to the entropy of the string. As a consequence, the previous results can be further improved by suitably compressing the Burrows-Wheeler Transform, thus obtaining the full *FM-index* [57]; we will not give further details here.

3.4 The Co-Lexicographic Variant

In the previous sections, we find the occurrences of a pattern by determining the interval corresponding to the suffixes starting with the pattern. A complementary, yet equivalent, approach is possible: we can determine the interval corresponding to the *prefixes ending* with the pattern. Note that now prefixes are sorted *co-lexicographically* (see Section 2.1). The suffix array of a string S corresponds to the *prefix array* of the reverse string S^R, and all the results of the previous sections can be immediately adapted by assuming to be working with the reverse string and the reverse pattern.

In the rest of this thesis, we will generalize to autoamta the co-lexicographical variant, and not the lexicographic variant. One reason is that we will mainly extend the results in Section 3.3, and it will be more natural to describe a *forward search* rather than a backward search. A second reason is that, in automata theory, many structural properties depend on the strings *reaching* a state (and not leaving a state), which can be seen as a generalization of prefixes, not suffixes. However, the most important reason is that, as we will see, our results are simpler if we consider *deterministic* automata: if we considered the lexicographic variant, we should introduce *co-deterministic* automata, that is, automata where for each state u and each character c there exists at most one edge labeled c *reaching* state u.

Chapter 4

Co-lex Orders: Compression And Pattern Matching

In this chapter we extend the results in Chapter 3 from strings to NFAs. We will consider the *co-lexicographic order* $\preceq$ on Σ^* (see Section 2.1), and we will see how to lift it to the states of an NFA, thus obtaining the *co-lex order* of an NFA.

We capture co-lex orders on an NFA by means of two axioms, inspired by Lemma 3.3, which ensure a *local* comparability between pairs of states. Given two states u and v such that $u < v$, Axiom 1 of Definition 4.1 imposes that all words in I_u end with letters being smaller than or equal to letters ending words in I_v; Axiom 2, instead, requires that the order among states u and v propagates backwards when following pairs of equally-labeled transitions.

Definition 4.1. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA. A *co-lex order* on $\mathcal{N}$ is a partial order $\leq$ 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 $u' \leq v'$.

Remark 4.2. 1. Since $\# \in \lambda(s)$ and $\# \notin \lambda(u)$ for $u \neq s$, then from Axiom 1 it follows that for every $u \in Q$ it holds $u \not< s$.

2. If $\mathcal{D}$ is a DFA, then we can restate Axiom 2 as follows: for every $a \in \Sigma$, if $u = \delta(u', a)$, $v = \delta(v', a)$, and $u < v$, then $u' < v'$ (it must be $u' \neq v'$ because u and v are distinct).

We note that Axiom 2 implies that the order between two states is not defined whenever their predecessors cannot be unambiguously compared, as observed in the following remark.

Remark 4.3. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let $\leq$ be a co-lex order on $\mathcal{N}$. Let $u, v \in Q$ be two distinct states. Then, $u \parallel v$ if at least one of the following holds:

1. There exist $u', v' \in Q$ and $a \in \Sigma$ such that $u \in \delta(u', a)$, $v \in \delta(v', a)$ and $u' \parallel v'$.
2. There exist $u', v', u'', v'' \in Q$ and $a, b \in \Sigma$ such that $u \in \delta(u', a) \cap \delta(u'', b)$, $v \in \delta(v', a) \cap \delta(v'', b)$, $u' < v'$ and $v'' < u''$.

Indeed, if e.g. it were $u < v$, then Axiom 2 would imply that in case 1 it should hold $u' \leq v'$ and in case 2 it should hold $u'' \leq v''$ (which is forbidden by the antisymmetry of $\leq$).

Co-lex orders can be seen as a generalization of prefix array from strings to automata. The prefix array of a string is a permutation of the text positions, while in general a co-lex order does not yield a permutation of the set of all states, because in general a co-lex order is only a partial order. The special case where a co-lex order is a total order is both of theoretical and practical interest, as we will see, so let us give the following definition.

Definition 4.4. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA.

1. If a co-lex order $\leq$ on $\mathcal{N}$ is a total order, we say that $\leq$ is a *Wheeler order*.
2. If $\mathcal{N}$ admits a Wheeler order, then $\mathcal{N}$ is a *Wheeler NFA*.

Note that a co-lex order $\leq$ is Wheeler if and only if its width is equal to 1. Wheeler orders were first introduced in [61] in a slightly less general setting. The class of Wheeler languages — that is, the class of all regular languages recognized by some Wheeler NFA — is rather small: for example, unary languages are Wheeler only if they are finite or co-finite, and all Wheeler languages are star-free (see [5]). Moreover, Wheeler languages are not closed under union, complement, concatenation, and Kleene star [5]. In contrast, as observed in the following remark, any regular automaton admits a co-lex order.

Remark 4.5. Every NFA $\mathcal{N}$ admits some 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 on $\mathcal{N}$.

More than one non-trivial co-lex order can be given on the same automaton. As an example, consider Figure 4.1: the automaton on the left admits the two co-lex orders whose Hasse diagrams are depicted on the right. The first, $\leq_1$, is total and states that $2 <_1 3$, while the width of the second one, $\leq_2$, is equal to 2 and $3 <_2 2$ holds. As a matter of fact, in any co-lex order $\leq$ for this automaton in which $3 < 2$ holds, nodes 4 and 5 must be incomparable.

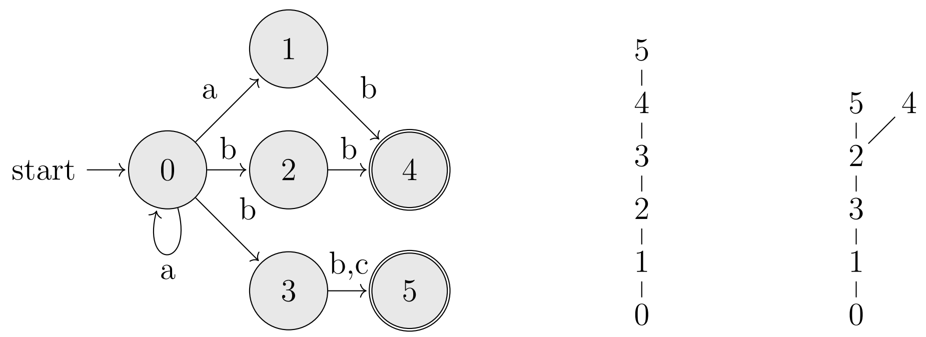


Figure 4.1: An NFA and the Hasse diagrams — that is, graphs depicting the *transitive reductions* of the partial orders — of two of its (maximal) co-lex orders. Characters are sorted according to the standard alphabetical order.

4.1 The Co-lex Width of NFAs and DFAs

In Sections 4.3 and 4.4 we will prove that a co-lex order over an automaton enables compression and indexing mechanisms whose efficiency is parameterized by the width of the co-lex order (the smaller, the better, so the maximum efficiency is reached on Wheeler NFAs): this justifies introducing the *co-lex width* of an NFA (Definition 4.6) as a meaningful measure for compression and indexing. In fact, the co-lex width can also be used for further, interesting, language-theoretic consequences — more on this in Chapter 5.

Definition 4.6. The *co-lex width* of an NFA $\mathcal{N}$ is the minimum width of a co-lex order on $\mathcal{N}$:

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

In Example 4.18 below we shall see that the value width($\mathcal{N}$) may depend on the choice of the total order $\preceq$ on Σ.

As a matter of fact, string sorting stands at the core of the most popular string compression and indexing paradigms, which for this reason also suffer from a sharp dependence on the total alphabet order. For example, the number r of equal-letter runs of the Burrows-Wheeler transform (BWT) of a string [27] is an important string compressibility parameter (see [62]) and its value depends on the choice of the total order on the alphabet; deciding whether there exists an ordering of the alphabet of a string such that r is bounded by a given value, however, is an NP-complete problem [14]. Despite this limitation, the BWT and the data structures based on it — such as the FM-index [56] and the r-index [62] — are widely used in applications with a fixed (often sub-optimal) alphabet order.

Similarly, in our scenario it is natural to wonder whether it is possible to determine an ordering of the alphabet that minimizes the width of an automaton. Unfortunately, also this problem is not tractable: deciding whether there exists a total alphabet order under which a given DFA is Wheeler (that is, it has co-lex width equal to one) is already an NP-complete problem [42]. In such situations, one possible way to tame the problem's complexity is to study a more constrained version of the problem, with the goal of shedding new light on the more general (unconstrained) scenario. For this reason, we start by fixing a total order on the alphabet and investigating the implications of this choice. In particular we will prove that, if we fix an order on the alphabet, then the width of a DFA with respect to that order can be determined in polynomial time. This finding can already be used, for example, as a black-box to test candidate alphabet orderings in search for the one minimizing the automaton's width.

In the following, we establish preliminary useful properties of the new measure width($\mathcal{N}$). Our first observation is that this measure is linked with the graph's *sparsity.*

Lemma 4.7. *Let $\mathcal{N}$ be an NFA on an alphabet of cardinality σ with n states and $|\delta|$ transitions, and let $p =$ width($\mathcal{N}$). Then:*

$$|\delta| \leq (2n - p)p\sigma.$$

Proof. Let $\mathcal{N}$ be an NFA on an alphabet of cardinality σ with n states and $|\delta|$ transitions, and let $p =$ width($\mathcal{N}$). Let $\leq$ be a co-lex order of width p, $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of the set of states, and for all $1 \leq i \leq p$ let $Q_i = \{v_{i,1}, \dots, v_{i,n_i}\}$, where $n_i = |Q_i|$ and $v_{i,k} < v_{i,k'}$ whenever $k < k'$. Fix $1 \leq i, j \leq p$ and $a \in \Sigma$, and consider all the transitions $(v_{i,k}, v_{j,l}, a)$ labeled with a that leave Q_i and reach Q_j. We denote with $e_{i,j,a}$ the number of such transitions; our goal is to establish an upper bound to this quantity for all i, j, a. Sort these $e_{i,j,a}$ edges $(v_{i,k}, v_{j,l}, a)$ by the index l of their destination state, breaking ties by the index k of their source state. Now, let us prove that the value $k + l$ is strictly increasing with respect to this order. In order words, we want to prove that if we pick two edges $(v_{i,k}, v_{j,l}, a)$ and $(v_{i,k'}, v_{j,l'}, a)$ being consecutive with respect to the edge order, then $k + l < k' + l'$. By the definition of the edge order, we have $l \leq l'$. If $l = l'$, again by the definition of the edge order we have $k < k'$ and so $k + l < k' + l'$. If $l < l'$, by Axiom 2 of co-lex orders we have $k \leq k'$, and again we conclude $k + l < k' + l'$. Since we have proved that $k + l$ is strictly increasing with respect to the edge order, then from $2 \leq k + l \leq n_i + n_j$ we obtain $e_{i,j,a} \leq n_i + n_j - 1$. Observing that $\sum_{i=1}^{p} n_i = n$, we conclude:

$$|\delta| = \sum_{a\in\Sigma}\sum_{i=1}^{p}\sum_{j=1}^{p} e_{i,j,a} \leq \sum_{a\in\Sigma}\sum_{i=1}^{p}\sum_{j=1}^{p}(n_i + n_j - 1) = 2\sigma pn - \sigma p^2 = (2n - p)p\sigma.$$

□

The above lemma will be useful later, when measuring the size of our NFA encodings as a function of the number of states. Note that Wheeler automata ($p = 1$) have a number of transitions proportional to $O(\sigma n)$. This relation was already noted in the literature [69, Thm. 4].

Next, we move on to studying some preliminary properties of the smallest-width co-lex order. Recall (see Section 2.3) that $\leq^*$ is a *refinement* of $\leq$ if, for all $u, v \in Q$, $u \leq v$ implies $u \leq^* v$. Since there are only finitely many co-lex orders over an automaton, every co-lex order $\leq$ is maximally refined by a co-lex order. Moreover, if $\leq^*$ is a refinement of $\leq$, then it must be that width($\leq^*$) is less than or equal to width($\leq$), since every $\leq$-chain partition is also a $\leq^*$-chain partition. This implies that there is always a *maximal* co-lex order $\leq$ on an NFA $\mathcal{N}$ such that width($\mathcal{N}$) = width($\leq$). In general an NFA admits several maximal co-lex orders of different widths. For example, the two co-lex orders presented in Figure 4.1 are both maximal and have different widths. This cannot happen over DFAs: in the following lemma we prove that a DFA always admits a unique maximal co-lex order (the *maximum* co-lex order) so that this order realizes the width of the DFA. In particular, the maximum co-lex order refines every co-lex order on the DFA.

Definition 4.8. Let $\mathcal{D}$ be a DFA. The relation $<_{\mathcal{D}}$ over Q is defined by:

$$u <_{\mathcal{D}} v \text{ if and only if } (\forall \alpha \in I_u)(\forall \beta \in I_v)\ (\alpha \prec \beta).$$

One can easily prove that $\leq_{\mathcal{D}}$ (that is, $<_{\mathcal{D}} \cup \{(u,u) \mid u \in Q\}$) is a partial order over Q. Moreover:

Lemma 4.9. *If $\mathcal{D}$ is a DFA then $(Q, \leq_{\mathcal{D}})$ 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 $\alpha \prec \beta$.

Let us now prove that $\leq_{\mathcal{D}}$ is the maximum co-lex order.

Suppose, reasoning for contradiction, that $\leq$ is a co-lex order on $\mathcal{D}$ and for some distinct $u, v \in Q$, $u < v$, and $u \not<_{\mathcal{D}} v$. Then, there exist $\alpha \in I_u, \beta \in I_v$ with $\beta \prec \alpha$. 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 word, otherwise v would be the initial state s, while $z \not< s$ for all $z \in Q$ (see Remark 4.2). 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 by 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 and Remark 4.2 we get $u' < v'$; however, the pair α', β' witnesses $u' \not\leq_{\mathcal{D}} v'$, contradicting the minimality of β. $\square$

Having proved that $\leq_{\mathcal{D}}$ is the maximum co-lex order over $\mathcal{D}$, we immediately deduce that its characterizing property is satisfied by *any* co-lex order.

Corollary 4.10. *If $\leq$ is a co-lex order over a DFA and $u < v$ then $(\forall \alpha \in I_u)(\forall \beta \in I_v)$ $(\alpha \prec \beta)$.*

Remark 4.11. The previous corollary shows that Axiom 1 of co-lex orders *propagates* from a *local* level (i.e. letters in $\lambda(u), \lambda(v)$, for which it holds $(\forall e \in \lambda(u))(\forall f \in \lambda(v))(e \preceq f)$) to a *global* one (i.e. words in I_u, I_v, for which it holds $(\forall \alpha \in I_u)(\forall \beta \in I_v)(\alpha \preceq \beta)$). This works for DFAs because different states are reached by disjoint sets of words: if $u \neq v$ then $I_u \cap I_v = \emptyset$. On NFAs things become more complicated and the existence of a maximum co-lex order is no longer guaranteed.

Since $\leq_{\mathcal{D}}$ extends any possible co-lex order on Q, it realizes the width of the automaton $\mathcal{D}$, as stated in the following lemma.

Lemma 4.12. *If $\mathcal{D}$ is a DFA then width$(\leq_{\mathcal{D}})$ = width$(\mathcal{D})$.*

Now, let us prove that $\leq_{\mathcal{D}}$ can be computed in polynomial time. We start with a characterization of $\leq_{\mathcal{D}}$ in terms of graph reachability.

Definition 4.13. 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 4.14. *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 $\alpha \prec \beta$.

$\square$

Using the previous lemma we are now able to describe a polynomial time algorithm for computing $<_\mathcal{D}$. Let $G = (V, F)$ where $V = \{(u, v) \in Q \times Q \mid u \neq v\}$ and $F = \{((u', v'), (u, v)) \in V \times V \mid (\exists e \in \Sigma)(\delta(u', e) = u \wedge \delta(v', e) = v)\}$, where $|F| \leq |\delta|^2$. Intuitively, we will use G to propagate the complement $\not<_\mathcal{D}$ of $<_\mathcal{D}$. First, mark all nodes (u, v) of G for which $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$ does not hold. This process takes $O(|Q|^2)$ time: for any state u we find the minimum and the maximum of $\lambda(u)$ by scanning the transitions of the automaton (total time $O(|\delta|)$); then we decide in constant time when $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$ does not hold. Then, mark all nodes reachable in G from marked nodes. This can be done with a simple DFS visit of G, initiating the stack with all marked nodes. This process takes $O(|\delta|^2)$ time. By Lemma 4.14, the set of unmarked pairs is $<_\mathcal{D}$. Hence, we proved:

Lemma 4.15. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. We can find the order $\leq_\mathcal{D}$ in $O(|\delta|^2)$ time.*

The width of a partial order can be determined in polynomial time [54].

Lemma 4.16. *Let $(V, \leq)$ be a partial order, with $|V| = n$. The smallest $\leq$-chain decomposition of V can be found in $O(n^{5/2})$ time.*

From Lemma 4.12 and Lemma 4.16 we conclude that the width of a DFA can be computed in polynomial time.

Corollary 4.17. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. Then, width($\leq_\mathcal{D}$) can be computed in $O(|\delta|^2 + |Q|^{5/2})$ time.*

Lemma 4.15, Lemma 4.16 and Corollary 4.17 show that the problem of computing width($\leq_\mathcal{D}$) can be solved in polynomial time, but this is not the end of the story; improving the bounds is algorithmically challenging and requires advanced techniques, as we will see in Chapter 7. As for the width-complexity over NFAs, it is known that the problem is NP-hard, since already deciding whether the width of an NFA is equal to 1 (i.e. deciding whether the NFA is Wheeler) is an NP-complete problem (see [69]).

With the next example we show that the co-lex width of an automaton may depend on the total order on the alphabet.

Example 4.18. Let $\mathcal{D}$ be a DFA. Let us show that, in general, the value $\text{width}(\mathcal{D}) = \text{width}(\leq_{\mathcal{D}})$ (Lemma 4.12) may depend on the total order $\preceq$ on the alphabet. Let $\mathcal{D}$ be the DFA in Figure 4.2.

First, assume that $\preceq$ is the standard alphabetical order such that $a \prec b \prec c \prec d$. We have $q_1 <_{\mathcal{D}} q_2 <_{\mathcal{D}} q_4$ and $q_1 <_{\mathcal{D}} q_3 <_{\mathcal{D}} q_4$, so $\text{width}(\leq_{\mathcal{D}})$ is at most two. Notice that q_2 and q_3 are not $\leq_{\mathcal{D}}$-comparable because $acc, ac \in I_{q_2}$, $bc \in I_{q_3}$ and $ac \prec bc \prec acc$, so $\text{width}(\leq_{\mathcal{D}})$ is equal to two.

Next, assume that $\preceq$ is the total order such that $a \prec c \prec b \prec d$. Then, $q_1 <_{\mathcal{D}} q_2 <_{\mathcal{D}} q_3 <_{\mathcal{D}} q_4$, hence $\text{width}(\leq_{\mathcal{D}})$ is equal to one.

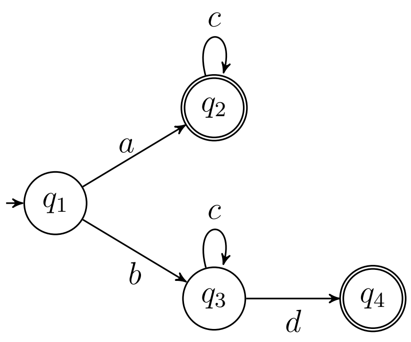


Figure 4.2: A DFA $\mathcal{D}$ where the value $\text{width}(\mathcal{D})$ depends on the total order $\preceq$ on the alphabet.

We now generalize Corollary 4.10 to NFAs, coping with the fact that, on NFAs, sets I_u, I_v may intersect for $u \neq v$.

Lemma 4.19. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let $\leq$ be a co-lex order on $\mathcal{N}$. If $u < v$, then $(\forall \alpha \in I_u)(\forall \beta \in I_v)(\{\alpha, \beta\} \not\subseteq I_u \cap I_v \implies \alpha \prec \beta)$.*

Proof. Let $\alpha \in I_u$ and $\beta \in I_v$ such that $\{\alpha, \beta\} \not\subseteq I_u \cap I_v$. We must prove that $\alpha \prec \beta$. Let $\gamma \in \Sigma^*$ be the longest string such that $\alpha = \alpha'\gamma$ and $\beta = \beta'\gamma$, for some $\alpha', \beta' \in \text{Pref}(\mathcal{L}(\mathcal{N}))$. If $\alpha' = \varepsilon$ the claim follows, therefore we can assume $|\alpha'| \geq 1$.

Let $\gamma = c_p \dots c_1$, with $c_i \in \Sigma$ for $i \in \{1, \dots, p\}$ $(p \geq 0)$, $\alpha' = a_q \dots a_1$, with $a_i \in \Sigma$ for $i \in \{1, \dots, q\}$ $(q \geq 1)$, and $\beta' = b_r \dots b_1$, with $b_i \in \Sigma$ for $i \in \{1, \dots, r\}$ $(r \geq 0)$.

Assume $|\gamma| > 0$. Since $\alpha \in I_u$ and $\beta \in I_v$, then there exist $u_1, v_1 \in Q$ such that $\alpha' c_p \dots c_2 \in I_{u_1}$, $\beta' c_p \dots c_2 \in I_{v_1}$, $u \in \delta(u_1, c_1)$ and $v \in \delta(v_1, c_1)$. By Axiom 2, we obtain $u_1 \leq v_1$. However, it cannot be $u_1 = v_1$ because this would imply

$\{\alpha, \beta\} \subseteq I_u \cap I_v$, so it must be $u_1 < v_1$. By iterating this argument, we conclude that there exist $u', v' \in Q$ such that $\alpha' \in I_{u'}$, $\beta' \in I_{v'}$ and $u' < v'$, and the same conclusion holds if $\gamma = \varepsilon$ as well.

Now, it cannot be $r = 0$ because this would imply $v' = s$, and $u' < s$ contradicts Remark 4.2. Hence, it must be $|\beta| \geq 1$. By Axiom 1, it must be $a_1 \preceq b_1$. At the same time, the definition of γ implies that it cannot be $a_1 = b_1$, so we obtain $a_1 \prec b_1$ and we can conclude $\alpha \prec \beta$.

□

Lemma 4.19 has an important implication (which will stand at the core of the encoding and indexing results in Sections 4.2): given a co-lex order on an NFA, then the sets I_α's are convex w.r.t this order.

Corollary 4.20. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let $\leq$ be a co-lex order on $\mathcal{N}$. If $\alpha \in Pref(\mathcal{L}(\mathcal{N}))$ then I_α is convex in $(Q, \leq)$.*

Proof. Suppose $u, z \in I_\alpha$ and let $v \in Q$ be such that $u < v < z$. We have to prove that $v \in I_\alpha$. If this were not true, we would have $\alpha \in (I_u \cap I_z) \setminus I_v$. Consider any $\beta \in I_v$. By Lemma 4.19 we would have $\alpha \prec \beta \prec \alpha$, a contradiction. □

A natural question is whether there is a connection between the notion of width and the complexity of regular expressions. The case width$(\mathcal{N}) = 1$ corresponds to the class of Wheeler automata [61]. In [5], it was shown that *Wheeler languages*, that is, regular languages recognized by Wheeler automata, are closed essentially only under intersection and under concatenation with a finite language. On the other hand, with the next two remarks we point out that our notion of width can easily be used to capture complementation, intersection and union.

Remark 4.21. Let $\mathcal{D}$ be a DFA. Then, there exists a DFA $\mathcal{D}'$ such that $\mathcal{L}(\mathcal{D}') = \Sigma^* \setminus \mathcal{L}(\mathcal{D})$ and width$(\mathcal{D}') \leq$ 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 $\leq_{\mathcal{D}'}$-comparable with any other state and therefore the width cannot increase by more than one.

Remark 4.22. 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, with $\text{width}(\mathcal{D}_1) = p_1$ and $\text{width}(\mathcal{D}_2) = p_2$. Let us 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 $\text{width}(\mathcal{D}) \leq p_1 \cdot p_2$ and a DFA $\mathcal{D}'$ such that $\mathcal{L}(\mathcal{D}') = \mathcal{L}(\mathcal{D}_1) \cup \mathcal{L}(\mathcal{D}_2)$ and $\text{width}(\mathcal{D}') \leq p_1 \cdot p_2 + p_1 + p_2$. In the following, we will implicitly use Lemma 4.12. Let $\{Q_1, \dots, Q_{p_1}\}$ and $\{Q_1^*, \dots, Q_{p_2}^*\}$ be a $\leq_{\mathcal{D}_1}$-chain partition of Q_1 and a $\leq_{\mathcal{D}_2}$-chain partition of Q_2, respectively. In order to build both $\mathcal{D}$ and $\mathcal{D}'$, 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 Remark 4.21), 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 $a \in \Sigma$. The difference between $\mathcal{D}$ and $\mathcal{D}'$ lies in how the set of final states is defined.

We define $\mathcal{D}$ by letting $F = F_1 \times F_2$ being the set of all final states, and then removing (i) all states that are not reachable from the initial state and (ii) 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 $i = 1, \dots, p_1$ and for every $j = 1, \dots, p_2$, 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 $\leq_{\mathcal{D}}$-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 $(u_1, v_1) <_{\mathcal{D}} (u_2, v_2)$.

Next, we define $\mathcal{D}'$ by letting $F' = (F_1 \times Q_2) \cup (Q_1 \times F_2)$ be the set of all final states, and then removing (i) all states that are not reachable from the initial state and removing (ii) all states that neither are final, nor allow to reach a final state.

Let $\mathcal{D}' = (Q', (s_1, s_2), \delta, F')$ the resulting DFA at the end of the construction. 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:

$$Q'_{i,j} = \{(u, v) \in Q' \mid u \in Q_i, v \in Q^*_j\}$$
$$Q'_{i,\star_2} = \{(u, \star_2) \in Q' \mid u \in Q_i\}$$
$$Q'_{\star_1,j} = \{(\star_1, v) \in Q' \mid v \in Q^*_j\}.$$

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

The above two remarks suggest that our notion of width is related with the structural complexity of the regular expressions accepting a given regular language (see also Remark 5.2 for the consequences of the above two remarks on the smallest-width DFA recognizing a regular language).

4.2 Path Coherence and Lower Bounds

The reason why Wheeler automata admit an efficient indexing mechanism lies in two key observations: (i) on finite total orders a convex set can be expressed with $O(1)$ words by specifying its endpoints, and (ii) the set of states reached by a path labeled with a given string α forms a convex set (*path-coherence*). We now show that the convex property holds true also for co-lex orders by generalizing the result in [61]. Intuitively, the next lemma generalizes from strings to automata the property that the set of all prefixes ending with a given pattern are consecutive in the prefix array (see Section 3.4).

Lemma 4.23 (Path-coherence)**.** *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, $\alpha \in \Sigma^*$, 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 still a (possibly empty) $\leq$-convex set.*

Proof. We proceed by induction on $|\alpha|$. If $|\alpha| = 0$, then $\alpha = \varepsilon$ and we are done. Now assume $|\alpha| \geq 1$. We can write $\alpha = \alpha' a$, with $\alpha' \in \Sigma^*$, $a \in \Sigma$. Let $u, v, z \in Q$ such that $u < v < z$ and $u, z \in U'$. We must prove that $v \in U'$. By the inductive hypothesis, the set U'' of all states in Q that can be reached from some state in U by following edges whose labels, when concatenated, yield α', is a $\leq$-convex set. In particular, there exist $u', z' \in U''$ such that $u \in \delta(u', a)$ and $z \in \delta(z', a)$. Since $a \in \lambda(u) \cap \lambda(z)$ and $u < v < z$, then $\lambda(v) = \{a\}$ (otherwise by Axiom 1 we would obtain a contradiction), so there exists $v' \in Q$ such that $v \in \delta(v', a)$. From $u < v < z$ and Axiom 2 we obtain $u' \leq v' \leq z'$. Since $u', z' \in U''$ and U'' is a $\leq$-convex set, then $v' \in U''$, and so $v \in U'$. □

Note that Corollary 4.20 also follows from Lemma 4.23 by picking $U = \{s\}$, because then $U' = I_\alpha$.

As we will see, the above result implies that indexing mechanism can be extended to arbitrary finite automata by updating *one* $\leq$-convex set for each character of the query pattern. This, however, does not mean that,in general, indexing can be performed as efficiently as on Wheeler automata: as we show next, in general it is not possible to represent a $\leq$-convex set in a partial order using constant space.

Lemma 4.24. *The following hold:*

1. *Any partial order $(V, \leq)$ of width p has at least 2^p distinct $\leq$-convex subsets.*

2. *For any n and p such that $1 \leq p \leq n$, there exists a partial order $(V, \leq)$ of width p and $|V| = n$ with at least $(n/p)^p$ distinct $\leq$-convex subsets.*

Proof. (1) Since V has width p, there exists an antichain A of cardinality p. It is easy to see that any subset $I \subseteq A$ is a distinct $\leq$-convex set. The bound 2^p follows. (2) Consider a partial order formed by p mutually-incomparable total orders V_i, all having n/p elements. Since any total order of cardinality n/p has $(n/p+1)(n/p)/2+1$ distinct convex sets and any combination of $\leq_{V_i}$-convex sets forms a distinct $\leq$-convex set, we obtain at least

$$\prod_{i=1}^{p}((n/p+1)(n/p)/2+1) \geq \prod_{i=1}^{p} n/p = (n/p)^p$$

distinct $\leq$-convex sets. □

Remark 4.25. Given an NFA $\mathcal{N}$ with n states and a co-lex order $\leq$ of width p on $\mathcal{N}$, Lemma 4.24 implies an information-theoretic lower bound of p bits for expressing a $\leq$-convex set, which increases to $\Omega(p \log(n/p))$ bits in the worst case. This means that, up to (possibly) a logarithmic factor, in the word RAM model $\Omega(p)$ time is needed to manipulate one $\leq$-convex set.

Remark 4.26. If $(V, \leq)$ is a partial order, $V' \subseteq V$ and U is a convex subset of $(V, \leq)$, then $U \cap V'$ is a convex set over the restricted partial order $(V', \leq_{V'})$. In particular, if $\{V_i \mid 1 \leq i \leq p\}$ is a partition of V then any $\leq$-convex set U is the disjoint union of p (possibly empty) sets $U_1, \ldots, U_p$, where $U_i = U \cap V_i$ is a convex set over the restriction $(V_i, \leq_{V_i})$.

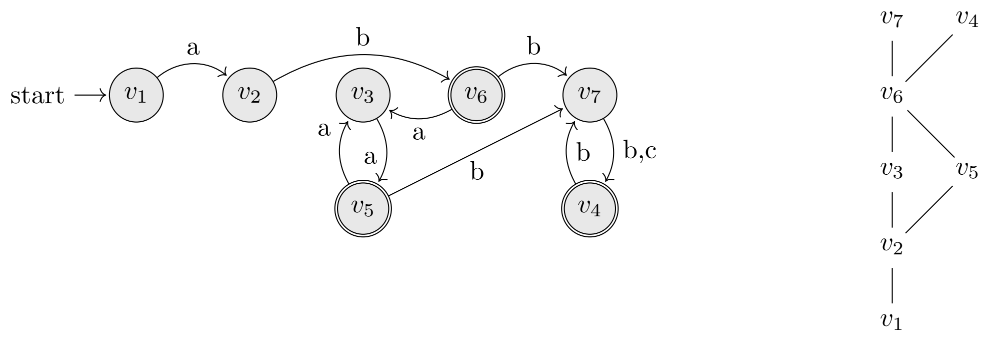
The above remarks motivate the following strategy. Letting p be the width of a partial order $\leq$, by Dilworth's theorem [45] there exists a $\leq$-chain partition $\{Q_i \mid 1 \leq i \leq p\}$ of Q into p chains. Then, Remark 4.26 implies that a $\leq$-convex set can be encoded by at most p convex sets, each contained in a distinct chain, using $O(p)$ words. This encoding is essentially optimal by Remark 4.25.

Using the above mentioned strategy we can now refine Lemma 4.23 and Corollary Corollary 4.20.

Lemma 4.27. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, $\{Q_i\}_{i=1}^{p}$ be a $\leq$-chain partition of $\mathcal{N}$, $\alpha \in \Sigma^*$, 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 $i = 1, \ldots, p$.*
In particular, if $\alpha \in Pref(\mathcal{L}(\mathcal{A}))$ then, I_α is the disjoint union of p (possibly empty) sets $I_\alpha^1, \ldots, I_\alpha^p$, where $I_\alpha^i = I_\alpha \cap Q_i$ is $\leq_{Q_i}$-convex, for $i = 1, \ldots, p$.

4.3 Encoding DFAs and Languages: the Automaton BWT (aBWT)

Let us define a representation of an automaton that is a generalization of the Burrows-Wheeler transform (BWT) of a string. We call this generalization the *automaton Burrows-Wheeler transform* (aBWT) and, just like the BWT of a string is an encoding of the string (that is, distinct strings have distinct BWTs), we will show that the aBWT of a DFA is an encoding of the DFA. We will also see that, on NFAs, the aBWT allow us to reconstruct the accepted language and to efficiently solve pattern matching queries, but in general it is not an encoding since it is not sufficient to reconstruct the NFA's topology. A variant, using slightly more space and encoding NFAs, will be presented in Section 4.5.



		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 4.3: 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 4.28). The aBWT can be linearized in five sequences, as shown here and in Example 4.29.

The aBWT is given for an automaton $\mathcal{N} = (Q, s, \delta, F)$ and it depends on a co-lex order $\leq$ endowed with a fixed $\leq$-chain partition $\{Q_i \mid 1 \leq i \leq p\}$ of Q (we assume $s \in Q_1$, so s is the first element of Q_1). An intuition behind the aBWT is provided in

Figure 4.3: after sorting the states in a total order which agrees with the co-lex order $\leq$ on pairs whose elements belong to the same class of the partition $\{Q_i \mid 1 \leq i \leq p\}$ and drawing the transition function's adjacency matrix in this order, we build five sequences collecting the chain borders (`CHAIN`), a boolean flag per state marking final states (`FINAL`), the states' in-degrees (`IN_DEG`), the states' out-degrees (`OUT_DEG`), and the states' labels and destination chains (`OUT`).

Definition 4.28 (aBWT of an automaton)**.** Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let $e = |\delta|$ be the number of $\mathcal{N}$-transitions. Let $\leq$ be a co-lex order on $\mathcal{N}$, and let $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-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 $Q = \{v_1, \dots, v_n\}$ 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* $\mathtt{aBWT}(\mathcal{N}, \leq, \{Q_i \mid 1 \leq i \leq p\})$ of $(\mathcal{N}, \leq, \{Q_i \mid 1 \leq i \leq p\})$ consists of the following sequences.

- `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.

- `FINAL` $\in \{0,1\}^n$ is such that the i-th bit is equal to 1 if and only if $v_i \in F$.

- `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.

- `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

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

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.

Example 4.29. The aBWT of $(\mathcal{D}, \leq, \{Q_i \mid 1 \leq i \leq 2\})$ in Figure 4.3 consists of the following sequences:

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

It is not hard to show that the aBWT generalizes all existing approaches [27, 55, 89, 21, 87, 61], for which $p = 1$ always holds (and so sequences `CHAIN` and the first components of the pairs in `OUT` are uninformative). For example, on strings also `OUT_DEG` and `IN_DEG` are uninformative (`FINAL` does not apply); the only sequence left is the concatenation of the second components of the pairs in `OUT`, that is, the classic Burrows-Wheeler transform (to be precise, its co-lexicographic variant, see Section 3.4).

In this section we will prove that if we only know the aBWT of an automaton we can reconstruct all the sets I_α^i of Lemma 4.27 (we recall that I_α^i is the set of all states in the i-th chain being connected with the source by a path labeled α), and in particular we can retrieve the language of the automaton. To this end, we first define some auxiliary sets of states of an NFA — $S(\alpha)$ and $L(\alpha)$ — and we prove that, for any $1 \leq i \leq p$, on the i-th chain the convex set corresponding to I_α^i lays between (the convex sets) $S(\alpha) \cap Q_i$ and $L(\alpha) \cap Q_i$. Intuitively, $S(\alpha)$ (respectively, $L(\alpha)$) is the set of all states u whose associated regular language I_u contains only strings co-lexicographically strictly smaller (respectively, larger) than α.

Definition 4.30. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, and $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q. Let $\alpha \in \Sigma^*$. Define:

$$S(\alpha) = \{u \in Q \mid (\forall \beta \in I_u)(\beta \prec \alpha)\}$$
$$L(\alpha) = \{u \in Q \mid (\forall \beta \in I_u)(\alpha \prec \beta)\}.$$

Moreover, for every $i = 1, \dots, p$ define $S_i(\alpha) = S(\alpha) \cap Q_i$ and $L_i(\alpha) = L(\alpha) \cap Q_i$.

In the following, we see a $\leq$-chain Q_i as an array of sorted elements, so $Q_i[j]$ and $Q_i[1, k]$ denote the j-th smallest state in Q_i and the k smallest states in Q_i, respectively.

In Lemma 4.31 we show that in order to compute I_α it will be sufficient to compute $S(\alpha)$ and $L(\alpha)$.

Lemma 4.31. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q, and $\alpha \in \Sigma^*$.*

1. *If $u, v \in Q$ are such that $u \leq v$ and $v \in S(\alpha)$, then $u \in S(\alpha)$. In particular, for every $i = 1, \dots, p$ there exists $0 \leq l_i \leq |Q_i|$ such that $S_i(\alpha) = Q_i[1, l_i]$ (namely, $l_i = |S_i(\alpha)|$).*

2. *If $u, v \in Q$ are such that $u \leq v$ and $u \in L(\alpha)$, then $v \in L(\alpha)$. In particular, for every $i = 1, \dots, p$ there exists $1 \leq r_i \leq |Q_i| + 1$ such that $L_i(\alpha) = Q_i[r_i, |Q_i|]$ (namely, $r_i = |Q_i| - |L_i(\alpha)| + 1$).*

3. *I_α, $S(\alpha)$, and $L(\alpha)$ are pairwise disjoint. In particular, it always holds that $l_i < r_i$.*

4. *Let $1 \leq i \leq p$. If $I_\alpha^i \neq \emptyset$, then $I_\alpha^i = Q_i[l_i + 1, r_i - 1]$, that is, $\{S_i(\alpha), I_\alpha^i, L_i(\alpha)\}$ is an ordered partition of Q_i.*

Proof. 1. Let $\beta \in I_u$. We must prove that $\beta \prec \alpha$. Now, if $\beta \in I_v$, from $v \in S(\alpha)$ we obtain $\beta \prec \alpha$. If $\beta \notin I_v$, then for any $\gamma \in I_v$ we have $\beta \prec \gamma$ by Lemma 4.19. Again, we have $\gamma \prec \alpha$, so we conclude $\beta \prec \alpha$.

2. Analogous to the previous point.

3. We have $I_\alpha \cap S(\alpha) = \emptyset$ because if $u \in I_\alpha$, then $\alpha \in I_u$, so $u \notin S(\alpha)$. Similarly, $I_\alpha \cap L(\alpha) = \emptyset$. Finally, we have $S(\alpha) \cap L(\alpha) = \emptyset$ because if there existed $u \in S(\alpha) \cap L(\alpha)$, then for any $\beta \in I_u$ (there exists at least one such β since $I_u \neq \emptyset$) we would obtain $\beta \prec \alpha \prec \beta$, a contradiction.

4. To begin with, let us prove that, for every $v \in I_\alpha$, (1) if $u < v$, then either $u \in I_\alpha$ or $u \in S(\alpha)$, and (2) if $v < z$, then either $z \in I_\alpha$ or $z \in L(\alpha)$. We only prove (1), the proof of (2) being analogous. Assume that $u \notin I_\alpha$, and let $\beta \in I_u$. We must prove that $\beta \prec \alpha$ and, since $\alpha \in I_v \setminus I_u$, this follows from Lemma 4.19.

   Now, let $1 \leq i \leq p$ be such that $I^i_\alpha \neq \emptyset$, and let us prove that $\{S_i(\alpha), I^i_\alpha, L_i(\alpha)\}$ is an ordered partition of Q_i. Consider $u \in I^i_\alpha$. Then, if $v \in Q_i \setminus I^i_\alpha$ we have either $v < u$ or $u < v$, hence what we have proved above implies that either $v \in S_i(\alpha)$ or $v \in L_i(\alpha)$. Therefore, if $I^i_\alpha \neq \emptyset$ then $\{S_i(\alpha), I^i_\alpha, L_i(\alpha)\}$ is an ordered partition of Q_i and point 4 follows.

□

Remark 4.32. Notice that if $I^i_\alpha = \emptyset$ then $\{S_i(\alpha), L_i(\alpha)\}$ is not, in general, an ordered partition of Q_i, as shown in Fig. 4.4.

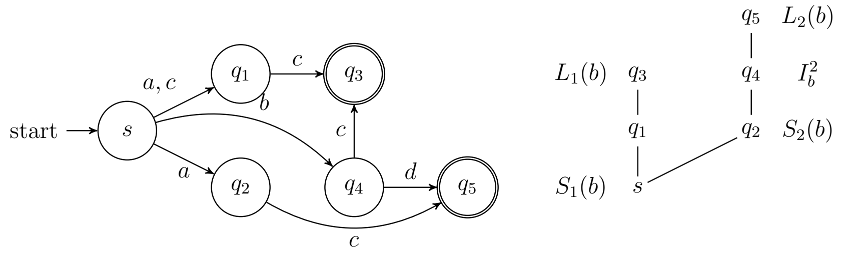


Figure 4.4: Consider the NFA in the figure and the Hasse diagram of a co-lex order $\leq$ with chain partition $Q_1 = \{s, q_1, q_3\}$ and $Q_2 = \{q_2, q_4, q_5\}$. If we consider the word b, then, on the one hand, $I^1_b = \emptyset$ and $\{S_1(b) = \{s\}, L_1(b) = \{q_3\}\}$ is not a partition of Q_1. On the other hand, since $I^2_b = \{q_4\} \neq \emptyset$, then $\{S_2(b) = \{q_2\}, I^2_b, L_1(b) = \{q_5\}\}$ is an ordered Q_2-partition.

Our next step is to show how to recursively compute the sets $S(\alpha)$ and $L(\alpha)$ defined above. We begin with the following two lemmas.

Lemma 4.33. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q, $\alpha' \in \Sigma^*$, $a \in \Sigma$, and $u \in Q$.*

1. *$u \in S(\alpha' a)$ if and only if (1) $max_{\lambda(u)} \preceq a$ and (2) if $u' \in Q$ is such that $u \in \delta(u', a)$, then $u' \in S(\alpha')$.*

2. *$u \in L(\alpha' a)$ if and only if (1) $a \preceq min_{\lambda(u)}$ and (2) if $u' \in Q$ is such that $u \in \delta(u', a)$, then $u' \in L(\alpha')$.*

Proof. Let us prove the first statement.

($\Rightarrow$) Let $c \in \Sigma$ such that $c \in \lambda(u) \setminus \{\#\}$. Let $u' \in Q$ such that $u \in \delta(u', c)$, and let $\beta' \in I_{u'}$. Then $\beta' c \in I_u$, so from $u \in S(\alpha' a)$ we obtain $\beta' c \prec \alpha' a$, which implies $c \preceq a$. Now assume that $c = a$. Suppose for sake of contradiction that $u' \notin S(\alpha')$. This means that there exists $\gamma' \in I_{u'}$ such that $\alpha' \preceq \gamma'$. This implies $\alpha' a \preceq \gamma' a$ and, since $\gamma' a \in I_u$, we obtain $u \notin S(\alpha' a)$, a contradiction.

($\Leftarrow$) Let $\beta \in I_u$. We must prove that $\beta \prec \alpha' a$. If $\beta = \varepsilon$ we are done, because $\varepsilon \prec \alpha' a$. Now assume that $\beta = \beta' b$. This means that there exists $u' \in Q$ such that $u \in \delta(u', b)$ and $\beta' \in I_{u'}$. We know that $b \preceq a$. If $b \prec a$, then $\beta \prec \alpha' a$ and we are done. If $b = a$, then $u' \in S(\alpha')$, so $\beta' \prec \alpha'$, which implies $\beta \prec \alpha' a$.

The proof of the second statement is analogus (the only difference being that in ($\Leftarrow$) it must necessarily be $\beta \neq \varepsilon$, because $a \preceq \min_{\lambda(u)}$). □

Lemma 4.34. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q, $\alpha' \in \Sigma^*$, and $a \in \Sigma$. Fix $1 \leq i \leq p$, and let $S_i(\alpha' a) = Q_i[1, l_i]$ and $L_i(\alpha' a) = Q_i[r_i, |Q_i|]$.*

1. *If $u' \in S(\alpha')$ and $u \in Q_i$ are such that $u \in \delta(u', a)$, then $u \in Q_i[1, \min\{l_i + 1, |Q_i|\}]$.*

2. *If $u' \in L(\alpha')$ and $u \in Q_i$ are such that $u \in \delta(u', a)$, then $u \in Q_i[\max\{r_i - 1, 1\}, |Q_i|]$.*

Proof. We only prove the first statement, the proof of the second statement being entirely analogous. We can assume $l_i < |Q_i| - 1$, otherwise the conclusion is trivial. If $a \prec \max(\lambda(Q_i[l_i + 1]))$ the conclusion is immediate by Axiom 1, so we can assume $\max(\lambda(Q_i[l_i + 1])) \preceq a$. We know that $Q_i[l_i + 1] \notin S(\alpha' a)$, so by Lemma 4.33 there exists $v' \in Q$ such that $Q_i[l_i + 1] \in \delta(v', a)$ and $v' \notin S(\alpha')$. Suppose for sake of contradiction that $Q_i[l_i + 1] < u$. By Axiom 2 we obtain $v' \leq u'$. From $u' \in S(\alpha')$ and Lemma 4.31 we conclude $v' \in S(\alpha')$, a contradiction. □

The following definition is instrumental in giving an operative variant of Lemma 4.33 (i.e. Lemma 4.36) to be used in our algorithms.

Definition 4.35. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, and $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q. Let $U \subseteq Q$. We denote by $\mathtt{in}(U, a)$ the number of edges labeled with character a that enter states in U:

$$\mathtt{in}(U, a) = |\{(u', u) \mid u' \in Q, u \in U, u \in \delta(u', a)\}|.$$

We denote by $\mathtt{out}(U, i, a)$ the number of edges labeled with character a that leave states in U and enter the i-th chain:

$$\mathtt{out}(U, i, a) = |\{(u', u) \mid u' \in U, u \in Q_i, u \in \delta(u', a)\}|.$$

In the following lemma we show how to compute the convex sets corresponding to $S_i(\alpha' a)$ and $L_i(\alpha' a)$, for every $i = 1, \ldots, p$, using the above definitions.

Lemma 4.36. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q, $\alpha' \in \Sigma^*$, $a \in \Sigma$, and $\alpha = \alpha' a$. For every $j = 1, \ldots, p$, let $S_j(\alpha') = Q_j[1, l'_j]$ and $L_j(\alpha') = Q_j[r'_j, |Q_j|]$. Fix $1 \leq i \leq p$, and let $S_i(\alpha) = Q_i[1, l_i]$ and $L_i(\alpha) = Q_i[r_i, |Q_i|]$.*

1. *Let $x = \mathtt{out}(S(\alpha), i, a) = \sum_{j=1}^{p} \mathtt{out}(Q_j[1, l'_j], i, a)$. Then, l_i is the largest integer $0 \leq k \leq |Q_i|$ such that (i) $\mathtt{in}(Q_i[1, k], a) \leq x$, and (ii) if $k \geq 1$, then $\max(\lambda(Q_i[k])) \preceq a$.*

2. *Let $y = \mathtt{out}(L(\alpha), i, a) = \sum_{j=1}^{p} \mathtt{out}(Q_j[r'_j, |Q_j|], i, a)$. Then, r_i is the smallest integer $1 \leq k \leq |Q_i| + 1$ such that (i) $\mathtt{in}(Q_i[k, |Q_i|], a) \leq y$, and (ii) if $k \leq |Q_i|$, then $a \preceq \min(\lambda(Q_i[k]))$.*

Proof. Again, we just prove the first statement since the proof of the second one is analogous.

Let z_i be the largest integer $0 \leq k \leq |Q_i|$ such that (i) $\mathtt{in}(Q_i[1, k], a) \leq x$, and (ii) if $k \geq 1$, then $\max(\lambda(Q_i[k])) \preceq a$. We want to prove that $l_i = z_i$.

($\leq$) The conclusion is immediate if $l_i = 0$, so we can assume $l_i \geq 1$. It will suffice to prove that $\mathtt{in}(Q_i[1, l_i], a) \leq x$ and $\max(\lambda(Q_i[l_i])) \preceq a$. This follows from Lemma 4.33 and the definition of x.

($\geq$) The conclusion is immediate if $l_i = |Q_i|$, so we can assume $l_i < |Q_i|$. We only have to prove that if $l_i + 1 \leq k \leq |Q_i|$, then either $\texttt{in}(Q_i[1,k],a) > x$ or $\max(\lambda(Q_i[k])) \succ a$. By Axiom 1, it will suffice to prove that we have $\texttt{in}(Q_i[1,l_i + 1],a) > x$ or $\max(\lambda(Q_i[l_i + 1])) \succ a$. Assume that $\max(\lambda(Q_i[l_i + 1])) \preceq a$. Since $Q_i[l_i + 1] \notin S(\alpha)$, by Lemma 4.33 there exists $v' \in Q$ such that $Q_i[l_i + 1] \in \delta(v', a)$ and $v' \notin S(\alpha')$. We will conclude that $\texttt{in}(Q_i[1,l_i+1],a) > x$ if we show that for every $j = 1, \dots, p$, if $u' \in Q_j$ and $u \in Q_i$ are such that $u' \in Q_j[1, l'_j]$ (and so $u' \in S(\alpha')$) and $u \in \delta(u', a)$, then it must be $u \in Q_i[1, l_i + 1]$. This follows from Lemma 4.34. □

We now use Lemma 4.36 to retrieve the language of the automaton starting from the aBWT.

Lemma 4.37. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, and $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q, with $s \in Q_1$. Let $v_1, \dots, v_n$ be the ordering of Q defined in Definition 4.28. Assume that we do not know $\mathcal{N}$, but we only know $\texttt{aBWT}(\mathcal{N}, \leq, \{Q_i \mid 1 \leq i \leq p\})$. Then, for every $\alpha \in \Sigma^*$ we can retrieve the set $\{i \in \{1, \dots, n\} \mid \alpha \in I_{v_i}\}$, which yields $\delta(s, \alpha)$.*

Proof. First, let us prove that for every $k = 1, \dots, n$, we can retrieve the labels — with multiplicities — of all edges entering v_k. By scanning CHAIN we can retrieve the integers k_1 and k_2 such that the states in chain Q_i are $v_{k_1}, v_{k_1+1}, \dots, v_{k_2-1}, v_{k_2}$. By scanning OUT, which stores the label and the destinational chain of each edge, we can retrieve how many edges enter chain Q_i, and we can retrieve the labels - with multiplicities - of all edges entering chain Q_i. By considering the substring of IN_DEG between the $(k_1 - 1)$-th one and the k_2-th one we can retrieve the in-degrees of all states in chain Q_i. Since we know the labels - with multiplicities - of all edges entering chain Q_i and the in-degrees of all states in chain Q_i, by Axiom 1 we can retrieve the labels - with multiplicities - of all edges entering each node in chain Q_i: order the multiset of incoming edge labels, scan the nodes in Q_i in order, and assign the labels to each node in Q_i in agreement with their in-degrees.

Let us prove that for every $i = 1, \dots, p$ we can retrieve the integers l_i and r_i such that $S_i(\alpha) = Q_i[1, l_i]$ and $L_i(\alpha) = Q_i[r_i, |Q_i|]$. We proceed by induction on $|\alpha|$. If $|\alpha| = 0$, then $\alpha = \varepsilon$, so for every $i = 1, \dots, p$ we have $l_i = 0$, for every $i = 2, \dots, p$ we have $r_i = 1$, and $r_1 = 2$. Now, assume $|\alpha| > 0$. We can write $\alpha = \alpha' a$, with $\alpha' \in \Sigma^*$

and $a \in \Sigma$. By the inductive hypothesis, for $j = 1, \dots, p$ we know the integers l'_j and r'_j such that $S_j(\alpha') = Q_j[1, l'_j]$ and $L_j(\alpha') = Q_j[r'_j, |Q_j|]$. Notice that by using `OUT_DEG` and `OUT` we can compute $\texttt{out}(Q_j[1, l'_j], i, a)$ for every $j = 1, \dots, p$ (see Definition 4.35). Since we know the labels - with multiplicities - of all edges entering each state, we can also compute $\texttt{in}(Q_i[c, d], a)$ and $\lambda(Q_i[k]))$ for every $i = 1, \dots, p$, $1 \leq c \leq d \leq |Q_i|$, and $1 \leq k \leq |Q_i|$. By Lemma 4.36 we conclude that we can compute l_i and r_i for every $i = 1, \dots, p$, and we are done.

Now, let us prove that for every $\alpha \in \Sigma^*$ we can retrieve the set $\{i \in \{1, \dots, n\} \mid \alpha \in I_{v_i}\}$. We proceed by induction on $|\alpha|$. If $|\alpha| = 0$, then $\alpha = \varepsilon$ and $\{i \in \{1, \dots, n\} \mid \varepsilon \in I_{v_i}\} = \{1\}$. Now, assume $|\alpha| > 0$. We can write $\alpha = \alpha' a$, with $\alpha' \in \Sigma^*$ and $a \in \Sigma$. By the inductive hypothesis, we know $\{i \in \{1, \dots, n\} \mid \alpha' \in I_{v_i}\}$. For every $i = 1, \dots, p$ we decide whether $I^i_\alpha \neq \emptyset$ by using $\{i \in \{1, \dots, n\} \mid \alpha' \in I_{v_i}\}$, `OUT_DEG` and `OUT`. If $I^i_\alpha \neq \emptyset$, then by Lemma 4.31 we know that $I^i_\alpha = Q_i[l_i + 1, r_i - 1]$, and we know how to determine l_i and r_i. Hence, we can easily compute $\{i \in \{1, \dots, n\} \mid \alpha \in I_{v_i}\}$. □

Corollary 4.38. *If* $\texttt{aBWT}(\mathcal{N}, \leq, \{Q_i \mid 1 \leq i \leq p\}) = \texttt{aBWT}(\mathcal{N}', \leq', \{Q'_i \mid 1 \leq i \leq p'\})$, *then:*

1. $p = p'$;
2. *for every* $1 \leq i \leq p$ *we have* $|Q_i| = |Q'_i|$;
3. $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{N}')$.

Proof. Since $\mathcal{N}$ and $\mathcal{N}'$ share the sequence `CHAIN`, it must be $p = p'$ and $|Q_i| = |Q'_i|$ for every i . Fix a string $\alpha \in \Sigma^*$. Then, by Lemma 4.37 we conclude that the set $\{i \in \{1, \dots, n\} \mid \alpha \in I_{v_i}\}$ is the same for both $\mathcal{N}$ and $\mathcal{N}'$. Since $\mathcal{N}$ and $\mathcal{N}'$ share also the sequence `FINAL`, we conclude that α is accepted by $\mathcal{N}$ if and only if it is accepted by $\mathcal{N}'$. □

Corollary 4.38 ensures that $\texttt{aBWT}(\mathcal{N}, \leq, \{Q_i \mid 1 \leq i \leq p\})$ is enough to reconstruct the language $\mathcal{L}(\mathcal{N})$ of an NFA. Similarly to the string case, however (where the BWT is augmented with light data structures in order to achieve efficient indexing with the *FM-index* [56]), we will need additional data structures built on top of the aBWT in order to solve efficiently string matching queries. In Section 4.4 we will show

how to extend the FM-index to automata by augmenting the aBWT with light data structures.

While Corollary 4.38 establishes that the aBWT preserves the automaton's language, it does not state anything about whether it preserves the automaton's topology. In fact we now show that this is not, in general, the case.

Definition 4.39. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA. We say that $\mathcal{N}$ is *distinguished by its paths* if for every $v \in Q$ there exists $\alpha \in \Sigma^*$ such that $I_\alpha = \{v\}$.

Remark 4.40. If an NFA is not distinguished by its paths, then in general we cannot retrieve its topology from its aBWT, because there exist two non-isomorphic NFAs having the same aBWT: see Figure 4.5 for an example.

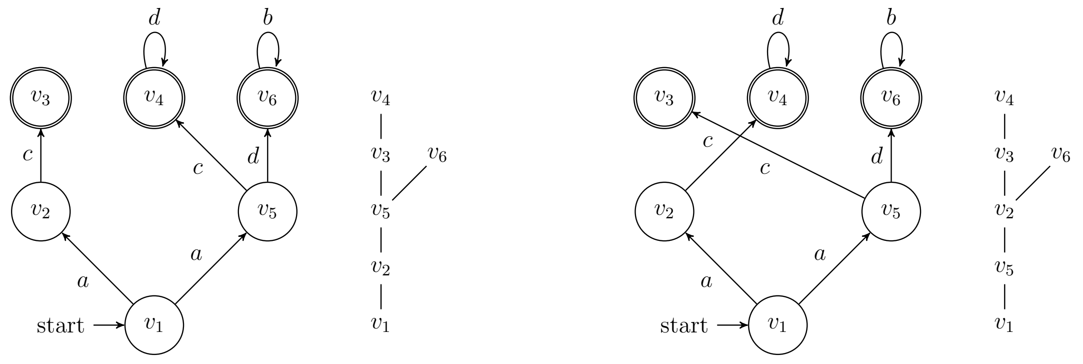


Figure 4.5: Consider the two non-isomorphic NFAs $\mathcal{N}_1$ and $\mathcal{N}_2$ in the figure, both with set of states $Q = \{v_1, v_2, v_3, v_4, v_5, v_6\}$. Let $\leq_1$ and $\leq_2$ be the maximal co-lex orders given by the Hasse diagrams shown in the figure, and notice that in both cases if we consider $Q_1 = \{v_1, v_2, v_3, v_4\}$ and $Q_2 = \{v_5, v_6\}$ we obtain a minimum-size chain partition $\mathcal{Q} = \{Q_1, Q_2\}$. It is easy to check that $\texttt{aBWT}(\mathcal{N}_1, \leq_1, \mathcal{Q}) = \texttt{aBWT}(\mathcal{N}_2, \leq_2, \mathcal{Q})$ because in both cases we have $\texttt{CHAIN} = 100010$, $\texttt{FINAL} = 001101$, $\texttt{OUT\_DEG} = 0010110100101$, $\texttt{OUT} = (1, a)(2, a)(1, c)(1, d)(1, c)(2, d)(2, b)$, and $\texttt{IN\_DEG} = 1010100101001$. Consistently with Theorem 4.41, we have that $\mathcal{N}_1$ and $\mathcal{N}_2$ are not distinguished by their paths.

Let us prove that $\texttt{aBWT}(\mathcal{N}, \leq, \{Q_i \mid 1 \leq i \leq p\})$ is a one-to-one encoding for the class of automata which are distinguished by paths.

Theorem 4.41. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, and $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q, with $s \in Q_1$. Assume that we do not know $\mathcal{N}$, but we only know $\texttt{aBWT}(\mathcal{N}, \leq, \{Q_i \mid 1 \leq i \leq p\})$. Then, we can decide whether $\mathcal{N}$ is distinguished by its paths and, if so, we can retrieve $\mathcal{N}$.*

Proof. Let $v_1, \ldots, v_n$ be the ordering of Q in Definition 4.28. We know that v_1 is the initial state and for every $j = 1, \ldots, n$ we can decide whether v_j is final by using `FINAL`. Now, for every $\alpha \in \Sigma^*$, let $C_\alpha = \{i \in \{1, \ldots, n\} \mid \alpha \in I_{v_i}\}$. Notice that we can compute C_α for every $\alpha \in \Sigma^*$ by Lemma 4.37. Consider a list that contains pairs of the form (α, C_α). Initially, the list contains only $(\varepsilon, \{1\})$. Remove recursively an element (α, C_α) and for every $a \in \Sigma$ add $(\alpha a, C_{\alpha a})$ to the list if and only if $C_{\alpha a}$ is nonempty and it is not the second element of a pair which is or has already been in the list. This implies that after at most $|\Sigma| \cdot 2^n$ steps the list is empty, and any non-empty C_α has been the second element of some pair in the list. Then, we conclude that $\mathcal{N}$ is distinguished by its paths if and only for every $k = 1, \ldots, n$ the set $\{v_k\}$ has been the second element of some pair in the list. In particular, if $\mathcal{N}$ is distinguished by its paths, then for every $k = 1, \ldots, n$ we know a string $\alpha' \in \Sigma^*$ such that $C_{\alpha'} = \{k\}$. We are only left with showing that we can use α' to retrieve all edges leaving v_k. Fix a character $a \in \Sigma$. Then, compute $C_{\alpha' a}$ using again Lemma 4.37. Then, v_k has $|C_{\alpha' a}|$ outgoing edges labeled a, whose indexes are given by $C_{\alpha' a}$. □

Since any DFA is distinguished by its paths, we obtain the following corollary:

Corollary 4.42. *The aBWT is a one-to-one encoding over DFAs.*

By counting the number of bits required by the aBWT, we can determine the size of our encoding for NFAs that are distinguished by their paths:

Corollary 4.43. *Let $\mathcal{N}$ be an NFA that is distinguished by its paths (for example, a DFA), and let $p = width(\mathcal{N})$. Then, we can store $\mathcal{N}$ using $\log(p\sigma) + O(1)$ bits per transition. If $\mathcal{N}$ is a DFA, this space can also be expressed as (at most) $\sigma \log(p\sigma) + O(\sigma)$ bits per state. If $\mathcal{N}$ is an NFA, this space can also be expressed as (at most) $2p\sigma \log(p\sigma) + O(p\sigma)$ bits per state.*

Proof. The bound of $\log(p\sigma) + O(1)$ bits per transition follows directly from Definition 4.28 and Theorem 4.41. Letting $|\delta|$ denote the number of transitions and n denote the number of states, on DFAs the naive bound $|\delta| \leq n\sigma$ holds; this allows us to derive the bound of $\sigma \log(p\sigma) + O(\sigma)$ bits per state on DFAs. On arbitrary NFAs, we can use the bound $|\delta| \leq 2p\sigma n$ implied by Lemma 4.7, yielding the bound of $2p\sigma \log(p\sigma) + O(p\sigma)$ bits per state on NFAs. □

We stress that, while not being an encoding of the NFA, the aBWT still allows to reconstruct the language of the automaton and — as we will show in the next subsection — to solve pattern matching queries by returning the convex set of all states reached by a path labeled with a given input query string. In Section 4.5 we will augment the aBWT and obtain an injective encoding of arbitrary NFAs.

4.4 An Index for NFAs and Languages

We now show how to support subpath queries by augmenting the aBWT with light data structures and turning it into an index. In fact, our structure is a generalization of the FM-index to arbitrary automata.

Solving subpath queries on an NFA requires finding the subset $T(\alpha)$ of its states reached by some path labeled by the query string α. In turn, note that there is a path labeled α ending in state u if and only if I_u contains a string suffixed by α. This motivates the following definition.

Definition 4.44. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q, and $\alpha \in \Sigma^*$. Define:

$$T(\alpha) = \{u \in Q \mid (\exists \beta \in I_u)(\alpha \dashv \beta)\},$$
$$R(\alpha) = S(\alpha) \cup T(\alpha) = \{u \in Q \mid (\forall \beta \in I_u)(\beta \prec \alpha) \vee (\exists \beta \in I_u)(\alpha \dashv \beta)\}.$$

Moreover, for every $i = 1, \ldots, p$ define $T_i(\alpha) = T(\alpha) \cap Q_i$ and $R_i(\alpha) = R(\alpha) \cap Q_i$.

Intuitively, $T(\alpha)$ contains all states reached by a path labeled with α, while $R(\alpha)$ contains all the states that are either reached by a string suffixed by α, or only reached by strings co-lexicographically smaller than α. Note that the goal of an index solving pattern matching queries is to compute the (cardinality of the) set $T(\alpha)$. The aim of the next lemma is to show that, once a co-lex order is fixed, $T(\alpha)$ always forms a range (a convex set). Indeed, we now prove a counterpart of Lemma 4.31, where for any $\alpha \in \Sigma^*$ we showed that $S_i(\alpha) = Q_i[1, l_i]$, for some $0 \leq l_i \leq |Q_i|$.

Lemma 4.45. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, and $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q. Let $\alpha \in \Sigma^*$. Then:*

1. *$S(\alpha) \cap T(\alpha) = \emptyset$.*

2. *$T(\alpha)$ is $\leq$-convex.*

3. *If $u, v \in Q$ are such that $u \leq v$ and $v \in R(\alpha)$, then $u \in R(\alpha)$. In particular, for every $i = 1, \dots, p$ there exists $0 \leq t_i \leq |Q_i|$ such that $T_i(\alpha) = Q_i[|S_i(\alpha)| + 1, t_i]$ (namely, $t_i = |R_i(\alpha)|$).*

Proof. 1. If $u \in T(\alpha)$, then there exists $\beta \in I_u$ such that $\alpha \dashv \beta$. In particular, $\alpha \preceq \beta$, so $u \notin S(\alpha) = \{v \in Q \mid (\forall \beta \in I_v)(\beta \prec \alpha)\}$.

2. It follows from Lemma 4.23 by picking $U = Q$.

3. If $v \in S(\alpha)$, then $u \in S(\alpha)$ by Lemma 4.31 and so $u \in R(\alpha)$. Now, assume that $v \in T(\alpha)$. If $u \in T(\alpha)$ we are done. If $u \notin T(\alpha)$ (and therefore $u \neq v$), we want to prove that $u \in S(\alpha)$, which implies $u \in R(\alpha)$. Fix $\beta \in I_u$; we must prove that $\beta \prec \alpha$. Since $v \in T(\alpha)$, then there exists $\gamma \in \Sigma^*$ such that $\gamma\alpha \in I_v$. Moreover, $\gamma\alpha \notin I_u$ because $u \notin T(\alpha)$. Since $u < v$, by Lemma 4.19 we conclude $\beta \prec \gamma\alpha$. Since $u \notin T(\alpha)$ implies $\alpha \not\dashv \beta$, from $\beta \prec \gamma\alpha$ we conclude $\beta \prec \alpha$.

□

We now show how to recursively compute the range on each chain Q_i corresponding to $R_i(\alpha)$. Note that, by Lemma 4.36, we can assume to be able to recursively compute the range on each chain Q_i corresponding to $S_i(\alpha)$. A computational variant of Lemma 4.45 will allow us to compute $T_i(\alpha)$ on each chain $1 \leq i \leq p$. Each recursive step of this procedure — dubbed here *forward search* (see Section 3.4) — will stand at the core of our index.

Lemma 4.46 (Forward search). *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, $\leq$ be a co-lex order on $\mathcal{N}$, and $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q. Let $\alpha' \in \Sigma^*$, $a \in \Sigma$ and $\alpha = \alpha' a$. For every $1 \leq i, j \leq p$, let:*

- *$S_j(\alpha') = Q_j[1, l'_j]$;*
- *$R_j(\alpha') = Q_j[1, t'_j]$;*
- *$S_i(\alpha) = Q_i[1, l_i]$;*
- *$R_i(\alpha) = Q_i[1, t_i]$.*

Fix $1 \le i \le p$, and define $c = \sum_{j=1}^{p} \texttt{out}(Q_j[1, l'_j], i, a)$ and $d = \sum_{j=1}^{p} \texttt{out}(Q_j[1, t'_j], i, a)$. Then $d \ge c$, and:

1. *If $d = c$, then $T_i(\alpha) = \emptyset$ and so $t_i = l_i$.*

2. *If $d > c$, then $T_i(\alpha) \ne \emptyset$ and t_i, with $1 \le t_i \le |Q_i|$, is the smallest integer such that $\texttt{in}(Q_i[1, t_i], a) \ge d$.*

In particular, l_i can be computed by means of Lemma 4.36, and:

$$T_i(\alpha) = Q_i[l_i + 1, t_i].$$

Proof. Since $S(\alpha) \subseteq R(\alpha)$, we have $d \ge c$. Now, notice that $T_i(\alpha) \ne \emptyset$ if and only there exists an edge labeled a leaving a state in $T(\alpha')$ and reaching chain Q_i, if and only if $d > c$. Hence, in the following we can assume $T_i(\alpha) \ne \emptyset$. In particular, this implies $l_i < |Q_i|$ and $t_i \ge l_i + 1$. By Lemma 4.34 all edges labeled a, leaving a state in $S(\alpha')$, and reaching chain Q_i must end in $Q_i[1, l_i + 1]$. At the same time, since $T_i(\alpha) \ne \emptyset$, the definition of t_i implies that there exists $v' \in T(\alpha')$ (and so $v' \in R(\alpha')$) such that $Q_i[t_i] \in \delta(v', a)$. Hence, the conclusion will follow if we prove that if $u', u \in Q$ are such that $u \in Q_i[1, t_i - 1]$ and $u \in \delta(u', a)$, then $u' \in R(\alpha')$. Since $u < Q_i[t_i]$, from Axiom 2 we obtain $u' \le v'$ and since $v' \in R(\alpha')$, from Lemma 4.45 we conclude $u' \in R(\alpha')$. $\square$

We are ready to present the main result of this section (Theorem 4.47): a linear-space index supporting subpath queries on any automaton in time proportional to $p^2 \cdot \log\log(p\sigma)$ per query character (p being the automaton's width).

Theorem 4.47 (aBWT-index of a finite-state automaton). *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA on alphabet Σ of size $\sigma = |\Sigma| \le e^{O(1)}$, where $e = |\delta|$ is the number of $\mathcal{N}$-transitions. Assume that we are given a $\le$-chain partition $\{Q_i \mid 1 \le i \le p\}$, for some co-lex order $\le$ of width p. Then, in expected time $O(e \log\log \sigma)$, we can build a data structure using $e \log(p\sigma)(1 + o(1)) + O(e)$ bits that, given a query string $\alpha \in \Sigma^m$, answers the following queries in $O(m \cdot p^2 \cdot \log\log(p\sigma))$ time:*

1. *compute the set $T(\alpha)$ of all states reached by a path on $\mathcal{N}$ labeled α, represented by means of p ranges on the chains in $\{Q_i \mid 1 \le i \le p\}$;*

2. *compute the set I_α of all states reached by a path labeled with α originating in the source, represented by means of p ranges on the chains in $\{Q_i \mid 1 \leq i \leq p\}$ and, in particular, decide whether $\alpha \in \mathcal{L}(\mathcal{N})$.*

Proof. Let $n = |Q|$. In this proof we assume that the states of Q have been sorted like in Definition 4.28: if $\pi(v)$, for $v \in Q$, is the unique integer such that $v \in Q_{\pi(v)}$, then we consider the ordering $v_1, \dots, v_n$ of Q such that for every $1 \leq i < j \leq n$ it holds $\pi(v_i) < \pi(v_j) \vee (\pi(v_i) = \pi(v_j) \wedge v_i < v_j)$. Moreover, we assume that $s \in Q_1$ (again like in Definition 4.28), so $s = v_1$. For every $i = 1, \dots, p$, let $e_i = |\{(u, v, a) \mid \delta(u, a) = v, u \in Q, v \in Q_i, a \in \Sigma\}|$ be the number of edges entering the i-th chain, let $\Sigma_i = (\bigcup_{u \in Q_i} \lambda(u)) \setminus \{\#\}$ be the set of characters labeling edges entering the i-th chain, and let $\sigma_i = |\Sigma_i|$.

We store the following data structures:

- One fully-indexable succinct dictionary (Lemma 2.3) on each Σ_i to map $\Sigma_i \subseteq [0, \sigma - 1]$ to $[0, \sigma_i - 1]$. The total number of required bits is $\sum_{i=1}^{p}(\sigma_i \log(\sigma/\sigma_i) + O(\sigma_i)) \leq \sum_{i=1}^{p}(e_i \log(\sigma/\sigma_i) + O(e_i)) = e \log \sigma - \sum_{i=1}^{p}(e_i \log \sigma_i) + O(e)$. As a consequence, we can solve rank, select, predecessor, strict-successor and membership queries on each dictionary in $O(\log\log(\sigma/\sigma_i)) \subseteq O(\log\log\sigma)$ time.

- The bitvector CHAIN $\in \{0, 1\}^n$ of Definition 4.28 represented by the data structure of Lemma 2.1. The number of required bits is $nH_0(\text{CHAIN})(1 + o(1)) + O(n) = O(n) \subseteq O(e)$. As a consequence, we can solve rank and select queries on CHAIN in $O(1)$ time. In particular, in $O(1)$ time we can compute $|Q_i|$, for $i = 1, \dots, p$, because $|Q_i| = \text{CHAIN}.\textit{select}(i+1, 1) - \text{CHAIN}.\textit{select}(i, 1)$.

- The bitvector FINAL $\in \{0, 1\}^n$ of Definition 4.28 represented by the data structure of Lemma 2.1. The number of required bits is again $O(n) \subseteq O(e)$.

- The bitvector `OUT_DEG` $\in \{0, 1\}^{e+n}$ of Definition 4.28 represented by the data structure of Lemma 2.1. The number of required bits is $(n+e)H_0(\texttt{OUT\_DEG})(1 + o(1)) + O(n+e) \subseteq O(e)$. As a consequence, we can solve rank and select queries on `OUT_DEG` in $O(1)$ time.

- The string `OUT` $\in ([1, p] \times \Sigma)^e$ of Definition 4.28 represented by the data structure of Lemma 2.2 (the assumption on the size on the alphabet in Lemma 2.2 is satisfied because $|[1,p] \times \sigma| = p \cdot \sigma \leq n \cdot \sigma \leq (e+1) \cdot \sigma = e^{O(1)}$). The number of required bits is $eH_0(\texttt{OUT})(1+o(1)) + O(e)$. We will bound the quantity $eH_0(\texttt{OUT})$ by exhibiting a prefix-free encoding of OUT. The key idea is that if $(i,c) \in [1,p] \times \Sigma$ occurs in OUT, then it must be $c \in \Sigma_i$, so we can encode (i,c) by using $\lceil \log(p+1) \rceil \leq \log p + 1$ bits encoding i, followed by $\lceil \log(\sigma_i + 1) \rceil \leq \log \sigma_i + 1$ bits encoding c (note that this part depends on i). We clearly obtain a prefix code, so we conclude $eH_0(\texttt{OUT}) \leq \sum_{i=1}^{p} e_i(\log p + \log \sigma_i + O(1)) = e \log p + \sum_{i=1}^{p}(e_i \log \sigma_i) + O(e)$ bits. Observing that $\sum_{i=1}^{p}(e_i \log \sigma_i) \leq e \log \sigma$, we conclude that the number of required bits for OUT is bounded by $eH_0(\texttt{OUT})(1+o(1)) + O(e) = \left(e \log p + \sum_{i=1}^{p}(e_i \log \sigma_i) + O(e)\right)(1+o(1)) + O(e) \leq (1+o(1)) e \log p + \sum_{i=1}^{p}(e_i \log \sigma_i) + o(e \log \sigma) + O(e)$ bits. Notice that in $O(\log\log(p\sigma))$ time we can solve rank and select queries on OUT (that is, queries $\texttt{OUT}.rank(j,(i,c))$ and $\texttt{OUT}.select(j,(i,c))$) for all $1 \leq i \leq p$ and for all $c \in \Sigma$. Indeed, given i and c, we first check whether $c \in \Sigma_i$ by solving a membership query on the dictionary for Σ_i in $O(\log\log\sigma)$ time. If $c \notin \Sigma_i$, then we immediately conclude that $\texttt{OUT}.rank(j,(i,c)) = 0$ and $\texttt{OUT}.select(j,(i,c))$ is undefined. If $c \in \Sigma_i$, then (i,c) appears in OUT, so the conclusion follows from Lemma 2.2.

- The bitvector $\texttt{IN\_DEG} \in \{0,1\}^{e+n}$ of Definition 4.28 represented by the data structure of Lemma 2.1. The number of required bits is again $O(e)$. As a consequence, we can solve rank and select queries on IN_DEG in $O(1)$ time.

- A bitvector IN′, represented by the data structure of Lemma 2.1, built as follows. We sort all edges (v_j, v_i, c) by end state v_i and, if the end state is the same, by label c. Then, we build a string $\texttt{IN} \in \Sigma^e$ by concatenating all labels of the sorted edges. Finally, $\texttt{IN}' \in \{0,1\}^e$ is the bitvector such that $\texttt{IN}'[k] = 1$ if and only if $k = 1$ or $\texttt{IN}[k] \neq \texttt{IN}[k-1]$ or the k-th edge and the $(k-1)$-th edge reach distinct chains. The number of required bits is $O(e)$.

For an example, consider the automaton of Figure 4.3. All sequences except for bitvector IN′ are reported in Example 4.29. To build bitvector IN′, we first build the string IN of all incoming labels of the sorted edges: $\texttt{IN} = aaabcabbbb$. Then, bitvector IN′ marks with a bit '1' (i) the first character of each maximal unary substring in IN, and (ii) the characters of IN labeling the first edge in each chain: $\texttt{IN}' = 1001111000$.

By adding up the space of all components (note that the terms $-\sum_{i=1}^{p}(e_i \log \sigma_i)$ in the dictionaries Σ_i and $\sum_{i=1}^{p}(e_i \log \sigma_i)$ in sequence `OUT` cancel out), we conclude that our data structures take at most $e\log(p\sigma)(1+o(1))+O(e)$ bits.

We proceed by showing how to solve queries 1 (string matching) and 2 (membership).

(1) Let us prove that, given a query string $\alpha \in \Sigma^m$, we can use our data structure to compute, in time $O(m \cdot p^2 \cdot \log\log(p\sigma))$, the set $T(\alpha)$ of all states reached by a α-path on $\mathcal{N}$, presented by p convex sets on the chains $\{Q_i \mid 1 \leq i \leq p\}$. By Lemma 4.46, it will suffice to show how to compute $R(\alpha)$ and $S(\alpha)$. We can recursively compute each $R(\alpha)$ and $S(\alpha)$ in time proportional to m by computing $R(\alpha')$ and $S(\alpha')$ for all prefixes α' of α. Hence, we only have to show that we can update $R(\alpha')$ and $S(\alpha')$ with a new character, in $O(p^2 \cdot \log\log(p\sigma))$ time. We start with the empty prefix ε, whose corresponding sets are $R(\varepsilon) = Q$ and $S(\varepsilon) = \emptyset$. For the update we apply Lemmas 4.36 and 4.46. An inspection of the two lemmas reveals the we can update $R(\alpha')$ and $S(\alpha')$ with a new character by means of $O(p^2)$ calls to the following queries.

- (op1) for any $1 \leq i, j \leq p$, $1 \leq k \leq |Q_j|$, and $a \in \Sigma$, compute $\texttt{out}(Q_j[1,k], i, a)$;
- (op2) for any $1 \leq i \leq p$, $a \in \Sigma$, and $h \geq 0$, find the largest integer $0 \leq k \leq |Q_i|$ such that $\texttt{in}(Q_i[1,k], a) \leq h$;
- (op3) for any $1 \leq i \leq p$, $a \in \Sigma$, and $z \geq 1$, find the smallest integer $1 \leq t \leq |Q_i|$ such that $\texttt{in}(Q_i[1,t], a) \geq z$, if it exists, otherwise report that it does not exist.
- (op4) for any $1 \leq i \leq p$ and $a \in \Sigma$, find the largest integer $0 \leq h \leq |Q_i|$ such that, if $h \geq 1$, then $\max(\lambda(Q_i[h])) \preceq a$.

As a consequence, we are left to show that we can solve each query in $O(\log\log(p\sigma))$ time.

(op1). The states in $Q_j[1,k]$ correspond to the convex set of all states (in the total order $v_1, \ldots, v_n$ of Definition 4.28) whose endpoints are v_l and v_r, where $l =$ CHAIN.*select*$(j, 1)$ and $r = l + k - 1$. Considering the order of edges used to define `OUT`, the set of all edges leaving a state in $Q_j[1,k]$ forms a convex set in `OUT` and in `OUT_DEG`. Define:

- $x = \texttt{OUT\_DEG}.rank(\texttt{OUT\_DEG}.select(l-1,1),0)$;
- $y = \texttt{OUT\_DEG}.rank(\texttt{OUT\_DEG}.select(r,1),0)$.

Notice that x is equal to the number of edges leaving all states before v_l, while y is the number of edges leaving all states up to v_r included. As a consequence, we have $x \leq y$. If $x = y$, then there are no edges leaving states in $Q_j[1,k]$ and we can immediately conclude $\texttt{out}(Q_j[1,k], i, a) = 0$. Assuming $x < y$, $x+1$ and y are the endpoints of the convex set of all edges leaving states in $Q_j[1,k]$. Hence, we are left with counting the number of such edges labeled a and reaching chain Q_i. Notice that $\texttt{OUT}.rank(x,(i,a))$ is the number of all edges labeled a and reaching chain i whose start state comes before v_l, whereas $\texttt{OUT}.rank(y,(i,a))$ is the number of all edges labeled a and reaching i whose start state comes before or is equal to v_r. We can then conclude that $\texttt{out}(Q_j[1,k], i, a) = \texttt{OUT}.rank(y,(i,a)) - \texttt{OUT}.rank(x,(i,a))$.

(op2). First, we check whether $a \in \Sigma_i$ by a membership query on the dictionary for Σ_i. If $a \notin \Sigma_i$, we immediately conclude that the largest k with the desired properties is $k = |Q_i|$. Assume $a \in \Sigma_i$ and notice that the states in Q_i correspond to the convex set of all states whose endpoints are v_l and v_r, where $l = \text{CHAIN}.select(i,1)$ and $r = \text{CHAIN}.select(i+1,1) - 1$. Considering the order of edges used to define $\texttt{IN}$, the set of all edges entering a state in $Q_j[1,k]$ forms a convex set in $\texttt{IN}$ and in $\texttt{IN\_DEG}$. Define

- $x = \texttt{IN\_DEG}.rank(\texttt{IN\_DEG}.select(l-1,1),0)$;
- $y = \texttt{IN\_DEG}.rank(\texttt{IN\_DEG}.select(r,1),0)$.

Notice that x is equal to the number of edges reaching all states before v_l, while y is the number of edges reaching all states coming before or equal to v_r. Since $a \in \Sigma_i$, we have $x < y$, and $x+1$ and y are the endpoints of the convex set of all edges reaching a state in Q_i. The next step is to determine the smallest edge labeled a reaching a state in Q_i. First, notice that the number of characters smaller than or equal to a in Σ_i can be retrieved, by Lemma 2.3, as $\Sigma_i.rank(a)$. Notice that $f = \texttt{IN}'.rank(x,1)$ yields the number of 0-runs in $\texttt{IN}'$ pertaining to chains before chain Q_i so that, since we know that $a \in \Sigma_i$, then $g = \texttt{IN}'.select(f + \Sigma_i.rank(a), 1)$ yields the smallest edge labeled a in the convex set of all edges reaching a state in Q_i. We distinguish two cases for the parameter h of op2:

- $h = 0$. In this case, the largest k with the desired properties is equal to the position on chain Q_i of the state reached by the g-th edge minus one. The index of the state reached by the g-th edge is given by $p = \texttt{IN\_DEG}.rank(\texttt{IN\_DEG}.select(g, 0), 1) + 1$, so the largest k with the desired properties is $k = p - l$.

- $h > 0$. The quantity $h' = \texttt{IN}'.select(f + \Sigma_i.rank(a) + 1, 1) - g = \texttt{IN}'.select(f + \Sigma_i.rank(a) + 1, 1) - \texttt{IN}'.select(f + \Sigma_i.rank(a), 1)$ yields the number of edges labeled a in the convex set of all edges reaching a state in Q_i. If $h' \leq h$, then we conclude that the largest k is $|Q_i|$. Hence, assume that $h' > h$. We immediately obtain that the $(h+1)$-th smallest edge labeled a reaching a state in Q_i is the $(g+h)$-th edge, and the largest k with the desired properties is equal to the position on chain Q_i of the state reached by the $(g+h)$-th edge minus one. Analogously to case 1, the index of the state reached by this edge is given by $p = \texttt{IN\_DEG}.rank(\texttt{IN\_DEG}.select(g+h, 0), 1) + 1$, so the largest k with the desired properties is $k = p - l$.

(op3). We simply use operation (op2) to compute the largest integer $0 \leq k \leq |Q_i|$ such that $\texttt{in}(Q_i[1,k], a) \leq z - 1$. If $k = |Q_i|$, then the desired integer does not exist, otherwise it is equal to $k + 1$.

(op4). We first decide whether $\Sigma_i.succ(a)$ is defined. If it is not defined, then the largest integer with the desired property is $|Q_i|$. Now assume that $\Sigma_i.succ(a)$ is defined. Then the largest integer with the desired property is simply the largest integer $0 \leq k \leq |Q_i| - 1$ such that $\texttt{in}(Q_i[1,k], \Sigma_i.succ(a)) \leq 0$, which can be computed using (op2).

(2) Let us prove that, given a query string $\alpha \in \Sigma^m$, we can use our data structure to compute I_α in $O(m \cdot p^2 \cdot \log\log(p\sigma))$ time, represented as p ranges on the chains in $\{Q_i \mid 1 \leq i \leq p\}$. We claim that it will suffice to run the same algorithm used in the previous point, starting with $R = \{v_1\}$ and $S = \emptyset$. Indeed, consider the automaton $\mathcal{N}'$ obtained from $\mathcal{N}$ by adding a new initial state v_0 and adding exactly one edge from v_0 to v_1 (the old initial state) labeled with $\#$, a character smaller than every character in the alphabet Σ. Let $\leq'$ the co-lex order on $\mathcal{N}'$ obtained from $\leq$ by adding the pair $\{(v_0, v_1)\}$, and consider the $\leq'$-chain partition obtained from $\{Q_i \mid 1 \leq i \leq p\}$ by adding v_0 to Q_1. It is immediate to notice that for every $k = 1, \dots, n$ and for every string $\alpha \in \Sigma^*$ we have that $v_k \in I_\alpha$ on $\mathcal{N}$ if and only if $v_k \in T(\#\alpha)$ on $\mathcal{N}'$. Since

v_0 has no incoming edges and on $\mathcal{N}'$ we have $R(\#) = \{v_0, v_1\}$ and $S(\#) = \{v_0\}$, the conclusion follows. Now, given I_α, we can easily check whether $\alpha \in \mathcal{L}(\mathcal{N})$. Indeed, for every $i = 1, \dots, p$ we know the integers l_i and t_i such that $I^i_\alpha = Q_i[l_i + 1, t_i]$, and we decide whether some of these states are final by computing $f = \text{CHAIN}.\textit{select}(i, 1)$, and then checking whether $\text{FINAL}.\textit{rank}(f + l_i - 1, 1) - \text{FINAL}.\textit{rank}(f + t_i - 1, 1)$ is larger than zero. □

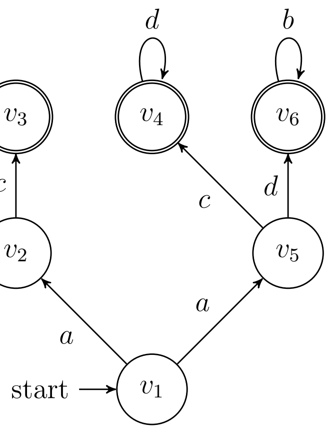
4.5 Encoding NFAs

We now show a simple extension of the aBWT of Definition 4.28, yielding an injective encoding of NFAs. It turns out that in order to achieve these goals it is sufficient to collect the components of the aBWT and, in addition, the origin chain of every edge (see Figure 4.6 for an example):

Definition 4.48. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let $e = |\delta|$ be the number of $\mathcal{N}$-transitions. Let $\leq$ be a co-lex order on $\mathcal{N}$, and let $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition of Q, where w.l.o.g. $s \in Q_1$. Let $\pi(v)$ and $Q = \{v_1, \dots, v_n\}$ be the map and the total state order defined in Definition 4.28. Define a new sequence $\texttt{IN\_CHAIN} \in [1, p]^e$, storing the edges' origin chains, as follows. Sort all edges (v_j, v_i, c) by increasing destination index i, breaking ties by label c and then by origin index j. Then, $\texttt{IN\_CHAIN}$ is obtained by concatenating the elements $\pi(v_j)$ for all edges (v_j, v_i, c) sorted in this order.

The following lemma presents a function that will ultimately allow us to show that our augmented aBWT is indeed an injective encoding on NFAs.

Lemma 4.49. *Let $1 \leq i, j \leq p$ and $a \in \Sigma$. Let w be the number of edges labeled with character a leaving any state in Q_j and entering any state in Q_i. Let $B_{j,i,a} = (f_1, f_2, \dots, f_w)$ be state indices such that (i) $1 \leq f_1 \leq f_2 \leq \cdots \leq f_w \leq n$, (ii) if $1 \leq k \leq n$ occurs in $B_{j,i,a}$, then $\pi(v_k) = j$ and (iii) if $1 \leq k \leq n$ occurs $t \geq 1$ times in $B_{j,i,a}$, then there exist exactly t edges labeled with character a leaving v_k and entering a state in Q_i. Let $C_{j,i,a} = (g_1, g_2, \dots, g_w)$ be state indices such that (i) $1 \leq g_1 \leq g_2 \leq \cdots \leq g_w \leq n$, (ii) if $1 \leq h \leq n$ occurs in $C_{j,i,a}$, then $\pi(v_h) = i$ and (iii) if $1 \leq h \leq n$ occurs $t \geq 1$ times in $C_{j,i,a}$, then there exist exactly t edges labeled with*



		CHAIN	1	0	0	0	1	0
		FINAL	0	0	1	1	0	1
		IN_DEG	1	01	01	001	01	001
		IN_CHAIN		1	1	21	1	22
OUT_DEG	OUT		1	2	3	4	5	6
001	(1,a),(2,a)	1		(1,1,a)			(1,2,a)	
01	(1,c)	2			(1,1,c)			
1		3						
01	(1,d)	4				(1,1,d)		
001	(1,c),(2,d)	5				(2,1,c)		(2,2,d)
01	(2,b)	6						(2,2,b)

Figure 4.6: Augmented aBWT of an NFA (the one in Figure 4.5 on the left), using the chain partition $\{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6\}\}$. In addition to the aBWT of Definition 4.28, we add a sequence IN_CHAIN collecting the origin chain of every edge. For each labeled edge (u, v, a), in the adjacency matrix we show the triple $(\pi(u), \pi(v), a)$, that is, the origin chain, destination chain, and label of the edge (the matrix is visually divided in 4 sectors, corresponding to all combinations of origin and destination chains). Vector IN_CHAIN collects vertically the incoming chains, that is, the first component of each triple in the corresponding column of the adjacency matrix.

character a entering v_h and leaving a state in Q_j. Then, $\{(v_{f_\ell}, v_{g_\ell}, a) \mid 1 \leq \ell \leq w\}$ is the set of all edges labeled with character a, leaving a state in Q_j and entering a state in Q_i.

Proof. Consider the set of all edges labeled with characater a, leaving a state in Q_j and entering a state in Q_i. Then, $B_{j,i,a}$ is obtained by picking and sorting all start states of these edges, and $C_{j,i,a}$ is obtained by picking and sorting all end states of these edges. The conclusion follows from Axiom 2. □

Lemma 4.49 allows reconstructing the topology of a NFA starting from our augmented aBWT, as we show in the next lemma.

Lemma 4.50. *The aBWT of Definition 4.28, in addition to sequence* `IN_CHAIN` *of Definition 4.48, is a one-to-one encoding over the NFAs.*

Proof. From `CHAIN` and `FINAL` we can retrieve the chain of each state, and we can decide which states are final. We only have to show how to retrieve the set $\{(v_f, v_g, a) \mid v_g \in \delta(v_f, a),\ v_f, v_g \in Q, a \in \Sigma\}$ of all NFA's transitions. By Lemma 4.49, we only have to prove that for every $1 \leq i, j \leq p$ and for every $a \in \Sigma$ we can retrieve $B_{j,i,a}$ and $C_{j,i,a}$. We will use ideas similar to those employed in the proof of Lemma 4.37.

Let us show how to retrieve $B_{j,i,a}$ for every $1 \leq i, j \leq p$ and for every $a \in \Sigma$. Fix i, j and a. From `OUT_DEG` we can retrieve the number of edges leaving each state in the j-th chain. Then, the definition of `OUT` implies that, for every state in the j-th chain, we can retrieve the label and the destination chain of each edge leaving the state, so we can decide how many times a state in the j-th chain occurs in $B_{j,i,a}$.

Let us show how to retrieve $C_{j,i,a}$ for every $1 \leq i, j \leq p$ and for every $a \in \Sigma$. Fix i, j and a. From `IN_DEG` we can retrieve the number of edges entering each state in the i-th chain. From `OUT` we can retrieve all characters (with multiplicities) labeling some edge entering the ith-chain. As a consequence, Axiom 1 implies that, for every state in the i-th chain, we can retrieve the label of each edge entering the state. Moreover, the definition of `IN_CHAIN` implies that, for every state in the i-th cain, we can retrieve the the start chain of each edge entering the state, so we can decide how many times a state in the i-th chain occurs in $C_{j,i,a}$. □

By analyzing the space required by our extension of the aBWT and applying Lemma 4.50, we obtain:

Corollary 4.51. *Let $\mathcal{N}$ be an NFA, and let $p = width(\mathcal{N})$. Then, we can store $\mathcal{N}$ using $\log(p^2\sigma) + O(1)$ bits per transition. This space can also be expressed as (at most) $2p\sigma\log(p^2\sigma) + O(p\sigma)$ bits per state.*

Proof. The bound of $\log(p^2\sigma) + O(1)$ bits per transition follows easily from the definitions of aBWT (Definition 4.28) and `IN_CHAIN` (Definition 4.48). In order to bound this space as a function of the number of states, we use the bound $|\delta| \leq 2p\sigma n$ implied by Lemma 4.7, where $|\delta|$ is the number of transitions. □

In Theorem 4.47 we provided an aBWT-index supporting pattern matching queries on any NFA. Being a superset of the aBWT, our (indexed) augmented aBWT can clearly support the same operations (in the same running times) of Theorem 4.47, albeit using additional $\log p$ bits per edge. Actually, these operations turn out to be much simpler on the augmented aBWT than on the aBWT (thanks to the new sequence IN_CHAIN). This is possible because by means of our augmented aBWT, given *any* convex set U of states (represented by means of p ranges on the chains) and a character c, one can compute the convex set of all states that can be reached from U by following edges labeled c (see Lemma 4.23). However, the queries' running times remain the same as in Theorem 4.47 so we will not describe them here.

Chapter 5

Co-lex Orders: Nondeterminism, Entanglement and Minimization

In Chapter 4 we studied the relationship between automata and co-lex orders. In this chapter we study the relationship between regular languages and co-lex orders, that is, we aim to study the properties of a regular languages (and not a specific automaton) through co-lex orders.

We start by defining the width of a regular language based on the co-lex orders of the automata recognizing it.

Definition 5.1. Let $\mathcal{L}$ be a regular language.

1. The *non-deterministic co-lex width* of $\mathcal{L}$, denoted by $\text{width}^N(\mathcal{L})$, is the smallest integer p for which there exists an NFA $\mathcal{N}$ such that $\mathcal{L}(\mathcal{N}) = \mathcal{L}$ and $\text{width}(\mathcal{N}) = p$.
2. 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 $\text{width}(\mathcal{D}) = p$.

In Example 4.18 we showed that the width of an automaton may depend on the total order $\preceq$ on the alphabet. In Example 5.27 below, we will show that the deterministic and nondeterministic widths of a language may also depend on the order $\preceq$ on the alphabet.

On the grounds of Remarks 4.21 and 4.22, we observe the following relations, which allow us to conclude that already constant-width regular languages form an interesting class:

Remark 5.2. Let $\mathcal{L}$, $\mathcal{L}_1$, and $\mathcal{L}_2$ be any regular languages. Then:

1. $\text{width}^D(\Sigma^* \setminus \mathcal{L}) \leq \text{width}^D(\mathcal{L}) + 1$
2. $\text{width}^D(\mathcal{L}_1 \cap \mathcal{L}_2) \leq \text{width}^D(\mathcal{L}_1) \cdot \text{width}^D(\mathcal{L}_2)$

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

These inequalities are a direct consequence of Remarks 4.21 and 4.22 by starting from smallest-width DFAs recognizing $\mathcal{L}$, $\mathcal{L}_1$, and $\mathcal{L}_2$.

By Remark 5.2, if $\mathcal{L}$ can be written as the boolean combination of a constant number of Wheeler languages (for example, of a constant number of finite languages), then $\text{width}^N(\mathcal{L}) \leq \text{width}^D(\mathcal{L}) \in O(1)$. Furthermore, this bound holds for any total order $\preceq$ on the alphabet if the starting languages are finite (because finite languages are Wheeler independent of the alphabet order).

5.1 The Powerset Construction

In this section we show that the notion of width can be used to prove some crucial relationships between an NFA $\mathcal{N}$ and the powerset automaton $\text{Pow}(\mathcal{N})$ obtained from $\mathcal{N}$. First, we bound the width of $\text{Pow}(\mathcal{N})$ in terms of the width of $\mathcal{N}$ and prove that the number of $\text{Pow}(\mathcal{N})$'s states is exponential in $\text{width}(\mathcal{N})$ rather than in the number of $\mathcal{N}$'s states. This implies that several problems easy on DFAs but difficult on NFAs are in fact fixed-parameter tractable with respect to the width.

Recall that, given an NFA $\mathcal{N} = (Q, s, \delta, F)$, the powerset construction algorithm builds an equivalent DFA $\text{Pow}(\mathcal{N}) = (Q^*, s^*, \delta^*, F^*)$ defined as:

- $Q^* = \{I_\alpha \mid \alpha \in \text{Pref}(\mathcal{L}(\mathcal{N}))\}$;
- $s^* = \{s\}$;
- $\delta^*(I_\alpha, a) = I_{\alpha a}$ for all $\alpha \in \Sigma^*$ and $a \in \Sigma$ such that $\alpha a \in \text{Pref}(\mathcal{L}(\mathcal{N}))$;
- $F^* = \{I_\alpha \mid \alpha \in \mathcal{L}(\mathcal{N})\}$.

For $\alpha, \alpha' \in \text{Pref}(\mathcal{L}(\mathcal{N}))$ we have:

$$\delta^*(s^*, \alpha') = I_\alpha \iff I_{\alpha'} = I_\alpha$$

and defining as usual $I_{u^*} = \{\alpha \in \text{Pref}(\mathcal{L}(\text{Pow}(\mathcal{N}))) \mid u^* \in \delta^*(s^*, \alpha)\}$ for $u^* \in Q^*$, we have that for $\alpha \in \text{Pref}(\mathcal{L}(\mathcal{N}))$:

$$I_{I_\alpha} = \{\alpha' \in \text{Pref}(\mathcal{L}(\mathcal{N})) \mid I_{\alpha'} = I_\alpha\}. \tag{5.1}$$

We start with a characterization of the maximum co-lex order on Pow($\mathcal{N}$) (which exists by Lemma 4.9).

Lemma 5.3. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let $Pow(\mathcal{N}) = (Q^*, s^*, \delta^*, F^*)$ be the powerset automaton obtained from $\mathcal{N}$. Let $\leq_{Pow(\mathcal{N})}$ be the maximum co-lex order on $Pow(\mathcal{N})$. Then, for $I_\alpha \neq I_\beta$:*

$$(I_\alpha <_{Pow(\mathcal{N})} I_\beta) \iff (\forall \alpha', \beta' \in Pref(\mathcal{L}(\mathcal{N})))((I_{\alpha'} = I_\alpha) \land (I_{\beta'} = I_\beta) \to \alpha' \prec \beta')$$

Moreover, let $\leq$ be a co-lex order on $\mathcal{N}$, and fix $\alpha, \beta \in Pref(\mathcal{L}(\mathcal{N}))$. Then:

$$(\exists u \in I_\alpha)(\exists v \in I_\beta)(\{u, v\} \not\subseteq I_\alpha \cap I_\beta \land u < v) \implies (I_\alpha <_{Pow(\mathcal{N})} I_\beta).$$

Proof. The first part follows immediately from the characterization of the maximum co-lex order over a DFA (Lemma 4.9) and Equation 5.1. Let us prove the second part. Consider $u \in I_\alpha$ and $v \in I_\beta$ such that $\{u, v\} \not\subseteq I_\alpha \cap I_\beta$ and $u < v$. We prove that $I_\alpha <_{\mathrm{Pow}(\mathcal{N})} I_\beta$ using the characterization of $<_{\mathrm{Pow}(\mathcal{N})}$ given in the first part of the proof. Fix $\alpha', \beta' \in \mathrm{Pref}(\mathcal{L}(\mathcal{N}))$ such that $I_{\alpha'} = I_\alpha$ and $I_{\beta'} = I_\beta$. We must prove that $\alpha' \prec \beta'$. From the hypothesis it follows $u \in I_{\alpha'}, v \in I_{\beta'}$, and $\{u, v\} \not\subseteq I_{\alpha'} \cap I_{\beta'}$ so that $\alpha' \in I_u, \beta' \in I_v$, and $\{\alpha', \beta'\} \not\subseteq I_u \cap I_v$ hold. Hence, $\alpha' \prec \beta'$ follows from $u < v$ and Lemma 4.19. $\square$

We can now prove the main result of this section.

Theorem 5.4. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let $Pow(\mathcal{N}) = (Q^*, s^*, \delta^*, F^*)$ be the powerset automaton obtained from $\mathcal{N}$. Let $n = |Q|$ and $p = width(\mathcal{N})$. Then:*

1. *$width(Pow(\mathcal{N})) \leq 2^p - 1$;*

2. *$|Q^*| \leq 2^p(n - p + 1) - 1$.*

Proof. Let $\leq$ be a co-lex order on $\mathcal{N}$ such that $width(\leq) = p$, and let $\{Q_i \mid 1 \leq i \leq p\}$ be a $\leq$-chain partition. Let $\leq_{\mathrm{Pow}(\mathcal{N})}$ be the maximum co-lex order on Pow($\mathcal{N}$). For every nonempty $K \subseteq \{1, \ldots, p\}$, define:

$$\mathcal{I}_K = \{I_\alpha \mid (\forall i \in \{1, \ldots, p\})(I_\alpha \cap Q_i \neq \emptyset \iff i \in K)\}.$$

Notice that Q^* is the disjoint union of all $\mathcal{I}_K$. More precisely:

$$Q^* = \bigsqcup_{\emptyset \neq K \subseteq \{1, \ldots, p\}} \mathcal{I}_K. \tag{5.2}$$

Let us prove that each $\mathcal{I}_K$ is a $\leq_{\mathrm{Pow}(\mathcal{N})}$-chain. Fix $I_\alpha, I_\beta \in \mathcal{I}_K$, with $I_\alpha \neq I_\beta$. We must prove that I_α and I_β are $\leq_{\mathrm{Pow}(\mathcal{N})}$-comparable. Since $I_\alpha \neq I_\beta$, there exists either $u \in I_\alpha \setminus I_\beta$ or $v \in I_\beta \setminus I_\alpha$. Assume that there exists $u \in I_\alpha \setminus I_\beta$ (the other case is analogous). In particular, let $i \in \{1, \dots, p\}$ be the unique integer such that $u \in Q_i$. Since $I_\alpha, I_\beta \in \mathcal{I}_K$, from the definition of $\mathcal{I}_K$ it follows that there exists $v \in I_\beta \cap Q_i$. Notice that $\{u, v\} \not\subseteq I_\alpha \cap I_\beta$ (so in particular $u \neq v$), and since $u, v \in Q_i$ we conclude that u and v are $\leq$-comparable. By Lemma 5.3 we conclude that I_α and I_β are $\leq_{\mathrm{Pow}(\mathcal{N})}$-comparable.

1. The first part of the theorem follows from Equation 5.2, because each $\mathcal{I}_K$ is a $\leq_{\mathrm{Pow}(\mathcal{N})}$-chain and there are $2^p - 1$ choices for K.

2. Let us prove the second part of the theorem. Fix $\emptyset \neq K \subseteq \{1, \dots, p\}$. For every $I_\alpha \in \mathcal{I}_K$ and for every $i \in K$, let m^i_α be the smallest element of $I_\alpha \cap Q_i$ (this makes sense because $(Q_i, \leq)$ is totally ordered), and let M^i_α be the largest element of $I_\alpha \cap Q_i$. Fix $I_\alpha, I_\beta \in \mathcal{I}_K$, and note the following:

   (a) Assume that for some $i \in K$ it holds $m^i_\alpha < m^i_\beta \vee M^i_\alpha < M^i_\beta$. Then, it must be $I_\alpha <_{\mathrm{Pow}(\mathcal{N})} I_\beta$. Indeed, assume that $m^i_\alpha < m^i_\beta$ (the other case is analogous). We have $m^i_\alpha \in I_\alpha$, $m^i_\beta \in I_\beta$, $\{m^i_\alpha, m^i_\beta\} \not\subseteq I_\alpha \cap I_\beta$ and $m^i_\alpha < m^i_\beta$, so the conclusion follows from Lemma 5.3. Equivalently, we can state that if $I_\alpha <_{\mathrm{Pow}(\mathcal{N})} I_\beta$ then $(\forall i \in K)(m^i_\alpha \leq m^i_\beta \wedge M^i_\alpha \leq M^i_\beta)$.

   (b) Assume that for some $i \in K$ it holds $m^i_\alpha = m^i_\beta \wedge M^i_\alpha = M^i_\beta$. By Corollary 4.20, the sets I_α and I_β are convex in $(Q, \leq)$. This implies that $I_\alpha \cap Q_i$ and $I_\beta \cap Q_i$ are $\leq_{Q_i}$-convex, and having the same minimum and maximum they must be equal, that is, $I_\alpha \cap Q_i = I_\beta \cap Q_i$.

   (c) Assume that $(\forall i \in K)(m^i_\alpha = m^i_\beta \wedge M^i_\alpha = M^i_\beta)$. Then, it must be $I_\alpha = I_\beta$. Indeed, from point (b) we obtain $(\forall i \in K)(I_\alpha \cap Q_i = I_\beta \cap Q_i)$, so $I_\alpha = \bigcup_{i \in K}(I_\alpha \cap Q_i) = \bigcup_{i \in K}(I_\beta \cap Q_i) = I_\beta$. Notice that we can equivalently state that if $I_\alpha \neq I_\beta$, then $(\exists i \in K)(m^i_\alpha \neq m^i_\beta \vee M^i_\alpha \neq M^i_\beta)$.

   Fix $I_\alpha, I_\beta \in \mathcal{I}_K$. Now it is easy to show that:

$$\begin{aligned} I_\alpha <_{\mathrm{Pow}(\mathcal{N})} I_\beta \iff & (\forall i \in K)(m^i_\alpha \leq m^i_\beta \wedge M^i_\alpha \leq M^i_\beta) \wedge \\ & \wedge (\exists i \in K)(m^i_\alpha < m^i_\beta \vee M^i_\alpha < M^i_\beta). \end{aligned} \tag{5.3}$$

Indeed, ($\Leftarrow$) follows from point (a). As for ($\Rightarrow$), notice that $(\forall i \in K)(m^i_\alpha \leq m^i_\beta \wedge M^i_\alpha \leq M^i_\beta)$ again follows from point (a), whereas $(\exists i \in K)(m^i_\alpha < m^i_\beta \vee M^i_\alpha < M^i_\beta)$ follows from point (c).

Let $|m^i_\alpha|$ and $|M^i_\alpha|$ be the positions of m^i_α and M^i_α in the total order $(Q_i, \leq)$ (so $|m^i_\alpha|, |M^i_\alpha| \in \{1, \dots, |Q_i|\}$). For every $I_\alpha \in \mathcal{I}_K$, define:

$$T(I_\alpha) = \sum_{i \in K} (|m^i_\alpha| + |M^i_\alpha|).$$

By Equation 5.3, we have that $I_\alpha <_{\text{Pow}(\mathcal{N})} I_\beta$ implies $T(I_\alpha) < T(I_\beta)$, so since $\mathcal{I}_K$ is a $<_{\text{Pow}(\mathcal{N})}$-chain we have that $|\mathcal{I}_K|$ is bounded by the values that $T(I_\alpha)$ can take. For every $I_\alpha \in \mathcal{I}_K$ we have $2|K| \leq T(I_\alpha) \leq 2\sum_{i \in K} |Q_i|$ (because $|m^i_\alpha|, |M^i_\alpha| \in \{1, \dots, |Q_i|\}$), so:

$$|\mathcal{I}_K| \leq 2\sum_{i \in K} |Q_i| - 2|K| + 1. \tag{5.4}$$

From Equations 5.2 and 5.4, we obtain:

$$\begin{aligned} |\mathcal{Q}^*| &= \sum_{\emptyset \subsetneq K \subseteq \{1,\dots,p\}} |\mathcal{I}_K| \leq \sum_{\emptyset \subsetneq K \subseteq \{1,\dots,p\}} (2\sum_{i \in K} |Q_i| - 2|K| + 1) \\ &= 2 \sum_{\emptyset \subsetneq K \subseteq \{1,\dots,p\}} \sum_{i \in K} |Q_i| - 2 \sum_{\emptyset \subsetneq K \subseteq \{1,\dots,p\}} |K| + \sum_{\emptyset \subsetneq K \subseteq \{1,\dots,p\}} 1. \end{aligned}$$

Notice that $\sum_{\emptyset \subsetneq K \subseteq \{1,\dots,p\}} \sum_{i \in K} |Q_i| = 2^{p-1} \sum_{i \in \{1,\dots,p\}} |Q_i| = 2^{p-1} n$ because every $i \in \{1, \dots, p\}$ occurs in exactly 2^{p-1} subsets of $\{1, \dots, p\}$. Analogously, we can write $\sum_{\emptyset \subsetneq K \subseteq \{1,\dots,p\}} |K| = 2^{p-1} p$ and $\sum_{\emptyset \subsetneq K \subseteq \{1,\dots,p\}} 1 = 2^p - 1$, We conclude:

$$|\mathcal{Q}^*| \leq 2^p n - 2^p p + 2^p - 1 = 2^p (n - p + 1) - 1.$$

□

As a first consequence of Theorem 5.4 we start comparing the non-deterministic and deterministic width hierarchies of regular languages. Clearly, for every regular language $\mathcal{L}$ we have $\text{width}^N(\mathcal{L}) \leq \text{width}^D(\mathcal{L})$ since DFAs are particular cases of NFAs. Moreover:

Corollary 5.5. *Let $\mathcal{L}$ be a regular language. Then, $width^D(\mathcal{L}) \leq 2^{width^N(\mathcal{L})} - 1$.*

Proof. Let $\mathcal{N}$ be an NFA such that $\mathcal{L}(\mathcal{N}) = \mathcal{L}$ and $\text{width}(\mathcal{N}) = \text{width}^N(\mathcal{L})$. By Theorem 5.4, we have $\text{width}^D(\mathcal{L}) \leq \text{width}(Pow(\mathcal{N})) \leq 2^{\text{width}(\mathcal{N})} - 1 = 2^{\text{width}^N(\mathcal{L})} - 1$. □

The above corollary shows that $\text{width}^N(\mathcal{L}) = 1$ implies $\text{width}^D(\mathcal{L}) = 1$, that is, the non-deterministic and deterministic widths are equal for Wheeler languages.

Theorem 5.4 has another intriguing consequence: the PSPACE-complete NFA equivalence problem [115] is fixed-parameter tractable with respect to the widths of the automata. In order to prove this result, we first update the analysis of Hopcroft et al. [74] of the powerset construction algorithm.

Lemma 5.6 (Adapted from [74])**.** *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let $Pow(\mathcal{N}) = (Q^*, s^*, \delta^*, F^*)$ be the powerset automaton obtained from $\mathcal{N}$. Let $n = |Q|$ and $p = width(\mathcal{N})$. Then, the powerset construction algorithm runs in $O(2^p(n-p+1)n^2\sigma)$ time.*

Proof. By Theorem 5.4, we know that $N = 2^p(n-p+1)$ is an upper bound to the number of states of the equivalent DFA. Each state in Q^* consists of $k \leq n$ states $u_1, \ldots, u_k$ of Q. For each character $a \in \Sigma$, we need to follow all edges labeled a leaving $u_1, \ldots, u_k$. In the worst case (a complete transition function), this leads to traversing $O(k \cdot n) \subseteq O(n^2)$ edges of the NFA. The final complexity is thus $O(N \cdot n^2 \cdot \sigma)$. □

Corollary 5.7. *We can check the equivalence between two NFAs over an alphabet of size σ, both with number of states at most n and width at most p, in $O(2^p(n-p+1)n^2\sigma)$ time.*

Proof. First, build the powerset automata, both having at most $N = 2^p(n-p+1)$ states by Theorem 5.4. This takes $O(Nn^2\sigma)$ time by Lemma 5.6. Finally, DFA equivalence can be tested in $O(N\sigma \log N)$ time by DFA minimization using Hopcroft's algorithm. □

Similarly, the powerset construction can be used to test membership of a word of length m in a regular language expressed as an NFA. When m is much larger than n and 2^p, this simple analysis of a classical method yields a faster algorithm than the state-of-the-art solution by Thorup and Bille, running in time $O(m \cdot e \cdot \log\log m/(\log m)^{3/2} + m + e)$ [17] where e is the NFA's (equivalently, the regular expression's) size:

Corollary 5.8. *We can test membership of a word of length m in the language recognized by an NFA with n states and co-lex width p on alphabet of size σ in $O(2^p(n-p+1)n^2\sigma + m)$ time.*

5.2 The Deterministic Width

It is natural to wonder whether the deterministic width of a language equals the width of its minimum DFA. As we show in Example 5.9, unfortunately this is not the case. Moreover, there is, in general, no unique (up to isomorphism) minimum DFA of minimum width recognizing a given regular language.

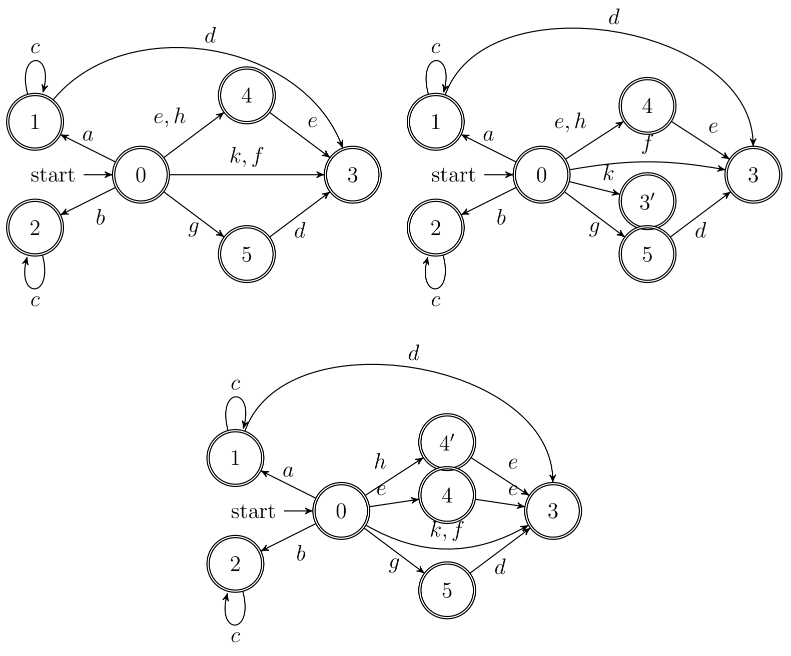


Figure 5.1: Three DFAs recognizing the same language.

Example 5.9. In Figure 5.1, three DFAs recognizing the same language $\mathcal{L}$ are shown. 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 recognising $\mathcal{L}$ of width 2. Consider the DFA $\mathcal{D}_1$ on the left of Figure 5.1 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: $I_0 = \{\varepsilon\}$, $I_1 = ac^*$, $I_2 = bc^*$, $I_3 = ac^*d \cup \{gd, ee, he, f, k\}$, $I_4 = \{e, h\}$, $I_5 = \{g\}$. 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 5.2.

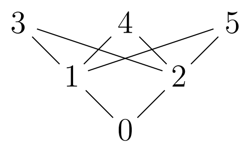


Figure 5.2: The Hasse diagram of $\leq_{\mathcal{D}_1}$

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 5.1 and let $\mathcal{D}_3$ be the DFA on the right of Figure 5.1. 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 $\mathcal{D}_3$).

Motivated by the problem of computing the deterministic width of a regular language is not a trivial problem, in the next sections we develop a set of tools that will ultimately allow us to derive an algorithm solving the problem.

Example 5.9 triggers a further natural and important observation. It is known that languages with deterministic width equal to 1 (that is, Wheeler languages) admit a (unique) minimum-size Wheeler DFA [4]. Note that Example 5.9 implies that no such minimality result holds true for higher levels of the deterministic width hierarchy. In Section 5.7 we will explain why Example 5.9 is not the end of the story and we will derive an adequate notion of minimality.

5.3 The Entanglement of a Regular Language

We now exhibit a measure that on the minimum DFA will capture exactly the width of the accepted language: the *entanglement number* of a DFA. We shall use the following terminology: if $\mathcal{D}$ is a DFA and $V \subseteq \operatorname{Pref}(\mathcal{L}(\mathcal{D}))$, then a state u *occurs* in V if there exists $\alpha \in V$ such that $u = \delta(s, \alpha)$.

Definition 5.10. Let $\mathcal{D}$ be a DFA with 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 $\operatorname{Pref}(\mathcal{L}(\mathcal{D}))$ 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 $V \subseteq \operatorname{Pref}(\mathcal{L}(\mathcal{D}))$ 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 :

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

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

As an example consider the entanglement of all DFAs in Figure 5.1. For any of them the entanglement is two, because the only entangled subset of states is $\{1, 2\}$, as witnessed by the sequence $a \prec b \prec ac \prec bc \prec acc \prec bcc \prec \cdots$.

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. In other words, two distinct states u, u' belonging to an entangled set are always $\leq_{\mathcal{D}}$-incomparable. Since by Lemma 4.12 we have $\operatorname{width}(\leq_{\mathcal{D}}) = \operatorname{width}(\mathcal{D})$, it easily follows that the entanglement of a DFA is always smaller than or equal to its width.

Lemma 5.11. *Let $\mathcal{D}$ be a DFA. Then ent$(\mathcal{D}) \leq$ width$(\mathcal{D})$.*

The converse of the above inequality is not always true: for the (minimum) DFA $\mathcal{D}_1$ on the left of Figure 5.1 we have $\operatorname{ent}(\mathcal{D}_1) = 2$ and $\operatorname{width}(\mathcal{D}_1) = 3$.

Contrary to what happens with the width, we now prove that the entanglement of a regular language is realized by the minimum-size automaton accepting it.

Lemma 5.12. *If $\mathcal{D}_\mathcal{L}$ is the minimum-size DFA recognizing $\mathcal{L}$, then ent$(\mathcal{D}_\mathcal{L}) = $ ent$(\mathcal{L})$.*

Proof. It is enough to prove that $\text{ent}(\mathcal{D}_\mathcal{L}) \leq \text{ent}(\mathcal{D})$, for any DFA $\mathcal{D}$ such that $\mathcal{L}(\mathcal{D}_\mathcal{L}) = \mathcal{L}(\mathcal{D})$. Suppose $u_1, \dots, u_k$ 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 $v_1, \dots, v_k$ infinitely often and $v_1, \dots, v_k$ are entangled in $\mathcal{D}$.

□

5.4 The Hasse Automaton of a Regular Language

Our aim is to prove that the entanglement measure over the minimum DFA $\mathcal{D}_\mathcal{L}$ captures the deterministic width of a language $\mathcal{L}$:

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

In order to prove the previous equality we shall describe an automaton, the *Hasse automaton* of $\mathcal{L}$, realizing the width and the entanglement of the language as its width (Theorem 5.24). 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 5.22).

To give an intuition on the construction of the automaton $\mathcal{D}'$ we use the *trace* of the DFA $\mathcal{D}$, that is, the (in general) transfinite sequence: $(\delta(s, \alpha))_{\alpha\in\text{Pref}(\mathcal{L})}$, indexed over the totally ordered set $(\text{Pref}(\mathcal{L}), \preceq)$, where $\mathcal{L} = \mathcal{L}(\mathcal{D})$. We depict below a hypothetical $(\text{Pref}(\mathcal{L}), \preceq)$, together with the trace left by a DFA $\mathcal{D}$ with set of states $\{u_1, u_2, u_3\}$ and $\delta(s, \alpha_i) = \delta(s, \alpha') = u_1$, $\delta(s, \beta_i) = \delta(s, \beta'_i) = u_2$, and $\delta(s, \gamma_i) = u_3$:

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

Consider the entanglement and width of $\mathcal{D}$. Notice that the sets $\{u_1, u_2\}$ and $\{u_2, u_3\}$ are entangled. 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 $\text{ent}(\mathcal{D}) = 2 < \text{width}(\mathcal{D}) = 3$.
In general, an automaton where incomparability and entanglement coincide would have $\text{ent}(\mathcal{D}) = \text{width}(\mathcal{D})$. 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 $\mathcal{V} = \{V_1, \ldots, V_r\}$ of $\text{Pref}(\mathcal{L})$ composed of convex sets which are entangled in $\mathcal{D}$. In the example above we can write $\text{Pref}(\mathcal{L}) = V_1 \cup V_2 \cup V_3$, where:

$$V_1 = \{\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i, \ldots\}, V_2 = \{\beta_1', \gamma_1, \ldots, \beta_i', \gamma_i \ldots\}, V_3 = \{\alpha'\}$$

and the states occurring (and entangled) in V_1, V_2, V_3 are, respectively: $\{u_1, u_2\}$, $\{u_2, u_3\}$, and $\{u_1\}$. In order 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 $\text{width}(\mathcal{D}') = 2$, 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 $\text{Pref}(\mathcal{L})$ 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 $\prec_{\mathcal{D}'}$-incomparable states $\{u_1, \ldots, u_k\}$ 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 $\text{width}(\mathcal{D}') = \text{ent}(\mathcal{D})$ will follow.

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

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

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

In order to give a formal definition of $\mathcal{D}'$ we need some properties of entangled sets. Since these properties hold true with respect to a partition of a generic total order, we state and prove them in a more general setting in Appendix A, while we state them without proof in this section. In particular, most of the properties of $\mathcal{D}'$ will be proved using a finite partition of $(\text{Pref}(\mathcal{L}(\mathcal{D})), \preceq)$, composed by entangled, convex sets.

Definition 5.13. If $\mathcal{D}$ is a DFA, we say that a partition $\mathcal{V}$ of $\text{Pref}(\mathcal{L}(\mathcal{D}))$ is an *entangled, convex decomposition of* $\mathcal{D}$ (*e.c. decomposition*, for short) if all the elements of $\mathcal{V}$ are convex in $(\text{Pref}(\mathcal{L}(\mathcal{D})), \preceq)$ and entangled in $\mathcal{D}$.

Theorem 5.14. *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, applied to $(Z, \leq) = (\text{Pref}(\mathcal{L}(\mathcal{D})), \preceq)$ and $\mathcal{P} = \{I_u \mid u \in Q\}$. $\square$

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

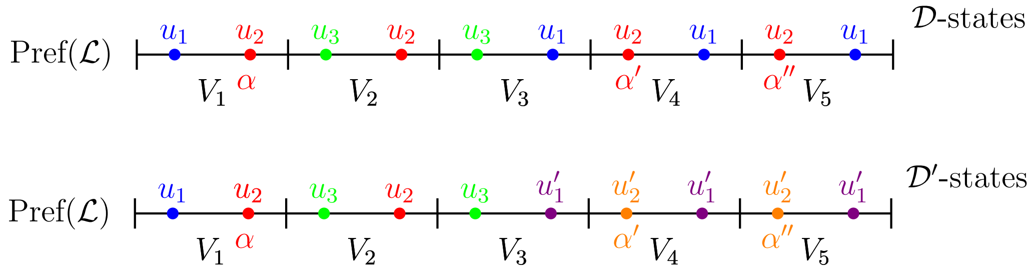
Lemma 5.15. *Let $\mathcal{D}$ be a DFA. Suppose $V_1 \prec V_2 \prec \cdots \prec V_m$ is an e.c. decomposition of $\mathcal{D}$ such that for every $u \in Q$ there are $1 \leq i \leq j \leq m$ with:*

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

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

Proof. We already proved that $\text{ent}(\mathcal{D}) \leq \text{width}(\mathcal{D})$ in Lemma 5.11. In order to prove the reverse inequality, let $\text{width}(\mathcal{D}) = p$ and let $u_1, \ldots, u_p$ be p $\preceq_{\mathcal{D}}$-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. 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. $\square$

The previous lemma suggests that in order to construct an automaton $\mathcal{D}'$ recognizing the same language as $\mathcal{D}$ and satisfying $\text{width}(\mathcal{D}') = \text{ent}(\mathcal{D}') = \text{ent}(\mathcal{D})$, we might



Pref($\mathcal{L}$)
$\mathcal{D}$-states
u_1 u_2 u_3 u_2 u_3 u_1 u_2 u_1 u_2 u_1
V_1 α V_2 V_3 α' V_4 α'' V_5
Pref($\mathcal{L}$)
$\mathcal{D}'$-states
u_1 u_2 u_3 u_2 u_3 u'_1 u'_2 u'_1 u'_2 u'_1
V_1 α V_2 V_3 α' V_4 α'' V_5

Figure 5.3: The upper line represents the words in Pref($\mathcal{L}$), partitioned by the e.c. decomposition $\mathcal{V}$, where some words are highlighted and labelled by the state they reach in $\mathcal{D}$. The lower line still represents Pref($\mathcal{L}$), with the same e.c. decomposition, but now the highlighted words are labelled by states of $\mathcal{D}'$. Note that the strings α, α' reach the same state in $\mathcal{D}$, but in different states in $\mathcal{D}'$, because $V_3 \subseteq [\alpha, \alpha']$, and $V_3 \cap I_{u_2} = \emptyset$. However, the words α', α'' reach the same state in both automata.

duplicate some states in $\mathcal{D}$ in order to ensure that the new automaton $\mathcal{D}'$ satisfies the condition of the previous Lemma. Consider two words $\alpha \prec \alpha'$ reaching the same state u of $\mathcal{D}$: if the convex $[\alpha, \alpha']$ is not contained in a union of consecutive elements of the partition $\mathcal{V}$, all having an occurrence of u, then in $\mathcal{D}'$ we duplicate the state u into u and u', with α reaching u and α' reaching u' (see Fig. 5.3). As usual, an equivalence relation $\sim_{\mathcal{D}}$ over 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 use generic entangled convex sets instead of elements of an e.c. decomposition. In Lemma 5.20 we prove that this is equivalent to using elements of an e.c. decomposition of minimum cardinality.

Definition 5.16. Let $\mathcal{D}$ be a DFA and let $\sim_{\mathcal{D}}$ be the equivalence relation on Pref($\mathcal{L}(D)$) defined as follows: $\alpha \sim_{\mathcal{D}} \alpha'$ if and only if:

- $\delta(s, \alpha) = \delta(s, \alpha')$ and
- there are entangled convex sets $C_1, \ldots, C_n \subseteq$ 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 $\sim_{\mathcal{D}}$.

In the following lemma we prove that the equivalence $\sim$ has finite index and $\mathcal{L}(\mathcal{D})$ is equal to the union of some of its classes.

Lemma 5.17. *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 $\mathcal{V} = \{V_1, \dots, V_m\}$ of $\mathcal{D}$, whose existence is guaranteed by Theorem 5.14. 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. $\square$

Lemma 5.18. *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 $\alpha a \in$ Pref($\mathcal{L}$). We must prove that $\alpha' a \in$ 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 $C_1, \dots, C_n$ such that $[\alpha, \alpha']^{\pm} \subseteq C_1 \cup \dots \cup C_n$ and $C_i \cap I_{\delta(s,\alpha)} \neq \emptyset$ for all $i = 1, \dots, n$. We must prove that $\alpha' a \in$ Pref($\mathcal{L}$), $\delta(s, \alpha a) = \delta(s, \alpha' a)$, and there exist entangled convex sets $C'_1, \dots, C'_{n'}$ such that $[\alpha a, \alpha' a]^{\pm} \subseteq C'_1 \cup \dots \cup C'_{n'}$ and $C'_i \cap I_{\delta(s,\alpha a)} \neq \emptyset$ for all $i = 1, \dots, n'$.

From $\delta(s, \alpha) = \delta(s, \alpha')$ and $\alpha a \in$ Pref($\mathcal{L}$) we immediately obtain $\alpha' a \in$ Pref($\mathcal{L}$) and $\delta(s, \alpha a) = \delta(s, \alpha' a)$. Moreover, from $[\alpha, \alpha']^{\pm} \subseteq C_1 \cup \dots \cup C_n$ we obtain $[\alpha a, \alpha' a]^{\pm} = [\alpha, \alpha']^{\pm} a \subseteq C_1 a \cup \dots \cup C_n a$, 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 co-lex 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. $\square$

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

Definition 5.19. 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 5.18), has finite index, and $\mathcal{L}$ is the union of some $\sim$-classes (Lemma 5.17). Hence, $\mathcal{D}'$ is a well-defined DFA, and $\alpha \in [\beta]_\sim \iff \delta'(s', \alpha) = [\beta]_\sim$, which implies that for every $\alpha \in \text{Pref}(\mathcal{L})$ it holds:

$$I_{[\alpha]_\sim} = [\alpha]_\sim \tag{5.5}$$

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

In the following lemma we prove that it is safe to replace $C_1, \dots, C_n$ in Definition 5.16 by the elements of a *minimum-size* e.c. decomposition, that is, an e.c. decomposition with minimum cardinality.

Lemma 5.20. *Let $\mathcal{D}$ be a DFA and let $\mathcal{V} = \{V_1, \dots, V_r\}$, 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 $i \le j$ such that:*
  - *$[\alpha, \alpha']^\pm \subseteq \bigcup_{h=i}^{j} V_h$;*
  - *$V_h \cap I_{\delta(s,\alpha)} \neq \emptyset$, for all $h \in \{i, \dots, j\}$.*

Proof. To prove that $\alpha \sim \alpha'$ holds under the above hypotheses, it is sufficient to recall that $V_i, \dots, V_j$ are entangled convex sets and apply Definition 5.16.

Let us prove the reverse implication. Pick $\alpha \sim \alpha' \in \text{Pref}(\mathcal{L}(\mathcal{D}))$ and let $C_1, \dots, C_n$ 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 $i = 1, \dots, n$, where $u = \delta(s, \alpha) = \delta(s, \alpha')$. By Lemma A.12 of Appendix A we can assume that $C_1 \prec C_2 \prec \cdots \prec C_n$. Let $i \le j$ be such that $[\alpha, \alpha']^\pm \subseteq \bigcup_{h=i}^{j} V_h$ and $V_h \cap [\alpha, \alpha']^\pm \neq \emptyset$ for all $h \in \{i, \dots, j\}$. We just have to prove that $V_h \cap I_u \neq \emptyset$, for all $h \in \{i, \dots, j\}$. 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. $\square$

Let $\mathcal{D}'$ be the automaton of Definition 5.19. 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 5.21. *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 5.20 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 5.20, 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 5.5 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 5.20 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 5.22. *If $\mathcal{D}$ is a DFA, then $ent(\mathcal{D}) = ent(\mathcal{D}') = width(\mathcal{D}')$.*

Proof. Let us prove that $\mathrm{ent}(\mathcal{D}') \leq \mathrm{ent}(\mathcal{D})$. Let $\{[\alpha_1]_\sim, \ldots, [\alpha_h]_\sim\}$ be an entangled collection of h states in $\mathcal{D}'$. Then, there is a monotone sequence $(\gamma_i)_{i \in \mathbb{N}}$ such that, for each $j \in \{1, \dots, h\}$ 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 $\beta_1, \dots, \beta_h$ in V such that $\delta'(s', \beta_k) = [\alpha_k]_\sim$, for every $k = 1, \dots, h$. Define $u_k = \delta(s, \beta_k)$ and notice that the states $u_1, \dots, u_h$ are pairwise distinct. In fact, if $u_r = u_s$ for $r \neq s$, then by Lemma 5.20 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, $\{u_1, \dots, u_h\}$ is an entangled set in $\mathcal{D}$, because all these states occur in V (as witnessed by $\beta_1, \dots, \beta_h$) 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 $\text{ent}(\mathcal{D}') \leq \text{ent}(\mathcal{D})$.

Let us now prove that $\text{ent}(\mathcal{D}) \leq \text{ent}(\mathcal{D}')$. Let $\{u_1, \dots, u_h\}$ 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 $k = 1, \dots h$ 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 $u'_1, \dots, u'_h$ are pairwise distinct and $\{u'_1, \dots, u'_h\}$ is an entangled set of states in $\mathcal{D}'$, which implies $\text{ent}(\mathcal{D}) \leq \text{ent}(\mathcal{D}')$.

Finally, we prove that $\text{width}(\mathcal{D}') = \text{ent}(\mathcal{D}')$. If $\mathcal{V}$ is an e.c. decomposition of $\mathcal{D}$ of minimum size, then Corollary 5.21 implies that $\mathcal{V}$ is an e.c. decomposition of $\mathcal{D}'$ satisfying the hypothesis of Lemma 5.15 so that $\text{width}(\mathcal{D}') = \text{ent}(\mathcal{D}')$ follows from this lemma. □

If we start from the minimum DFA $\mathcal{D}_\mathcal{L}$ of a regular language $\mathcal{L}$, then, as we shall see in Theorem 5.24, the automaton $\mathcal{D}'_\mathcal{L}$ acquires a special role because it realizes the deterministic width of the language $\mathcal{L}$.

Definition 5.23. 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 above definition is motivated by 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 5.24. *If $\mathcal{D}_\mathcal{L}$ is the minimum DFA of the regular language $\mathcal{L}$, then:*

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

Proof. By Lemma 5.12 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 5.11), we obtain $\text{ent}(\mathcal{L}) \leq \text{width}^{D}(\mathcal{L})$, while from Theorem 5.22 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. □

The previous theorem allows us to provide an automata-free characterization of the deterministic width of a regular language. Recall that a property is *eventually* true for a sequence if it holds true for all but finitely many elements of the sequence.

Corollary 5.25. *Let $\mathcal{L}$ be a regular language. Then $\text{width}^{D}(\mathcal{L}) \leq p$ iff every (co-lexicographically) monotone sequence in Pref($\mathcal{L}$) is eventually included in at most p classes of the Myhill-Nerode equivalence $\equiv_{\mathcal{L}}$.*

Proof. Let $\mathcal{D}_{\mathcal{L}}$ be the minimum DFA for $\mathcal{L}$. By definition, k states $u_1, \ldots, u_k$ are entangled in $\mathcal{D}_{\mathcal{L}}$ iff there exists a monotone sequence $(\alpha_j)_{j \in \mathbb{N}}$ such that, for each $i = 1, \ldots, k$, we have $\delta(s, \alpha_j) = u_i$ for infinitely many j's. Moreover, since $\mathcal{D}_{\mathcal{L}}$ is minimum, if $i \neq i'$ a word arriving in u_i and a word arriving in $u_{i'}$ belong to different $\equiv_{\mathcal{L}}$-classes. Hence, $\text{ent}(\mathcal{D}_{\mathcal{L}}) > p$ iff there exists a monotone sequence in $\text{Pref}(\mathcal{L})$ which eventually reaches more than p classes of the Myhill-Nerode equivalence $\equiv_{\mathcal{L}}$ infinitely often, and the corollary follows from the previous theorem.

□

Summarizing, the Hasse automaton $\mathcal{H}_{\mathcal{L}}$ captures the deterministic width of a language. An interesting open question is whether it is possible to devise an effective procedure to build the Hasse automaton.

5.5 Computing the Deterministic Width of a Regular Language

In this section we shall use Theorem 5.24 — stating that the deterministic width of $\mathcal{L}$ is equal to the entanglement of the minimum DFA for $\mathcal{L}$ — to study the complexity of the problem of finding the deterministic width of a language recognized by a given automaton. We show that if we are given a regular language $\mathcal{L}$ by means of a DFA $\mathcal{D}$ accepting $\mathcal{L}$ and a positive integer p, then the problem

$$\text{width}^{D}(\mathcal{L}) \overset{?}{\leq} p$$

is solvable in polynomial time for constant values of p. More precisely, we show that the problem of computing $\text{width}^D(\mathcal{L})$ is in the class XP with parameter p. This result is achieved by exhibiting a dynamic programming algorithm that extends the ideas introduced in [5] when solving the corresponding problem for Wheeler languages.

Theorem 5.24 suggests that the minimum DFA contains all "topological" information required to compute the width of a language. In the next theorem we clarify this intuition by providing a graph-theoretical characterization of the deterministic width of a language based on the minimum DFA recognizing the language.

Theorem 5.26. *Let $\mathcal{L}$ be a regular language and let $\mathcal{D}_\mathcal{L}$ be the minimum DFA of $\mathcal{L}$, with set of states Q. Let $k \geq 2$ be an integer. Then, $\text{width}^D(\mathcal{L}) \geq k$ if and only if there exist strings $\mu_1, \dots, \mu_k, \gamma$ and pairwise distinct states $u_1, \dots, u_k \in Q$, such that for every $j = 1, \dots, k$:*

1. *μ_j labels a path from the initial state s to u_j;*
2. *γ labels a cycle starting (and ending) at u_j;*
3. *either $\mu_1, \dots, \mu_k \prec \gamma$ or $\gamma \prec \mu_1, \dots, \mu_k$;*
4. *γ is not a suffix of μ_j.*

Proof. By Theorem 5.24 we have $\text{width}^D(\mathcal{L}) = \text{ent}(\mathcal{D}_\mathcal{L})$. We begin by proving that, if the stated conditions hold true, then $\text{ent}(\mathcal{D}_\mathcal{L}) \geq k$. Notice that for every integer i we have $\mu_j\gamma^i \in I_{u_j}$. Moreover, the μ_j's are pairwise distinct because the u_j's are pairwise distinct, so without loss of generality we can assume $\mu_1 \prec \dots \prec \mu_k$.

1. If $\mu_1 \prec \dots \prec \mu_k \prec \gamma$, consider the increasing sequence:

$$\mu_1 \prec \dots \prec \mu_k \prec \mu_1\gamma \prec \dots \prec \mu_k\gamma \prec \mu_1\gamma^2 \prec \dots \prec \mu_k\gamma^2 \prec \mu_1\gamma^3 \prec \dots \prec \mu_k\gamma^3 \prec \dots$$

2. If $\gamma \prec \mu_1 \prec \dots \prec \mu_k$, consider the decreasing sequence:

$$\mu_k \succ \dots \succ \mu_1 \succ \mu_k\gamma \succ \dots \succ \mu_1\gamma \succ \mu_k\gamma^2 \succ \dots \succ \mu_1\gamma^2 \succ \mu_k\gamma^3 \succ \dots \succ \mu_1\gamma^3 \succ \dots$$

   where $\mu_1\gamma^i \succ \mu_k\gamma^{i+1}$ holds because $\mu_1 \succ \gamma$ and γ is not a suffix of μ_1.

In both cases the sequence witnesses that $\{u_1, \dots, u_k\}$ is an entangled set of distinct states, so $\text{ent}(\mathcal{D}_\mathcal{L}) \geq k$.

Conversely, assume that $\text{ent}(\mathcal{D}_\mathcal{L}) \geq k$. This means that there exist distinct states $v_1, \dots, v_k$ and a monotone sequence $(\alpha_i)_{i \in \mathbb{N}}$ that reaches each of the v_j's infinitely many times. Let us show that, up to taking subsequences, we can assume not only that $(\alpha_i)_{i \in \mathbb{N}}$ reaches each of the v_j's infinitely many times, but it also satisfies additional properties.

- Since $|\Sigma|$ and $|Q|$ are finite and $(\alpha_i)_{i \in \mathbb{N}}$ is monotone, then up to removing a finite number of initial elements we can assume that all α_i's end with the same $m = |Q|^k$ characters, and we can write $\alpha_i = \alpha'_i \theta$, for some $\theta \in \Sigma^m$. Notice that such a new monotone sequence $(\alpha_i)_{i \in \mathbb{N}}$ still reaches each of the v_j's infinitely many times.

- Up to taking a subsequence of the new $(\alpha_i)_{i \in \mathbb{N}}$, we can assume that α_i reaches v_j if and only if $i - j$ is a multiple of k, that is, $\alpha_j, \alpha_{k+j}, \alpha_{2k+j}, \dots \in I_{v_j}$. Notice that such such a new monotone sequence $(\alpha_i)_{i \in \mathbb{N}}$ still satifies $\alpha_i = \alpha'_i \theta$ for every i.

- Since $|Q|$ is finite, up to taking a subsequence of the new $(\alpha_i)_{i \in \mathbb{N}}$ we can assume that all α_i's reaching the same v_j spell the suffix θ visiting the same $m+1$ states $x^j_0, x^j_1, \dots, x^j_m = v_j$.

Consider the k-tuples $(x^1_s, \dots, x^k_s)$, for $s \in \{0, \dots, m\}$ (corresponding to the states in column in Figure 5.4). There are $m + 1 = |Q|^k + 1$ such k-tuples and therefore two of them must be equal. That is, there exist h, ℓ, with $0 \leq h < \ell \leq m$, such that $(x^1_h, \dots, x^k_h) = (x^1_\ell, \dots, x^k_\ell)$. Hence, for all $j \in \{1, \dots, k\}$ there is a cycle $x^j_h, x^j_{h+1}, \dots, x^j_\ell$, all these cycles are labelled by the same string γ', and we can write $\theta = \phi\gamma'\psi$ for some ϕ and ψ.

Let $u_1, u_2, \dots, u_k$ be the pairwise distinct states $x^1_h, x^2_h, \dots, x^k_h$ (they are distinct, because if $u_i = u_j$ for some $i \neq j$ we would have $v_i = v_j$). Hence, we have k pairwise distinct states and k equally labelled cycles. In order to fulfill the remaining conditions of the theorem, we proceed as follows. Considering the monotone sequence $(\alpha'_i \phi)_{i \in \mathbb{N}}$, reaching each of the u_i's infinitely many times, we may suppose without loss of generality (possibly eliminating a finite number of initial elements) that all $\alpha'_i \phi$'s

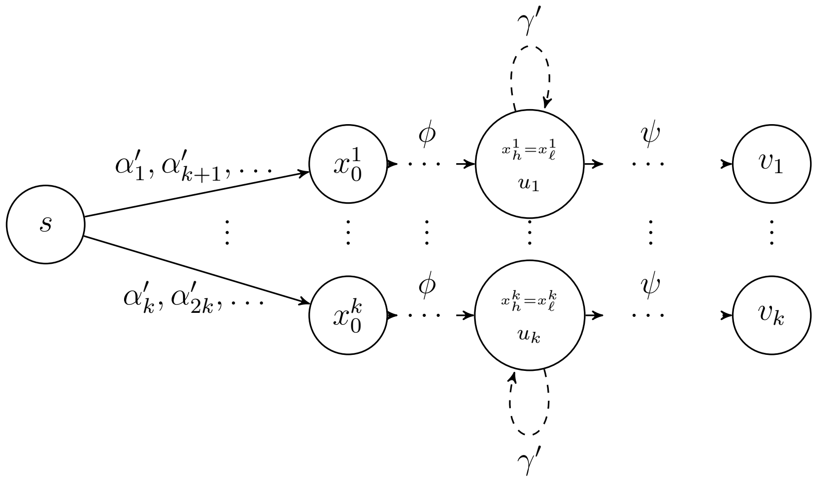


Figure 5.4

are co-lexicographically larger than γ' or they are all co-lexicographically smaller than γ'.

If γ' is not a suffix of any $\alpha'_i\phi$ we can choose $\gamma = \gamma'$ and, considering k words $\mu_1, \dots, \mu_k$ of the sequence $(\alpha'_i\phi)_{i\in\mathbb{N}}$ arriving in $u_1, \dots, u_k$, respectively, we are done.

Otherwise, if γ' is a suffix of some $\alpha'_i\phi$, pick $2k-1$ strings $\delta_1, \dots, \delta_{2k-1}$ in the sequence $(\alpha'_i\phi)_{i\in\mathbb{N}}$ such that δ_k ends in u_k, while δ_i and δ_{k+i} end in u_i for $i = 1, \dots, k-1$, and

$$\delta_1 \prec \dots \prec \delta_k \prec \dots \prec \delta_{2k-1}.$$

Let r be an integer such that $|(\gamma')^r| > |\delta_i|$ for every $i = 1, \dots, 2k-1$. Then $\gamma = (\gamma')^r$ is the label of a cycle from u_i, for every $i = 1, \dots, k$, and γ is not a suffix of δ_i, for every $i = 1, \dots, 2k-1$. We distinguish two cases:

1. $\delta_k \prec \gamma$. In this case, let $\mu_1, \dots, \mu_k$ be equal to $\delta_1, \dots, \delta_k$, respectively.

2. $\gamma \prec \delta_k$. In this case, let $\mu_1, \dots, \mu_k$ be equal to $\delta_k, \dots, \delta_{2k-1}$, respectively.

In both cases, we have either $\mu_1, \dots, \mu_k \prec \gamma$ or $\gamma \prec \mu_1, \dots, \mu_k$ and the conclusion follows.

□

Example 5.27. Let $\mathcal{L}$ be a regular language. Let us prove that, in general, $\text{width}^D(\mathcal{L})$ and $\text{width}^N(\mathcal{L})$ may depend on the total order $\preceq$ on the alphabet. Let $\mathcal{D}$ be the DFA in Figure 4.2, and let $\mathcal{L}$ be the language recognized by $\mathcal{D}$. Notice that $\mathcal{D}$ is the minimum DFA recognizing $\mathcal{L}$.

First, assume that $\preceq$ is the standard alphabetical order such that $a \prec b \prec c \prec d$. Let us prove that $\text{width}^D(\mathcal{L}) = 2$. From Example 4.18, we obtain $\text{width}^D(\mathcal{L}) \leq 2$, and from Theorem 5.26 we obtain $\text{width}^D(\mathcal{L}) \geq 2$ by choosing $u_1 = q_2$, $u_2 = q_3$, $\mu_1 = a$, $\mu_2 = b$, $\gamma = c$. Notice that Corollary 5.5 implies that $\text{width}^N(\mathcal{L}) = \text{width}^D(\mathcal{L}) = 2$.

Next, let $\preceq$ be the total order such that $a \prec c \prec b \prec d$. From Example 4.18 we immediately obtain $\text{width}^N(\mathcal{L}) = \text{width}^D(\mathcal{L}) = 1$.

The strings $\mu_1, \ldots, \mu_k$, and γ of Theorem 5.26 can be determined by a dynamic programming algorithm whose running time can be computed using the following lemma.

Lemma 5.28. *Let $\mathcal{D}$ be a DFA with set of states Q, and let $s_1, q_1, \ldots, s_h, q_h \in Q$. Suppose there are strings $\nu_1 \preceq \cdots \preceq \nu_h$ such that $\delta(s_i, \nu_i) = q_i$, for every $i = 1, \ldots, h$. Then, there exist strings $\nu'_1 \preceq \cdots \preceq \nu'_h$ such that, for every $i, j \in \{1, \ldots, h\}$, it holds:*

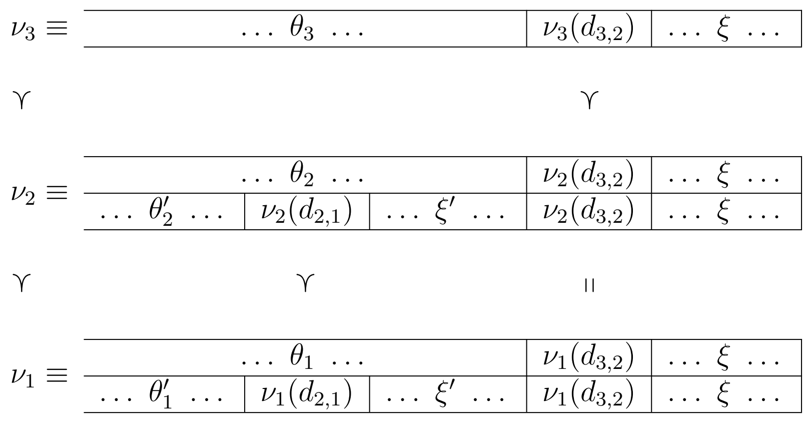
- $\delta(s_i, \nu'_i) = q_i$;
- $\nu_i = \nu_j$ *iff* $\nu'_i = \nu'_j$;
- $\nu_i \dashv \nu_j$ *iff* $\nu'_i \dashv \nu'_j$;
- $|\nu'_i| \leq h - 2 + \sum_{t=1}^{h} |Q|^t$.

Proof. We will prove the lemma for $h = 3$ (the extension to the general case is straightforward). Given $\varphi \in \Sigma^*$, we denote by $\varphi(k)$ the k-th letter of φ from the right (if $|\varphi| < k$ we write $\varphi(k) = \varepsilon$, where ε is the empty string); therefore, if $\varphi \neq \varepsilon$, then $\varphi(1)$ is the last letter of φ.

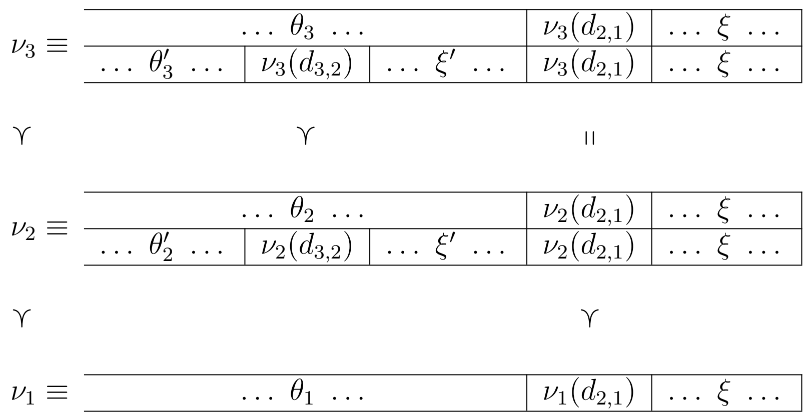
Let $\nu_1 \preceq \nu_2 \preceq \nu_3$ be strings with $\delta(s_i, \nu_i) = q_i$, for $i = 1, 2, 3$. Let $d_{3,2}$ be the first position from the right where ν_3 and ν_2 differ (if $\nu_3 = \nu_2$, let $d_{3,2} = |\nu_3|$). Since $\nu_2 \preceq \nu_3$, we have $d_{3,2} \leq |\nu_3|$. Defining $d_{2,1}$ similarly, we have $d_{2,1} \leq |\nu_2|$.

We distinguish three cases.

1. $d_{3,2} = d_{2,1}$.
2. $d_{3,2} < d_{2,1}$ (see Fig. 5.5a, assuming that $\nu_1 \prec \nu_2 \prec \nu_3$).



(a) Case $\nu_1 \prec \nu_2 \prec \nu_3$ and $d_{3,2} < d_{2,1}$.



(b) Case $\nu_1 \prec \nu_2 \prec \nu_3$ and $d_{2,1} < d_{3,2}$.

Figure 5.5

3. $d_{2,1} < d_{3,2}$ (see Fig. 5.5b, assuming that $\nu_1 \prec \nu_2 \prec \nu_3$).

Case 3 is analogous to case 2 and will not be considered. In cases 1 and 2, $|\nu_3| \geq d_{3,2}$ and $|\nu_2| \geq d_{2,1} \geq d_{3,2}$, so that ν_3, ν_2, and ν_1 end with the same (possibly empty) word ξ with $|\xi| = d_{3,2} - 1$. Summing up:

$$\nu_1 = \theta_1 \nu_1(d_{3,2})\xi \preceq \nu_2 = \theta_2 \nu_2(d_{3,2})\xi \preceq \nu_3 = \theta_3 \nu_3(d_{3,2})\xi$$

for some (possibly empty) strings $\theta_1, \theta_2, \theta_3$.

Without loss of generality, we may assume that $|\xi| \leq |Q|^3$. Indeed, if $|\xi| > |Q|^3$ then when we consider the triples of states visited while reading the last $|\xi|$ letters in computations from s_i to q_i following ν_i, for $i = 1, 2, 3$, we should have met a repetition. If this were the case, we could erase a common factor from ξ, obtaining a shorter word ξ_1 such that $\theta_1\nu_1(d_{3,2})\xi_1 \preceq \theta_2\nu_2(d_{3,2})\xi_1 \preceq \theta_3\nu_3(d_{3,2})\xi_1$, with the three new strings starting in s_1, s_2, s_3 and ending in q_1, q_2, q_3, respectively, respecting equalities and suffixes. Then we can repeat the argument until we reach a word not longer than $|Q|^3$.

If $d_{3,2} = d_{2,1}$, the order between the $\nu_i's$ is settled in position $d_{3,2}$. Let r_1, r_2, r_3 be the states reached from s_1, s_2, s_3 by reading $\theta_1, \theta_2, \theta_3$, respectively. For $i = 1, 2, 3$, let $\bar{\theta}_i$ be the label of a simple path from s_i to r_i and let $\nu'_i = \bar{\theta}_i\nu_i(d_{3,2})\xi$. Then $\delta(s_i, \nu'_i) = q_i$, $\nu'_1 \preceq \nu'_2 \preceq \nu'_3$, $|\nu'_1|, |\nu'_2|, |\nu'_3| \leq |Q| + |Q|^3$.

If $d_{3,2} < d_{2,1}$ we have $\nu_1(d_{3,2}) = \nu_2(d_{3,2})$. Moreover, θ_1 and θ_2 end with the same word ξ' with $|\xi'| = d_{2,1} - d_{3,2} - 1$ (see Fig. 5.5a), and we can write

$$\theta_1 = \theta'_1\nu_1(d_{2,1})\xi' \preceq \theta_2 = \theta'_2\nu_2(d_{2,1})\xi'.$$

Arguing as before we can assume, without loss of generality, that $|\xi'| \leq |Q|^2$. Moreover, we have $\nu_1(d_{2,1}) \preceq \nu_2(d_{2,1})$ and, as before, we can assume that θ'_1, θ'_2 and θ_3 label simple paths. Therefore, $|\theta'_1|, |\theta'_2|, |\theta_3| \leq |Q| - 1$. Hence, in this case we can find ν'_1, ν'_2, ν'_3 such that $\delta(s_i, \nu'_i) = q_i$, $\nu'_1 \preceq \nu'_2 \preceq \nu'_3$, and $|\nu'_1|, |\nu'_2|, |\nu'_3| \leq 1 + |Q| + |Q|^2 + |Q|^3$.

Finally, the construction implies that $\nu_i = \nu_j$ iff $\nu'_i = \nu'_j$, and $\nu_i \dashv \nu_j$ iff $\nu'_i \dashv \nu'_j$. $\square$

We are now ready for a computational variant of Theorem 5.26.

Corollary 5.29. *Let $\mathcal{L}$ be a regular language and let $\mathcal{D}_{\mathcal{L}}$ be the minimum DFA of $\mathcal{L}$, with set of states Q. Let $k \geq 2$ be an integer. Then, width$^D(\mathcal{L}) \geq k$ if and only if there exist strings $\mu_1, \ldots, \mu_k$, and γ and there exist pairwise distinct $u_1, \ldots, u_k \in Q$ such that, for every $j = 1, \ldots, k$:*

1. *μ_j labels a path from the initial state s to u_j;*
2. *γ labels a cycle starting (and ending) at u_j;*
3. *either $\mu_1, \ldots, \mu_k \prec \gamma$ or $\gamma \prec \mu_1, \ldots, \mu_k$;*
4. *$|\mu_1|, \ldots, |\mu_k| < |\gamma| \leq 2(2k - 2 + \sum_{t=1}^{2k} |Q|^t)$.*

Proof. ($\Leftarrow$) Since condition 4. implies that γ is not a suffix of any of the μ_j, $\text{width}^D(\mathcal{L}) \geq k$ follows from Theorem 5.26.
($\Rightarrow$) If $\text{width}^D(\mathcal{L}) \geq k$, we use Theorem 5.26 and find words $\mu'_1, \dots, \mu'_k, \gamma'$ and pairwise distinct states $u_1, \dots, u_k \in Q$ such that for every $j = 1, \dots, k$:

1. μ'_j labels a path from the initial state s to u_j;

2. γ' labels a cycle starting (and ending) at u_j;

3. either $\mu'_1, \dots, \mu'_k \prec \gamma'$ or $\gamma' \prec \mu'_1, \dots, \mu'_k$;

4. γ' is not a suffix of μ'_j.

We only consider the case $\gamma' \prec \mu'_1, \dots, \mu'_k$, since in the case $\mu'_1, \dots, \mu'_k \prec \gamma'$ the proof is similar. Up to an index permutation, we may suppose without loss of generality that $\gamma' \prec \mu'_1 \prec \cdots \prec \mu'_k$. Consider the $2k$-words ν_i and states s_i, q_i defined, for $i = 1, \dots 2k$, as follows:

- $\nu_1 = \cdots = \nu_k = \gamma'$, $s_1 = q_1 = u_1, \dots, s_k = q_k = u_k$;

- $\nu_{k+i} = \mu'_i$, $s_{k+i} = s$, $q_{k+i} = u_i$ for $i = 1, \dots, k$, where s is the initial state of $\mathcal{D}_{\mathcal{L}}$.

If we apply Lemma 5.28 to these $2k$-words, we obtain words $\nu'_1 = \cdots = \nu'_k \prec \nu'_{k+1} \prec \cdots \prec \nu'_{2k}$ such that, for all $i = 1, \dots, k$:

1. $\delta(u_i, \nu'_1) = u_i$, that is, ν'_1 labels a cycle from every u_i;

2. $\delta(s, \nu'_{k+i}) = u_i$;

3. ν'_1 is not a suffix of ν'_{k+i};

4. $|\nu'_1|, |\nu'_{k+i}| \leq 2k - 2 + \sum_{t=1}^{2k} |Q|^t$.

Let r be the smallest integer such that $|(\nu'_1)^r| > max\{|\nu'_{k+i}| \mid i \in \{1, \dots, k\}\}$. Let $\mu_1 = \nu'_{k+1}, \dots, \mu_k = \nu'_{2k}, \gamma = (\nu'_1)^r$. Since ν'_1 is not a suffix of μ_i, for all $i = 1, \dots, k$, from $\nu'_1 \prec \mu_1 \prec \cdots \prec \mu_k$ it follows $\gamma = (\nu'_1)^r \prec \mu_1 \prec \cdots \prec \mu_k$, Moreover:

$$|\mu_1|, \dots, |\mu_k| < |\gamma| \leq max\{|\nu'_{k+i}| \mid i \in \{1, \dots, k\}\} + |\nu'_1| \leq 2(2k - 2 + \sum_{t=1}^{2k} |Q|^t)$$

and the conclusion follows. $\square$

We can finally provide the main theorem of this section.

Theorem 5.30. *Let $\mathcal{L}$ be a regular language, given as input by means of any DFA $\mathcal{D} = (Q, s, \delta, F)$ recognizing $\mathcal{L}$. Then, for any integer $p \geq 1$ we can decide whether $\text{width}^D(\mathcal{L}) \leq p$ in time $|\delta|^{O(p)}$.*

Proof. We exhibit a dynamic programming algorithm based on Corollary 5.29, plugging in the value $k = p + 1$ and returning true if and only if $\text{width}^D(\mathcal{L}) \geq k$ is false.

First, note that the alphabet's size is never larger than the number of transitions: $\sigma \leq |\delta|$, and that $|Q| \leq |\delta| + 1$ since we assume that each state can be reached from s. Up to minimizing $\mathcal{D}$ (with Hopcroft's algorithm, running in time $O(|Q|\sigma \log |Q|) \leq |\delta|^{O(1)}$) we can assume that $\mathcal{D} = \mathcal{D}_{\mathcal{L}}$ is the minimum DFA recognizing $\mathcal{L}$. Let $N' = 2(2k - 2 + \sum_{t=1}^{2k} |Q|^t)$ be the upper bound to the lengths of the strings μ_i $(1 \leq i \leq k)$ and γ that need to be considered, and let $N = N'+1$ be the number of states in a path labeled by a string of length N'. Asymptotically, note that $N \leq |Q|^{O(k)} \leq |\delta|^{O(k)}$. The high-level idea of the algorithm is as follows. First, in condition (3) of Corollary 5.29, we focus on finding paths μ_j's smaller than γ, as the other case (all μ_j's larger than γ) can be solved with a symmetric strategy. Then:

1. For each state u and for each length $2 \leq \ell \leq N$, we compute the co-lexicographically smallest path of length (number of states) ℓ connecting s with u.

2. For each k-tuple $u_1, \ldots, u_k$ and for each length $\ell \leq N$, we compute the co-lexicographically largest string γ labeling k cycles of length (number of states) ℓ originating (respectively, ending) from (respectively, in) all the states $u_1, \ldots, u_k$.

Steps (1) and (2) could be naively solved by enumerating the strings $\mu_1, \ldots, \mu_k$, and γ and trying all possible combinations of states $u_1, \ldots, u_k$. Because of the string enumeration step, however, this strategy would be exponential in N, i.e. doubly-exponential in k. We show that a dynamic programming strategy is exponentially faster.

Step (1). This construction is identical to the one used in [5] for the Wheeler case $(p = 1)$. For completeness, we report it here. Let $\pi_{u,\ell}$, with $u \in Q$ and $2 \leq \ell \leq N$, denote the predecessor of u such that the co-lexicographically smallest path of length (number of states) ℓ connecting the source s to u passes through $\pi_{u,\ell}$ as follows: $s \rightsquigarrow \pi_{u,\ell} \to u$. The node $\pi_{u,\ell}$ coincides with s if $\ell = 2$ and u is a successor of s; in this case, the path is simply $s \to u$. If there is no path of length ℓ connecting s with u, then we write $\pi_{u,\ell} = \perp$. We show that the set $\{\pi_{u,\ell} \ : \ 2 \leq \ell \leq N, \ u \in Q\}$ stores in just polynomial space all co-lexicographically smallest paths of any fixed length $2 \leq \ell \leq N$ from the source to any node u. We denote such a path — to be intended as a sequence $u_1 \to \cdots \to u_\ell$ of states — with $\alpha_\ell(u)$. The node sequence $\alpha_\ell(u)$ can be obtained recursively (in $O(\ell)$ steps) as $\alpha_\ell(u) = \alpha_{\ell-1}(\pi_{u,\ell}) \to u$, where $\alpha_1(s) = s$ by convention. Note also that $\alpha_\ell(u)$ does not fully specify the sequence of edges (and thus labels) connecting those ℓ states, since two states may be connected by multiple (differently labeled) edges. However, the corresponding co-lexicographically smallest sequence $\lambda^-(\alpha_\ell(u))$ of $\ell - 1$ labels is uniquely defined as follows:

$$\begin{cases} \lambda^-(\alpha_\ell(u)) = \min\{a \in \Sigma \mid \delta(s,a) = u\} & \text{if } \ell = 2 \\ \lambda^-(\alpha_\ell(u)) = \lambda^-(\alpha_{\ell-1}(\pi_{u,\ell}) \to u) = \lambda^-(\alpha_{\ell-1}(\pi_{u,\ell})) \cdot \min\{a \in \Sigma \mid \delta(\pi_{u,\ell},a) = u\} & \text{if } \ell > 2. \end{cases}$$

It is not hard to see that each $\pi_{u,\ell}$ can be computed in $|\delta|^{O(k)}$ time using dynamic programming. First, we set $\pi_{u,2} = s$ for all successors u of s. Then, for $\ell = 3, \ldots, N$:

$$\pi_{u,\ell} = \operatorname*{argmin}_{v \in \mathrm{Pred}(u)} \Big(\lambda^-(\alpha_{\ell-1}(v)) \cdot \min\{a \in \Sigma \mid \delta(v,a) = u\} \Big)$$

where $\mathrm{Pred}(u)$ is the set of all predecessors of u and the argmin operator compares strings in co-lex order. In the equation above, if none of the $\alpha_{\ell-1}(v)$ are well-defined (because there is no path of length $\ell - 1$ from s to v), then $\pi_{u,\ell} = \perp$. Note that computing any particular $\pi_{u,\ell}$ requires comparing co-lexicographically $|\mathrm{Pred(u)}| \leq |Q|$ strings of length at most $\ell \leq N \leq |\delta|^{O(k)}$, which overall amounts to $|\delta|^{O(k)}$ time. Since there are $|Q| \times N = |\delta|^{O(k)}$ variables $\pi_{u,\ell}$ and each can be computed in time $|\delta|^{O(k)}$, overall Step (1) takes time $|\delta|^{O(k)}$. This completes the description of Step (1).

Step (2). Fix a k-tuple $u_1, \ldots, u_k$ and a length $2 \leq \ell \leq N$. Our goal is now to show how to compute the co-lexicographically largest string γ of length $\ell - 1$ labeling k cycles of length (number of states) ℓ originating (respectively, ending) from (respectively, in) all the states $u_1, \ldots, u_k$. Our final strategy will iterate over all such k-tuples of states (in time exponential in k) in order to find one satisfying the conditions of Corollary 5.29.

Our goal can again be solved by dynamic programming. Let $u_1, \dots, u_k$ and $u'_1, \dots, u'_k$ be two k-tuples of states, and let $2 \leq \ell \leq N$. Let moreover $\pi_{u_1,\dots,u_k,u'_1,\dots,u'_k,\ell}$ be the k-tuple $\langle u''_1, \dots, u''_k \rangle$ of states (if it exists) such that there exists a string γ of length $\ell - 1$ with the following properties:

- For each $1 \leq i \leq k$, there is a path $u_i \rightsquigarrow u''_i \rightarrow u'_i$ of length (number of nodes) ℓ labeled with γ, and

- γ is the co-lexicographically largest string satisfying the above property.

If such a string γ does not exist, then we set $\pi_{u_1,\dots,u_k,u'_1,\dots,u'_k,\ell} = \perp$.

Remember that we fix $u_1, \dots, u_k$. For $\ell = 2$ and each k-tuple $u'_1, \dots, u'_k$, it is easy to compute $\pi_{u_1,\dots,u_k,u'_1,\dots,u'_k,\ell}$: this k-tuple is $\langle u_1, \dots, u_k \rangle$ (all paths have length 2) if and only if there exists $c \in \Sigma$ such that $u'_i = \delta(u_i, c)$ for all $1 \leq i \leq k$ (otherwise it does not exist). Then, γ is formed by one character: the largest such c.

For $\ell > 2$, the k-tuple $\pi_{u_1,\dots,u_k,u'_1,\dots,u'_k,\ell}$ can be computed as follows. Assume we have computed those variables for all lengths $\ell' < \ell$. Note that for each such $\ell' < \ell$ and k-tuple $u''_1, \dots, u''_k$, the variables $\pi_{u_1,\dots,u_k,u''_1,\dots,u''_k,\ell'}$ identify k paths $u_i \rightsquigarrow u''_i$ of length (number of nodes) ℓ'. Let us denote with $\alpha_{\ell'}(u''_i)$ such paths, for $1 \leq i \leq k$.

Then, $\pi_{u_1,\dots,u_k,u'_1,\dots,u'_k,\ell}$ is equal to $\langle u''_1, \dots, u''_k \rangle$ maximizing co-lexicographically the string $\gamma' \cdot c$ defined as follows:

1. $u'_i = \delta(u''_i, c)$ for all $1 \leq i \leq k$,

2. $\pi_{u_1,\dots,u_k,u''_1,\dots,u''_k,\ell-1} \neq \perp$, and

3. γ' is the co-lexicographically largest string labeling all the paths $\alpha_{\ell-1}(u''_i)$. Note that this string exists by condition (2), and it can be easily built by following those paths in parallel (choosing, at each step, the largest character labeling all the k considered edges of the k paths).

If no $c \in \Sigma$ satisfies condition (1), or condition (2) cannot be met, then $\pi_{u_1,\dots,u_k,u'_1,\dots,u'_k,\ell} = \perp$.

Note that $\pi_{u_1,\dots,u_k,u_1,\dots,u_k,\ell}$ allows us to identify (if it exists) the largest string γ of length $\ell - 1$ labeling k cycles originating and ending in each u_i, for $1 \leq i \leq k$.

Each tuple $\pi_{u_1,\dots,u_k,u'_1,\dots,u'_k,\ell}$ can be computed in $|\delta|^{O(k)}$ time by dynamic programming (in order of increasing ℓ), and there are $|\delta|^{O(k)}$ such tuples to be computed (there are $|Q|^{O(k)} \leq |\delta|^{O(k)}$ ways of choosing $u_1, \dots, u_k, u'_1, \dots, u'_k$, and $N \leq |\delta|^{O(k)}$). Overall, also Step (2) can therefore be solved in $|\delta|^{O(k)}$ time.

To sum up, we can check if the conditions of Corollary 5.29 hold as follows:

1. We compute $\pi_{u,\ell}$ for each $u \in Q$ and $\ell \leq N$. This identifies a string μ_u^ℓ for each such pair $u \in Q$ and $\ell \leq N$: the co-lexicographically smallest one, of length $\ell - 1$, labeling a path connecting s with u.

2. For each k-tuple $u_1, \dots, u_k$ and each $\ell \leq N$, we compute $\pi_{u_1,\dots,u_k,u_1,\dots,u_k,\ell}$. This identifies a string $\gamma^\ell_{u_1,\dots,u_k}$ for each such tuple $u_1, \dots, u_k$ and $\ell \leq N$: the co-lexicographically largest one, of length $\ell - 1$, labeling k cycles originating and ending in each u_i, for $1 \leq i \leq k$.

3. We identify the k-tuple $u_1, \dots, u_k$ and the lengths $\ell_i < \ell \leq N$ (if they exist) such that $\mu_{u_i}^{\ell_i} \prec \gamma^\ell_{u_1,\dots,u_k}$ for all $1 \leq i \leq k$.

The conditions of Corollary 5.29 hold if and only if step 3 above succeeds for at least one k-tuple $u_1, \dots, u_k$ and lengths $\ell_i < \ell \leq N$, for $1 \leq i \leq k$. Overall, the algorithm terminates in $|\delta|^{O(k)} = |\delta|^{O(p)}$ time.

□

5.6 Relation with Star-Free Languages

Theorem 5.26 allows us to describe the levels of the width hierarchy looking to cycles in the minimum automata for the languages. This result resembles another very well known result on a class of subregular languages, the star-free ones, which can also be described by inspecting the cycles in the minimum DFA for the language.

Definition 5.31. A regular language is said to be star-free if it can be described by a regular expression constructed from the letters of the alphabet, the empty set symbol, all boolean operators (including complementation), and concatenation (but no Kleene star).

A well-known automata characterization of star-free languages is given by using counters. A counter in a DFA is a sequence of pairwise-distinct states $u_0, \dots, u_n$ (with $n \geq 1$) such that there exists a non-empty string α with $\delta(u_0, \alpha) = u_1, \dots, \delta(u_{n-1}, \alpha) = u_n, \delta(u_n, \alpha) = u_0$. A language is star-free if and only if its minimum DFA has no counters [111, 91].

We can easily prove that a Wheeler language, i.e. a language $\mathcal{L}$ with $\text{width}^N(\mathcal{L}) = \text{width}^D(\mathcal{L}) = 1$ for a fixed order of the alphabet, is always star-free. Indeed, if the minimum DFA for a language has a counter $u_0, \dots, u_n$ with string α, and $\gamma \in I_{u_0}$, then $(\gamma\alpha^n)_{n\in\mathbb{N}}$ is a monotone sequence (increasing or decreasing depending on which string between γ and $\gamma\alpha$ is smaller) which is not ultimately included in one class of the Myhill-Nerode equivalence $\equiv_{\mathcal{L}}$ (because in a minimum DFA the I_u's are exactly equal to Nerode classes). Hence, the language is not Wheeler by Corollary 5.25.

This implies that the first level of the deterministic width hierarchy is included in the class of star-free languages. On the other hand, in the next example we prove that there is an infinite sequence of star-free languages $(\mathcal{L}_n)_{n\in\mathbb{N}}$ over the two letter alphabet $\{a, b\}$ such that $\text{width}^D(\mathcal{L}_n) = n$, for both total orders $\preceq$ on $\{a, b\}$.

Example 5.32. In Figure 5.6 we depicted a DFA $\mathcal{D}_n$ with $3n$ states accepting the language $\mathcal{L}_n = \bigcup_{j=0}^{n-1} b^j ab^* a^{j+1}$. Notice that:

1. for every state u and for every $1 \leq j \leq n$, we have that $\delta(u, aba^j)$ is defined and final if and only if $u = q_j$;

2. for every state u in the second or third row and for every $1 \leq j \leq n$, we have that $\delta(u, ba^j)$ is defined and final if and only if $u = r_j$;

3. for every state u in the third row and for every $1 \leq j \leq n$, we have that $\delta(u, a^{j-1})$ is defined and final if and only if $u = s_j$.

We conclude that $\mathcal{D}_n$ is the minimum DFA of $\mathcal{L}_n$.

Since $\mathcal{D}_n$ has no counters (because every cycle is a self-loop), the above mentioned characterization of star-free languages tells us that $\mathcal{L}_n$ is star-free. Let us prove that $\text{ent}(\mathcal{D}_n) = n$, so that $\text{width}^D(\mathcal{L}_n) = n$ follows from Theorem 5.24. Notice that (1) states in the first row are reached by only one string, (2) states in the second row are reached infinitely many times only by string ending with b, and (3) states in the third row are reached only by strings ending with a. This implies $\text{ent}(\mathcal{D}_n) \leq n$, because the words belonging to a monotone sequence witnessing an entanglement between states will definitely end by the same letter, so only states belonging to the same row may belong to an entangled set. Finally, the n states in the second level are entangled , as it witnessed by the monotone sequence:

$$a \prec ba \prec bba \prec \dots \prec b^{n-1}a \prec ab \prec bab \prec$$
$$bbab \prec \dots \prec b^{n-1}ab \prec \dots \prec ab^k \prec bab^k \prec bbab^k \prec \dots$$

if $a \prec b$, and by the monotone sequence:

$$b^{n-1}a \succ b^{n-2}a \succ \dots \succ a \succ b^{n-1}ab \succ$$
$$b^{n-2}ab \succ \dots \succ ab \succ \dots \succ b^{n-1}ab^k \succ b^{n-2}ab^k \succ \dots \succ ab^k \succ \dots$$

if $b \prec a$. Hence, in both cases we have $\text{ent}(\mathcal{D}_n) = n$.

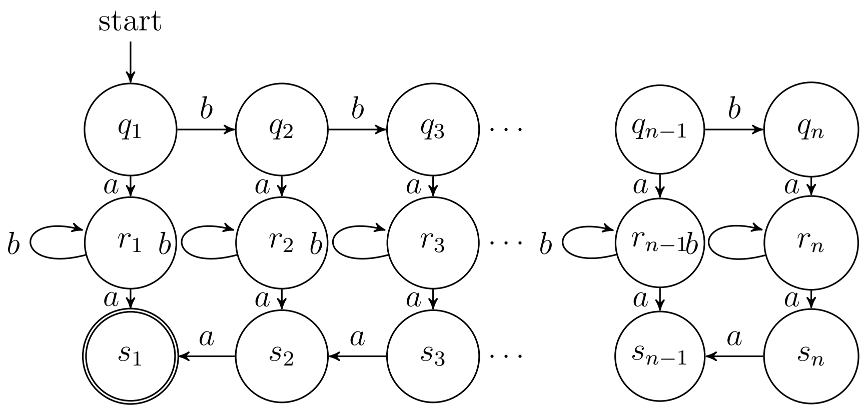


Figure 5.6: A minimum DFA $\mathcal{D}_n$ recognizing a star-free language $\mathcal{L}_n$ with $\text{width}^D(\mathcal{L}_n) = n$ for the two possible orders on the alphabet $\{a, b\}$.

5.7 The Convex Myhill-Nerode Theorem

In the previous sections we described a hierarchy of regular languages by means of their deterministic widths. A natural question is whether a corresponding Myhill-Nerode theorem can be provided for every level of the hierarchy: given a regular language $\mathcal{L}$, if we consider all DFAs recognizing $\mathcal{L}$ and having width equal to $\text{width}^D(\mathcal{L})$, is there a unique such DFA having the minimum number of states? In general, the answer is "no", as showed in Example 5.9.

The non-uniqueness can be explained as follows. If a DFA of width p recognizes $\mathcal{L}$, then $\text{Pref}(\mathcal{L})$ can be partitioned into p sets, each of which consists of the (disjoint) union of some pairwise comparable I_q's. However, in general the partition into p sets is not unique, so it may happen that two distinct partitions lead to two non-isomorphic minimal DFAs with the same number of states. For example, in Figure 5.1, we see two non-isomorphic DFAs (center and right) realizing the width of the language and with the minimum number of states among all DFAs recognizing the same language and realizing the width of the language: the chain partition $\{\{0,1,4\},\{2,3,5,3'\}\}$ of the DFA in the center induces the partition $\{ac^*\cup\{\varepsilon,e,h\}, bc^*\cup ac^*d\cup\{gd,ee,he,f,k,g\}\}$ of $\text{Pref}(\mathcal{L})$, whereas the chain partition $\{\{0,1,3\},\{2,4,5,4'\}\}$ of the DFA on the right induces the partition $\{ac^*\cup ac^*d\cup\{\varepsilon,gd,ee,he,f,k\}, bc^*\cup\{e,h,g\}\}$ of $\text{Pref}(\mathcal{L})$.

This example shows that no uniqueness results can be ensured as long as partitions are not fixed. But what happens if we fix a partition? As we will prove in this section, once a partition is fixed, it is possible to prove a full Myhill-Nerode theorem, thereby providing a DFA-free characterization of languages of width equal to p and a minimum DFA for these languages. In particular, if $p = 1$, then the partition contains a single set equal to $\text{Pref}(\mathcal{L})$, and we retrieve a Myhill-Nerode theorem for Wheeler language [5].

More formally, let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA, and let $\{Q_i \mid 1 \le i \le p\}$ be a $\le_{\mathcal{D}}$-chain partition of Q. For every $i \in \{1,\dots,p\}$, define:

$$\text{Pref}(\mathcal{L}(\mathcal{D}))^i = \{\alpha \in \text{Pref}(\mathcal{L}(\mathcal{D})) \mid \delta(s,\alpha) \in Q_i\}.$$

Then $\{\text{Pref}(\mathcal{L}(\mathcal{D})^i \mid 1 \le i \le p\}$ is a partition of $\text{Pref}(\mathcal{L}(\mathcal{D}))$, and from now on we will think of such a partition as fixed. We now consider the class of all DFAs accepting $\mathcal{L}$ and inducing the considered partition.

Definition 5.33. Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA, and let $\mathcal{P} = \{U_1,\dots,U_p\}$ be a partition of $\text{Pref}(\mathcal{L}(\mathcal{D}))$. We say that $\mathcal{D}$ is $\mathcal{P}$-sortable if there exists a $\le_{\mathcal{D}}$-chain partition $\{Q_i \mid 1 \le i \le p\}$ such that for every $i \in \{1,\dots,p\}$:

$$\text{Pref}(\mathcal{L}(\mathcal{D}))^i = U_i.$$

We wish to give a DFA-free characterization of languages $\mathcal{L}$ and partitions $\mathcal{P}$ of $\text{Pref}(\mathcal{L})$ for which there exists a $\mathcal{P}$-sortable DFA. As in the Myhill-Nerode theorem, we aim to determine which properties an equivalence relation $\sim$ should satisfy to ensure that a canonical construction provides a $\mathcal{P}$-sortable DFA. First, $\mathcal{L}$ must be regular, so $\sim$ is expected to be right-invariant. In order to develop some intuition on the required properties, let us consider an equivalence relation which plays a key role in the classical Myhill-Nerode theorem. Let $\mathcal{D} = (Q, s, \delta, F)$ be a $\mathcal{P}$-sortable DFA, and let $\equiv_{\mathcal{D}}$ be the equivalence relation on $\text{Pref}(\mathcal{L}(\mathcal{D}))$ defined by

$$\alpha \equiv_{\mathcal{D}} \beta \Leftrightarrow \delta(s, \alpha) = \delta(s, \beta).$$

Notice that equivalent strings end up in the same element of $\mathcal{P}$ (*$\mathcal{P}$-consistency*), and since all states in each $\leq_{\mathcal{D}}$-chain Q_i are comparable, then each I_q must be convex in the corresponding element of $\mathcal{P}$ (*$\mathcal{P}$-convexity*). More formally we consider the following definition, where, for every $\alpha \in \text{Pref}(\mathcal{L})$, we denote by U_α the unique element U_i of $\mathcal{P}$ such that $\alpha \in U_i$.

Definition 5.34. Let $\mathcal{L} \subseteq \Sigma^*$ be a language, and let $\sim$ be an equivalence relation on $\text{Pref}(\mathcal{L})$. Let $\mathcal{P} = \{U_1, \dots, U_p\}$ be a partition of $\text{Pref}(\mathcal{L})$.

1. We say that $\sim$ is *$\mathcal{P}$-consistent* if for every $\alpha, \beta \in \text{Pref}(\mathcal{L})$, if $\alpha \sim \beta$, then $U_\alpha = U_\beta$.

2. Assume that $\sim$ is $\mathcal{P}$-consistent. We say that $\sim$ is *$\mathcal{P}$-convex* if for every $\alpha \in \text{Pref}(\mathcal{L})$ we have that $[\alpha]_\sim$ is a convex in $(U_\alpha, \preceq)$.

As we now prove, these are exactly the required properties for a DFA-free characterization.

Let $\mathcal{L} \subseteq \Sigma^*$ be a language, and let $\sim$ be an equivalence relation on $\text{Pref}(\mathcal{L})$. We say that $\sim$ *respects* $\text{Pref}(\mathcal{L})$ if:

$$(\forall \alpha, \beta \in \text{Pref}(\mathcal{L}))(\forall \phi \in \Sigma^*)(\alpha \sim \beta \wedge \alpha\phi \in \text{Pref}(\mathcal{L}) \to \beta\phi \in \text{Pref}(\mathcal{L})).$$

Now, let us define the right-invariant, $\mathcal{P}$-consistent and $\mathcal{P}$-convex refinements of an equivalence relation $\sim$.

1. Assume that $\sim$ respects $\text{Pref}(\mathcal{L})$. For every $\alpha, \beta \in \text{Pref}(\mathcal{L})$, define:

$$\alpha \sim^r \beta \iff (\forall \phi \in \Sigma^*)(\alpha\phi \in \text{Pref}(\mathcal{L}) \to \alpha\phi \sim \beta\phi).$$

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

2. Let $\mathcal{P} = \{U_1, \ldots, U_p\}$ be a partition of Pref($\mathcal{L}$). For every $\alpha, \beta \in$ Pref($\mathcal{L}$), define:
$$\alpha \sim^{cs} \beta \iff (\alpha \sim \beta) \wedge (U_\alpha = U_\beta)$$
We say that $\sim^{cs}$ is the $\mathcal{P}$-*consistent refinement* of $\sim$.

3. Let $\mathcal{P} = \{U_1, \ldots, U_p\}$ be a partition of Pref($\mathcal{L}$). Assume that $\sim$ is $\mathcal{P}$-consistent. For every $\alpha, \gamma \in$ Pref($\mathcal{L}$), define:
$$\begin{aligned}&\alpha \sim^{cv} \gamma \iff (\alpha \sim \gamma) \wedge \\ &\quad \wedge (\forall \beta \in \mathrm{Pref}(\mathcal{L}))(((U_\alpha = U_\beta) \wedge (\min\{\alpha, \gamma\} \prec \beta \prec \max\{\alpha, \gamma\}) \to \alpha \sim \beta).\end{aligned}$$
We say that $\sim^{cv}$ is the $\mathcal{P}$-*convex refinement* of $\sim$.

It is easy to check that $\sim^r$ is the coarsest right-invariant equivalence relation refining $\sim$, $\sim^{cs}$ is the coarsest $\mathcal{P}$-consistent equivalence relation refining $\sim$ and $\sim^{cv}$ is the coarsest $\mathcal{P}$-convex equivalence relation refining $\sim$.

We wish to prove that any equivalence relation that respects Pref($\mathcal{L}$) admits a coarsest refinement being $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant at once, because then we will be able to define an equivalence relation inducing the minimum ($\mathcal{P}$-sortable) DFA. We first prove that if we use the operators *cv* and *r*, in this order, over a $\mathcal{P}$-consistent and right-invariant equivalence relation we do not lose $\mathcal{P}$-consistency, nor right-invariance, and we gain $\mathcal{P}$-convexity.

Lemma 5.35. *Let $\mathcal{L} \subseteq \Sigma^*$ be a language, and let $\mathcal{P}$ be a partition of Pref($\mathcal{L}$). If $\sim$ is a $\mathcal{P}$-consistent and right-invariant equivalence relation on Pref($\mathcal{L}$), then the relation $(\sim^{cv})^r$ is $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant.*

Proof. By definition $(\sim^{cv})^r$ is a right-invariant refinement. Moreover, $\sim^{cv}$ and $(\sim^{cv})^r$ are $\mathcal{P}$-consistent because they are refinements of the $\mathcal{P}$-consistent equivalence relation $\sim$. Let us prove that $(\sim^{cv})^r$ is $\mathcal{P}$-convex. Assume that $\alpha, \beta, \gamma \in$ Pref($\mathcal{L}$) are such that $\alpha(\sim^{cv})^r\gamma$, $\alpha \prec \beta \prec \gamma$ and $U_\alpha = U_\beta$. Being $(\sim^{cv})^r$ a $\mathcal{P}$-consistent relation, we have $U_\alpha = U_\beta = U_\gamma$. We must prove that $\alpha(\sim^{cv})^r\beta$. Fix $\phi \in \Sigma^*$ such that $\alpha\phi \in$ Pref($\mathcal{L}$). We must prove that $\alpha\phi \sim^{cv} \beta\phi$. Now, $\alpha(\sim^{cv})^r\gamma$ implies $\alpha \sim^{cv} \gamma$. Since $\alpha \prec \beta \prec \gamma$ and $U_\alpha = U_\beta = U_\gamma$, then the $\mathcal{P}$-convexity of $\sim^{cv}$ implies $\alpha \sim^{cv} \beta$. In particular, $\alpha \sim \beta$. Since $\sim$ is right-invariant we have $\alpha\phi \sim \beta\phi$, and from the $\mathcal{P}$-consistency of

$\sim$ we obtain $U_{\alpha\phi} = U_{\beta\phi}$. Moreover, $\alpha(\sim^{cv})^r\gamma$ implies $\alpha\phi(\sim^{cv})^r\gamma\phi$ by right-invariance, so $\alpha\phi \sim^{cv} \gamma\phi$. By $\mathcal{P}$-convexity, from $\alpha\phi \sim^{cv} \gamma\phi$, $U_{\alpha\phi} = U_{\beta\phi}$ and $\alpha\phi \prec \beta\phi \prec \gamma\phi$ (since $\alpha \prec \beta \prec \gamma$) we conclude $\alpha\phi \sim^{cv} \beta\phi$. □

Corollary 5.36. *Let $\mathcal{L} \subseteq \Sigma^*$ be a nonempty language, and let $\mathcal{P}$ be a partition of Pref($\mathcal{L}$). Let $\sim$ be an equivalence relation that respects Pref($\mathcal{L}$). Then, there exists a (unique) coarsest $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant equivalence relation refining $\sim$.*

Proof. The equivalence relation $(\sim^{cs})^r$ is $\mathcal{P}$-consistent (because it is a refinement of the $\mathcal{P}$-consistent equivalence relation $\sim^{cs}$) and right-invariant (by definition it is a right-invariant refinement), so by Lemma 5.35 the equivalence relation $(((\sim^{cs})^r)^{cv})^r$ is $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant. Moreover, every $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant equivalence relation refining $\sim$ must also refine $(((\sim^{cs})^r)^{cv})^r$, so $(((\sim^{cs})^r)^{cv})^r$ is the coarsest $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant equivalence relation refining $\sim$. □

Corollary 5.36 allows us to give the following definition.

Definition 5.37. Let $\mathcal{L} \subseteq \Sigma^*$ be a language, and let $\mathcal{P} = \{U_1, \dots, U_p\}$ be a partition of Pref($\mathcal{L}$). Denote by $\equiv^{\mathcal{P}}_{\mathcal{L}}$ the coarsest $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant equivalence relation refining the Myhill-Nerode equivalence $\equiv_{\mathcal{L}}$.

In particular, since $\mathcal{L}$ is the union of some $\equiv_{\mathcal{L}}$-classes, we also have that $\mathcal{L}$ is the union of some $\equiv^{\mathcal{P}}_{\mathcal{L}}$-classes.

Recall that, given a DFA $\mathcal{D} = (Q, s, \delta, F)$, the equivalence relation $\equiv_{\mathcal{D}}$ on Pref($\mathcal{L}(\mathcal{D})$) is the one such that:

$$\alpha \equiv_{\mathcal{D}} \beta \iff \delta(s, \alpha) = \delta(s, \beta).$$

Here are the key properties of $\equiv_{\mathcal{D}}$, when $\mathcal{D}$ is a $\mathcal{P}$-sortable DFA.

Lemma 5.38. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a $\mathcal{P}$-sortable DFA, where $\mathcal{P} = \{U_1, \dots, U_p\}$ is a partition of Pref($\mathcal{L}$) for $\mathcal{L} = \mathcal{L}(\mathcal{D})$. Then, $\equiv_{\mathcal{D}}$ has finite index, it respects Pref($\mathcal{L}$), it is right-invariant, $\mathcal{P}$-consistent, $\mathcal{P}$-convex, it refines $\equiv^{\mathcal{P}}_{\mathcal{L}}$, and $\mathcal{L}$ is the union of some $\equiv_{\mathcal{D}}$-classes. In particular, $\equiv^{\mathcal{P}}_{\mathcal{L}}$ has finite index.*

Proof. The relation $\equiv_\mathcal{D}$ has index equal to $|Q|$. It respects $\text{Pref}(\mathcal{L})$ because if $\alpha \equiv_\mathcal{D} \beta$ and $\phi \in \Sigma^*$ satisfies $\alpha\phi \in \text{Pref}(\mathcal{L})$, then there exists γ with $\alpha\phi\gamma \in \mathcal{L}$ so $\delta(s, \alpha\phi\gamma) \in F$. Since $\delta(s, \alpha) = \delta(s, \beta)$ we obtain $\delta(s, \alpha\phi\gamma) = \delta(s, \beta\phi\gamma)$ and so $\beta\phi\gamma \in \mathcal{L}$ and $\beta\phi \in \text{Pref}(\mathcal{L})$ follows. Moreover, it is right-invariant because if $\alpha \equiv_\mathcal{D} \beta$ and $\phi \in \Sigma^*$ is such that $\alpha\phi\gamma \in \mathcal{L}$, then $\beta\phi \in \text{Pref}(\mathcal{L})$ and from $\delta(s, \alpha) = \delta(s, \beta)$ we obtain $\delta(s, \alpha\phi) = \delta(s, \beta\phi)$.

For every $\alpha \in \text{Pref}(\mathcal{L})$ we have $[\alpha]_{\equiv_\mathcal{D}} = I_{\delta(s,\alpha)}$, which implies that $\equiv_\mathcal{D}$ is $\mathcal{P}$-consistent. Moreover, $\equiv_\mathcal{D}$ is $\mathcal{P}$-convex, that is, for every $\alpha \in \text{Pref}(\mathcal{L})$ we have that $[\alpha]_{\equiv_\mathcal{D}} = I_{\delta(s,\alpha)}$ is convex in U_α, because if $u_1, \dots, u_k \in Q$ are such that $U_\alpha = \bigcup_{i=1}^k I_{u_i}$, then the u_i's must be pairwise $\leq_\mathcal{D}$-comparable, being in the same $\leq_\mathcal{D}$-chain. Moreover, $\equiv_\mathcal{D}$ refines $\equiv_\mathcal{L}$, because $\alpha \equiv_\mathcal{D} \beta$ implies that for every $\phi \in \Sigma^*$ we have $\delta(s, \alpha\phi) = \delta(s, \beta\phi)$ and so $\alpha\phi \in \mathcal{L}$ iff $\beta\phi \in \mathcal{L}$. Since $\equiv_\mathcal{L}^\mathcal{P}$ is the coarsest $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant equivalence relation refining $\equiv_\mathcal{L}$, and $\equiv_\mathcal{D}$ is a $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant equivalence relation refining $\equiv_\mathcal{L}$, we conclude that $\equiv_\mathcal{D}$ also refines $\equiv_\mathcal{L}^\mathcal{P}$, which in particular implies that $\mathcal{L}$ is the union of some $\equiv_\mathcal{D}$-classes. We know that $\equiv_\mathcal{D}$ has finite index, so $\equiv_\mathcal{L}^\mathcal{P}$ has finite index. □

We can now explain how to canonically build a $\mathcal{P}$-sortable DFA starting from an equivalence relation.

Lemma 5.39. *Let $\mathcal{L} \subseteq \Sigma^*$ be a language, and let $\mathcal{P} = \{U_1, \dots, U_p\}$ be a partition of Pref($\mathcal{L}$). Assume that $\mathcal{L}$ is the union of some classes of a $\mathcal{P}$-consistent, $\mathcal{P}$-convex, right-invariant equivalence relation $\sim$ on Pref($\mathcal{L}$) of finite index. Then, $\mathcal{L}$ is recognized by a $\mathcal{P}$-sortable DFA $\mathcal{D}_\sim = (Q_\sim, s_\sim, \delta_\sim, F_\sim)$ such that:*

1. *$|Q_\sim|$ is equal to the index of $\sim$;*
2. *$\equiv_{D_\sim}$ and $\sim$ are the same equivalence relation (in particular, $|Q_\sim|$ is equal to the index of $\equiv_{D_\sim}$).*

Moreover, if $\mathcal{B}$ is a $\mathcal{P}$-sortable DFA that recognizes $\mathcal{L}$, then $\mathcal{D}_{\equiv_\mathcal{B}}$ is isomorphic to $\mathcal{B}$.

Proof. Define the DFA $\mathcal{D}_\sim = (Q_\sim, s_\sim, \delta_\sim, F_\sim)$ as follows.

- $Q_\sim = \{[\alpha]_\sim \mid \alpha \in \text{Pref}(\mathcal{L})\}$;
- $s_\sim = [\varepsilon]_\sim$, where ε is the empty string;

- $\delta_\sim([\alpha]_\sim, a) = [\alpha a]_\sim$, for every $\alpha \in \Sigma^*$ and $a \in \Sigma$ such that $\alpha a \in \text{Pref}(\mathcal{L})$.
- $F_\sim = \{[\alpha]_\sim \mid \alpha \in \mathcal{L}\}$.

Since $\sim$ is right-invariant, it has finite index and $\mathcal{L}$ is the union of some $\sim$-classes, then $\mathcal{D}_\sim$ is a well-defined DFA and:

$$\alpha \in [\beta]_\sim \iff \delta_\sim(s_\sim, \alpha) = [\beta]_\sim. \tag{5.6}$$

which implies that for every $\alpha \in \text{Pref}(\mathcal{L})$ it holds $I_{[\alpha]_\sim} = [\alpha]_\sim$, and so $\mathcal{L}(\mathcal{D}_\sim) = \mathcal{L}$.

For every $i \in \{1, \dots, p\}$, define:

$$Q_i = \{[\alpha]_\sim \mid U_\alpha = U_i\}.$$

Notice that each Q_i is well-defined because $\sim$ is $\mathcal{P}$-consistent, and each Q_i is a $\leq_{\mathcal{D}_\sim}$-chain because $\sim$ is $\mathcal{P}$-convex. It follows that $\{Q_i \mid 1 \leq i \leq p\}$ is a $\leq_{\mathcal{D}_\sim}$-chain partition of $Q_\sim$.

From Equation 5.6 we obtain:

$$\begin{aligned}\text{Pref}(\mathcal{L}(\mathcal{D}_\sim))^i &= \{\alpha \in \text{Pref}(\mathcal{L}(\mathcal{D}_\sim)) \mid \delta_\sim(s_\sim, \alpha) \in Q_i\} \\ &= \{\alpha \in \text{Pref}(\mathcal{L}(\mathcal{D}_\sim)) \mid (\exists [\beta]_\sim \in Q_i \ \alpha \in [\beta]_\sim)\} \\ &= \{\alpha \in \text{Pref}(\mathcal{L}(\mathcal{D}_\sim)) \mid U_\alpha = U_i\} = U_i.\end{aligned}$$

In other words, $\mathcal{D}_\sim$ witnesses that $\mathcal{L}$ is recognized by a $\mathcal{P}$-sortable DFA. Moreover:

1. The number of states of $\mathcal{D}_\sim$ is clearly equal to the index of $\sim$.
2. By Equation 5.6:

$$\alpha \equiv_{D_\sim} \beta \iff \delta_\sim(s_\sim, \alpha) = \delta_\sim(s_\sim, \beta) \iff [\alpha]_\sim = [\beta]_\sim \iff \alpha \sim \beta$$

   so $\equiv_{D_\sim}$ and $\sim$ are the same equivalence relation.

Finally, suppose $\mathcal{B}$ is a $\mathcal{P}$-sortable DFA that recognizes $\mathcal{L}$. Notice that by Lemma 5.38 we have that $\equiv_\mathcal{B}$ is a $\mathcal{P}$-consistent, $\mathcal{P}$-convex, right-invariant equivalence relation on $\text{Pref}(\mathcal{L})$ of finite index such that $\mathcal{L}$ is the union of some $\equiv_\mathcal{B}$-classes, so $\mathcal{D}_{\equiv_\mathcal{B}}$ is well-defined. Call $Q_\mathcal{B}$ the set of states of $\mathcal{B}$, and let $\phi : Q_{\equiv_\mathcal{B}} \to Q_\mathcal{B}$ be the function sending $[\alpha]_{\equiv_\mathcal{B}}$ into the state in $Q_\mathcal{B}$ reached by reading α. Notice that ϕ is well-defined because by the definition of $\equiv_\mathcal{B}$ we obtain that all strings in $[\alpha]_{\equiv_\mathcal{B}}$ reach the same state of $\mathcal{B}$. It is easy to check that ϕ determines an isomorphism between $\mathcal{D}_{\equiv_\mathcal{B}}$ and $\mathcal{B}$. $\square$

We now have all the required definitions to state our Myhill-Nerode theorem, which generalizes the one for Wheeler languages [4].

Theorem 5.40 (Convex Myhill-Nerode Theorem)**.** *Let $\mathcal{L}$ be a language. Let $\mathcal{P}$ be a partition of Pref($\mathcal{L}$). The following are equivalent:*

1. *$\mathcal{L}$ is recognized by a $\mathcal{P}$-sortable DFA.*
2. *$\equiv_{\mathcal{L}}^{\mathcal{P}}$ has finite index.*
3. *$\mathcal{L}$ is the union of some classes of a $\mathcal{P}$-consistent, $\mathcal{P}$-convex, right-invariant equivalence relation on Pref($\mathcal{L}$) of finite index.*

Moreover, if one of the above statements is true (and so all the above statements are true), then there exists a unique minimum $\mathcal{P}$-sortable DFA recognizing $\mathcal{L}$ (that is, two $\mathcal{P}$-sortable DFAs recognizing $\mathcal{L}$ having the minimum number of states must be isomorphic).

Proof. (1) $\to$ (2) It follows from Lemma 5.38.

(2) $\to$ (3) The desired equivalence relation is simply $\equiv_{\mathcal{L}}^{\mathcal{P}}$.

(3) $\to$ (1) It follows from Lemma 5.39.

Now, let us prove that the minimum DFA is $\mathcal{D}_{\equiv_{\mathcal{L}}^{\mathcal{P}}}$ as defined in Lemma 5.39. First, $\mathcal{D}_{\equiv_{\mathcal{L}}^{\mathcal{P}}}$ is well-defined because $\equiv_{\mathcal{L}}^{\mathcal{P}}$ is $\mathcal{P}$-consistent, $\mathcal{P}$-convex and right-invariant by definition; moreover, it has finite index and $\mathcal{L}$ is the union of some $\equiv_{\mathcal{L}}^{\mathcal{P}}$-equivalence classes by Lemma 5.38. Now, the number of states of $\mathcal{D}_{\equiv_{\mathcal{L}}^{\mathcal{P}}}$ is equal to the index of $\equiv_{\mathcal{L}}^{\mathcal{P}}$, or equivalently, of $\equiv_{\mathcal{D}_{\equiv_{\mathcal{L}}^{\mathcal{P}}}}$. On the other hand, let $\mathcal{B}$ be any $\mathcal{P}$-sortable DFA recognizing $\mathcal{L}$ non-isomorphic to $\mathcal{D}_{\equiv_{\mathcal{L}}^{\mathcal{P}}}$. Then $\equiv_{\mathcal{B}}$ is a refinement of $\equiv_{\mathcal{L}}^{\mathcal{P}}$ by Lemma 5.38, and it must be a strict refinement of $\equiv_{\mathcal{L}}^{\mathcal{P}}$, otherwise $\mathcal{D}_{\equiv_{\mathcal{L}}^{\mathcal{P}}}$ would be equal to $\mathcal{D}_{\equiv_{\mathcal{B}}}$, which by Lemma 5.39 is isomorphic to $\mathcal{B}$, a contradiction. We conclude that the index of $\equiv_{\mathcal{L}}^{\mathcal{P}}$ is smaller than the index of $\equiv_{\mathcal{B}}$, so again by Lemma 5.39 the number of states of $\mathcal{D}_{\equiv_{\mathcal{L}}^{\mathcal{P}}}$ is smaller than the number of states of $\mathcal{D}_{\equiv_{\mathcal{B}}}$ and so of $\mathcal{B}$. □

Notice that for a language $\mathcal{L}$ Definition 5.33 implies that $\text{width}^D(\mathcal{L}) = p$ if and only if (i) there exists a partition $\mathcal{P}$ of size p such that $\mathcal{L}$ is recognized by a $\mathcal{P}$-sortable DFA and (ii) for every partition $\mathcal{P}'$ of size less than p it holds that $\mathcal{L}$ is not recognized by a $\mathcal{P}'$-sortable DFA. As a consequence, $\text{width}^D(\mathcal{L}) = p$ if and only if the minimum cardinality of a partition $\mathcal{P}$ of $\text{Pref}(\mathcal{L})$ that satisfies any of the statements in Theorem 5.40 is equal to p.

5.8 Minimization of Wheeler DFAs

Definition 5.33 implies that a DFA is Wheeler if and only if it is $\{\text{Pref}(\mathcal{L})\}$-sortable. From Theorem 5.40 we obtain that if $\mathcal{L}$ is a Wheeler language, there there exists a unique state-minimal Wheeler DFA recognizing $\mathcal{L}$ — the *minimum Wheeler DFA* recognizing $\mathcal{L}$. The problem of minimizing the states a DFA $\mathcal{D} = (Q, s, \delta, F)$ is a classical problem in automata theory that can be solved by means of Hopcroft's algorithm [73]. In this section, given a Wheeler DFA $\mathcal{D} = (Q, s, \delta, F)$, we want to determine the minimum Wheeler DFA recognizing $\mathcal{L}(\mathcal{D})$.

Let us first revise Hopcroft's algorithm. Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. We will now consider equivalence relations defined on Q, not on $\text{Pref}(\mathcal{L}(\mathcal{D}))$. We say that an equivalence relation $\sim$ on Q is *right-invariant* if for every $u, v \in Q$ such that $u \sim v$ and for every $a \in \Sigma$, $\delta(u, a)$ is defined if and only if $\delta(v, a)$ is defined, and, if they are defined, it holds $\delta(u, a) \sim \delta(v, a)$. As usual, we denote the equivalence class of v with $[v]_\sim$.

Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA and let $\sim$ be a right-invariant equivalence relation on Q, Define$\mathcal{D}_{/\sim} = (Q_\sim, \delta_\sim, [s]_\sim, F_\sim)$, where $Q_\sim = \{[u]_\sim \mid u \in Q\}$, $\delta_\sim([u]_\sim, a) = [v]_\sim$ if and only if $\delta(u, a) = v$, and $F_\sim = \{[u]_\sim \mid u \in F\}$. A classic and simple result is that, since $\sim$ is right-invariant, then $\mathcal{D}_{/\sim}$ is a well-defined DFA such that $\mathcal{L}(\mathcal{A}_{/\sim}) = \mathcal{L}(\mathcal{A})$; moreover, $I_{[u]_\sim} = \bigcup_{u' \in [u]_\sim} I_{u'}$ for every $u \in Q$ [74].

Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. Consider the right-invariant equivalence relation $\approx_\mathcal{D}$ on Q such that for every $u, v \in Q$ it holds $u \approx_\mathcal{D} v$ if and only if for every $\alpha \in \Sigma^*$ we have $\delta(u, \alpha) \in F$ if and only if $\delta(v, \alpha) \in F$ (the expression $\delta(u, \alpha) \in F$ means that the state $\delta(u, \alpha)$ is defined and it is in F). The equivalence relation $\approx_\mathcal{D}$ can be seen as a state equivalent of the Myhill-Nerode equivalence $\equiv_{\mathcal{L}(\mathcal{D})}$. Then, $\mathcal{D}_{/\approx_\mathcal{D}}$ is the minimum DFA recognizing $\mathcal{L}(\mathcal{D})$ [74]. Given $\approx_\mathcal{D}$, the minimum DFA $\mathcal{D}_{/\approx_\mathcal{D}}$ can be built in linear time, so the time required to compute the minimum DFA is the time required to build $\approx_\mathcal{D}$. Hopcroft's algorithm shows that $\approx_\mathcal{D}$ can be built in $O(|\delta| \log |Q|)$ time.

Now, let $\mathcal{D} = (Q, s, \delta, F)$ be a Wheeler DFA, where $Q = \{u_1, \dots, u_n\}$ and $u_1 <$

$u_2 < \cdots < u_n$ in the Wheeler order $\leq$. Consider the right-invariant equivalence relation $\approx^c_{\mathcal{D}}$ on Q that puts in the same equivalence classes exactly all states belonging to the maximum runs of states $u_i, u_{i+1}, \dots, u_{i+t}$ for which $u_i \approx_{\mathcal{A}} u_{i+1} \approx_{\mathcal{A}} \cdots \approx_{\mathcal{A}} u_{i+t}$. Then, $\approx^c_{\mathcal{D}}$ is right-invariant by the following lemma.

Lemma 5.41. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a Wheeler DFA, where $Q = \{u_1, \dots, u_n\}$ and $u_1 < u_2 < \cdots < u_n$ in the Wheeler order $\leq$. Let $\alpha \in \Sigma^*$ and $1 \leq i < n$ be such that $\delta(u_i, \alpha)$ and $\delta(u_{i+1}, \alpha)$ are both-defined and distinct. Then, there exists $1 \leq j < n$ such that $u_j = \delta(u_i, \alpha)$ and $u_{j+1} = \delta(u_{i+1}, \alpha)$.*

Proof. Without loss of generality, we can assume that $\alpha = a \in \Sigma$ is a character (the claim will follow by extension).

Let $1 \leq j, k \leq n$ be such that $u_j = \delta(u_i, \alpha)$ and $u_k = \delta(u_i, \alpha)$. We must prove that $k = j + 1$. We know that $j \neq k$. If it were $k < j$, then by Axiom 2 and Remark 4.2 we would conclude $i + 1 < i$, a contradiction, so it must be $j < k$. Suppose for the sake of contradiction that $j + 1 < k$. By Axiom 1, there exists $1 \leq i' \leq n$ such that $u_{j+1} = \delta(u_{i'}, a)$. By Axiom 2 and Remark 4.2 we would conclude $i < i' < i + 1$, a contradiction. $\square$

Since $\approx^c_{\mathcal{D}}$ is right-invariant, then $\mathcal{D}_{/\approx^c_{\mathcal{D}}}$ is a well-defined DFA. It is easy to show that $\mathcal{D}_{/\approx^c_{\mathcal{D}}}$ is the minimum Wheeler DFA recognizing $\mathcal{L}(\mathcal{D})$, because by the proof of Theorem 5.40 the minimum Wheeler DFA is $\mathcal{D}_{\equiv^{\mathcal{P}}_{\mathcal{L}}}$, with $\mathcal{L} = \mathcal{L}(\mathcal{D})$ and $\mathcal{P} = \{\text{Pref}(\mathcal{L}(\mathcal{D}))\}$, and $\mathcal{D}_{/\approx^c_{\mathcal{D}}}$ is isomorphic to $\mathcal{D}_{\equiv^{\mathcal{P}}_{\mathcal{L}}}$ by construction. As a consequence, we are only left with the problem of determining $\approx^c_{\mathcal{D}}$. Since $\approx_{\mathcal{D}}$ can be computed in $O(|\delta| \log |Q|)$ time, we can also compute $\approx^c_{\mathcal{D}}$ in $O(|\delta| \log |Q|)$ time. In the remaining of this section, we show that the properties of a Wheeler DFA allow computing $\approx^c_{\mathcal{D}}$ in $O(|\delta|)$ time, thus leading to the following theorem.

Theorem 5.42. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a Wheeler DFA. We can compute the minimum Wheeler DFA recognizing $\mathcal{L}(\mathcal{D})$ in $O(|\delta|)$ time.*

In order to prove the previous theorem, let us define the *border graph* of a WDFA:

Definition 5.43 (border graph)**.** Let $\mathcal{D} = (Q, s, \delta, F)$ be a Wheeler DFA, where $Q = \{u_1, \dots, u_n\}$ and $u_1 < u_2 < \cdots < u_n$ in the Wheeler order $\leq$. The border graph of $\mathcal{D}$ is the (unlabeled) graph $\mathcal{B}(\mathcal{D}) = (B, Z)$ where $B = \{(u_i, u_{i+1}) \mid 1 \leq i < n\}$ and $Z = \{((u_i, u_{i+1}), (u_j, u_{j+1})) \in B \times B \mid u_i = \delta(u_j, a) \ \wedge \ u_{i+1} = \delta(u_{j+1}, a), a \in \Sigma\}$.

In other words, an edge of $\mathcal{B}(\mathcal{D})$ exists between borders (u_i, u_{i+1}) and (u_j, u_{j+1}) whenever u_i (respectively, u_{i+1}) can be reached by u_j (respectively, u_{j+1}) by an edge labeled with the same character a. Note that in this case it must necessarily be $a = \max_{\lambda(u_i)} = \min_{\lambda(u_{i+1})}$.

In general, $\mathcal{B}(\mathcal{D})$ may contain cycles. With the next lemma we put a bound to the size of $\mathcal{B}(\mathcal{D})$, by showing that the maximum out-degree in the graph is at most one.

Lemma 5.44. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a Wheeler DFA. Then, $\mathcal{B}(\mathcal{D})$ has at most $|Q|-1$ edges and $|Q|-1$ vertices. Moreover, $\mathcal{B}(\mathcal{A})$ can be constructed in $O(|\delta|)$ time.*

Proof. Clearly, $|B| \leq n-1$ since elements of B are pairs of adjacent (in Wheeler order) states of $\mathcal{D}$.

We now show that for every $(u_i, u_{i+1}) \in B$, there exists at most one $(u_j, u_{j+1}) \in B$ such that $((u_i, u_{i+1}), (u_j, u_{j+1})) \in Z$. Indeed, if $u_r, u_s \in Q$ are any states such that $u_i = \delta(u_r, a)$ and $u_{i+1} = \delta(u_s, a)$, then from $u_i < u_{i+1}$ and from Axiom 2 it follows $u_r < u_s$, and $(u_r, u_s) \in B$ if and only if r is the largest integer such that $u_i = \delta(u_r, a)$, s is the smallest integer such that $u_{i+1} = \delta(u_s, a)$, $s = r+1$ and $\max_{\lambda(u_r)} = \min_{\lambda(u_s)}$. In other words, $|Z| \leq |B| \leq n-1$.

Finally, $\mathcal{B}(\mathcal{D})$ can be built in $O(|\delta|)$ time as follows. Consider the list $u_1 < \cdots < u_n$ of $\mathcal{A}$'s states, sorted in Wheeler order. For each (u_i, u_{i+1}) and each letter a labeling a transition leaving u_i, let $v = \delta(u_i, a)$ and $v' = \delta(u_{i+1}, a)$ (note that the outgoing edges of each node can be sorted in linear time by their label to speed up this operation). If both v and v' exist and are distinct, they must indeed be adjacent in Wheeler order by Lemma 5.41: $v = u_j$ and $v' = u_{j+1}$, for some $1 \leq j < n$. Then, insert in $\mathcal{B}(\mathcal{D})$ an edge $((u_j, u_{j+1}), (u_i, u_{i+1}))$. □

We describe our minimization algorithm as Algorithm 1. In line 1 we compute the border graph $(B, Z) = \mathcal{B}(\mathcal{D})$ of $\mathcal{D}$. This is done in linear time, see Lemma 5.44. In Lines 3-5, we mark base-case nodes: for every pair of adjacent nodes, if they are not both final/not final, or if their sets of outgoing labels are not equal then they cannot be $\equiv^c_{\mathcal{D}}$-equivalent. This step takes $O(|\delta|)$ time. In Line 7 we perform a linear-time visit of $\mathcal{B}(\mathcal{D})$ starting from the nodes that have been marked in Lines 3-5. During this visit, we mark every visited node. In Lines 7-9 we compute

Algorithm 1 Input: A Wheeler DFA $\mathcal{D} = (Q, s, \delta, F)$, where $Q = \{u_1, \ldots, u_n\}$ and $u_1 < u_2 < \cdots < u_n$ in the Wheeler order $\leq$. Output: The minimum Wheeler DFA recognizing $\mathcal{L}(\mathcal{D})$.

```
1: $(B, Z) \leftarrow$ border_graph($\mathcal{D}$)                      ▷ Compute border graph $\mathcal{B}(\mathcal{D})$ of $\mathcal{D}$
2: for $i = 1, \ldots, n-1$ do
3:     if $out(u_i) \neq out(u_{i+1}) \vee final(u_i) \neq final(u_{i+1})$ then
4:         mark($(u_i, u_{i+1})$)                                   ▷ *Mark a "base-case" node of $\mathcal{B}(\mathcal{D})$
5:     end if
6: end for
7: mark_reachable($B, Z$)                                           ▷ Propagate "base-case" marked nodes
8: for $i = 1, \ldots, n-1$ do
9:     if not marked($(u_i, u_{i+1})$) then
10:        make_equivalent($u_i, u_{i+1}$)                          ▷ Record that $u_i \approx^c_{\mathcal{D}} u_{i+1}$
11:    end if
12: end for
13: return $\mathcal{D}_{/\approx^c_{\mathcal{D}}}$                  ▷ Compute and return quotient automaton
```

the equivalence classes of $\equiv^c_{\mathcal{D}}$. In Line 8, the predicate $marked((u_i, u_{i+1}))$ returns true if and only if $(u_i, u_{i+1}) \in B$ has been marked in the previous lines. Procedure make_equivalent(u_i, u_{i+1}) at Line 10 records that nodes u_i, u_{i+1} belong to the same equivalence class of $\approx^c_{\mathcal{D}}$. To conclude, at Line 13 we return the quotient automaton $\mathcal{D}_{/\approx^c_{\mathcal{D}}}$, the minimum Wheeler DFA recognizing $\mathcal{L}(\mathcal{D})$. The Wheeler DFA $\mathcal{D}_{/\approx^c_{\mathcal{D}}}$ can be computed in $O(|\delta|)$ time by collapsing each equivalence class of $\approx^c_{\mathcal{D}}$ (intervals in Wheeler order).

In Figure 5.7 we pictorially show how Algorithm 1 minimizes a WDFA.

In order to prove Theorem 5.42, we are only left with showing that Algorithm 1 is correct. Our claim will follow if we prove that $u_i \not\approx_{\mathcal{D}} u_{i+1}$ if and only (u_i, u_{i+1}) is marked in $\mathcal{B}(\mathcal{D})$.

($\Leftarrow$) Suppose that (u_i, u_{i+1}) is marked. Then, by the definition of $\mathcal{B}(\mathcal{D})$ this means that there exists a pair (u_j, u_{j+1}) (possibly, $i = j$) that was marked in Line 4 (in particular, $u_j \not\approx_{\mathcal{D}} u_{j+1}$) such that (u_i, u_{i+1}) is reachable from (u_j, u_{j+1}) in $\mathcal{B}(\mathcal{D})$. In turn, by the definition of $\mathcal{B}(\mathcal{D})$ this implies that there exists a string α such that $\delta(u_i, \alpha) = u_j$ and $\delta(u_{i+1}, \alpha) = u_{j+1}$. Since $\approx_{\mathcal{D}}$ is right-invariant, we conclude $u_i \not\approx_{\mathcal{D}} u_{i+1}$.

($\Rightarrow$) Conversely, suppose $u_i \not\approx_{\mathcal{D}} u_{i+1}$. Then, there must exist two states v, v' such that $\delta(u_i, \alpha) = v$ and $\delta(u_{i+1}, \alpha) = v'$ for some string α, with either $out(v) \neq out(v')$ or $final(v) \neq final(v')$ (in particular, $v \neq v'$). Indeed, let α' be a shortest string witnessing that $u_i \not\approx_{\mathcal{A}} u_{i+1}$. If $\delta(u_i, \alpha')$ and $\delta(u_{i+1}, \alpha')$ are both defined, let $v = \delta(u_i, \alpha')$, $v' = \delta(u_{i+1}, \alpha')$ and $\alpha = \alpha'$ (in this case $final(v) \neq final(v')$). If exactly one between $\delta(u_i, \alpha')$ and $\delta(u_{i+1}, \alpha')$ is not defined, then α' is not the empty string, so we can write $\alpha' = \alpha'' a$ with $\alpha'' \in \Sigma^*$ and $a \in \Sigma$, where both $\delta(u_i, \alpha'')$ and $\delta(u_{i+1}, \alpha'')$ are defined (by the minimality of α'), so let $v = \delta(u_i, \alpha'')$, $v' = \delta(u_{i+1}, \alpha'')$ and $\alpha = \alpha''$ (in this case $out(v) \neq out(v')$). From Lemma 5.41, v and v' must be adjacent in Wheeler order, i.e. $v = u_j$ and $v' = u_{j+1}$ for some $1 \leq j < n$. This implies that (i) (u_j, u_{j+1}) is marked in Line 4 and (ii) (u_i, u_{i+1}) is reachable from (u_j, u_{j+1}) in $\mathcal{B}(\mathcal{A})$. Finally, (i) and (ii) imply that (u_i, u_{i+1}) is marked during the visit of $\mathcal{B}(\mathcal{A})$ in Line 7.

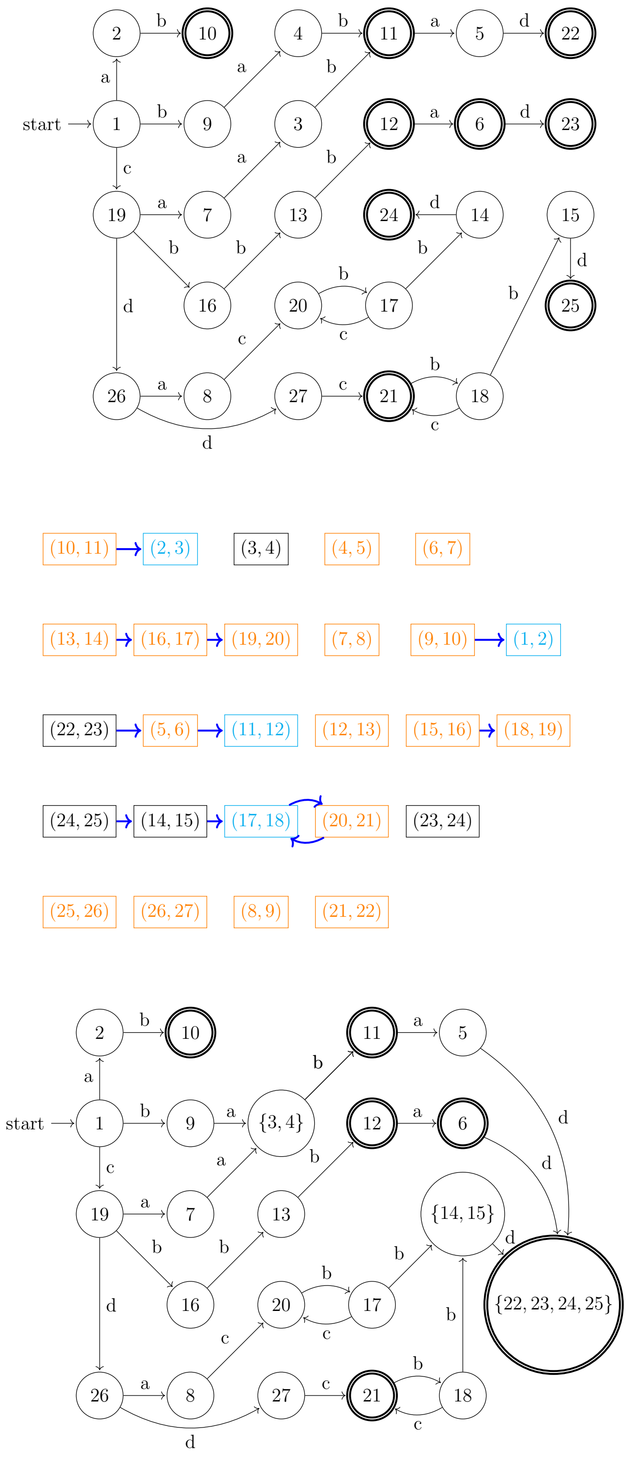


Figure 5.7: *Top*: a sorted WDFA $\mathcal{A}$ (node labels indicate the Wheeler order). *Center*: the border graph $\mathcal{B}(\mathcal{A})$ built at Line 2 of Algorithm 1. Nodes marked at Line 4 of the algorithm are orange, nodes marked at Line 7 are light blue. *Bottom*: the minimum WDFA recognizing $\mathcal{L}(\mathcal{A})$. Borders not marked (colored) in $\mathcal{B}(\mathcal{A})$ have been collapsed.

Chapter 6

Co-lex Relations

In Chapter 4 we defined co-lex orders, and we showed how they can be used to index and compress NFAs. If $\mathcal{N}$ is an NFA, then Theorem 4.47 requires having a co-lex order on $\mathcal{N}$, and the smaller its width, the better. As a consequence, we should determine a co-lex order of minimum width on $\mathcal{N}$. However, in Section 4.1, we saw that determining width($\leq_{\mathcal{N}}$) is NP-hard (but becomes a polynomial problem if $\mathcal{N}$ is a DFA, see Corollary 4.17).

In this chapter, we show that we can overcome this limitation by switching from co-lex order to *co-lex relations*. We will show that the problem of determining a minimum-width co-lex relation of an NFA (i) can be solved in polynomial time and (ii) leads to bounds at least as good as the one in Theorem 4.47.

Let us generalize the definition of co-lex order (Definition 4.1).

Definition 6.1. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA. A *co-lex relation* on G is a reflexive relation $R \subseteq Q \times Q$ that satisfies the following two axioms:

1. (Axiom 1) For every $u, v \in Q$ such that $u \neq v$, if $(u, v) \in R$, then $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$;

2. (Axiom 2) For every $a \in \Sigma$ and $u, v, u', v' \in Q$ such that $u \neq v$, if $u \in \delta(u', a)$, $v \in \delta(v', a)$ and $(u, v) \in R$, then $(u', v') \in R$.

A *co-lex preorder* is a co-lex relation that is also a preorder.

Remark 6.2. (1) Let $u \in Q$ be a state with no incoming edges, and let $v \in Q$ a state with incoming edges. From Axiom 1, it follows $(v, u) \notin R$. (2) If for distinct $u, v \in Q$ it holds $(u, v) \in R$ and $(v, u) \in R$, then by Axiom 1 we obtain $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$ and $\max_{\lambda(v)} \preceq \min_{\lambda(u)}$, so $\lambda(u) = \lambda(v)$ and $|\lambda(u)| = |\lambda(v)| = 1$.

Remark 6.3. Every NFA $\mathcal{N} = (Q, s, \delta, F)$ admits a co-lex relation. For example, $\{(v, v) \mid v \in Q\}$ and $\{(u, v) \in Q \times Q \mid \max_{\lambda(u)} \prec \min_{\lambda(v)}\} \cup \{(v, v) \mid v \in Q\}$ are co-lex relations on G.

We saw that the key property for indexing is *path coherence* (Lemma 4.23). Let us see how to generalize it to co-lex relations.

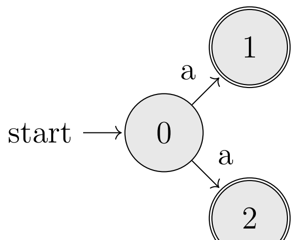
Lemma 6.4 (Path coherence)**.** *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let R be a co-lex relation on $\mathcal{N}$. Let $\alpha \in \Sigma^*$, and let $U \subseteq Q$ be R-convex. Then, the set U' of all states in Q that can be reached from U by following edges whose labels, when concatenated, yield α, is still R-convex (possibly U' is empty).*

Proof. We proceed by induction on $|\alpha|$. If $|\alpha| = 0$, then $\alpha = \varepsilon$ and we are done. Now assume $|\alpha| \geq 1$. We can write $\alpha = \alpha' a$, with $\alpha' \in \Sigma^*$, $a \in \Sigma$. Let $u, v, z \in Q$ such that $u, z \in U'$ and $(u, v), (v, z) \in R$. We must prove that $v \in U'$. If $v = u$ or $v = z$ the conclusion follows, so we can assume $v \neq u$ and $v \neq z$. By the inductive hypothesis, the set U'' of all states in Q that can be reached from some state in U by following edges whose labels, when concatenated, yield α', is R-convex. In particular, there exist $u', z' \in U''$ such that $u \in \delta(u', a)$ and $z \in \delta(z', a)$. Since $a \in \lambda(u) \cap \lambda(z)$ and $(u, v), (v, z) \in R$, then $\lambda(v) = \{a\}$ (otherwise by Axiom 1 we would obtain a contradiction), so there exists $v' \in Q$ such that $v \in \delta(v', a)$. From $(u, v), (v, z) \in R$ and Axiom 2 we obtain $(u', v'), (v', z') \in R$; since $u', z' \in U''$ and U'' is R-convex, then $v' \in U''$, which implies $v \in U'$. $\square$

We can already observe that switching from co-lex orders to co-lex relations simplifies the algebraic structure. In general, the union of two co-lex orders is not a co-lex order (see Figure 6.1). However, the union of two co-lex relations is always a co-lex relation:

Lemma 6.5. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $R_1, \ldots, R_m$ be co-lex relations on $\mathcal{N}$. Then, $\bigcup_{i=1}^m R_i$ is a co-lex relation on $\mathcal{N}$.*

Proof. First, $\bigcup_{i=1}^m R_i$ is reflexive because each R_i is reflexive. Let us prove Axiom 1. Assume that $(u, v) \in \bigcup_{i=1}^m R_i$, with $u \neq v$. We must prove that $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$. Notice that it must be $(u, v) \in R_j$ for some j, so the conclusion follows from Axiom 1 applied to the co-lex relation R_j. Let us prove Axiom 2. Assume that $u \in \delta(u', a), v \in \delta(v', a)$ are such that $u \neq v$ and $(u, v) \in \bigcup_{i=1}^m R_i$. We must prove that $(u', v') \in \bigcup_{i=1}^m R_i$. Notice that it must be $(u, v) \in R_j$ for some j, so the conclusion follows from Axiom 2 applied to the co-lex relation R_j. $\square$



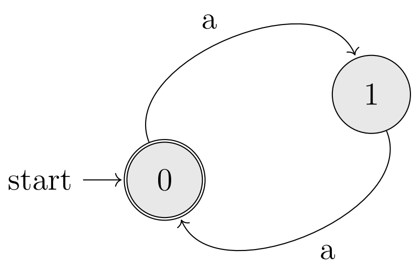


Figure 6.1: *Left*: Notice that $\{(0,0),(1,1),(2,2),(1,2)\}$ and $\{(0,0),(1,1),(2,2),(2,1)\}$ are co-lex orders, but their union is not a co-lex order (antisymmetry would be violated). In particular, the NFA does not admit the maximum co-lex order. *Right*: A NFA that admits the maximum co-lex order, which however is distinct from the maximum co-lex relation. Indeed, the maximum co-lex order is $\{(0,0),(1,1)\}$ and the maximum co-lex relation is $\{(0,0),(1,1),(0,1),(1,0)\}$.

We now prove that every co-lex relation is refined by a co-lex preorder, namely, its transitive closure.

Lemma 6.6. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let R be a co-lex relation on $\mathcal{N}$. Then, Trans(R) is a co-lex preorder on $\mathcal{N}$.*

Proof. First, $\text{Trans}(R)$ is reflexive because R is reflexive, and it is a preorder by definition.

Let us prove Axiom 1. Assume that $(u, v) \in \text{Trans}(R)$, with $u \neq v$. We must prove that $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$. Since $(u, v) \in \text{Trans}(R)$, then there exist $z_1, \dots, z_r \in Q$ ($r \geq 0$) such that $(u, z_1) \in R$, $(z_1, z_2) \in R$, $\dots$, $(z_r, v) \in R$ and $u \neq z_1$, $z_1 \neq z_2$, $\dots$, $z_r \neq v$. Then, Axiom 1 applied to R implies $\max_{\lambda(u)} \preceq \min_{\lambda(z_1)}$, $\max_{\lambda(z_1)} \preceq \min_{\lambda(z_2)}$, $\dots$, $\max_{\lambda(z_r)} \preceq \min_{\lambda(v)}$, so we conclude $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$.

Let us prove Axiom 2. Assume that $u \in \delta(u', a), v \in \delta(v', a)$ are such that $u \neq v$ and $(u, v) \in \text{Trans}(R)$. We must prove that $(u', v') \in \text{Trans}(R)$. Since $(u, v) \in \text{Trans}(R)$, then like before there exist $z_1, \dots, z_r \in Q$ ($r \geq 0$) such that $(u, z_1) \in R$, $(z_1, z_2) \in R$, $\dots$, $(z_r, v) \in R$ and $u \neq z_1$, $z_1 \neq z_2$, $\dots$, $z_r \neq v$, and it must be $\max_{\lambda(u)} \preceq \min_{\lambda(z_1)}$, $\max_{\lambda(z_1)} \preceq \min_{\lambda(z_2)}$, $\dots$, $\max_{\lambda(z_r)} \preceq \min_{\lambda(v)}$. Since $a \in \lambda(u) \cap \lambda(v)$, we conclude $\lambda(z_1) = \dots = \lambda(z_r) = \{a\}$. This implies that there exist $z'_1, \dots, z'_r \in Q$ such that $z_1 \in \delta(z'_1, a)$, $\dots$, $z_r \in \delta(z'_r, a)$. Then, Axiom 2 applied to R implies $(u', z'_1) \in R$, $(z'_1, z'_2) \in R$, $\dots$, $(z'_r, v') \in R$, so we conclude $(u', v') \in \text{Trans}(R)$. $\square$

Definition 6.7. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA. Let R be a co-lex relation on $\mathcal{N}$. We say that R is *maximum* if it refines every co-lex relation R' on $\mathcal{N}$.

It is clear that if a maximum co-lex relation exists, then it is unique. The following lemma shows that the maximum co-lex relation always exists. This is a crucial distinction between co-lex relations and co-lex orders: in general, the maximum co-lex order - that is, a co-lex order refining every co-lex order - does not exist (see Figure 6.1), and this provides some intuition about why determining the minimum width p of a co-lex order on an NFA is NP-hard.

Lemma 6.8. *Every NFA $\mathcal{N} = (Q, s, \delta, F)$ admits the maximum co-lex relation (in the following denoted by $\leq_{\mathcal{N}}$). Moreover, $\leq_{\mathcal{N}}$ is a co-lex preorder.*

Proof. Let $\leq_{\mathcal{N}}$ be the union of all co-lex relations on $\mathcal{N}$. Notice that such a union is nonempty by remark 6.3 and it is finite because the number of binary relations on Q is finite. Moreover, $\leq_{\mathcal{N}}$ is a co-lex relation by Lemma 6.5, and if for some co-lex relation it holds $(u, v) \in R$, then by definition $u \leq_G v$, so $\leq_{\mathcal{N}}$ is the maximum co-lex relation. Finally, $\text{Trans}(\leq_{\mathcal{N}})$ is a co-lex relation by Lemma 6.6, so the maximality of $\leq_{\mathcal{N}}$ implies $\text{Trans}(\leq_{\mathcal{N}}) = \leq_{\mathcal{N}}$, that is, $\leq_{\mathcal{N}}$ is transitive. □

Remark 6.9. Since $\leq_{\mathcal{N}}$ refines every co-lex relation on $\mathcal{N}$, then the width of $\leq_{\mathcal{N}}$ is smaller than or equal to the width of any co-lex relation on $\mathcal{N}$.

Lemma 6.8 implies that the notion of maximum co-lex preorder (a co-lex preorder refining every co-lex preorder) is pointless, because the maximum co-lex preorder always exists and it is always equal to the maximum co-lex relation. If the maximum co-lex relation is also antisymmetric, then it also the maximum co-lex order; however in general the maximum co-lex order does not exist, or if it exists it can be distinct from the maximum co-lex relation (and in this case the maximum co-lex relation is a strict refinement of the maximum co-lex order), see Figure 6.1.

If $\mathcal{D} = (Q, s, \delta, F)$ is a DFA, then the notation $\leq_{\mathcal{D}}$ looks ambiguous because it has two possible meanings: it can refer to the maximum co-lex relation (Lemma 6.8) and the maximum co-lex order (Lemma 4.9). Corollary 6.12 below shows that, in fact, there is no ambiguity.

First, by following the proof of Lemma 4.19 verbatim, we obtain the following result.

Lemma 6.10. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let R be a co-lex relation on $\mathcal{N}$. If $(u, v) \in R$, then $(\forall \alpha \in I_u)(\forall \beta \in I_v)(\{\alpha, \beta\} \not\subseteq I_u \cap I_v \implies \alpha \prec \beta)$.*

We immediately obtain the following corollaries.

Corollary 6.11. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA and let R be a co-lex relation on $\mathcal{N}$. If $(u, v) \in R$ and $(v, u) \in R$, then $I_u = I_v$. In particular, if $\leq$ is a co-lex preorder on $\mathcal{N}$ and $[u]_\leq = [v]_\leq$, then $I_u = I_v$.*

Corollary 6.12. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. Then, the maximum co-lex relation on $\mathcal{D}$ is equal to the maximum co-lex order on $\mathcal{D}$.*

Proof. By the previous discussion, we only have to show that the maximum co-lex relation is antisymmetric. It will suffice to show that every co-lex relation R on $\mathcal{D}$ is antisymmetric. Since $\mathcal{D}$ is a DFA, then for every $u, v \in Q$ it holds $I_u \cap I_v = \emptyset$, so the conclusion follows from Corollary 6.11. $\square$

We now show that the maximum co-lex relation can be computed in $O(|\delta|^2)$ time. To this end, we need the characterization in Lemma 6.15. Since the maximum co-lex relation is transitive, when indexing an NFA we can assume that we use a co-lex preorder.

Definition 6.13. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $(u', v'), (u, v) \in Q \times Q$ be pairs of distinct states. We say that (u', v') *precedes* (u, v) if there exist $u_1, \dots, u_r, v_1, \dots, v_r \in Q$ $(r \geq 1)$ and $a_1, \dots, a_{r-1} \in \Sigma$ such that:

1. $u_1 = u'$ and $v_1 = v'$;
2. $u_r = u$ and $v_r = v$;
3. $u_i \neq v_i$ for $i = 1, \dots, r$;
4. $u_{i+1} \in \delta(u_i, a_i), v_{i+1} \in \delta(v_i, a_i)$ for $i = 1, \dots, r - 1$.

Remark 6.14. Notice that if $u, v \in Q$ are distinct states, then (u, v) trivially precedes (u, v) itself.

Lemma 6.15. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $u, v \in Q$ be distinct states. Then, there exists a co-lex relation containing (u, v) if and only if for all pairs (u', v') preceding (u, v) it holds $max_{\lambda(u')} \preceq min_{\lambda(v')}$. In this case, there exists the* minimum *co-lex relation containing (u, v), that is, a co-lex relation containing (u, v) refined by every co-lex relation containing (u, v).*

Proof. ($\Rightarrow$) Let R be a co-lex relation containing (u, v). Assume that (u', v') precedes (u, v). We must prove that $\max_{\lambda(u')} \preceq \min_{\lambda(v')}$. Let $u_1, \dots, u_r$ and $v_1, \dots, v_r$ be states like in Definition 6.13. From Axiom 2 it follows $(u_{r-1}, v_{r-1}) \in R$, then again by Axiom 2 we obtain $(u_{r-2}, v_{r-2}) \in R$, and so on, until we obtain $(u', v') \in R$. By Axiom 1 we conclude $\max_{\lambda(u')} \preceq \min_{\lambda(v')}$.

($\Leftarrow$) Consider a stack that only contains (u, v) at the beginning. Now, process the element in the stack as follows. Pick (u_1, v_1) in the stack, remove it from the stack and add to the stack all the pairs (u'_1, v'_1) of distinct states that have not previously been in the stack such that for some $a \in \Sigma$ it holds $u_1 \in \delta(u'_1, a)$ and $v_1 \in \delta(v'_1, a)$. Process all the elements in the stack until the stack gets empty (which at some point happens because pairs of states are processed at most once), and let R be the reflexive closure of the relation obtained by considering all pairs of states that at some point have been in the stack. Let us prove that R is a co-lex order (and in particular $(u, v) \in R$). It is immediate to show by induction that all elements that go into the stack precede (u, v), so by our assumption, we have $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$, which proves Axiom 1. Finally, Axiom 2 follows from the rule according to which elements are added to the stack.

Lastly, if there exists a co-lex relation containing (u, v), then there exists the minimum co-lex relation containing (u, v), which is simply the relation R built in ($\Leftarrow$): indeed, all elements added to the stack must be in every co-lex relation containing (u, v) by Axiom 2. □

Corollary 6.16. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $u, v \in Q$ be distinct states. Then:*

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

Proof. ($\Rightarrow$) Since (u, v) is contained in a co-lex relation on G (namely, $\leq_{\mathcal{N}}$), the conclusion follows from Lemma 6.15.

($\Leftarrow$) By Lemma 6.15 (u, v) is contained in a co-lex order on $\mathcal{N}$, and so also in the maximum co-lex order $\leq_{\mathcal{N}}$. $\square$

Theorem 6.17. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA. Then, $\leq_{\mathcal{N}}$ can be computed in $O(|\delta|^2)$ time.*

Proof. Consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{(u, v) \in Q \times Q \mid u \neq v\}$ and $\mathcal{E} = \{((u', v'), (u, v)) \in \mathcal{V} \mid u \in \delta(u', a), v \in \delta(v', a) \text{ for some } a \in \Sigma\}$. First, mark all $(u, v) \in \mathcal{V}$ such that $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$ does not hold true (the property "$\max_{\lambda(u)} \preceq \min_{\lambda(v)}$" can be checked in constant time because one only needs to compare the largest element in $\lambda(u)$ and the smallest element in $\lambda(v)$). Then, mark all nodes in $\mathcal{V}$ reachable by a marked node. Notice that at the end a pair (u, v) is marked if and only if there exists a pair (u', v) preceding (u, v) for which $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$ does not hold true, if and only if it holds $u \not\leq_{\mathcal{N}} v$ (by Corollary 6.16). As a consequence, $\leq_{\mathcal{N}}$ is the reflexive closure of the relation consisting of all non-marked nodes in $\mathcal{V}$. Notice that $\leq_{\mathcal{N}}$ can be computed in $O(|\delta|^2)$ time because $|\mathcal{E}| \leq |\delta|^2$ and states in $\mathcal{V}$ can be marked by means of a graph traversal. $\square$

6.1 Quotienting a Preorder

Our aim is to build a quotient NFA that captures all information required for pattern matching. Broadly speaking, we will construct the quotient NFA starting from a co-lex preorder $\leq$ on $\mathcal{N}$ and considering the *partial* order $(Q/_{\leq}, \leq^{\sim})$, see Section 2.3. In this section, we present some preliminary results that will be useful in the following.

Lemma 6.18. *Let $(V, \leq)$ be a preorder. Then, the width of the partial order $(V/_{\leq}, \leq^{\sim}$ $)$ is equal to the width of $(V, \leq)$.*

Proof. Let m_1 be the width of $(V/_{\leq}, \leq^{\sim})$ and let m_2 be the width of $(V, \leq)$. We must prove that $m_1 = m_2$. On the one hand, if $\{U_i\}_{i=1}^{m_1}$ is a $\leq^{\sim}$-chain decomposition of $V/_{\leq}$, then $\{V_i\}_{i=1}^{m_1}$ is a $\leq$-chain decomposition of V, where V_i is the union of all elements of V being in some $\sim_{\leq}$-class of U_i. This proves that $m_2 \leq m_1$. On the other hand, if $\{V_i\}_{i=1}^{m_2}$ is a $\leq$-chain decomposition of V, then $\{U_i\}_{i=1}^{m_2}$ is a cover of $V/_{\leq}$, where $U_i = \{[v]_{\leq} | v \in V_i\}$, and each U_i is a $\leq^{\sim}$ chain, so by extracting an arbitrary partition from the cover we obtain a $\leq^{\sim}$-chain decomposition of $V/_{\leq}$ of cardinality at most m_2. This proves that $m_1 \leq m_2$. □

Incidentally, Lemma 6.18 is the start point for proving a Dilworth theorem-like for preorders. *Dilworth theorem* [45] states that the width of a partial order is equal to the maximum size of an antichain. The same results holds true for preorders. This results is likely to have been implicitly proved previously, but since we did not find a statement for for preorders in the literature, we provide an explicit proof.

Theorem 6.19 (Dilworth theorem for preorders)**.** *Let $(V, \leq)$ be a preorder. Then, the width of $(V, \leq)$ is equal to the maximum size of a $\leq$-antichain.*

Proof. Consider the partial order $(V/_{\leq}, \leq^{\sim})$. By Dilworth theorem for partial orders [45], the width of $(V/_{\leq}, \leq^{\sim})$ is equal to the the maximum size of a $\leq^{\sim}$-antichain in $V/_{\leq}$. The theorem will follow if we prove that the width of $(V/_{\leq}, \leq^{\sim})$ is equal to the width of $(V, \leq)$, and the maximum size of a $\leq^{\sim}$-antichain in $V/_{\leq}$ is equal to the maximum size of a $\leq$-antichain in V. The first statement is Lemma 6.18, so we only have to prove the second statement.

Let M_1 be the maximum size of an $\leq^{\sim}$-antichain in $V/_{\leq}$ and let M_2 be the maximum size of a $\leq$-antichain in V. We must prove prove that $M_1 = M_2$. On the one hand, if $\{[v_1]_{\leq}, \dots, [v_{M_1}]_{\leq}\}$ is a $\leq^{\sim}$-antichain in $V/_{\leq}$, then $\{v_1, \dots, v_{M_1}\}$ is a $\leq$-antichain in V. This prove that $M_1 \leq M_2$. On the other hand, if $\{v_1, \dots, v_{M_2}\}$ is a $\leq$-antichain in V, then the elements of the antichain are in pairwise distinct $\sim_{\leq}$-classes, so $\{[v_1]_{\leq}, \dots, [v_{M_2}]_{\leq}\}$ is a $\leq^{\sim}$-antichain in $V/_{\leq}$ and, in fact, it has cardinality M_2. This proves that $M_2 \leq M_1$. □

Let us prove a simple result relating convexity and quotients: every convex set is the union of some $\sim_{\leq}$-classes. This result is crucial for showing that without loss of generality we can perform pattern matching on the quotient graph.

Lemma 6.20. *Let $(V, \leq)$ be a preorder, and let $U \subseteq V$ be $\leq$-convex. If $v \in U$, then $[v]_{\sim} \subseteq U$. In other words, every $\leq$-convex set is the union of some $\sim_{\leq}$-classes.*

Proof. Assume that $u \in [v]_{\sim}$. We must prove that $u \in U$. We know that $v \in U$, $v \leq u$ and $u \leq v$, so we conclude $u \in U$ because U is $\leq$-convex. □

More generally, we can prove that there is a natural 1-1 correspondence between $\leq$-convex sets in V and $\leq^{\sim}$-convex sets in $V/_{\leq}$.

Lemma 6.21 (Correspondence theorem - convex sets)**.** *Let $(V, \leq)$ be a preorder. Let $\mathcal{U}$ be the family of all $\leq$-convex sets in V, and let $\mathcal{U}_{\leq}$ be the family of all $\leq^{\sim}$-convex sets in $V/_{\leq}$. Define:*

$$\begin{aligned} \phi : \mathcal{U} &\to \mathcal{U}_{\leq} \\ U &\mapsto \{[v]_{\leq} \mid v \in U\}. \end{aligned}$$

Then, ϕ is a bijective function, with inverse:

$$\begin{aligned} \psi : \mathcal{U}_{\leq} &\to \mathcal{U} \\ U_{\leq} &\mapsto \{v \in V \mid [v]_{\leq} \in U_{\leq}\}. \end{aligned}$$

Proof. First, let us prove that ϕ and ψ are well-defined.

1. Let us prove that if U is $\leq$-convex, then $U_{\leq} = \{[v]_{\leq} \mid v \in U\}$ is $\leq^{\sim}$-convex. Assume that $[u]_{\leq}, [v]_{\leq}, [z]_{\leq} \in V/_{\leq}$ satisfy $[u]_{\leq}, [z]_{\leq} \in U_{\leq}$, $[u]_{\leq} \leq^{\sim} [v]_{\leq}$ and $[v]_{\leq} \leq^{\sim} [z]_{\leq}$. We must prove that $[v]_{\leq} \in U_{\leq}$. From $[u]_{\leq} \leq^{\sim} [v]_{\leq}$ and $[v]_{\leq} \leq^{\sim} [z]_{\leq}$ it follows $u \leq v$ and $v \leq z$. Moreover, from $[u]_{\leq}, [z]_{\leq} \in U/_{\leq}$ and Lemma 6.20 it follows $u, z \in U$. Since U is $\leq$-convex, we conclude $v \in U$, and so $[v]_{\leq} \in U_{\leq}$.

2. Let us prove that if $U_{\leq}$ is $\leq^{\sim}$-convex, then $U = \{v \in V | [v]_{\leq} \in U_{\sim}\}$ is $\leq$-convex. Assume that $u, v, z \in V$ satisfy $u, z \in U$, $u \leq v$ and $v \leq z$. We must prove that $v \in U$. From $u \leq v$ and $v \leq z$ it follows $[u]_{\leq} \leq^{\sim} [v]_{\leq}$ and $[v]_{\leq} \leq^{\sim} [z]_{\leq}$. Moreover, from $u, z \in U$ it follows $[u]_{\leq}, [z]_{\leq} \in U_{\leq}$. Since $U_{\leq}$ is $\leq^{\sim}$-convex, we conclude $[v]_{\leq} \in U_{\leq}$, and so $v \in U$.

Now, we are only left with proving that $\psi \circ \phi = id_{\mathcal{U}}$ and $\phi \circ \psi = id_{\mathcal{U}_{\leq}}$. We have:

$$(\psi \circ \phi)(U) = \psi(\{[v]_{\leq} \mid v \in U\}) = \{v' \in U \mid \ [v']_{\leq} = [v]_{\leq} \text{ for some } v \in U\} = U$$

where $(\subseteq)$ in the last equality follows from Lemma 6.20. Finally:

$$(\phi \circ \psi)(U_{\leq}) = \phi(\{v \in V \mid [v]_{\leq} \in U_{\leq}\}) = \{[v']_{\leq} \mid [v']_{\leq} \in U_{\leq}\} = U_{\leq}.$$

$\square$

6.2 The Quotient NFA

We can now define our quotient NFA.

Definition 6.22. Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $\leq$ be a co-lex preorder on $\mathcal{N}$. Define $\mathcal{N}/_{\leq} = (Q/_{\leq}, s/_{\leq}, \delta/_{\leq}, F/_{\leq})$ by:

1. $Q/_{\leq} = \{[v]_{\leq} \mid v \in Q\}$.

2. $s/_{\leq} = \{[s]_{\leq}\}$.

3. $\delta/_{\leq}([u]_{\leq}, a) = \{[v]_{\leq} \mid v' \in \delta(u', a) \in E$ for some $u' \in [u]_{\leq}$ and $v' \in [v]_{\leq}\}$.

4. $F/_{\leq} = \{[v]_{\leq} \mid v' \in F$ for some $v' \in [v]_{\leq}\}$.

Remark 6.23. (1) Note that $[s]_{\leq} = \{s\}$ by Corollary 6.11 because there is a string belonging only to I_s, namely, the empy string. (2) If $u, v \in Q$ are distinct states such that $[u]_{\leq} = [v]_{\leq}$, then $\lambda(u) = \lambda(v)$ and $|\lambda(u)| = |\lambda(v)| = 1$ by Remark 6.2. (3) If $[u]_{\leq} = [v]_{\leq}$, then $\lambda(u) = \lambda(v)$. Indeed, if u and v are distinct states, the conclusion follows from the first point, otherwise the conclusion is trivial (in this case, if $[v]_{\leq} = \{v\}$, then $\lambda(v)$ may have cardinality larger than one. (4) For every $v \in Q$, it holds $\lambda(v) = \lambda([v]_{\leq})$, where $\lambda(v)$ refers to $\mathcal{N}$ and $\lambda([v]_{\leq})$ refers to $\mathcal{N}/_{\leq}$. Indeed, if $a \in \Sigma \cap \lambda(v)$, then there exists $u \in Q$ such that $v \in \delta(u, a)$, so $[v]_{\leq} \in \delta/_{\leq}([u]_{\leq}, a)$ and $a \in \lambda([v]_{\leq})$; conversely, if $a \in \Sigma \cap \lambda([v]_{\leq})$, then there exist $u', v' \in Q$ such that $v' \in \delta(u', a)$ and $[v']_{\leq} = [v]_{\leq}$, so $a \in \lambda(v')$ and, by the second point, $a \in \lambda(v)$.

Lemma 6.24. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $\leq$ be a co-lex preorder on $\mathcal{N}$.*

1. *For clarity, denote by I_α the set of all states reached by α on $\mathcal{N}$ starting from s, and denote by $I_\alpha^{\leq}$ the set of all states reached by α on $\mathcal{N}/_{\leq}$ starting from $s/_{\leq}$. For every $\alpha \in \Sigma^*$ and for every $v \in Q$, it holds:*

$$v \in I_\alpha \iff [v]_{\leq} \in I_\alpha^{\leq}.$$

2. *For clarity, denote by I_v the set of all strings that reach v on $\mathcal{N}$, and denote by $I_{[v]_{\leq}}^{\leq}$ the set of all strings that reach $[v]_{\leq}$ on $\mathcal{N}/_{\leq}$. For every $v \in Q$, it holds:*

$$I_v = I_{[v]_{\leq}}^{\leq}.$$

3. $\mathcal{L}(\mathcal{N}/_{\leq}) = \mathcal{L}(\mathcal{N})$.

Proof. 1. ($\Rightarrow$) Assume that $v \in I_\alpha$. We must prove that $[v]_\leq \in I^{\leq}_\alpha$. We proceed by induction on $|\alpha|$. If $|\alpha| = 0$, then α is the empty string ε, so it must be $v = s$, and indeed $[s]_\leq \in I^{\leq}_\varepsilon$. Now assume that $|\alpha| \geq 1$. We can write $\alpha = \alpha' a$, with $\alpha' \in \Sigma^*$ and $a \in \Sigma$. Since $v \in I_\alpha$, then there exists $u \in I_{\alpha'}$ such that $v \in \delta(u, a)$. Hence $[v]_\leq \in \delta/_\leq([u]_\leq, a)$, and by the inductive hypothesis $[u]_\leq \in I^{\leq}_{\alpha'}$, so we conclude $[v]_\leq \in I^{\leq}_\alpha$.

($\Leftarrow$) Assume that $[v]_\leq \in I^{\leq}_\alpha$. We must prove that $v \in I_\alpha$. We proceed by induction on $|\alpha|$. If $|\alpha| = 0$, then α is the empty string ε, so it must be $[v]_\leq = \{s\}$, hence $v = s$ and indeed $s \in I_\varepsilon$. Now assume that $|\alpha| \geq 1$. We can write $\alpha = \alpha' a$, with $\alpha' \in \Sigma^*$ and $a \in \Sigma$. Since $[v]_\leq \in I^{\leq}_\alpha$, then there exists $[u]_\leq \in I^{\leq}_{\alpha'}$ such that $[v]_\leq \in \delta/_\leq([u]_\leq, a)$. Hence there exist $u' \in [u]_\leq$ and $v' \in [v]_\leq$ such that $v' \in \delta(u', a)$. Since $[u']_\leq = [u]_\leq \in I^{\leq}_{\alpha'}$, by the inductive hypothesis $u' \in I_{\alpha'}$, so $v' \in I_\alpha$. Since $[v']_\leq = [v]_\leq$, then Corollary 6.11 implies that $I_{v'} = I_v$, hence we conclude $v \in I_\alpha$.

2. By the first point, for every $\alpha \in \Sigma^*$ we have:

$$\alpha \in I_v \iff v \in I_\alpha \iff [v]_\leq \in I^{\leq}_\alpha \iff \alpha \in I^{\leq}_{[v]_\leq}.$$

3. For every $\alpha \in \Sigma^*$, we have:

$$\alpha \in \mathcal{L}(\mathcal{N}/_\leq) \iff (\exists [u]_\leq \in F/_\leq)([u]_\leq \in I^{\leq}_\alpha)$$
$$\iff (\exists u \in F)(u \in I_\alpha) \iff \alpha \in \mathcal{L}(\mathcal{N})$$

where the second equivalence holds true because ($\Leftarrow$) if $u \in F$ is such that $u \in I_\alpha$, then $[u]_\leq \in F/_\leq$ and by the first point $[u]_\leq \in I^{\leq}_\alpha$, and ($\Rightarrow$) if $[u]_\leq \in F/_\leq$ is such that $[u]_\leq \in I^{\leq}_\alpha$, then there exists $u' \in F$ such that $u' \in [u]_\leq$, so $[u']_\leq \in I^{\leq}_\alpha$ and by the first point $u' \in I_\alpha$.

□

Let us prove that $\mathcal{N}/_\leq$ enjoys a number of properties. (1) If a state of $\mathcal{N}/_\leq$ has been obtained by collapsing two or more states of $\mathcal{N}/_\leq$, then that state has at most one incoming edge in $\mathcal{N}/_\leq$ (which is possibly a self-loop). (2) $\leq^\sim$ is a co-lex order on $\mathcal{N}/_\leq$. (3) The graph $\mathcal{N}/_{\leq_\mathcal{N}}$ always admits the maximum co-lex order (recall that in general a graph does not admit the maximum co-lex order). More precisely, the maximum co-lex order is $\leq_{\mathcal{N}}^{\sim}$ (the partial order on $Q/_{\leq_{\mathcal{N}}}$ induced by $\leq_{\mathcal{N}}$), which is also the maximum co-lex relation on $\mathcal{N}/_{\leq_{\mathcal{N}}}$. Notice that $\mathcal{N}/_{\leq_{\mathcal{N}}}$ is well-defined because $\leq_{\mathcal{N}}$ is a co-lex preorder by Lemma 6.8.

We prove the first property in Lemma 6.26. We need a preliminary result.

Lemma 6.25. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $\leq$ be a co-lex preorder on $\mathcal{N}$. Assume that $u, v \in Q$ are (non necessarily distinct) states such that $[u]_\leq = [v]_\leq$ and $|[u]_\leq| = |[v]_\leq| \geq 2$. If $u \in \delta(u', a), v \in \delta(v', a)$, then $[u']_\leq = [v']_\leq$.*

Proof. We distinguish two cases.

1. Assume that $u \neq v$. From $[u]_\leq = [v]_\leq$ we obtain $u < v$ and $v < u$, hence Axiom 2 applied to u', u, v', v', a implies $u' \leq v'$ and $v' \leq u'$, so $[u']_\leq = [v']_\leq$.

2. Assume that $u = v$. Since $|[u]_\leq| \geq 2$, then there exists $z \in Q$ such that $u \neq z$ and $[u]_\leq = [z]_\leq$. From Remark 6.23, we know that $\lambda(u) = \lambda(v) = \lambda(z) = \{a\}$, so there exists $z' \in Q$ such that $z \in \delta(z', a)$. From $[u]_\leq = [z]_\leq$ we obtain $u < z$ and $z < u$, hence Axiom 2 applied to u', u, z', z, a implies $u' \leq z'$ and $z' \leq u'$, and Axiom 2 applied to v', v, z', z, a implies $v' \leq z'$ and $z' \leq v'$. Hence, $[u']_\leq = [z']_\leq$ and $[v']_\leq = [z']_\leq$, and so $[u']_\leq = [v']_\leq$.

□

Lemma 6.26. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $\leq$ be a co-lex preorder on $\mathcal{N}$. If $[v]_\leq \in Q/_\leq$ is such that $|[v]_\leq| \geq 2$, then there exists at most one edge entering $[v]_\leq$ in $\mathcal{N}/_\leq$.*

Proof. Since $|[v]_\leq| \geq 2$, by Remark 6.23 we have $|\lambda([v]_\leq)| = 1$. If $\lambda([v]_\leq) = \{\#\}$, then there is no edge entering $[v]_\leq$ in $\mathcal{N}/_\leq$. Now, assume $\lambda([v]_\leq) = \{a\}$, with $a \in \Sigma$, and let $[v]_\leq \in \delta/_\leq([v']_\leq, a) \cap \delta/_\leq([v'_1]_\leq, a)$. We must prove that $[v']_\leq = [v'_1]_\leq$. Since $[v]_\leq \in \delta/_\leq([v']_\leq, a) \cap \delta/_\leq([v'_1]_\leq, a)$, then there exist $u, u_1, u', u'_1 \in Q$ such that $[v]_\leq = [u]_\leq = [u_1]_\leq$, $[v']_\leq = [u']_\leq$, $[v'_1]_\leq = [u'_1]_\leq$, $u \in \delta(u', a)$ and $u_1 \in \delta(u'_1, a)$. Since $[u]_\leq = [u_1]_\leq$ and $|[u]_\leq| = |[u_1]_\leq| = |[v]_\leq| \geq 2$, then from Lemma 6.25 we obtain $[u']_\leq = [u'_1]_\leq$, so from $[v']_\leq = [u']_\leq$ and $[v'_1]_\leq = [u'_1]_\leq$ we conclude $[v']_\leq = [v'_1]_\leq$. □

Next, we prove that $\leq^\sim$ is a co-lex order on $\mathcal{N}/_\leq$.

Lemma 6.27. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $\leq$ be a co-lex preorder on $\mathcal{N}$. Then, $\leq^\sim$ is a co-lex order on $\mathcal{N}/_\leq$, and the width of $\leq^\sim$ is equal to the width of $\leq$.*

Proof. Let us prove that $\leq^\sim$ is a co-lex order on $\mathcal{N}/_\leq$. We know that $\leq^\sim$ is a partial order, so we only have to prove that it satisfies Axiom 1 and Axiom 2.

Let us prove Axiom 1. Assume that $[u]_\leq, [v]_\leq \in Q/_\leq$ satisfy $[u]_\leq <^\sim [v]_\sim$. We must prove that $\max_{\lambda([u]_\leq)} \preceq \min_{\lambda(v]_\leq)}$. By the definition of $\leq^\sim$ we have $u < v$, so by Axiom 1 applied to $\leq$ we conclude $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$. The conclusion follows, because by Remark 6.23 we have $\lambda([u]_\leq) = \lambda(u)$ and $\lambda([v]_\leq) = \lambda(v)$.

Let us prove Axiom 2. Assume that $[u]_\leq \in \delta/_\leq([u']_\leq, a), [v]_\leq \in \delta/_\leq([v']_\leq, a)$ satisfy $[u]_\leq <^\sim [v]_\leq$. We must prove that $[u']_\leq \leq^\sim [v']_\leq$. Since $[u]_\leq \in \delta/_\leq([u']_\leq, a)$, then there exist $u'_1, u_1 \in Q$ such that $u_1 \in \delta(u'_1, a)$, $[u'_1]_\leq = [u']_\leq$ and $[u_1]_\leq = [u]_\leq$. Analogously, $[v]_\leq \in \delta/_\leq([v']_\leq, a)$ implies that there exist $v'_1, v_1 \in Q$ such that $v_1 \in \delta(v'_1, a)$, $[v'_1]_\leq = [v']_\leq$ and $[v_1]_\leq = [v]_\leq$. From $[u]_\leq <^\sim [v]_\leq$ we obtain $u_1 < v_1$, hence by Axiom 2 applied to $\leq$ we conclude $u'_1 \leq v'_1$, which implies $[u']_\leq \leq^\sim [v']_\leq$.

Lastly, $\leq^\sim$ and $\leq$ have the same width by Lemma 6.18. □

Let us prove that $\leq^\sim_\mathcal{N}$ is the maximum co-lex relation and the maximum co-lex order on $\mathcal{N}/_{\leq_\mathcal{N}}$.

Lemma 6.28 (Correspondence theorem - co-lex relations)**.** *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $\leq$ be a co-lex preorder on $\mathcal{N}$. Let $\mathcal{C}_\leq$ the set of all co-lex relations on $\mathcal{N}/_\leq$, and let $\mathcal{C}$ be set of all co-lex relations R on $\mathcal{N}$ such that, if $(u, v) \in R$, $[u]_\leq = [u']_\leq$ and $[v]_\leq = [v']_\leq$, then $(u', v') \in R$. Define:*

$$\begin{aligned} \rho : \mathcal{C}_\leq &\to \mathcal{C} \\ R^\leq &\mapsto \{(u, v) \in Q \times Q \mid ([u]_\leq, [v]_\leq) \in R^\leq\} \end{aligned}$$

Then, ρ is a bijective function, with inverse:

$$\begin{aligned} \sigma : \mathcal{C} &\to \mathcal{C}_\leq \\ R &\mapsto \{([u]_\leq, [v]_\leq) \in Q/_\leq \times Q/_\leq \mid (u, v) \in R\}. \end{aligned}$$

In particular, $\sigma(\leq)$ is equal to $\leq^\sim$.

Proof. First, let us prove that ρ and σ are well-defined.

1. Let us prove that if $R^{\leq}$ is a co-lex relation on $\mathcal{N}/_{\leq}$, then $R = \{(u, v) \in Q \times Q \mid (u, v) \in R\}$ is a co-lex relation on $\mathcal{N}$ which belongs to $\mathcal{C}$. Clearly, R is reflexive because $R^{\leq}$ is reflexive. Let us prove Axiom 1. Assume that $(u, v) \in R$, with $u \neq v$. We must prove that $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$. It must be $([u]_{\leq}, [v]_{\leq}) \in R$. If $[u]_{\leq} \neq [v]_{\leq}$, then from Axiom 1 applied to $R^{\leq}$ we obtain $\max_{\lambda([u]_{\leq})} \preceq \min_{\lambda([v]_{\leq})}$, and we conclude $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$ because $\lambda(u) = \lambda([u]_{\leq})$ and $\lambda(v) = \lambda([v]_{\leq})$ by Remark 6.23. If $[u]_{\leq} = [v]_{\leq}$, then again from Remark 6.23 we obtain $\lambda(u) = \lambda(v)$ and $|\lambda(u)| = |\lambda(v)| = 1$, so again $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$. Let us prove Axiom 2. Assume that $u \in \delta(u', a), v \in \delta(v', a)$, with $(u, v) \in R$ and $u \neq v$. We must prove that $(u', v') \in R$. We have $[u]_{\leq} \in \delta/_{\leq}([u']_{\leq}, a), [v]_{\leq} \in \delta/_{\leq}([v']_{\leq}, a)$ and $([u]_{\leq}, [v]_{\leq}) \in R^{\leq}$. If $[u]_{\leq} \neq [v]_{\leq}$, then from Axiom 2 applied to $R^{\leq}$ we obtain $([u']_{\leq}, [v']_{\leq}) \in R^{\leq}$, and so $(u', v') \in R$. If $[u]_{\leq} = [v]_{\leq}$, then we have $|[u]_{\leq}| = |[v]_{\leq}| \geq 2$ (because $u \neq v$), so by Lemma 6.26 we have $[u']_{\leq} = [v']_{\leq}$, hence trivially $([u']_{\leq}, [v']_{\leq}) \in R^{\leq}$ and in particular $(u', v') \in R$. Lastly, R belongs to $\mathcal{C}$ because if $(u, v) \in R$, $[u]_{\leq} = [u']_{\leq}$ and $[v]_{\leq} = [v']_{\leq}$, then $([u']_{\leq}, [v']_{\leq}) = ([u]_{\leq}, [v]_{\leq}) \in R^{\leq}$ and so $(u', v') \in R$.

2. Let us prove that if R a co-lex relation in $\mathcal{C}$, then $R^{\leq} = \{([u]_{\leq}, [v]_{\leq}) \in Q/_{\leq} \times Q/_{\leq} \mid (u, v) \in R\}$ is a well-defined co-lex relation on $\mathcal{N}/_{\leq}$. First, $R^{\leq}$ is well-defined (that is, if $[u']_{\leq} = [u]_{\leq}$, $[v']_{\leq} = [v]_{\leq}$ and $(u, v) \in R$, then $(u', v') \in R$) because R belongs to $\mathcal{C}$. Clearly, $R^{\leq}$ is reflexive because R is reflexive. Let us prove Axiom 1. Assume that $([u]_{\leq}, [v]_{\leq}) \in R^{\leq}$, with $[u]_{\leq} \neq [v]_{\leq}$. We must prove that $\max_{\lambda([u]_{\leq})} \preceq \min_{\lambda([v]_{\leq})}$. It must be $(u, v) \in R$, with $u \neq v$, so from Axiom 1 applied to R we obtain $\max_{\lambda(u)} \preceq \min_{\lambda(v)}$, and we conclude $\max_{\lambda([u]_{\leq})} \preceq \min_{\lambda([v]_{\leq})}$ because $\lambda(u) = \lambda([u]_{\leq})$ and $\lambda(v) = \lambda([v]_{\leq})$ by Remark 6.23. Let us prove Axiom 2. Assume that $[u]_{\leq} \in \delta/_{\leq}([u']_{\leq}, a), [v]_{\leq} \in \delta/_{\leq}([v']_{\leq}, a)$, with $([u]_{\leq}, [v]_{\leq}) \in R^{\leq}$ and $[u]_{\leq} \neq [v]_{\leq}$. We must prove that $([u']_{\leq}, [v']_{\leq}) \in R^{\leq}$. There must exist $u_1, v_1, u'_1, v'_1 \in Q$ such that $[u_1]_{\leq} = [u]_{\leq}$, $[v_1]_{\leq} = [v]_{\leq}$, $[u'_1]_{\leq} = [u']_{\leq}$, $[v'_1]_{\leq} = [v']_{\leq}$, $u_1 \in \delta(u'_1, a)$, $v_1 \in \delta(v'_1, a)$. From $([u]_{\leq}, [v]_{\leq}) \in R^{\leq}$, $[u_1]_{\leq} = [u]_{\leq}$ and $[v_1]_{\leq} = [v]_{\leq}$ it follows $(u_1, v_1) \in R$, where $u_1 \neq v_1$ (because $[u]_{\leq} \neq [v]_{\leq}$). From Axiom 2 applied to R we obtain $(u'_1, v'_1) \in R$, hence from $[u'_1]_{\leq} = [u']_{\leq}$ and $[v'_1]_{\leq} = [v']_{\leq}$ we conclude $([u']_{\leq}, [v']_{\leq}) \in R^{\leq}$.

Next, let us prove that $\sigma \circ \rho = id_{\mathcal{C}_{\leq}}$ and $\rho \circ \sigma = id_{\mathcal{C}}$. We have:

$$(\sigma \circ \rho)(R^{\leq}) = \sigma(\{(u,v) \in Q \times Q \mid ([u]_{\leq},[v]_{\leq}) \in R^{\leq}\}) =$$
$$= \{([u]_{\leq},[v]_{\leq}) \in Q/_{\leq} \times Q/_{\leq} \mid ([u]_{\leq},[v]_{\leq}) \in R^{\leq}\} = R^{\leq}$$

and:

$$(\rho \circ \sigma)(R) = \rho(\{([u]_{\leq},[v]_{\leq}) \in Q/_{\leq} \times Q/_{\leq} \mid (u,v) \in R\}) =$$
$$= \{(u,v) \in Q \times Q \mid (u,v) \in R\} = R.$$

Lastly, notice that $\leq$ belongs to $\mathcal{C}$, because if $u \leq v$, $[u]_{\leq} = [u']_{\leq}$ and $[v]_{\leq} = [v']_{\leq}$, then $u' \leq u \leq v \leq v'$. By the definition of σ, we conclude that $\sigma(\leq)$ is equal to $\leq^{\sim}$. □

Corollary 6.29. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA. Then, $\leq_{\mathcal{N}}^{\sim}$ is the maximum co-lex relation and the maximum co-lex order on $\mathcal{N}/_{\leq_{\mathcal{N}}}$.*

Proof. By Lemma 6.27 we know that $\leq_{\mathcal{N}}^{\sim}$ is a co-lex order on $\mathcal{N}/_{\leq_{\mathcal{N}}}$, so we only have to prove that $\leq_{\mathcal{N}}^{\sim}$ is the maximum co-lex relation on $\mathcal{N}/_{\leq_{\mathcal{N}}}$. Let $R^{\leq_{\mathcal{N}}}$ be a co-lex relation on $\mathcal{N}/_{\leq_{\mathcal{N}}}$ and assume that $([u]_{\leq_{\mathcal{N}}}, [v]_{\leq_{\mathcal{N}}}) \in R^{\leq_{\mathcal{N}}}$. We must prove that $[u]_{\leq_{\mathcal{N}}} \leq_{\mathcal{N}}^{\sim} [v]_{\leq_{\mathcal{N}}}$. By Lemma 6.28 we know that $R = \rho(R^{\leq_{\mathcal{N}}})$ is a co-lex relation on $\mathcal{N}$, and $(u,v) \in R$. Since $\leq_{\mathcal{N}}$ is the maximum co-lex relation on $\mathcal{N}$, we obtain $u \leq_{\mathcal{N}} v$, so we conclude $[u]_{\leq_{\mathcal{N}}} \leq_{\mathcal{N}}^{\sim} [v]_{\leq_{\mathcal{N}}}$. □

6.3 Pattern Matching

Corollary 6.29 ensures that $\mathcal{N}/_{\leq_{\mathcal{N}}}$ always admits the maximum co-lex order (which has minimum width), and it can be determined in polynomial time by Theorem 6.17. As a consequence, we have overcome the hardness of determining a co-lex order of minimum width of an *arbitrary* graph if we show that we can answer pattern matching queries on $\mathcal{N}$ *by answering a query on* $\mathcal{N}/_{\leq_{\mathcal{N}}}$. This is indeed the purpose of the following lemma. Intuitively, if we start from a $\leq_{\mathcal{N}}$-convex set U of states in $\mathcal{N}$, we can obtain the $\leq_{\mathcal{N}}$-convex set of states that can be reached through a string α by (1) passing to the quotient, (2) obtaining the $\leq_{\mathcal{N}}^{\sim}$-convex set of states that can reached through α in $\mathcal{N}/_{\leq_{\mathcal{N}}}$, and (3) going back to $\mathcal{N}$.

$$\begin{tikzcd}
\mathcal{U} \arrow[r, shift left, "\phi"] \arrow[d, "\theta_\alpha"] & \mathcal{U}_{\le} \arrow[l, shift left, "\psi"] \arrow[d, "\theta_\alpha^{\le}"] \\
\mathcal{U} \arrow[r, shift left, "\phi"] & \mathcal{U}_{\le} \arrow[l, shift left, "\psi"]
\end{tikzcd}$$

Figure 6.2: Lemma 6.30

Lemma 6.30 (Correspondence theorem - path coherence)**.** *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $\le$ be a co-lex preorder on $\mathcal{N}$. Let $\alpha \in \Sigma^*$. Let $\mathcal{U}$ be the family of all $\le$-convex sets in Q, and let $\mathcal{U}_{\le}$ be the family of all $\le^{\sim}$-convex sets in $Q/_{\le}$. Let:*

$$\theta_\alpha : \mathcal{U} \to \mathcal{U}$$

be the function such that if $U \in \mathcal{U}$, then $\theta_\alpha(U)$ is the set of all states of $\mathcal{N}$ that can be reached from U by following edges whose labels, when concatenated, yield α. Moreover, let:

$$\theta_\alpha^{\le} : \mathcal{U}_{\le} \to \mathcal{U}_{\le}$$

be the function such that if $U_{\le} \in \mathcal{U}_{\le}$, then $\theta_\alpha^{\le}(U_{\le})$ is the set of all states of $\mathcal{N}/_{\le}$ that can be reached from $U_{\le}$ by following edges whose labels, when concatenated, yield α. Let ϕ and ψ be the functions defined in Lemma 6.21. Then (see Figure 6.2):

$$\theta_\alpha \circ \psi = \psi \circ \theta_\alpha^{\le}$$
$$\phi \circ \theta_\alpha = \theta_\alpha^{\le} \circ \phi.$$

Proof. First, the codomains of θ_α and $\theta_\alpha^{\le}$ are correct by Lemma 6.4.

Let us prove the first equation. We proceed by induction on $|\alpha|$. If $|\alpha| = 0$, then $\alpha = \varepsilon$, so $\theta_\alpha = id_{\mathcal{U}}$, $\theta_\alpha^{\le} = id_{\mathcal{U}_{\le}}$ and we conclude $\theta_\alpha \circ \psi = \psi = \psi \circ \theta_\alpha^{\le}$. Now, assume

$|\alpha| \geq 1$. We can write $\alpha = \alpha' a$, with $\alpha' \in \Sigma^*$ and $a \in \Sigma$. By the inductive hypothesis, $\theta_{\alpha'} \circ \psi = \psi \circ \theta^{\leq}_{\alpha'}$. Moreover, notice that $\theta_\alpha = \theta_a \circ \theta_{\alpha'}$ and $\theta^{\leq}_{\alpha} = \theta^{\leq}_{a} \circ \theta^{\leq}_{\alpha'}$. Hence:

$$\theta_\alpha \circ \psi = \theta_a \circ \theta_{\alpha'} \circ \psi = \theta_a \circ \psi \circ \theta^{\leq}_{\alpha'}$$
$$\psi \circ \theta^{\leq}_{\alpha} = \psi \circ \theta^{\leq}_{a} \circ \theta^{\leq}_{\alpha'}$$

so the conclusion follows if we prove that:

$$\theta_a \circ \psi = \psi \circ \theta^{\leq}_{a}.$$

Fix $U_{\leq} \in \mathcal{U}_{\leq}$. We have to prove that $(\theta_a \circ \psi)(U_{\leq}) = (\psi \circ \theta^{\leq}_{a})(U_{\leq})$. We will use that ψ is the inverse of ϕ (Lemma 6.21).

($\subseteq$) If $v \in (\theta_a \circ \psi)(U_{\leq})$, then there exists $u \in \psi(U_{\leq})$ such that $v \in \delta(u, a)$. In particular, $[u]_{\leq} \in U_{\leq}$ and $[v]_{\leq} \in \delta/_{\leq}([u]_{\leq}, a)$, so $[v]_{\leq} \in \theta^{\leq}_{a}(U_{\leq})$ and we conclude $v \in (\psi \circ \theta^{\leq}_{a})(U_{\leq})$.

($\supseteq$) If $v \in (\psi \circ \theta^{\leq}_{a})(U_{\leq})$, then $[v]_{\leq} \in \theta^{\leq}_{a}(U_{\leq})$, so there exists $[u]_{\leq} \in U_{\leq}$ such that $[v]_{\leq} \in \delta/_{\leq}([u]_{\leq}, a)$. In particular, there exist $u', v' \in Q$ such that $v' \in \delta(u', a)$, $[u']_{\leq} = [u]_{\leq} \in U_{\leq}$ and $[v']_{\leq} = [v]_{\leq}$. This means that $v' \in (\theta_a \circ \psi)(U_{\leq})$, and by Lemma 6.20 we conclude $v \in (\theta_a \circ \psi)(U_{\leq})$.

Let us prove the second equation. Since ψ is the inverse of ϕ (Lemma 6.21), then from the first equation we obtain:

$$\phi \circ \theta_\alpha = \phi \circ \theta_\alpha \circ \psi \circ \phi = \phi \circ \psi \circ \theta^{\leq}_{\alpha} \circ \phi = \theta^{\leq}_{\alpha} \circ \phi.$$

□

Corollary 6.31. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA, and let $\leq$ be a co-lex preorder on $\mathcal{N}$. Let $\alpha \in \Sigma^*$.*

1. *Let $T(\alpha)$ and $T^{\leq}(\alpha)$ be the of all states reached by a path on $\mathcal{N}$ and $\mathcal{N}/_{\leq}$, respectively. Then, $T(\alpha) = \bigcup_{[u]_{\leq} \in T^{\leq}(\alpha)} [u]_{\leq}$.*

2. *Let I_α and $I^{\leq}_{\alpha}$ be the of all states reached by a path from the initial state on $\mathcal{N}$ and $\mathcal{N}/_{\leq}$, respectively. Then, $I_\alpha = \bigcup_{[u]_{\leq} \in I^{\leq}_{\alpha}} [u]_{\leq}$.*

Here is our main result.

Theorem 6.32. *Let $\mathcal{N} = (Q, s, \delta, F)$ be an NFA on alphabet Σ of size $\sigma = |\Sigma| \leq e^{O(1)}$, where $e = |\delta|$ is the number of $\mathcal{N}$-transitions and $e^{\leq} = |\delta/_{\leq_{\mathcal{N}}}|$. Assume that we are given a $\leq_{\mathcal{N}}$-chain partition $\{Q_i \mid 1 \leq i \leq q\}$, where q is the width of $\leq_{\mathcal{N}}$. Then, in expected time $O(e^{\leq} \log\log \sigma)$, we can build a data structure using $e^{\leq} \log(q\sigma)(1 + o(1)) + O(e^{\leq})$ bits that, given a query string $\alpha \in \Sigma^m$, answers the following queries in $O(m \cdot q^2 \cdot \log\log(q\sigma))$ time:*

1. *compute the set $T(\alpha)$ of all states reached by a path on $\mathcal{N}$ labeled α, represented by means of q ranges on the chains in $\{Q_i \mid 1 \leq i \leq q\}$;*

2. *compute the set I_α of all states reached by a path labeled with α originating in the source, represented by means of q ranges on the chains in $\{Q_i \mid 1 \leq i \leq q\}$ and, in particular, decide whether $\alpha \in \mathcal{L}(\mathcal{N})$.*

Moreover, $\leq_{\mathcal{N}}$ and $\{Q_i \mid 1 \leq i \leq q\}$ can be built in $O(e^2 + |Q|^{5/2})$ time.

Proof. First, $\leq_{\mathcal{N}}$ can be built in $O(e^2)$ time by Theorem 6.17, and then $\{Q_i \mid 1 \leq i \leq q\}$ can be built in $|Q|^{5/2}$ time by Lemma 4.16. Build the NFA $\mathcal{N}/_{\leq_{\mathcal{N}}}$ by a graph traversal. By Lemma 6.27 we know that $\leq_{\mathcal{N}}^{\sim}$ is a co-lex order on $\mathcal{N}/_{\leq_{\mathcal{N}}}$ of width q. Moreover, $\leq_{\mathcal{N}}^{\sim}$ is the maximum co-lex order on $\mathcal{N}/_{\leq_{\mathcal{N}}}$ by Corollary 6.29, and so it is a co-lex order of minimum width. Hence, just build the data structure from Theorem 4.47 on $\mathcal{N}/_{\leq_{\mathcal{N}}}$. Corollary 6.31 ensures that querying $\mathcal{N}/_{\leq_{\mathcal{N}}}$ is equivalent to querying $\mathcal{N}$. $\square$

Theorem 6.32 improves on Theorem 4.47 in several respects:

1. The data structure in Theorem 6.32 can be built in polynomial time, while the data structure in Theorem 4.47 requires determining a minimum-width co-lex order, which as we saw is a provably hard problem.

2. The parameter q in Theorem 6.32 is always smaller than or equal to the parameter p in Theorem 4.47, because the maximum co-lex relation on $\mathcal{N}$ refines every co-lex order on $\mathcal{N}$. Moreover, q can be arbitrarily smaller than p: for every integer n there exists an NFA for which $q = 1$ and $p = n$ (see Figure 6.3).

3. The bounds in Theorem 6.32 only depend on the graph $e^{\leq}$, which may be smaller than than e. In other words, $\mathcal{N}/_{\leq_{\mathcal{N}}}$ *eliminates the unnecessary redundancy in $\mathcal{N}$ to perform pattern matching.*

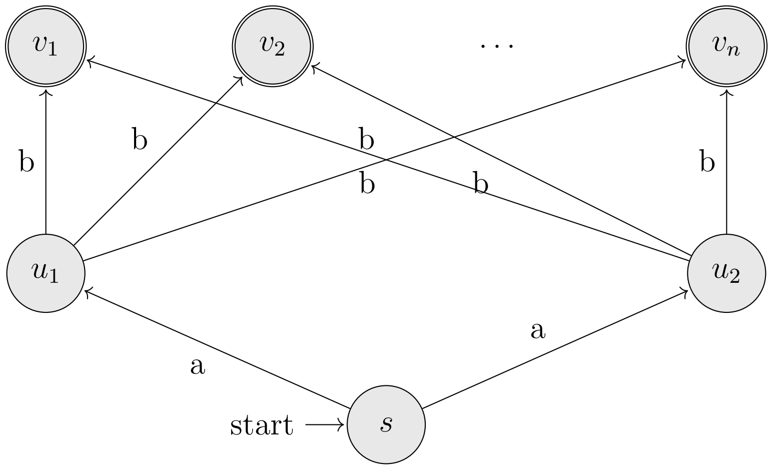


Figure 6.3: An NFA $\mathcal{N} = (Q, s, \delta, F)$ for which $q = 1$ and $p = n$. First, we have $\leq_{\mathcal{N}} = \{(s, z) \mid z \in Q\} \cup \{(v_i, v_j) \mid 1 \leq i, j \leq n\} \cup \{(u_1, v_i) \mid 1 \leq i \leq n\} \cup \{(u_2, v_i) \mid 1 \leq i \leq n\} \cup \{(u_i, u_j) \mid 1 \leq i, j \leq 2\}$, so $q = 1$. Second, let us prove that $p \geq n$. Let $\leq$ be any co-lex order on $\mathcal{N}$. We must prove that the width of $\leq$ is at least n. Notice that for every $1 \leq i < j \leq n$ states v_i and v_j are not $\leq$-comparable, because Axiom 2 would imply both $u_1 < u_2$ and $u_2 < u_1$, which contradicts antisymmetry. Hence $\{v_1, \ldots, v_n\}$ is a $\leq$-antichain and by Dilworth's theorem we conclude that the width of $\leq$ is at least n. In fact, a co-lex order of width n is $\{(s, z) \mid z \in Q\} \cup \{(u_1, v_i) \mid 1 \leq i \leq n\} \cup \{(u_2, v_i) \mid 1 \leq i \leq n\} \cup \{(v_i, v_i) \mid 1 \leq i \leq n\} \cup \{(u_i, u_i) \mid 1 \leq i \leq 2\}$.

Chapter 7

Building the Maximum Co-lex Order on DFAs

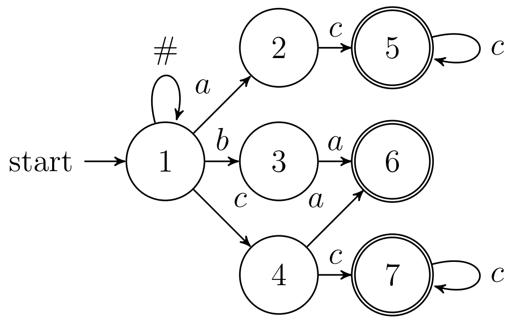
In Chapter 4, we proved that the maximum co-lex order $\leq_{\mathcal{D}}$ on a DFA $\mathcal{D} = (Q, s, \delta, F)$ and a smallest $\leq_{\mathcal{D}}$-chain partition can be determined in $O(|\delta|^2 + |Q|^{5/2})$ time (Lemma 4.15, Lemma 4.16, Corollary 4.17). In this chapter, we will improve the previous bound. The whole chapter is devoted to proving the following theorem.

Theorem 7.1. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. Then, $\leq_{\mathcal{D}}$ and a smallest $\leq_{\mathcal{D}}$-chain partition can be determined in can be determined in $O(|\delta| + |Q|^2)$ time.*

We remark that there exists an alternative algorithm that determine $\leq_{\mathcal{D}}$ and a smallest $\leq_{\mathcal{D}}$-chain partition in $O(|\delta| \log |Q|)$ time [12]. If the graph underlying the DFA is sparse, then the algorithm in [12] improves Theorem 7.1. Since the $O(|\delta| \log |Q|)$ algorithm uses different techniques (it is obtained by adapting Paige and Tarjan's partition refinement algorithm [99]), we are left with the intriguing open problem of determining whether, by possibly combining the ideas behind our algorithm and the algorithm in [12], it is possible to determine $\leq_{\mathcal{D}}$ and a smallest $\leq_{\mathcal{D}}$-chain partition in $O(|\delta|)$ time.

In this chapter and in Chapter 8, we use the *lexicographic order*; moreover, if u is a state, then I_u contains strings in Σ^ω, and $\min_u$ and $\max_u$ are the lexicographically smallest and largest such strings (see Section 2.1 and Section 2.2). For example, in Figure 7.1, we listed the minimum and the maximum string reaching each state of a DFA.

In order to prove Theorem 7.1, we will need the notion of *min/max partition*. Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. Let $Q' \subseteq Q$. Let $\mathcal{A}$ be the unique partition of Q' and let $\leq$ be the unique total order on $\mathcal{A}$ such that, for every $I, J \in \mathcal{A}$ and for every $u \in I$ and $v \in J$, (i) if $I = J$, then $\min_u = \min_v$ and (ii) if $I < J$, then $\min_u \prec \min_v$. Then, we say that $(\mathcal{A}, \leq)$, or more simply $\mathcal{A}$, is the *min-partition* of Q'. The *max-partition* of Q' is defined analogously. Now, consider the set $Q' \times \{\min, \max\}$, and define $\rho((u, \min)) = \min_u$ and $\rho((u, \max)) = \max_u$ for every $u \in Q'$. Let $\mathcal{B}$ be the unique



i	$\min_i$	$\max_i$
1	$\#\#\#\#\#\dots$	$\#\#\#\#\#\dots$
2	$a\#\#\#\#\dots$	$a\#\#\#\#\dots$
3	$b\#\#\#\#\dots$	$b\#\#\#\#\dots$
4	$c\#\#\#\#\dots$	$c\#\#\#\#\dots$
5	$ca\#\#\#\dots$	$ccccc\dots$
6	$ab\#\#\#\dots$	$ac\#\#\#\dots$
7	$cc\#\#\#\dots$	$ccccc\dots$

Figure 7.1: A DFA $\mathcal{D}$, with the minimum and maximum string reaching each state (we assume $\# \prec a \prec b \prec c$). The min/max partition is given by $\{(1,\min),(1,\max)\} < \{(2,\min),(2,\max)\} < \{(6,\min)\} < \{(6,\max)\} < \{(3,\min),(3,\max)\} < \{(4,\min),(4,\max)\} < \{(5,\min)\} < \{(7,\min)\} < \{(5,\max),(7,\max)\}$, meaning that $\min_1 = \max_1 \prec \min_2 = \max_2 \prec \min_6 \prec \max_6 \prec \min_3 = \max_3 \prec \min_4 = \max_4 \prec \min_5 \prec \min_7 \prec \max_5 = \max_7$.

partition of $V' \times \{\min, \max\}$ and let $\leq$ be the unique total order on $\mathcal{B}$ such that, for every $I, J \in \mathcal{B}$ and for every $x \in I$ and $y \in J$, (i) if $I = J$, then $\rho(x) = \rho(y)$ and (ii) if $I < J$, then $\rho(x) \prec \rho(y)$. Then, we say that $(\mathcal{B}, \leq)$, or more simply $\mathcal{B}$, is the *min/max-partition* of Q'. See again Figure 7.1 for an example.

Kim et al. showed that, given a DFA $\mathcal{D} = (Q, s, \delta, F)$, it we have the min/max-partition of Q, then we can build $\leq_{\mathcal{D}}$ and a smallest $\leq_{\mathcal{D}}$-chain partition in $O(|Q|)$ time by means of a reduction to the interval partitioning problem [82]. As a consequence, in order to prove Theorem 7.1, we only have to prove the following theorem.

Theorem 7.2. *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. Then, the min/max-partition of Q can be determined in $O(|\delta| + |Q|^2)$ time.*

In Chapter 4 we saw that the maximum co-lex order can be seen as a generalization of the suffix array from strings to automata. As a consequence, the problem of building the the min/max-partition can be attacked by generalizing some of the algorithms for building a suffix array from strings to graphs. The impact of suffix arrays has led to a big effort in the attempt to design efficient algorithms to construct suffix arrays, where "efficient" refers to various metrics (worst-case running time, average running time, space, performance on real data and so on); see [104] for a comprehensive survey on the topic. Let us focus on worst-case running time. Manber and Myers build the suffix array of a string of length n in $O(n \log n)$ by means of a *prefix-doubling* algorithm [88]. In 1997, Farach proposed a recursive algorithm to build the suffix *tree* of a string in linear time for integer alphabets [51]. In the following years, the recursive paradigm of Farach's algorithm was used to develop a multitude of linear-time algorithms for building the suffix array [83, 80, 81, 94]. All these algorithms carefully exploit the lexicographic structure of the suffixes of a string, recursively reducing the problem of computing the suffix array of a string to the problem of computing the suffix array of a smaller string (*induced sorting*). We will prove Thereom 7.2 by suitably generalizing one of these algorithms, namely, Ko and Aluru's algorithm [83].

Let $\mathcal{D}$ be a DFA. Note that the initial state s and the set F of final states play no role in the definition of min/max partition. Moreover, here we are not interested in the language recognized $\mathcal{D}$, so it will be expedient to focus on the set E rather than the transition function δ (see Section 2.2). As a consequence, in the remainder of the chapter we will denote a DFA $\mathcal{D}$ by $\mathcal{D} = (Q, E)$, where Q is the set of states and E is the set of edges.

In order to build the min/max partition of a DFA, we will first show how to build the min partition and the max/partition; then, we will show how to jointly build the min/max partition. To this end, let us define the min/max partition of *pairs of DFAs*. Assume that we have two DFAs $\mathcal{D}_1 = (Q_1, E_1)$ and $\mathcal{D}_2 = (Q_2, E_2)$ on the same alphabet $(\Sigma, \preceq)$, with $Q_1 \cap Q_2 = \emptyset$ (we allow $\mathcal{D}_1$ and $\mathcal{D}_2$ to possibly be the *null DFA*, that is, the DFA without states). Let $Q_1' \subseteq Q_1$, $Q_2' \subseteq Q_2$, $W = Q_1' \cup Q_2'$, and for every $u \in W$ define $\rho(u) = \min_u$ if $u \in Q_1'$, and $\rho(u) = \max_u$ if $u \in Q_2'$. Let $\mathcal{A}$ be the unique partition of W and let $\leq$ be the unique total order on $\mathcal{A}$ such that, for every $I, J \in \mathcal{A}$ and for every $u \in I$ and $u \in J$, (i) if $I = J$, then $\rho(u) = \rho(u)$ and (ii) if $I < J$, then $\rho(u) \prec \rho(u)$. Then, we say that $(\mathcal{A}, \leq)$, or more simply $\mathcal{A}$, is the *min/max-partition* of (Q_1', Q_2').

Notice that, in order to prove Theorem 7.2, we only have to prove the following theorem.

Theorem 7.3. *Let $\mathcal{D}_1 = (Q_1, E_1)$ and $\mathcal{D}_2 = (Q_2, E_2)$ be two DFAs on the same alphabet $(\Sigma, \preceq)$, with $Q_1 \cap Q_2 = \emptyset$, such that $|\lambda(u)| = 1$ for every $u \in Q_1 \cup Q_2$. Then, we can build the min/max partition of (Q_1, Q_2) in $O((|Q_1| + |Q_2|)^2)$ time.*

Indeed, we can prove Theorem 7.2 as follows. Let $\mathcal{D} = (Q, E)$ be a DFA. First, let $\mathcal{D}_1 = (Q_1, E_1)$ the DFA obtained by picking a copy of $\mathcal{D}$ and, for every $u \in Q_1$ removing all edges entering u not labeled with $\min_{\lambda(u)}$. Similarly, let $\mathcal{D}_2 = (Q_2, E_2)$

the DFA obtained by picking a distinct copy of $\mathcal{D}$ and, for every $u \in Q_2$ removing all edges entering u not labeled with $\max_{\lambda(u)}$. Then, $\mathcal{D}_1$ and $\mathcal{D}_2$ are such that for every $u \in Q_1 \cup Q_2$ it holds $|\lambda(u)| = 1$. By Theorem 7.3, in $O(|Q|^2)$ time we can build the min/max partition of (Q_1, Q_2), which by definition yields the min/max partition of Q with respect to $\mathcal{D}$. Then Theorem 7.2 follows, because we can build $\mathcal{D}_1$ and $\mathcal{D}_2$ in $O(|\delta|)$ time.

We conclude that we are only left with proving Theorem 7.3. In view of the statement of Theorem 7.3, in the remainder of the chapter we will implicitly assume to consider only DFAs $\mathcal{D} = (Q, E)$ in which $|\lambda(u)| = 1$ for every $u \in Q$ (that is, all edges entering the same state have the same label). This implies that every state has at most $|Q|$ incoming edges, and so $|E| \leq |Q|^2$ (in particular, an $O(|E|)$ algorithm is also an $O(|Q|^2)$ algorithm).

7.1 Our Approach

Let $\mathcal{D} = (Q, E)$ be a DFA. We will first show how to build the min-partition of Q in $O(n^2)$ time, where $n = |Q|$ (Section 7.3); then, we will show how the algorithm can be adapted so that Theorem 7.3 follows (Section 7.10).

In order to build a min-partition of Q, we will first classify all minima into three categories (Section 7.2), so that we can split Q into three pairwise-disjoint sets Q_1, Q_2, Q_3. Then, we will show that in $O(n^2)$ time:

- we can compute Q_1, Q_2, Q_3 (Section 7.4);
- we can define a DFA $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ having $|Q_3|$ states (Section 7.5);
- assuming that we have already determined the min-partition of $\bar{Q}$, we can determine the min-partition of Q (Section 7.6).

Analogously, in $O(n^2)$ time we can reduce the problem of determining the min-partition of Q to the problem of determining the min-partition of the set of all states of a DFA having $|Q_1|$ (not $|Q_3|$) states (Sections 7.7, 7.8, 7.9). As a consequence, since $\min\{|Q_1|, |Q_3|\} \leq |Q|/2 = n/2$, we obtain a recursive algorithm whose running time is given by the recurrence:

$$T(n) = T(n/2) + O(n^2)$$

and we conclude that the running time of our algorithm is $O(n^2)$.

7.2 Classifying Strings

In [83], Ko and Aluru divide the suffixes of a string into two groups. Here we follow an approach purely based on stringology, without fixing a string or a graph from the start. We divide the strings of Σ^ω into three groups, which we call group 1, group 2 and group 3 (Corollary 7.6 provides the intuition behind this choice).

Definition 7.4. Let $\alpha \in \Sigma^\omega$. Let $a \in \Sigma$ and $\alpha' \in \Sigma^\omega$ such that $\alpha = a\alpha'$. Then, we define $\tau(\alpha)$ as follows:

1. $\tau(\alpha) = 1$ if $\alpha' \prec \alpha$.

2. $\tau(\alpha) = 2$ if $\alpha' = \alpha$.

3. $\tau(\alpha) = 3$ if $\alpha \prec \alpha'$.

We will constantly use the following characterization.

Lemma 7.5. *Let $\alpha \in \Sigma^\omega$. Let $a \in \Sigma$ and $\alpha' \in \Sigma^\omega$ such that $\alpha = a\alpha'$. Then:*

1. *$\tau(\alpha) = 2$ if and only if $\alpha' = a^\omega$, if and only if $\alpha = a^\omega$.*

2. *$\tau(\alpha) \neq 2$ if and only if $\alpha' \neq a^\omega$, if and only if $\alpha \neq a^\omega$.*

Assume that $\tau(\alpha) \neq 2$. Then, there exist unique $c \in \Sigma \setminus \{a\}$, $\alpha'' \in \Sigma^\omega$ and $i \geq 0$ such that $\alpha' = a^i c\alpha''$ (and so $\alpha = a^{i+1}c\alpha''$). Moreover:

1. *$\tau(\alpha) = 1$ if and only if $c \prec a$, if and only if $\alpha' \prec a^\omega$, if and only if $\alpha \prec a^\omega$.*

2. *$\tau(\alpha) = 3$ if and only if $a \prec c$, if and only if $a^\omega \prec \alpha'$, if and only if $a^\omega \prec \alpha$,*

Proof. 1. Let us prove that $\tau(\alpha) = 2$ if and only if $\alpha' = a^\omega$.

($\Leftarrow$) If $\alpha' = a^\omega$, we have $\alpha = a\alpha' = aa^\omega = a^\omega = \alpha'$, so $\tau(\alpha) = 2$.

($\Rightarrow$) Assume that $\tau(\alpha) = 2$ and so $\alpha = \alpha'$. In order to prove that $\alpha' = a^\omega$, it will suffice to show that for every $i \geq 1$ it holds $\alpha' = a^i\alpha'$. We proceed by induction on i. Notice that $\alpha' = \alpha = a\alpha'$, which proves our claim for $i = 1$. Now, assume that $i > 1$. By the inductive hypothesis, we can assume $\alpha' = a^{i-1}\alpha'$. Hence, $\alpha' = \alpha = a\alpha' = a(a^{i-1}\alpha') = a^i\alpha'$.

Let us prove that $\alpha' = a^\omega$ if and only if $\alpha = a^\omega$. We have $\alpha' = a^\omega$ if and only if $a\alpha' = aa^\omega$, if and only if $\alpha = a^\omega$.

2. It follows by negating the first point.

Now assume that $\tau(\alpha) \neq 2$. Since $\alpha' \neq a^\omega$, then there exist unique $c \in \Sigma \setminus \{a\}$, $\alpha'' \in \Sigma^\omega$ and $i \geq 0$ such that $\alpha' = a^i c\alpha''$. Moreover:

1. We have $\tau(\alpha) = 1$ if and only if $\alpha' \prec \alpha$, if and only if $\alpha' \prec a\alpha'$, if and only if $a^i c\alpha'' \prec a(a^i c\alpha'')$, if and only if $a^i c\alpha'' \prec a^i(ac\alpha'')$, if and only if $c\alpha'' \prec ac\alpha''$, if and only if $c \prec a$, if and only if $a^i c\alpha'' \prec a^\omega$, if and only if $\alpha' \prec a^\omega$, if and only if $a\alpha' \prec aa^\omega$, if and only if $\alpha \prec a^\omega$.

2. It follows by negating the previous points.

□

The following corollary will be a key ingredient in our recursive approach.

Corollary 7.6. *Let $\alpha, \beta \in \Sigma^\omega$. Let $a, b \in \Sigma$ and $\alpha', \beta' \in \Sigma^\omega$ such that $\alpha = a\alpha'$ and $\beta = b\beta'$. Then:*

1. *If $a = b$ and $\tau(\alpha) = \tau(\beta) = 2$, then $\alpha = \beta$.*

2. *If $a = b$ and $\tau(\alpha) < \tau(\beta)$, then $\alpha \prec \beta$. Equivalently, if $a = b$ and $\alpha \preceq \beta$, then $\tau(\alpha) \leq \tau(\beta)$.*

Proof. 1. By Lemma 7.5 we have $\alpha = a^\omega = b^\omega = \beta$.

2. Let us prove that, if $a = b$ and $\tau(\alpha) < \tau(\beta)$, then $\alpha \prec \beta$. We distinguish three cases.

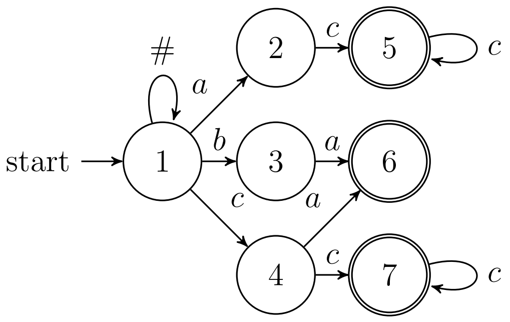
   (a) $\tau(\alpha) = 1$ and $\tau(\beta) = 2$. Then by Lemma 7.5 we have $\alpha \prec a^\omega = b^\omega = \beta$.

   (b) $\tau(\alpha) = 2$ and $\tau(\beta) = 3$. Then by Lemma 7.5 we have $\alpha = a^\omega = b^\omega \prec \beta$.

   (c) $\tau(\alpha) = 1$ and $\tau(\beta) = 3$. Then by Lemma 7.5 we have $\alpha \prec a^\omega = b^\omega \prec \beta$.

   Lastly, if $a = b$ and $\alpha \preceq \beta$, then $\tau(\alpha) \leq \tau(\beta)$, because if it were $\tau(\beta) \leq \tau(\alpha)$, then we would conclude $\beta \prec \alpha$, a contradiction.

□



i	min_i	$\tau(i)$ $(= \tau(\text{min}_i))$
1	$\#\#\#\#\#\dots$	2
2	$a\#\#\#\#\dots$	1
3	$b\#\#\#\#\dots$	1
4	$c\#\#\#\#\dots$	1
5	$ca\#\#\#\dots$	1
6	$ab\#\#\#\dots$	3
7	$cc\#\#\#\dots$	1

Figure 7.2: The DFA from Figure 7.1, with the values min_i's and $\tau(i)$'s.

7.3 Computing the min-partition

Let $\mathcal{D} = (Q, E)$ be a DFA. We will prove that we can compute the min-partition of Q in $O(|Q|^2)$ time. In the following, for every $u \in Q$ we define $\tau(u) = \tau(\text{min}_u)$ (see Figure 7.2).

Let $u \in Q$, and let $(u_i)_{i \geq 1}$ be an occurrence of min_u starting at u. It is immediate to realize that (i) if $\tau(u) = 1$, then $\lambda(u_2) \preceq \lambda(u_1)$, (ii) if $\tau(u) = 2$, then $\lambda(u_k) = \lambda(u_1)$ for every $k \geq 1$ and (iii) if $\tau(u) = 3$, then $\lambda(u_1) \preceq \lambda(u_2)$.

As a first step, let us prove that without loss of generality we can remove some edges from Q without affecting the min/max-partition. This preprocessing will be helpful in Lemma 7.26.

Definition 7.7. Let $\mathcal{D} = (Q, E)$ be a DFA. We say that $\mathcal{N}$ is *trimmed* if contains no edge $(u, v) \in E$ such that $\tau(v) = 1$ and $\lambda(v) \prec \lambda(u)$.

In order to simplify the readability of our proofs, we will not directly remove some edges from $\mathcal{D} = (Q, E)$, but we will first build a copy of $\mathcal{D}$ where every state u is a mapped to a state u^*, and then we will trim the DFA. In this way, when we write min_u and min_{u^*} it will be always clear whether we refer to the original DFA or the trimmed DFA. We will use the same convention in Section 7.5 when we define the DFA $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ that we will use for the recursive step.

Lemma 7.8. *Let $\mathcal{D} = (Q, E)$ be a DFA. Then, in $O(|E|)$ time we can build a trimmed DFA $\mathcal{D}^* = (Q^*, E^*)$, with $Q^* = \{u^* \mid u \in Q\}$, such that for every $u \in Q$ it holds $\text{min}_{u^*} = \text{min}_u$. In particular, $\tau(u^*) = \tau(u)$ for every $u \in Q$.*

Proof. Define $\lambda(u^*) = \lambda(u)$ for every $u^* \in Q^*$, $F = \{(u, v) \in E \mid \tau(v) = 1, \lambda(v) \prec \lambda(u)\}$, and $E^* = \{(u^*, v^*) \mid (u, v) \in E \setminus F\}$. Essentially, we define $\mathcal{D}^*$ by removing from $\mathcal{D}$ all edges that violate the definition of trimmed graph. Let us show that $\mathcal{D}^*$ is still a DFA in which $|\lambda(u^*)| = 1$ for every $u^* \in Q^*$.

1. All edges entering the same state in Q^* have the same label by construction.

2. Every $v^* \in Q^*$ has an incoming edge in $\mathcal{D}^*$. Indeed, if $\tau(v) \neq 1$, then, if $u \in Q$ is any state such that $(u, v) \in E$, then $(u, v) \notin F$ and so $(u^*, v^*) \in E^*$. If $\tau(v) = 1$, then there exists $u \in Q$ such that $(u, v) \in E$ and $\lambda(u) \preceq \lambda(v)$, so $(u, v) \notin F$ and $(u^*, v^*) \in E^*$.

3. $\mathcal{D}^*$ is a DFA because it is essentially a subgraph of a DFA. More precisely, assume that $(u^*, v^*).(u^*, z^*) \in E^*$ are such that $\lambda(v^*) = \lambda(z^*)$. We must prove that $v^* = z^*$, or equivalently, $v = z$. From $(u^*, v^*), (u^*, z^*) \in E^*$ we obtain $(u, v), (u, z) \in E$, so from $\lambda(v) = \lambda(v^*) = \lambda(z^*) = \lambda(z)$ we obtain $v = z$ because $\mathcal{D}$ is a DFA.

Now, let us prove that for every $u \in Q$ it holds $\min_{u^*} = \min_u$. To this end we have to prove that (i) $\min_u \in I_{u^*}$ and (ii) if $\alpha \in I_{u^*}$, then $\min_u \preceq \alpha$.

1. Let us prove that $\min_u \in I_{u^*}$. To this end, it will suffice to prove that, if $(u_i)_{i \geq 1}$ is an occurrence of $\min_u$ starting at u, then $(u_i^*)_{i \geq 1}$ is an occurrence of $\min_u$ starting at u^*. For every $i \geq 1$, we have $\lambda(u_i^*) = \lambda(u_i) = \min_u[i]$. We are only left with showing that $(u_{i+1}^*, u_i^*) \in E^*$ for every $i \geq 1$. We know that $(u_{i+1}, u_i) \in E$, so we only have to prove that $(u_{i+1}, u_i) \notin F$. Assume that $\tau(u_i) = 1$. We must prove that $\lambda(u_{i+1}) \preceq \lambda(u_i)$. Since $(u_j)_{j \geq i}$ is an occurrence of $\min_{u_i}$ starting at u_i, this is equivalent to proving that $\min_{u_i}[2] \preceq \min_{u_i}[1]$, which indeed follows from $\tau(u_i) = 1$.

2. Let us prove that if $\alpha \in I_{u^*}$, then $\alpha \in I_u$ (which implies $\min_u \preceq \alpha$). Let $(v_i^*)_{i \geq 1}$ be an occurrence of α starting at u^*. It will suffice to prove that $(v_i)_{i \geq 1}$ is an occurrence of α starting at u. For every $i \geq 1$ we have $(v_{i+1}, v_i) \in E$ because $(v_{i+1}^*, v_i^*) \in E$, and $\lambda(v_i) = \lambda(v_i) = \alpha[i]$, so the conclusion follows.

In particular, $\tau(u^*) = \tau(u)$ for every $u \in Q$, and $\mathcal{D}^* = (Q^*, E^*)$ is a trimmed DFA. $\square$

7.4 Classifying Minima

Let us first show how to compute all $u \in Q$ such that $\tau(u) = 1$.

Lemma 7.9. *Let $\mathcal{D} = (Q, E)$ be a DFA, and let $u, v \in Q$.*

1. *If $(u, v) \in E$ and $\lambda(u) \prec \lambda(v)$, then $\tau(v) = 1$.*

2. *If $(u, v) \in E$, $\lambda(u) = \lambda(v)$ and $\tau(u) = 1$, then $\tau(v) = 1$.*

Proof. 1. Since $\lambda(u) \prec \lambda(v)$, we have $\min_u \prec \lambda(v)^\omega$. Moreover, $\lambda(v) \min_u \in I_v$, hence $\min_v \preceq \lambda(v) \min_u \prec \lambda(v)\lambda(v)^\omega = \lambda(v)^\omega$, so $\tau(v) = 1$ by Lemma 7.5.

2. Since $\tau(u) = 1$, then $\min_u \prec \lambda(u)^\omega$ by Lemma 7.5. From $(u, v) \in E$ we obtain $\lambda(v) \min_u \in I_v$, hence $\min_v \preceq \lambda(v) \min_u \prec \lambda(v)\lambda(u)^\omega = \lambda(v)\lambda(v)^\omega = \lambda(v)^\omega$, so again by Lemma 7.5 we conclude $\tau(v) = 1$.

□

Corollary 7.10. *Let $\mathcal{D} = (Q, E)$ be a DFA, and let $u \in Q$. Then, $\tau(u) = 1$ if and only if there exist $k \geq 2$ and $z_1, \dots, z_k \in Q$ such that (i) $(z_i, z_{i+1}) \in E$ for every $1 \leq i \leq k-1$, (ii) $z_k = u$, (iii) $\lambda(z_1) \prec \lambda(z_2)$ and (iv) $\lambda(z_2) = \lambda(z_3) = \dots = \lambda(z_k)$.*

Proof. ($\Leftarrow$) Let $k \geq 2$ and $z_1, \dots, z_k \in Q$ be such that (i) $(z_i, z_{i+1}) \in E$ for every $1 \leq i \leq k-1$, (ii) $z_k = u$, (iii) $\lambda(z_1) \prec \lambda(z_2)$ and (iv) $\lambda(z_2) = \lambda(z_3) = \dots = \lambda(z_k)$. We must prove that $\tau(u) = 1$. Let us prove by induction that for every $2 \leq i \leq k$ it holds $\tau(z_i) = 1$ (and in particular $\tau(u) = \tau(z_k) = 1$). If $i = 2$, then from $(z_1, z_2) \in E$ and $\lambda(z_1) \prec \lambda(z_2)$ we obtain $\tau(z_2) = 1$ by Lemma 7.9. Now, assume that $3 \leq i \leq k$. By the inductive hypothesis, we have $\tau(z_{i-1}) = 1$. Since $(z_{i-1}, z_i) \in E$ and $\lambda(z_{i-1}) = \lambda(z_i)$, then by Lemma 7.9 we conclude $\tau(z_i) = 1$.

($\Rightarrow$) Assume that $\tau(u) = 1$. Then, by Lemma 7.5 there exist $c \in \Sigma \setminus \{\lambda(u)\}$, $\gamma' \in \Sigma^\omega$ and $j \geq 1$ such that $\min_u = \lambda(u)^j c \gamma'$ and $c \prec \lambda(u)$. Let $(u_i)_{i \geq 1}$ be an occurrence of $\min_u$ starting at u. Then, (i) $u = u_1$, (ii) $(u_{i+1}, u_i) \in E$ for every $i \geq 1$, (iii) $\lambda(u_i) = \lambda(u)$ for $1 \leq i \leq j$ and (iv) $\lambda(u_{j+1}) = c$. As a consequence, if $k = j + 1$ and $z_i = u_{k+1-i}$ for every $1 \leq i \leq k$, then (i) $(z_i, z_{i+1}) = (u_{k-i+1}, u_{k-i}) \in E$ for every $1 \leq i \leq k-1$, (ii) $z_k = u_1 = u$, (iii) $\lambda(z_1) = \lambda(u_k) = \lambda(u_{j+1}) = c \prec \lambda(u) = \lambda(u_j) = \lambda(u_{k-1}) = \lambda(z_2)$ and (iv) $\lambda(z_2) = \lambda(z_3) = \dots = \lambda(z_k)$ because $\lambda(u_j) = \lambda(u_{j-1}) = \dots = \lambda(u_1)$. □

Corollary 7.10 yields an algorithm to decide whether $u \in Q$ is such that $\tau(u) = 1$.

Corollary 7.11. *Let $\mathcal{D} = (Q, E)$ be a DFA. We can determine all $u \in Q$ such that $\tau(u) = 1$ in time $O(|E|)$.*

Proof. In our algorithm, we will mark exactly all states u such that $\tau(u) = 1$ by means of a traversal. We will use an (initially empty) queue. Initially, no state is marked. First, process all edges $(u, v) \in E$ and, if $\lambda(u) \prec \lambda(v)$ and v has not been marked before, then mark v and add v to the queue. This step takes $O(|E|)$ time. Next, recursively pick an element u from the queue, and consider the unique $v \in Q$ such that $(u, v) \in E$ and $\lambda(u) = \lambda(v)$ (if it exists). If v has not been marked before, then mark v and add v to the queue. This step also takes $O(|E|)$ time because each edge is considered at most once. At the end of the algorithm, by Corollary 7.10 a state u has been marked if and only if $\tau(u) = 1$. □

Now, let us show how to determine all $u \in Q$ such that $\tau(u) = 2$. We can assume that we have already determined all $u \in Q$ such that $\tau(u) = 1$.

Lemma 7.12. *Let $\mathcal{D} = (Q, E)$ be a DFA, and let $u \in Q$ such that $\tau(u) \neq 1$. Then, we have $\tau(u) = 2$ if and only if there exist $k \geq 2$ and $z_1, \dots, z_k \in Q$ such that (i) $(z_{i+1}, z_i) \in E$ for every $1 \leq i \leq k-1$, (ii) $z_1 = u$, (iii) $z_k = z_j$ for some $1 \leq j \leq k-1$ and (iv) $\lambda(z_1) = \lambda(z_2) = \dots = \lambda(z_k)$. In particular, such $z_1, \dots, z_k \in Q$ must satisfy $\tau(z_i) = 2$ for every $1 \leq i \leq k$.*

Proof. ($\Leftarrow$) Let $k \geq 2$ and $z_1, \dots, z_k \in Q$ such that (i) $(z_{i+1}, z_i) \in E$ for every $1 \leq i \leq k-1$, (ii) $z_1 = u$, (iii) $z_k = z_i$ for some $1 \leq j \leq k-1$ and (iv) $\lambda(z_1) = \lambda(z_2) = \dots = \lambda(z_k)$. We must prove that $\tau(u) = 2$. Notice that it holds $\min_u = \min_{z_1} \preceq \lambda(z_1)\lambda(z_2)\dots\lambda(z_{j-1}) \min_{z_j} = \lambda(u)^{j-1} \min_{z_j}$. Similarly, we have $\min_{z_j} \preceq \lambda(z_j)\lambda(z_{j+1})\dots\lambda(z_{k-1}) \min_{z_k} = \lambda(u)^{k-j} \min_{z_j}$. Since $\min_{z_j} \preceq \lambda(u)^{k-i} \min_{z_j}$, then by induction we obtain $\min_{z_i} \preceq (\lambda(u)^{k-j})^h \min_{z_i}$ for every $h \geq 1$, and so $\min_{z_j} \preceq \lambda(u)^\omega$. As a consequence, $\min_u \preceq \lambda(u)^{j-1} \min_{z_j} \preceq \lambda(u)^{j-1}\lambda(u)^\omega = \lambda(u)^\omega$, and so $\tau(u) \preceq 2$ by Lemma 7.5. Since $\tau(u) \neq 1$, we conclude $\tau(u) = 2$.

($\Rightarrow$) Assume that $\tau(u) = 2$. In particular, $\lambda(u)^\omega \in I_u$, so there exists an occurrence $(z_i)_{i \geq 1}$ of $\lambda(u)^\omega$ starting at u. This means that (i) $(z_{i+1}, z_i) \in E$ for every $i \geq 1$, (ii) $z_1 = u$ and $\lambda(z_i) = \lambda(u)$ for every $i \geq 1$. Since Q is finite, there exist $1 \leq j < k$ such that $z_j = z_k$. Then, $k \geq 2$ and $z_1, \dots, z_k \in Q$ have the desired properties.

Now, let us prove that, if there exist $z_1, \ldots, z_k \in Q$ with the desired properties, then $\tau(z_i) = 2$ for every $1 \leq i \leq k$. Notice that it must be $\tau(z_i) \neq 1$ for every $1 \leq i \leq k$, because otherwise by Corollary 7.10 we would conclude $\tau(u) = 1$. As a consequence, it must be $\tau(z_i) = 2$ for every $1 \leq i \leq k$ by the characterization that we have just proved. □

Corollary 7.13. *Let $\mathcal{D} = (Q, E)$ be a DFA. We can determine all $u \in Q$ such that $\tau(u) = 2$ in time $O(|E|)$.*

Proof. By Corollary 7.11, we can assume that we have already computed all $u \in Q$ such that $\tau(u) = 1$. In order to compute all $u \in Q$ such that $\tau(u) = 2$, we explore $\mathcal{D}$ by using a breadth-first search algorithm, with the following modifications: (i) we only start from states $u \in Q$ such that $\tau(u) \neq 1$, (ii) we follow edges in a backward fashion, not in a forward fashion and (iii) if we start from $u \in Q$, we explore the graph by only following edges labeled $\lambda(u)$ (that is, we first consider only all $u' \in Q$ such that $(u', u) \in E$ and $\lambda(u') = \lambda(u)$, then we repeat this step in BFS fashion). Note that during the search we can never encounter a state v such that $\tau(v) = 1$, otherwise by Corollary 7.10 we would obtain $\tau(u) = 1$. During the search, we will infer the value $\tau(v)$ for every state v that we encounter. Assume that we start the exploration from state $u \in Q$. If during the exploration at u at some point we encounter a state v for which we have already concluded that $\tau(v) = 3$, then we do not consider the edges entering v and we backtrack (because all the z_i's in the characterization of Lemma 7.12 must satisfy $\tau(z_i) = 2$). If during the exploration at u at some point we encounter a state v for which we have already concluded that $\tau(v) = 2$, then we backtrack straight to u and, by Lemma 7.12, we can also conclude that all z that we encounter during the backtracking (including u) are such that $\tau(z) = 2$. If during the exploration of u at some point we encounter the same state twice, then we backtrack straight to u and, by Lemma 7.12, we can also conclude that all z that we encounter during the backtracking (including u) are such that $\tau(z) = 2$. If during the exploration of u we never encounter a state v for which we have already determined $\tau(v)$ and we never encounter the same state twice (and so at some point we get stuck), by Lemma 7.12 we can conclude that all v that we have encountered (including u) are such that $\tau(v) = 3$. The algorithm runs in $O(|E|)$ time because we never follow the same edge twice. □

From Corollary 7.11 and Corollary 7.13 we immediately obtain the following result.

Corollary 7.14. *Let $\mathcal{D} = (Q, E)$ be a DFA. Then, in time $O(|E|)$ we can compute $\tau(u)$ for every $u \in Q$.*

7.5 Recursive Step

Let us sketch the general idea to build a smaller DFA for the recursive step. We consider each $u \in Q$ such that $\tau(u) = 3$, and we follow edges in a backward fashion, aiming to determine a prefix of $\min_u$. As a consequence, we discard edges through which no occurrence of $\min_u$ can go, and by Corollary 7.6 we can restrict our attention to the states v such that $\tau(v)$ is minimal. We proceed like this until we encounter states v' such that $\tau(v') = 3$.

Let us formalize our intuition. We will first present some properties that the occurrences of a string $\min_u$ must satisfy.

Lemma 7.15. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u, v \in Q$ be such that $\min_u = \min_v$. Let $(u_i)_{i \geq 1}$ be an occurrence of $\min_u$ and let $(v_i)_{i \geq 1}$ be an occurrence of $\min_v$. Then:*

1. *$\lambda(u_i) = \lambda(v_i)$ for every $i \geq 1$.*

2. *$\min_{u_i} = \min_{v_i}$ for every $i \geq 1$.*

3. *$\tau(u_i) = \tau(v_i)$ for every $i \geq 1$.*

In particular, the previous results hold if $u = v$ and $(u_i)_{i \geq 1}$ and $(v_i)_{i \geq 1}$ are two distinct occurrences of $\min_u$.

Proof. 1. For every $i \geq 1$ we have $\lambda(u_i) = \min_u[i] = \min_v[i] = \lambda(v_i)$.

2. Fix $i \geq 1$. We have $\min_u[1, i-1] \min_{u_i} = \min_u = \min_v = \min_v[1, i-1] \min_{v_i}$ and so $\min_{u_i} = \min_{v_i}$

3. Fix $i \geq 1$. By the previous points we have $\min_{u_i} = \min_{v_i}$ and $\lambda(u_i) = \lambda(v_i)$, so by Corollary 7.6 we conclude that it must necessarily be $\tau(u_i) = \tau(v_i)$.

□

Lemma 7.16. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u \in Q$ and let $(u_i)_{i\geq 1}$ an occurrence of $\min_u$ starting at u. Let $k \geq 1$ be such that $\tau(u_1) = \tau(u_2) = \cdots = \tau(u_{k-1}) = \tau(u_k) \neq 2$. Then, $u_1, \dots, u_k$ are pairwise distinct. In particular, $k \leq |Q|$.*

Proof. We assume that $\tau(u_1) = \tau(u_2) = \cdots = \tau(u_{k-1}) = \tau(u_k) = 1$, because the case $\tau(u_1) = \tau(u_2) = \cdots = \tau(u_{k-1}) = \tau(u_k) = 3$ is symmetrical. Notice that for every $1 \leq l \leq k$ we have that $(u_i)_{i\geq l}$ is an occurrence of $\min_{u_l}$ starting at u_l and $\tau(u_l) = 1$, so $\lambda(u_{l+1}) \preceq \lambda(u_l)$. In particular, for every $1 \leq r \leq s \leq k+1$ we have $\lambda(u_s) \preceq \lambda(u_r)$.

Suppose for sake of contradiction that there exist $1 \leq i < j \leq k$ such that $u_i = u_j$. Then, $\min_{u_i} = \min_{u_i}[1, j-i] \min_{u_j} = \min_{u_i}[1, j-i] \min_{u_i}$. By induction, we obtain $\min_{u_i} = (\min_{u_i}[1, j-i])^t \min_{u_i}$ for every $t \geq 1$, and so $\min_{u_i} = (\min_{u_i}[1, j-i])^\omega$. For every $1 \leq h \leq j$, we have $\lambda(u_j) \preceq \lambda(u_h) \preceq \lambda(u_i)$. Since $u_i = u_j$, then $\lambda(u_i) = \lambda(u_j)$, so $\lambda(u_h) = \lambda(u_i)$ for every $i \leq h \leq j$, which implies $\min_{u_i}[1, j-i] = \lambda(u_i)\lambda(u_{i+1}) \dots \lambda(u_{j-1}) = \lambda(u_i)^{j-i}$ and so $\min_{u_i} = (\min_{u_i}[1, j-i])^\omega = \lambda(u_i)^\omega$. By Lemma 7.5 we conclude $\tau(u_i) = 2$, a contradiction. □

The previous results allow us to give the following definition.

Definition 7.17. Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u \in Q$ such that $\tau(u) = 3$. Let ℓ_u to be the smallest integer $k \geq 2$ such that $\tau(u_k) \geq 2$, where $(u_i)_{i\geq 1}$ is an occurrence of $\min_u$ starting at u.

Note that ℓ_u is well-defined, because (i) it cannot hold $\tau(u_k) = 1$ for every $k \geq 2$ by Lemma 7.16 (indeed, if $\tau(u_2) = 1$, then $(u_i)_{i\geq 2}$ is an occurrence of $\min_{u_2}$ starting at u_2, and by Lemma 7.16 there exists $2 \leq k \leq |Q| + 2$ such that $\tau(u_k) \neq 1$) and (ii) ℓ_u does not depend on the choice of $(u_i)_{i\geq 1}$ by Lemma 7.15. In particular, it must be $\ell_u \leq |Q| + 1$ because $u_1, u_2, \dots, u_{\ell_u - 1}$ are pairwise distinct (u_1 is distinct from $u_2, \dots, u_{\ell_u-1}$ because $\tau(u_1) = 3$ and $\tau(u_2) = \tau(u_3) = \dots \tau(u_{\ell_u-1}) = 1$ by the minimality of ℓ_u).

Lemma 7.18. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u \in Q$ such that $\tau(u) = 3$. Then, $\min_u[i+1] \preceq \min_u[i]$ for every $2 \leq i \leq \ell_u - 1$. In particular, if $2 \leq i \leq j \leq \ell_u$, then $\min_u[j] \preceq \min_u[i]$.*

Proof. Fix $2 \leq i \leq \ell_u$. By the definition of ℓ_u) we have $\tau(u_i) = 1$, so $\min_u[i+1] = \min_{u_i}[2] \preceq \min_{u_i}[1] = \min_u[i]$. □

If $R \subseteq Q$ is a nonempty set of states such that for every $u, v \in R$ it holds $\lambda(u) = \lambda(v)$, we define $\lambda(R) = \lambda(u) = \lambda(v)$. If $R \subseteq Q$ is a nonempty set of states such that for every $u, v \in R$ it holds $\tau(u) = \tau(v)$, we define $\tau(R) = \tau(u) = \tau(v)$.

Let $R \subseteq Q$ be a nonempty set of states. Let $\mathtt{F}(R) = \arg\min_{u \in R'} \tau(u)$, where $R' = \arg\min_{v \in R} \lambda(v)$. Notice that $\mathtt{F}(R)$ is nonempty, and both $\lambda(\mathtt{F}(R))$ and $\tau(\mathtt{F}(R))$ are well-defined. In other words, $\mathtt{F}(R)$ is obtained by first considering the subset $R' \subseteq \mathtt{F}(R)$ of all states v such that $\lambda(v)$ is as small as possible, and then considering the subset of R' of all states v such that $\tau(v)$ is as small as possible. This is consistent with our intuition on how we should be looking for a prefix of $\min_u$.

Define:

$$G_i(u) = \begin{cases} \{u\} & \text{if } i = 1; \\ \mathtt{F}(\{v' \in Q \mid (\exists v \in G_{i-1}(u))((v', v) \in E)\}) \setminus \bigcup_{j=2}^{i-1} G_j(u) & \text{if } 1 < i \leq \ell_u. \end{cases}$$

Notice that we also require that a states in $G_i(u)$ has not been encountered before. Intuitively, this does not affect our search for a prefix of $\min_u$ because, if we met the same state twice, then we would have a cycle where all edges are equally labeled (because by Lemma 7.18 labels can only decrease), and since $\tau(G_i(u)) = 1$ for every $2 \leq i \leq \ell_u - 1$, then no occurrence of the minimum can go through the cycle because if we remove the cycle from the occurrence we obtain a smaller string by Lemma 7.5.

The following technical lemma is crucial to prove that our intuition is correct.

Lemma 7.19. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u \in Q$ such that $\tau(u) = 3$.*

1. *$G_i(u)$ is well-defined and nonempty for every $1 \leq i \leq \ell_u$.*

2. *Let $(u_i)_{i \geq 1}$ be an occurrence of $\min_u$ starting at u. Then, $u_i \in G_i(u)$ for every $1 \leq i \leq \ell_u$. In particular, $\tau(u_i) = \tau(G_i(u))$ and $\min_u[i] = \lambda(u_i) = \lambda(G_i(u))$ for every $1 \leq i \leq \ell_u$.*

3. *For every $1 \leq i \leq \ell_u$ and for every $v \in G_i(u)$ there exists an occurrence of $\min_u[1, i-1]$ starting at u and ending at v.*

Proof. We proceed by induction on $i \geq 1$. If $i = 1$, then $G_i(u) = \{u\}$ is well-defined and nonempty, and $u_1 = u \in G_i(u)$. Moreover, if $v \in G_i(u)$, then $v = u$, and there exists an occurrence of $\varepsilon = \min[1, 0]$ starting and ending at u. Now, assume that $2 \leq i \leq \ell_u$.

First, notice that $\mathtt{F}(\{v' \in Q \mid (\exists v \in G_{i-1}(u))((v', v) \in E)\})$ is well-defined and nonempty. Indeed, by the inductive hypothesis $G_{i-1}(u)$ is well-defined and nonempty, so $\{v' \in Q \mid (\exists v \in G_{i-1}(u))((v', v) \in E)\}$ is nonempty because every state has an incoming edge.

Let us prove that $u_i \in \mathtt{F}(\{v' \in Q \mid (\exists v \in G_{i-1}(u))((v', v) \in E)\})$. Let $R = \{v' \in Q \mid (\exists v \in G_{i-1}(u))((v', v) \in E)\}$ and $R' = \arg\min_{v \in R} \lambda(v)$. We must prove that $u_i \in \arg\min_{u \in R'} \tau(u)$.

1. Let us prove that $u_i \in R$. By the inductive hypothesis, we know that $u_{i-1} \in G_{i-1}(u)$. We know that $(u_i, u_{i-1}) \in E$, so $u_i \in R$.

2. Let us prove that $u_i \in R'$. Let $v' \in R$. We must prove that $\lambda(u_i) \preceq \lambda(v')$. Since $v' \in R$, there exists $v \in G_{i-1}(u)$ such that $(v', v) \in E$. From $u_{i-1}, v \in G_{i-1}(u)$ we obtain $\lambda(u_{i-1}) = \lambda(v)$. By the inductive hypothesis, there exists an occurrence of $\min_u[1, i-2]$ starting at u and ending at v, so there exists an occurrence of $\min_u[1, i-2]\lambda(v) = \min_u[1, i-2]\lambda(u_{i-1}) = \min_u[1, i-1]$ starting at u and ending at v'. Since $\min_u = \min_u[1, i-1] \min_{u_i}$, it must be $\min_{u_i} \preceq \min_{v'}$, and in particular $\lambda(u_i) \preceq \lambda(v')$.

3. Let us prove that $u_i \in \arg\min_{u \in R'} \tau(u)$. Let $v' \in R'$. We must prove that $\tau(u_i) \leq \tau(v)$. In particular, $v' \in R$, so as before we obtain $\min_{u_i} \preceq \min_{v'}$. Moreover, from $u_i, v' \in R'$ we obtain $\lambda(u_i) = \lambda(v')$. From Corollary 7.6 we conclude $\tau(u_i) \leq \tau(v)$.

Let us prove that $u_i \notin \bigcup_{j=2}^{i-1} G_j(u)$. If $i = \ell_u$, we have $\tau(u_i) = \tau(u_{\ell_u}) \geq 2$ and the conclusion follows because $\tau(G_j(u)) = \tau(u_j) = 1$ for every $2 \leq j \leq i-1$. As a consequence, in the following we can assume $2 \leq i \leq \ell_u - 1$. In particular, $\tau(u_i) = 1$, so by Lemma 7.5 there exists $k \geq 1$, $c \in \Sigma \setminus \{\lambda(u_i)\}$ and $\gamma' \in \Sigma^\omega$ such that $\min_u = \lambda(u)^k c\gamma'$ and $c \prec \lambda(u_i)$. Suppose for sake of contradiction that there exists $2 \leq j \leq i-1 \leq \ell_u - 2$ such that $u_i \in G_j(u)$. We know that $u_j \in G_j(u)$, and in particular, $\lambda(u_i) = \lambda(G_j(u)) = \lambda(u_j)$. From Lemma 7.18 we obtain $\min_u[i] \preceq \min_u[h] \preceq \min_u[j]$ for every $j \leq h \leq i$, or equivalently $\lambda(u_i) \preceq \lambda(u_h) \preceq \lambda(u_j)$ for every $j \leq h \leq i$. Since $\lambda(u_j) = \lambda(u_i)$, we conclude $\lambda(u_h) = \lambda(u_i)$ for every $j \leq h \leq i$. As a consequence, $\min_u = \min_u[1, i-1] \min_{u_i} = \min_u[1, j-1] \min_u[j, i-1] \min_{u_i} = \min_u[1, j-1]\lambda(u_i)^{i-j}\lambda(u_i)^k c\gamma' = \min_u[1, j-1]\lambda(u_i)^{k+i-j} c\gamma'$. On the other hand, since $u_i \in G_j(u)$ and $j < i$, by the inductive hypothesis there exists an occurrence of $\min_u[1, j-1]$ starting at u and ending at u_i, so $\min_u[1, j-1] \min_{u_i} \in I_u$. As a consequence, by the minimality of $\min_u$ we obtain $\min_u \preceq \min_u[1, j-1] \min_{u_i}$, or equivalently, $\min_u[1, j-1]\lambda(u_i)^{k+i-j}c\gamma' \preceq \min_u[1, j-1]\lambda(u_i)^k c\gamma'$. Since $j < i$, we obtain $\lambda(u_i) \preceq c$, a contradiction.

Let us prove that $G_i(u)$ is well-defined and nonempty, and $u_i \in G_i(u)$. This follows from $u_i \in \mathtt{F}(\{v' \in Q \mid (\exists v \in G_{i-1}(u))((v', v) \in E)\})$ and $u_i \notin \bigcup_{j=2}^{i-1} G_j(u)$.

Lastly, let us prove that if $v' \in G_i(u)$, then there exists an occurrence of $\min[1, i-1]$ starting at u and ending at v'. Since $v' \in G_i(u)$, then there exists $v \in G_{i-1}(u)$ such that $(v', v) \in E$. In particular, $\lambda(v) = \lambda(G_{i-1}(u)) = \lambda(u_{i-1})$. By the inductive hypothesis, there exists an occurrence of $\min_u[1, i-2]$ starting at u and ending at v, so there exists an occurrence of $\min_u[1, i-2]\lambda(v) = \min_u[1, i-2]\lambda(u_{i-1}) = \min_u[1, i-1]$ starting at u and ending at v'. $\square$

Let $u \in Q$ such that $\tau(u) = 3$. We define:

- $\gamma_u = \min_u[1, \ell_u]$;
- $\mathtt{t}_u = \tau(G_{\ell_u}(u)) \in \{2, 3\}$

Now, in order to define the smaller graph for the recursive step, we also need a new alphabet $(\Sigma', \preceq')$, which must be defined consistently with the mutual ordering of the minima. The next lemma yields all the information that we need.

Lemma 7.20. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u, v \in Q$ such that $\tau(u) = \tau(v) = 3$. Assume that one of the following statements is true:*

1. *γ_u is not a prefix of γ_v and $\gamma_u \prec \gamma_v$.*
2. *$\gamma_u = \gamma_v$, $\mathtt{t}_u = 2$ and $\mathtt{t}_v = 3$.*
3. *γ_v is a strict prefix of γ_u.*

Then, $\min_u \prec \min_v$.

Equivalently, if $\min_u \preceq \min_v$, *then one the following is true: (i) γ_u is not a prefix of γ_v and $\gamma_u \prec \gamma_v$; (ii) $\gamma_u = \gamma_v$ and $\mathtt{t}_u \leq \mathtt{t}_v$; (iii) γ_v is a strict prefix of γ_u.*

Proof. Let us prove that, if one of the three statements (1) - (3) is true, then $\min_u \prec \min_v$.

1. If $\min_u[1, \ell_u] \prec \min_v[1, \ell_v]$ and $\min_u[1, \ell_u]$ is not a prefix of $\min_v[1, \ell_v]$, then $\min_u \prec \min_v$.

2. Assume that $\min_u[1, \ell_u] = \min_v[1, \ell_v]$ (so in particular $\ell_u = \ell_v$), $\mathtt{t}_u = 2$ and $\mathtt{t}_v = 3$. Let $(u_i)_{i \geq 1}$ be an occurrence of u, and let $(v_i)_{i \geq 1}$ be an occurrence of v. It holds $\min_u = \min_u[1, \ell_u - 1] \min_{u_{\ell_u}}$ and $\min_v = \min_v[1, \ell_v - 1] \min_{v_{\ell_v}}$. Since $\min_u[1, \ell_u - 1] = \min_v[1, \ell_v - 1]$, in order to prove that $\min_u \prec \min_v$ we only have to show that $\min_{u_{\ell_u}} \prec \min_{v_{\ell_v}}$. By Lemma 7.19 we have $u_{\ell_u} \in G_{\ell_u}(u)$ and $v_{\ell_v} \in G_{\ell_v}(v)$, so $\lambda(u_{\ell_u}) = \min_u[\ell_u] = \min_v[\ell_v] = \lambda(v_{\ell_v})$. Since $\tau(u_{\ell_u}) = \tau(G_{\ell_u}(u)) = \mathtt{t}_u = 2$ and $\tau(v_{\ell_v}) = \tau(G_{\ell_v}(v)) = \mathtt{t}_v = 3$, we conclude $\min_{u_{\ell_u}} \prec \min_{v_{\ell_v}}$ by Corollary 7.6.

3. Assume that γ_v is a strict prefix of γ_u (hence $\ell_v < \ell_u$). In particular, $\min_u[1, \ell_v] = \min_v[1, \ell_v]$, so similarly to the previous case we only have to prove that $\min_{u_{\ell_v}} \prec \min_{v_{\ell_v}}$. Again, we obtain $u_{\ell_v} \in G_{\ell_v}(u)$, $v_{\ell_v} \in G_{\ell_u}(v)$, $\lambda(u_{\ell_v}) = \lambda(v_{\ell_v})$ and $\tau(v_{\ell_v}) = 3$. Since $\ell_v < \ell_u$, the minimality of ℓ_u implies $\tau(u_{\ell_v}) = \tau(G_{\ell_v}(u)) = 1$. By Corollary 7.6 we conclude $\min_{u_{\ell_v}} \prec \min_{v_{\ell_v}}$.

Now, let us prove that if $\min_u \preceq \min_v$, then one the following is true: (i) γ_u is not a prefix of γ_v and $\gamma_u \prec \gamma_v$; (ii) $\gamma_u = \gamma_v$ and $\mathtt{t}_u \leq \mathtt{t}_v$; (iii) γ_v is a strict prefix of γ_u. If this were not true, then one of the following would be true: (1) γ_v is not a prefix of γ_u and $\gamma_v \prec \gamma_u$; (2) $\gamma_v = \gamma_u$, $\mathtt{t}_v = 2$ and $\mathtt{t}_u = 3$; (3) γ_u is a strict prefix of γ_v. We would then conclude $\min_v \prec \min_u$, a contradiction. □

Now, let $\Sigma' = \{(\gamma_u, \mathtt{t}_u) \mid u \in Q, \tau(u) = 3\}$, and let $\preceq'$ be the total order on Σ' such that for every distinct $(\alpha, x), (\beta, y) \in \Sigma'$, it holds $(\alpha, x) \prec' (\beta, y)$ if and only if one of the following is true:

1. α is not a prefix of β and $\alpha \prec \beta$.

2. $\alpha = \beta$, $x = 2$ and $y = 3$.

3. β is a strict prefix of α.

It is immediate to verify that $\preceq'$ is a total order: indeed, $\preceq'$ is obtained (i) by first comparing the γ_u's using the variant of the (total) lexicographic order on Σ^* in which a string is smaller than every strict prefix of it and (ii) if the γ_u's are equal by comparing the $\mathtt{t}_u$'s, which are elements in $\{2, 3\}$.

Starting from $\mathcal{D} = (Q, E)$, we define a new DFA $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ as follows:

- $\bar{Q} = \{\bar{u} \mid u \in Q, \tau(u) = 3\}$.
- The new totally-ordered alphabet is $(\Sigma', \preceq')$.
- For every $\bar{u} \in \bar{Q}$, we define $\lambda(\bar{u}) = (\gamma_u, \mathtt{t}_u)$.
- $\bar{E} = \{(\bar{v}, \bar{v}) \mid \mathtt{t}_v = 2\} \cup \{(\bar{u}, \bar{v}) \mid \mathtt{t}_v = 3, u \in G_{\ell_v}(v)\}$.

Note that for every $\bar{v} \in \bar{Q}$ such that $\mathtt{t}_v = 3$ and for every $u \in G_{\ell_v}(v)$ it holds $\tau(u) = \tau(G_{\ell_v}(v)) = \mathtt{t}_v = 3$, so $\bar{u} \in \bar{V}$ and $(\bar{u}, \bar{v}) \in E$. Moreover, in $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$, (i) all edges entering the same state have the same label (by definition), (ii) every state has at least one incoming edge (because if $\bar{v} \in \bar{Q}$, then $G_{\ell_v}(v) \neq \emptyset$ by Lemma 7.19) and (iii) $\bar{\mathcal{D}}$ is a DFA (because if $(\bar{u}, \bar{v}), (\bar{u}, \bar{v}') \in \bar{E}$ and $\lambda(\bar{v}) = \lambda(\bar{v}')$, then $\gamma_v = \gamma_{v'}$ and $\mathtt{t}_v = \mathtt{t}_{v'}$, so by the definition of $\bar{E}$ if $\mathtt{t}_v = \mathtt{t}_{v'} = 2$ we immediately obtain $\bar{v} = \bar{u} = \bar{v}'$, and if $\mathtt{t}_v = \mathtt{t}'_v = 3$ we obtain $u \in G_{\ell_v}(v) \cap G_{\ell_{v'}}(v')$; since by Lemma 7.19 there exist two occurrences of $\min_v[1, \ell_v - 1] = \gamma_v[1, \ell_v - 1] = \gamma_{v'}[1, \ell_{v'} - 1] = \min_{v'}[1, \ell_{v'} - 1]$ starting at v and v' and both ending at u, the determinism of $\mathcal{D}$ implies $v = v'$ and so $\bar{v} = \bar{v}'$).

Notice that if $\bar{v} \in \bar{Q}$ is such that $\mathtt{t}_v = 2$, then $I_{\bar{v}}$ contains exactly one string, namely, $\lambda(\bar{v})^\omega$; in particular, $\min_{\bar{v}} = \max_{\bar{v}} = \lambda(\bar{v})^\omega$.

When we implement $\mathcal{D} = (Q, E)$ and $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$, we use integer alphabets $\Sigma = \{0, 1, \dots, |\Sigma| - 1\}$ and $\Sigma' = \{0, 1, \dots, |\Sigma'| - 1\}$; in particular, we will not store Σ' by means of pairs $(\gamma_u, \mathtt{t}_u)$'s, but we will remap Σ' to an integer alphabet consistently with the total order $\preceq'$ on Σ', so that the mutual order of the $\min_{\bar{u}}$'s is not affected.

Let us prove that we can use $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ for the recursive step. We will start with some preliminary results.

Lemma 7.21. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u, v \in Q$ be such that $\tau(u) = \tau(v) = 3$, $\gamma_u = \gamma_v$ and $\mathtt{t}_u = \mathtt{t}_v = 2$. Then,* $\min_u = \min_v$.

Proof. First, note that $\gamma_u = \gamma_v$ implies $\ell_u = \ell_v$, and it is equivalent to $\min_u[1, \ell_u] = \min_v[1, \ell_v]$. Now, let $(u_i)_{i\geq 1}$ be an occurrence of $\min_u$ starting at u, and let $(v_i)_{i\geq 1}$ be an occurrence of $\min_v$ starting at v. We have $\min_u = \min_u[1, \ell_u - 1] \min_{u_{\ell_u}}$ and $\min_v = \min_v[1, \ell_v - 1] \min_{v_{\ell_v}}$. Since $\min_u[1, \ell_u - 1] = \min_v[1, \ell_v - 1]$, we only have to show that $\min_{u_{\ell_u}} = \min_{v_{\ell_v}}$. Notice that $\lambda(u_{\ell_u}) = \min_u[\ell_u] = \min_v[\ell_v] = \lambda(v_{\ell_v})$, and by Lemma 7.19 we have $\tau(u_{\ell_u}) = \tau(G_{\ell_u}(u)) = \mathtt{t}_u = 2$ and $\tau(v_{\ell_v}) = \tau(G_{\ell_v}(v)) = \mathtt{t}_v = 2$. By Corollary 7.6 we conclude $\min_{u_{\ell_u}} = \min_{v_{\ell_v}}$. □

Lemma 7.22. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u \in Q$, and let $(u_i)_{i\geq 1}$ be an occurrence of $\min_u$ starting at u. Then, exactly one of the following holds true:*

1. *There exists $i_0 \geq 1$ such that $\tau(u_i) \neq 2$ for every $1 \leq i < i_0$ and $\tau(u_i) = 2$ for every $i \geq i_0$.*
2. *$\tau(u_i) \neq 2$ for every $i \geq 1$, and both $\tau(u_i) = 1$ and $\tau(u_i) = 3$ are true for infinitely many i's.*

Proof. If $\tau(u_i) \neq 2$ for every $i \geq 1$, then both $\tau(u_i) = 1$ and $\tau(u_i) = 3$ are true for infinitely many i's, because otherwise we could choose $j \geq 1$ such that either $\tau(u_i) = 1$ for every $i \geq j$ or $\tau(u_i) = 3$ for every $i \geq j$, and in both cases Lemma 7.16 leads to a contradiction, because $(u_j)_{j\geq i}$ is an occurrence of $\min_{u_j}$ starting at u_j and Q is a finite set.

Now, assume that there exists $i_0 \geq 1$ such that $\tau(u_{i_0}) = 2$; without loss of generality, we can assume that i_0 is the smallest integer with this property. We only have to prove that if $i \geq i_0$, then $\tau(u_i) = 2$. Since $\tau(u_{i_0}) = 2$, then $\min_{u_{i_0}} = (\lambda(u_{i_0}))^\omega$ by Lemma 7.5. Since $(u_j)_{j\geq i_0}$ in an occurrence of $\min_{u_{i_0}}$ starting at i_0, then $\lambda(u_j) = \lambda(u_{i_0})$ for every $j \geq i_0$, so $\min_{u_i} = (\lambda(u_i))^\omega$ for every $i \geq i_0$, $(u_j)_{j\geq i}$ being an occurrence of $\min_{u_i}$ starting at u_i. By Lemma 7.5, we conclude $\tau(u_i) = 2$ for every $i \geq i_0$. □

Crucially, the next lemma establishes a correspondence between minima of states in $\mathcal{D} = (Q, E)$ and minima of states in $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$.

Lemma 7.23. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u \in Q$ such that $\tau(u) = 3$. Let $(u_i)_{i\geq 1}$ be an occurrence of $\min_u$ starting at u. Let $(u'_i)_{i\geq 1}$ be the infinite sequence of states in Q obtained as follows. Consider $L = \{k \geq 1 \mid \tau(u_k) = 3\}$, and for every $i \geq 1$,* *let $j_i \geq 1$ be the i^{th} smallest element of L, if it exists. For every $i \geq 1$ such that j_i is defined, let $u'_i = u_{j_i}$, and if $i \geq 1$ is such that j_i is not defined (so L is a finite set), let $u'_i = u'_{|L|}$. Then, $(\bar{u}'_i)_{i \geq 1}$ is an occurrence of* $\min_{\bar{u}}$ *starting at $\bar{u}$ in $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$.*

Proof. First, notice that $\bar{u}'_i \in \bar{Q}$ for every $i \geq 1$ because $\tau(u'_i) = 3$, and $u'_1 = u_1 = u$ (so $\bar{u}'_1 = \bar{u}_1 = \bar{u}$). Now, let us prove that $(\bar{u}'_{i+1}, \bar{u}'_i) \in \bar{E}$ for every $i \geq 1$. Fix $i \geq 1$. We distinguish three cases.

1. j_i and j_{i+1} are both defined. This means that $\tau(u'_i) = \tau(u'_{i+1}) = 3$, and $\tau(u_k) \neq 3$ for every $j_i < k < j_{i+1}$. By Lemma 7.22, it must be $\tau(u_k) = 1$ for every $j_i < k < j_{i+1}$. Since $(u_k)_{k \geq j_i}$ is an occurrence of $\min_{u'_i}$ starting at u'_i, by Lemma 7.19 we obtain $\mathtt{t}_{u'_i} = 3$ and $u'_{i+1} \in G_{\ell_{u'_i}}(u'_i)$, so $(\bar{u}'_{i+1}, \bar{u}'_i) \in \bar{E}$.

2. j_i is defined and j_{i+1} is not defined. This means that $i = |L|$, $\tau(u_{j_{|L|}}) = 3$, and by Lemma 7.22 there exists $h > j_{|L|}$ such that $\tau(u_h) = 2$ and $\tau(u_k) = 1$ for every $j_{|L|} < k < h$. Since $(u_k)_{k \geq j_{|L|}}$ is an occurrence of $\min_{u'_{|L|}}$ starting at $u'_{|L|}$, by Lemma 7.19 we obtain $\mathtt{t}_{u'_{|L|}} = 2$, so $(\bar{u}'_{i+1}, \bar{u}'_i) = (\bar{u}'_{|L|}, \bar{u}'_{|L|}) \in \bar{E}$.

3. j_i and j_{i+1} are both non-defined. Then, by the previous case we conclude $(\bar{u}'_{i+1}, \bar{u}'_i) = (\bar{u}'_{|L|}, \bar{u}'_{|L|}) \in \bar{E}$.

We are left with proving that $\min_{\bar{u}}[i] = \lambda(\bar{u}'_i)$ for every $i \geq 1$. Let $\alpha \in (\Sigma')^\omega$ be such that $\alpha[i] = \lambda(\bar{u}'_i)$ for every $i \geq 1$. We have to prove that $\alpha = \min_{\bar{u}}$. Since $(\bar{u}'_{i+1}, \bar{u}'_i) \in \bar{E}$ for every $i \geq 1$, we have $\alpha \in I_{\bar{u}}$, and $(\bar{u}'_i)_{i \geq 1}$ is an occurrence of α starting at $\bar{u}$. The conclusion follows if we prove that for every $\beta \in I_{\bar{u}}$ we have $\alpha \preceq' \beta$. Fix $\beta \in I_{\bar{u}}$; it will suffice to show that, for every $k \geq 1$, if $\alpha[1, k-1] = \beta[1, k-1]$, then $\alpha[k] \preceq' \beta[k]$. Fix $k \geq 1$, and let $(\bar{v}_i)_{i \geq 1}$ be an occurrence of β starting at $\bar{u}$. Notice that for every $1 \leq h \leq k-1$ we have $\alpha[h] = \beta[h]$, or equivalently, $\lambda(\bar{u}'_h) = \lambda(\bar{v}_h)$, or equivalently, $\gamma_{u'_h} = \gamma_{v_h}$ (so $\ell_{u'_h} = \ell_{v_h}$) and $\mathtt{t}_{u'_h} = \mathtt{t}_{v_h}$. Note that $(\bar{v}_{i+1}, \bar{v}_i) \in \bar{E}$ for every $i \geq 1$. We distinguish two cases:

1. There exists $1 \leq h \leq k-1$ such that $\mathtt{t}_{u'_h} = \mathtt{t}_{v_h} = 2$. In this case, the definition of u'_h implies $u'_h = u'_{h+1} = u'_{h+2} = \ldots$, and in particular $u'_h = u'_k$. Moreover, since $\mathtt{t}_{v_h} = 2$ and $(\bar{v}_{i+1}, \bar{v}_i) \in \bar{E}$, the definition of $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ implies $v_h = v_{h+1} = v_{h+2} = \ldots$, and in particular $v_h = v_k$. We conclude $\alpha[k] = \lambda(u'_k) = \lambda(u'_h) = \lambda(v_h) = \lambda(v_k) = \beta[k]$.

2. For every $1 \leq h \leq k-1$ it holds $\mathtt{t}_{u'_h} = \mathtt{t}_{v_h} = 3$. In this case, for every $1 \leq h \leq k-1$ we have $\mathtt{t}_{v_h} = 3$ and $(\bar{v}_{h+1}, \bar{v}_h) \in \bar{E}$, so $v_{h+1} \in G_{\ell_{v_h}}(v_h)$ and by Lemma 7.19 there exists an occurrence of $\min_{v_h}[1, \ell_{v_h} - 1] = \gamma_{v_h}[1, \ell_{v_h} - 1] = \gamma_{u'_h}[1, \ell_{u'_h} - 1] = \min_{u'_h}[1, \ell_{u'_h} - 1]$ starting at v_h and ending at v_{h+1}. As a consequence, there is an occurrence of $\min_{u'_1}[1, \ell_{u'_1} - 1] \min_{u'_2}[1, \ell_{u'_2} - 1] \dots \min_{u'_{k-1}}[1, \ell_{u'_{k-1}} - 1] = \min_u[1, j_k - 1]$ starting at u and ending at v_k and so, if $\beta' = \min_u[1, j_k - 1] \min_{v_k}$, then $\beta' \in I_u$. This implies $\min_u \preceq \beta'$, so from $\min_u = \min_u[1, j_k - 1] \min_{u'_k}$ we obtain $\min_{u'_k} \preceq \min_{v_k}$. By Lemma 7.20 and $\min_{u'_k} \preceq \min_{v_k}$, we obtain that one of the following statements must be true:

   (a) $\gamma_{u'_k}$ is not a prefix of γ_{v_k}, γ_{v_k} is not a prefix of $\gamma_{u'_k}$ and $\gamma_{u'_k} \prec \gamma_{v_k}$.

   (b) $\gamma_{u'_k} = \gamma_{v_k}$ and $\mathtt{t}_{u'_k} \leq \mathtt{t}_{v_k}$.

   (c) γ_{v_k} is a strict prefix of $\gamma_{u'_k}$.

   In all three cases, we conclude $(\gamma_{u'_k}, \mathtt{t}_{u'_k}) \preceq' (\gamma_{v_k}, \mathtt{t}_{v_k})$, or equivalently, $\lambda(u'_k) \preceq' \lambda(v_k)$, or equivalently, $\alpha[k] \preceq' \beta[k]$.

□

The following theorem shows that our reduction to $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ is correct.

Theorem 7.24. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u, v \in Q$ be such that $\tau(u) = \tau(v) = 3$.*

1. *If $\min_u = \min_v$, then $\min_{\bar{u}} = \min_{\bar{v}}$.*

2. *If $\min_u \prec \min_v$, then $\min_{\bar{u}} \prec' \min_{\bar{v}}$.*

Proof. Let $(u_i)_{i\geq 1}$ be and occurrence of $\min_u$ starting at u, and let $(v_i)_{i\geq 1}$ be and occurrence of $\min_v$ starting at v . Let $(\bar{u}'_i)_{i\geq 1}$ be the occurrence of $\min_{\bar{u}}$ defined by means of $(u_i)_{i\geq 1}$ in Lemma 7.23, and let $(\bar{v}'_i)_{i\geq 1}$ be the occurrence of $\min_{\bar{v}}$ defined by means of $(v_i)_{i\geq 1}$ in Lemma 7.23. Let $L = \{i \geq 1 \mid \tau(u_i) = 3\}$, and let $j_i \geq 1$ be the i^{th} smallest element of L, if it exists. Moreover, let $M = \{i \geq 1 \mid \tau(v_i) = 3\}$, and let $k_i \geq 1$ be the i^{th} smallest element of K, if it exists. Notice that $j_1 = k_1 = 1$.

In the rest of the proof, we say that an integer $h \geq 1$ is *nice* if it satisfies the following properties:

- $j_1, \dots, j_h$ and $k_1, \dots, k_h$ are all defined, and $j_i = k_i$ for every $1 \leq i \leq h$.

- $\gamma_{u'_i} = \gamma_{v'_i}$ for every $1 \le i \le h-1$.

Note the following properties:

- 1 is always nice because $j_1 = k_1 = 1$.

- If h is nice, than every $1 \le h' \le h$ is nice. This implies that either every $h \ge 1$ is nice, or there exists a unique $h^* \ge 1$ such that every $h \le h^*$ is nice and every $h > h^*$ is not nice.

- If h is nice, $\gamma_{u'_h} = \gamma_{v'_h}$ and $\mathtt{t}_{u'_h} = \mathtt{t}_{v'_h} = 3$, then $h+1$ is nice. Indeed, $\gamma_{u'_h} = \gamma_{v'_h}$ implies $\ell_{u'_h} = \ell_{v'_h}$. Moreover, since $(u_i)_{i \ge j_h}$ is an occurrence of $\min_{u'_h}$ starting at u'_h, then by the minimality of $\ell_{u'_h}$ and Lemma 7.19 we have $\tau(u_{j_h+i}) = \tau(G_{i+1}(u'_h)) = 1$ for every $1 \le i \le \ell_{u'_h} - 2$, and $\tau(u_{j_h+\ell_{u'_h}-1}) = \tau(G_{\ell_{u'_h}}(u'_h)) = \mathtt{t}_{u'_h} = 3$, which implies that j_{h+1} is defined. Analogously, one obtains $\tau(u_{k_h+i}) = 1$ for every $1 \le i \le \ell_{v'_h} - 2$ and $\tau(v_{k_h+\ell_{v'_h}-1}) = 3$, so k_{h+1} is defined and $j_{h+1} = k_{h+1}$, which implies that $h+1$ is nice.

- If h is nice, then $\mathtt{t}_{u'_i} = \mathtt{t}_{v'_i} = 3$ for every $1 \le i \le h-1$. Indeed, assume for sake of contradiction that $\mathtt{t}_{u'_l} = 2$ for some $1 \le l \le h-1$ (the case $\mathtt{t}_{v'_l} = 2$ is analogous). Since $(u_i)_{i \ge j_l}$ is an occurrence of $\min_{u'_l}$ starting at u'_l, then by Lemma 7.19 we have $\tau(u_{j_l+i}) = \tau(G_{i+1}(u'_l)) = 1$ for every $1 \le i \le \ell_{u'_l} - 2$, and $\tau(u_{j_l+\ell_{u'_l}-1}) = \tau(G_{\ell_{u'_l}}(u'_l)) = \mathtt{t}_{u'_l} = 2$, which by Lemma 7.22 implies $\tau(u_i) = 2$ for every $i \ge j_l + \ell_{u'_l} - 1$. This implies that j_{l+1} is not defined, which is a contradiction because $1 \le l \le h-1$.

Let us prove that, if $h \ge 1$ is nice, then:

- $\min_u[1, j_h - 1] = \min_v[1, k_h - 1]$.

- $\min_{\bar{u}}[1, h-1] = \min_{\bar{v}}[1, h-1]$.

We will prove these two properties separately.

- Let us prove that $\min_u[1, j_h - 1] = \min_v[1, j_h - 1]$. Since h is nice, then for every $1 \le i \le h$ we have that j_i and k_i are defined, and $j_i = k_i$. As a consequence, it will suffice to prove that $\min_u[j_i, j_{i+1} - 1] = \min_v[k_i, k_{i+1} - 1]$ for every $1 \le i \le h-1$. This is equivalent to proving that $\min_{u'_i}[1, \ell_{u'_i} - 1] = \min_{v'_i}[1, \ell_{v'_i} - 1]$ for every $1 \le i \le h-1$. This follows from $\gamma_{u'_i} = \gamma_{v'_i}$ for every $1 \le i \le h-1$.

- Let us prove that $\min_{\bar{u}}[1, h-1] = \min_{\bar{v}}[1, h-1]$. Fix $1 \leq i \leq h-1$. We must prove that $\min_{\bar{u}}[i] = \min_{\bar{v}}[i]$, or equivalently, $\lambda(\bar{u}'_i) = \lambda(\bar{v}'_i)$. This follows from $\gamma(u'_i) = \gamma(v'_i)$ and $\mathtt{t}_{u'_i} = \mathtt{t}_{v'_i}$.

We can now prove the two claims of the theorem. We will use that following observation: if every $h \geq 1$ is nice, then $\min_u = \min_v$ and $\min_{\bar{u}} = \min_{\bar{v}}$, because $\min_u[1, j_h - 1] = \min_v[1, k_h - 1]$ and $\min_{\bar{u}}[1, h-1] = \min_{\bar{v}}[1, h-1]$ for every $h \geq 1$.

1. Assume that $\min_u = \min_v$; we must prove that $\min_{\bar{u}} = \min_{\bar{v}}$. If every $h \geq 1$ is nice, we are done, so we can assume that $h^* \geq 1$ is the largest nice integer. By Lemma 7.15, we have $\tau(u_i) = \tau(v_i)$ for every $i \geq 1$, so $L = M$. In particular, for every $i \geq 1$ we have that j_i is defined if and only if k_i is defined and, if so $j_i = k_i$. Since h^* is nice, then j_{h^*} and k_{h^*} are defined, $j_{h^*} = k_{h^*}$ and $\min_{\bar{u}}[1, h^*-1] = \min_{\bar{v}}[1, h^*-1]$. We know that $\min_{\bar{u}} = \min_{\bar{u}}[1, h^*-1] \min_{\bar{u}'_{h^*}}$ and $\min_{\bar{v}} = \min_{\bar{v}}[1, h^*-1] \min_{\bar{v}'_{h^*}}$, so $\min_{\bar{u}}[1, h^*-1] = \min_{\bar{v}}[1, h^*-1]$ implies that in order to prove that $\min_{\bar{u}} = \min_{\bar{v}}$ we only have to prove that $\min_{\bar{u}'_{h^*}} = \min_{\bar{v}'_{h^*}}$.

   By Lemma 7.15, we have $\min_{u'_{h^*}} = \min_{v'_{h^*}}$. Since $\tau(u_i) = \tau(v_i)$ for every $i \geq 1$, $(u_i)_{i \geq j_{h^*}}$ is an occurrence of $\min_{u'_{h^*}}$ starting at u'_{h^*} and $(v_i)_{i \geq j_{h^*}}$ is an occurrence of $\min_{v'_{h^*}}$ starting at v'_{h^*}, we obtain $\ell_{u'_{h^*}} = \ell_{v'_{h^*}}$. As a consequence, $\gamma_{u'_{h^*}} = \min_{u'_{h^*}}[1, \ell_{u'_{h^*}}] = \min_{v'_{h^*}}[1, \ell_{v'_{h^*}}] = \gamma_{v'_{h^*}}$. In particular, it must be $\mathtt{t}_{u'_{h^*}} = \mathtt{t}_{v'_{h^*}}$, so from $\gamma_{u'_{h^*}} = \gamma(v'_{h^*})$ we conclude $\lambda(\bar{u}'_{h^*}) = \lambda(\bar{v}'_{h^*})$. Moreover, it must be $\mathtt{t}_{u'_{h^*}} = \mathtt{t}_{v'_{h^*}} = 2$, otherwise we would conclude that $h^* + 1$ is nice, which contradicts the maximality of h^*. As a consequence, by the definition of $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ we conclude that $\min_{\bar{u}'_{h^*}} = (\lambda(\bar{u}'_{h^*}))^\omega = \lambda(\bar{v}'_{h^*}))^\omega = \min_{\bar{v}'_{h^*}}$.

2. Assume that $\min_u \prec \min_v$; we must prove that $\min_{\bar{u}} \prec' \min_{\bar{v}}$. Notice that it cannot happen that every $h \geq 1$ is nice, otherwise we would obtain $\min_u = \min_v$, a contradiction. Let $h^* \geq 1$ be the biggest nice integer. In particular, j_{h^*} and k_{h^*} are defined, $j_{h^*} = k_{h^*}$, $\min_u[1, j_{h^*} - 1] = \min_v[1, k_{h^*} - 1]$ and $\min_{\bar{u}}[1, h^*-1] = \min_{\bar{v}}[1, h^*-1]$. We know that $\min_u = \min_u[1, j_{h^*} - 1] \min_{u'_{h^*}}$ and $\min_v = \min_v[1, k_{h^*} - 1] \min_{v'_{h^*}}$, so from $\min_u \prec \min_v$ we conclude $\min_{u'_{h^*}} \prec \min_{v'_{h^*}}$. Since $\min_{\bar{u}} = \min_{\bar{u}}[1, h^*-1] \min_{\bar{u}'_{h^*}}$, $\min_{\bar{v}} = \min_{\bar{v}}[1, h^*-1] \min_{\bar{v}'_{h^*}}$ and $\min_{\bar{u}}[1, h^*-1] = \min_{\bar{v}}[1, h^*-1]$, in order to prove that $\min_{\bar{u}} \prec' \min_{\bar{v}}$ we only have to prove that $\min_{\bar{u}'_{h^*}} \prec' \min_{\bar{v}'_{h^*}}$.

Notice that it cannot be $\gamma_{u'_{h^*}} = \gamma_{v'_{h^*}}$ and $\tau(u'_{h^*}) = \tau(v'_{h^*})$, because:

(a) If $\tau(u'_{h^*}) = \tau(v'_{h^*}) = 2$, then by Lemma 7.21 we would conclude $\min_{u'_{h^*}} = \min_{v'_{h^*}}$, a contradiction.

(b) If $\tau(u'_{h^*}) = \tau(v'_{h^*}) = 3$, then $h^* + 1$ would be a nice integer, which contradicts the maximality of h^*.

From this observation, Lemma 7.20 and $\min_{u'_{h^*}} \prec \min_{v'_{h^*}}$ we conclude that one of the following must be true:

(a) $\gamma_{u'_{h^*}}$ is not a prefix of $\gamma_{v'_{h^*}}$, $\gamma_{v'_{h^*}}$ is not a prefix of $\gamma_{u'_{h^*}}$ and $\gamma_{u'_{h^*}} \prec \gamma_{v'_{h^*}}$.

(b) $\gamma_{u'_{h^*}} = \gamma_{v'_{h^*}}$, $\mathtt{t}_{u'_{h^*}} = 2$ and $\mathtt{t}_{v'_{h^*}} = 3$.

(c) $\gamma_{u'_{h^*}}$ is a strict prefix of $\gamma_{v'_{h^*}}$.

In all three cases we conclude $(\gamma_{u'_{h^*}}, \mathtt{t}_{u'_{h^*}}) \prec' (\gamma_{v'_{h^*}}, \mathtt{t}_{v'_{h^*}})$, or equivalently, $\lambda(\bar{u}'_{h^*}) \prec' \lambda(\bar{v}'_{h^*})$, which implies $\min_{\bar{u}'_{h^*}} \prec' \min_{\bar{v}'_{h^*}}$.

□

Since $\preceq$ is a total order (so exactly one among $\min_u \prec \min_v$, $\min_u = \min_v$ and $\min_v \prec \min_u$ holds true), from Theorem 7.24 we immediately obtain the following result.

Corollary 7.25. *Let $\mathcal{D} = (Q, E)$ be a DFA. Let $u, v \in Q$ be such that $\tau(u) = \tau(v) = 3$.*

1. *It holds $\min_u = \min_v$ if and only if $\min_{\bar{u}} = \min_{\bar{v}}$.*

2. *It holds $\min_u \prec \min_v$ if and only if $\min_{\bar{u}} \prec' \min_{\bar{v}}$.*

In particular, if we have the min-partition of $\bar{Q}$ (with respect to $\bar{\mathcal{D}}$), then we also have the min-partition of $\{u \in Q \mid \tau(u) = 3\}$ (with respect to $\mathcal{D}$).

Lastly, we show that our reduction to $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ can be computed within $O(n^2)$ time.

Lemma 7.26. *Let $\mathcal{D} = (Q, E)$ be a trimmed DFA. Then, we can build $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$ in $O(|Q|^2)$ time.*

Proof. The definition of $\bar{\mathcal{D}}$ implies that in order to build $\bar{\mathcal{D}}$ it is sufficient to compute γ_u, $\mathtt{t}_u$ and $G_{\ell_u}(u)$ for every $u \in Q$ such that $\tau(u) = 3$; moreover, we also need to compute the total order $\preceq'$, so that we can remap Σ' to an integer alphabet consistently with the total order $\preceq'$ on Σ'.

Fix $u \in Q$ such that $\tau(u) = 3$. For every $1 \leq i \leq \ell_u - 1$, let $E_i(u)$ be the set of all edges entering a state in $G_i(u)$, and let $m_i(u) = |E_i(i)|$. Note that since the $G_i(u)$'s are pairwise distinct, then the $E_u(i)$'s are pairwise disjoint. Let us prove that $\sum_{i=1}^{\ell_u - 1} m_i(u) \in O(n)$. To this end, it will suffice to prove that for every $v \in Q$ there exist at most two edges in $\bigcup_{i=1}^{\ell_u-1} E_i(u)$ whose start states are equal to v. Fix $v \in Q$, and let $1 \leq i \leq \ell_u - 1$ be the smallest integer such that v is the first state of an edge $(v, v') \in E_i(u)$ for some $v' \in G_i(u)$, if such a v' exists. Notice that v' is uniquely determined because the value $\lambda(z)$ does not depend on the choice of $z \in G_i(u)$, and $\mathcal{D}$ is a DFA. This means that $(v, v') \in E_i(u)$ is uniquely determined. In order to prove our claim, it will suffice to show that if $(v, v'') \in E_j(u)$ for some $i+1 \leq j \leq \ell_u - 1$ and $v'' \in G_j(u)$, then $\lambda(v'') = \lambda(v)$, because we will conclude that j and v'' are uniquely determined, since $\mathcal{D}$ is a DFA. Since $v' \in G_i(u)$ and $(v, v') \in E_i(u)$, by the definition of $G_{i+1}(u)$, we have $\lambda(G_{i+1}(u)) \preceq \lambda(v)$. Since $2 \leq i + 1 \leq j \leq \ell_u - 1$, then by Lemma 7.18 we obtain $\min_u[j] \preceq \min_u[i+1]$, which by Lemma 7.19 is equivalent to $\lambda(G_j(u)) \preceq \lambda(G_{i+1}(u))$, and so we conclude $\lambda(G_j(u)) \preceq \lambda(v)$, or equivalently, $\lambda(v'') \preceq \lambda(v)$. Since $v'' \in G_j(u)$ and $2 \leq j \leq \ell_u - 1$, then $\tau(v'') = \tau(G_j(u)) = 1$, so from $(v, v'') \in E$ we conclude that it must be $\lambda(v) \preceq \lambda(v'')$ because $\mathcal{D}$ is trimmed, and so $\lambda(v'') = \lambda(v)$.

Let us show that in $O(|Q|^2)$ we can compute $G_{\ell_u}(u)$, γ_u and $\mathtt{t}_u$ for every $u \in Q$ such that $\tau(u) = 3$. It will suffice to show that, for a fixed $u \in Q$ such that $\tau(u) = 3$, we can compute $G_{\ell_u}(u)$, γ_u and $\mathtt{t}_u$ in $O(|Q|)$ time.

Let us show how we recursively compute each set $G_i(u)$, for $1 \leq i \leq \ell_u$. During the algorithm, we will mark some states. Initially, no state is marked, and after step $i \geq 1$, a state of Q is marked if and only if it belongs to $\bigcup_{j=2}^{i-1} G_j(u)$. If $i = 1$, we just let $G_1(u) = \{u\}$, and we define $c_1 = \lambda(u)$. Now, assume that $i \geq 2$, and assume that we have computed $G_{i-1}(u)$. We first scan all edges in $E_{i-1}(u)$ and we collect the set R_i the set of all start states of these edges. Then, by scanning R_i, we first determine $c_i = \min\{\lambda(v) \mid v \in R_i\}$, then we determine $R'_i = \{v \in R_i \mid \lambda(v) = c_i\}$. Next, we determine $t_i = \min\{\tau(v) \mid v \in R'_i\}$ and $R''_i = \{v \in R'_i \mid \lambda(v) = t_i\}$. By checking which states in R''_i are marked, we determine $G_i(u)$. Finally, we mark all states in $G_i(u)$. We can perform all these steps in $O(m_{i-1}(u))$ time, where the hidden constant in the asymptotic notation does not depend on i. Now, we check whether $t_i \geq 2$. If $t_i \geq 2$, then $i = \ell_u$ and we are done. Otherwise, we have $i < \ell_u$ and we proceed with step $i+1$.

We conclude that we can determine $G_{\ell_u}(u)$ in time $\sum_{i=1}^{\ell_u-1} O(m_i(u)) = O(\sum_{i=1}^{\ell_u-1} m_i(u)) = O(n)$, where we have used that the hidden constant in $O(m_i(u))$ does not depend on i. In addition, we have $\gamma_u = c_1c_2 \dots c_{\ell_u}$ and $\mathtt{t}_u = t_{\ell_u}$.

We are only left with showing how to compute $\preceq'$. Essentially, we only have to radix sort the strings $\gamma(u)$'s by taking into account that $\preceq'$ is defined by considering a slight variation of the lexicographic order. More precisely, we proceed as follows. We know that $|\gamma(u)| \leq |Q| + 1$ for each $u \in Q$ such that $\tau(u) = 3$, so we first pad the end of each $\gamma(u)$ with a special character larger than all characters in the alphabet until the length of each string is exactly $|Q| + 1$. Next, we consider two extra special characters d_2 and d_3 such that $d_2 \prec' d_3$, and we append exactly one of this character to each $\gamma(u)$: we append d_2 if $\mathtt{t}_u = 2$, and we append d_3 if $\mathtt{t}_u = 3$. Now, we radix sort the (modified) $\gamma(u)$'s in $O(|Q|^2)$ time, so obtaining $\preceq'$. $\square$

7.6 Merging

We want to determine the min-partition $\mathcal{A}$ of Q, assuming that we already have the min-partition $\mathcal{B}$ of $\{u \in Q \mid \tau(u) = 3\}$.

First, note that we can easily build the min-partition $\mathcal{B}'$ of $\{u \in Q \mid \tau(u) = 2\}$. Indeed, if $\tau(u) = 2$, then $\min_u = \lambda(u)^\omega$ by Lemma 7.5. As a consequence, if $\tau(u) = \tau(v) = 2$, then (i) $\min_u = \min_v$ if and only if $\lambda(u) = \lambda(v)$ and (ii) $\min_u \prec \min_v$ if and only if $\lambda(u) \prec \lambda(v)$, so we can build $\mathcal{B}'$ in $O(|Q|)$ time by using counting sort.

For every $c \in \Sigma$ and $t \in \{1, 2, 3\}$, let $Q_{c,t} = \{v \in Q \mid \lambda(v) = c, \tau(v) = t\}$. Consider $u, v \in Q$: (i) if $\lambda(u) \prec \lambda(v)$, then $\min_u \prec \min_v$ and (ii) if $\lambda(u) = \lambda(v)$ and $\tau(u) < \tau(v)$, then $\min_u \prec \min_v$ by Corollary 7.6. As a consequence, in order to build $\mathcal{A}$, we only have to build the min-partition $\mathcal{A}_{c,t}$ of $Q_{c,t}$, for every $c \in \Sigma$ and every $t \in \{1, 2, 3\}$.

A possible way to implement each $\mathcal{A}_{c,t}$ is by means of an array $A_{c,t}$ storing the elements of $Q_{c,t}$, where we also use a special character to delimit the border between consecutive elements of $A_{c,t}$.

It is immediate to build incrementally $\mathcal{A}_{c,3}$ for every $c \in \Sigma$, from its smallest element to its largest element. At the beginning, $A_{c,3}$ is empty for every $c \in \Sigma$. Then, scan the elements I in $\mathcal{B}$ from smallest to largest, and add I to $\mathcal{A}_{c,3}$, where $c = \lambda(u)$ for any $u \in I$ (the definition of c does not depend on the choice of u). We scan $\mathcal{B}$ only once, so this step takes $O(|Q|)$ time. Analogously, we can build $\mathcal{A}_{c,2}$ for every $c \in \Sigma$ by using $\mathcal{B}'$.

We are only left with showing how to build $\mathcal{A}_{c,1}$ for every $c \in \Sigma$. At the beginning, each $A_{c,1}$ is empty, and we will build each $\mathcal{A}_{c,1}$ from its smallest element to its largest element. During this step of the algorithm, we will gradually mark the states $u \in Q$ such that $\tau(u) = 1$. At the beginning of the step, no such state is marked, and at the end of the step all these states will be marked. Let $\Sigma = \{c_1, c_2, \dots, c_\sigma\}$, with $c_1 \prec c_2 \prec \cdots \prec c_\sigma$. Notice that it must be $Q_{c_1,1} = \emptyset$, because if there existed $u \in Q_{c_1,1}$, then it would be $\min_u \prec c_1^\omega$ by Lemma 7.5 and so c_1 would not be the smallest character in Σ. Now, consider $Q_{c_1,2}$; we have already fully computed $A_{c_1,2}$. Process each I in $A_{c_1,2}$ from smallest to largest, and for every $c_k \in \Sigma$ compute the set J_k of all non-marked states $v \in Q$ such that $\tau(v) = 1$, $\lambda(v) = c_k$, and $(u, v) \in E$ for some $u \in I$. Then, if $J_k \neq \emptyset$ add J_k to $A_{c_k,1}$ and mark the states in J_k. After processing the elements in $A_{c_1,2}$, we process the element in $A_{c_1,3}$, $A_{c_2,1}$, $A_{c_2,2}$, $A_{c_2,3}$, $A_{c_3,1}$ and so on, in this order. Each $A_{c_i,t}$ is processed from its (current) smallest element to its (current) largest element. We never remove or modify elements in any $A_{c,t}$, but we only add elements to the $A_{c,1}$'s. More precisely, when we process I in $A_{c,t}$, for every $c_k \in \Sigma$ we compute the set J_k of all non-marked states $v \in Q$ such that $\tau(v) = 1$, $\lambda(v) = c_k$, and $(u, v) \in E$ for some $u \in I$ and, if $J_k \neq \emptyset$, then we add J_k to $A_{c_k,1}$ and we mark the states in J_k.

The following lemma shows that our approach is correct. Let us give some intuition. A *prefix* of a min-partition $\mathcal{C}$ is a subset $\mathcal{C}'$ of $\mathcal{C}$ such that, if $I, J \in \mathcal{C}$, $I < J$ and $J \in \mathcal{C}'$, then $I \in \mathcal{C}'$. Notice that every prefix of $\mathcal{A}$ is obtained by taking the union of $\mathcal{A}_{c_1,2}$, $\mathcal{A}_{c_1,3}$, $\mathcal{A}_{c_2,1}$, $\mathcal{A}_{c_2,2}$, $\mathcal{A}_{c_2,3}$, $\mathcal{A}_{c_3,1}$, $\dots$ in this order up to some element $\mathcal{A}_{c,t}$, where possibly we only pick a prefix of the last element $\mathcal{A}_{c,t}$. Then, we will show that, when we process I in $A_{c,t}$, we have already built the prefix of $\mathcal{A}$ whose largest element is I. This means that, for every $v \in J_k$ and for any *any* occurrence $(v_i)_{i \geq 1}$ of $\min_v$ starting at v, it must hold that v_2 is in I.

Lemma 7.27. *Let $\mathcal{D} = (Q, E)$ be a DFA. If we know the min-partition of $\{u \in Q \mid \tau(u) = 3\}$, then we can build the min-partition of Q in $O(|E|)$ time.*

Proof. We have seen that the algorithm correctly builds $\mathcal{A}_{c,3}$ and $\mathcal{A}_{c,2}$ for every $c \in \Sigma$. Note that when the algorithm builds the $\mathcal{A}_{c,1}$'s, it never modifies the $A_{c,2}$'s and the $A_{c,3}$'s (because we only add elements to the $A_{c,1}$'s).

Let us prove that, when we consider I in $A_{c_i,t}$ (and before computing the J_k's), then:

1. I is an element of $\mathcal{A}$ and we have already built the prefix of $\mathcal{A}$ whose largest element is I.

2. every $A_{c,1}$ contains a prefix of $\mathcal{A}_{c,1}$.

At the beginning of the algorithm we consider I in $A_{c_1,t}$, with $t \in \{2, 3\}$, so our claim is true because we have already built $\mathcal{A}_{c_1,2}$ and $\mathcal{A}_{c_1,3}$, and all the $A_{c,1}$'s are empty.

Now, assume that our claim is true when we consider $I \in \mathcal{A}_{c_i,t}$. We want to prove that it is true when we process the next element (if it exists). When we consider I, we compute the nonempty J_k's, and we add each such J_k to $A_{c_k,1}$. We now want to prove that J_k is correctly identified as the next element in $A_{c_k,1}$.

- First, let us prove that, if $v_1, v_2 \in J_k$, then $\min_{v_1} = \min_{v_2}$. Since $v_1, v_2 \in J_k$, then there exist $u_1, u_2 \in I$ such that $(u_1, v_1), (u_2, v_2) \in E$. Since by the inductive hypothesis I is an element of $\mathcal{A}$ and we have already built the prefix of $\mathcal{A}$ whose largest element is I and since v_1 and v_2 were not marked before, then it must be $\min_{v_1} = c_k \min_{u_1}$, $\min_{v_2} = c_k \min_{u_2}$ and $\min_{u_1} = \min_{u_2}$, so we conclude $\min_{v_1} = \min_{v_2}$.

- We are only left with showing that if $v_1 \in J_k$ and $v_2 \in Q_{c_k,1}$ is not in some element of $A_{c_k,1}$ after J_k is added to $A_{c_k,1}$, then $\min_{v_1} \prec \min_{v_2}$. Let $u_1 \in I$ such that $(u_1, u_2) \in E$; we have shown that $\min_{v_1} = c_k \min_{u_1}$. The definition of v_2

implies that $\min_{v_2} = c_k \min_{u_2}$ for some u_2 that we have not processed yet. Since by the inductive hypothesis I is an element of $\mathcal{A}$ and we have already built the prefix of $\mathcal{A}$ whose largest element is I, then it must be $\min_{u_1} \prec \min_{u_2}$, and we conclude $\min_{v_1} \prec \min_{v_2}$.

Now, let us prove that, if after considering I in $A_{c_i,t}$, the next element to be considered is not in $A_{c_i,t}$, then the construction of $\mathcal{A}_{c_1,t}$ is complete. If $t \in \{2, 3\}$ then the conclusion is immediate because $\mathcal{A}_{c_i,2}$ and $\mathcal{A}_{c_i,3}$ had already been fully built. Now assume that $t = 1$. Suppose for sake of contradiction that there exists $v \in Q_{c_i,1}$ which has not been added to some element in $A_{c_i,1}$. Without loss of generality, we choose v such that $\min_v$ is as small as possible. Since $\tau(v) = 1$, then there exists $u \in Q$ such that $(u, v) \in E$ and $\min_u \prec \min_v$ (and so $\lambda(u) \preceq \lambda(v) = c_i$). If $\lambda(u) \prec c_i$, then we immediately obtain that u has already been processed (because we have already built the prefix of $\mathcal{A}$ whose largest element is I) and so v should have already been processed when u was processed, a contradiction. If $\lambda(u) = c_i$, then by Corollary 7.6 we have $\tau(u) \leq \tau(v)$, so $\tau(u) = 1$ and $u \in Q_{c_i,1}$; the minimality of v implies that u was previously added to some element of $A_{c_i,1}$ and so v should have been processed, again a contradiction.

As a consequence, when we process the next element after I in $A_{c_i,t}$, then our claim will be true (independently of whether the next element is in $A_{c_i,t}$ or not). In particular, if there is no next element, we conclude that all $\mathcal{A}_{c,1}$'s have been built,

Lastly, we can build each $\mathcal{A}_{c,1}$ in $O(|E|)$ time, because we only need to scan each state and each edge once.

□

7.7 The Complementary Case

We have shown that in $O(n^2)$ time we can reduce the problem of determining the min-partition of Q to the problem of determining the min-partition of the set of all states of a DFA having $|\{u \in Q \mid \tau(u) = 3\}|$ states. Now, we must show that (similarly) in $O(n^2)$ time we can reduce the problem of determining the min-partition of Q to the problem of determining the min-partition of the set of all states of a DFA having $|\{u \in Q \mid \tau(u) = 1\}|$ states. The merging step will be more complex, because the order in which we will process the $\mathcal{A}_{c,t}$ will be from largest to smallest ($\mathcal{A}_{c_\sigma,2}$, $\mathcal{A}_{c_\sigma,1}$, $\mathcal{A}_{c_{\sigma-1},3}$, $\mathcal{A}_{c_{\sigma-1},2}$, $\mathcal{A}_{c_{\sigma-1},1}$, $\mathcal{A}_{c_{\sigma-2},3}$ and so on) so we will need to update some elements of some $A_{c,t}$'s to include the information about minima that we may infer at a later stage of the algorithm.

We use the same notation that we used in the previous sections (with a complementary meaning) so that it will be easy to follow our argument.

7.8 The Complementary Case: the Recursive Step

For every $u \in Q$ such that $\tau(u) = 1$, let ℓ_u be the smallest integer $k \geq 2$ such that $\tau(u_k) \leq 2$, where $(u_i)_{i\geq 1}$ is an occurrence of $\min_u$ starting at u. For every $1 \leq i \leq \ell_u$, we can define each $G_i(u)$ as we did before; now, it will be $\tau(G_i(u)) = 3$ for every $2 \leq i \leq \ell_u - 1$. In fact, when we explore the DFA starting from u in a backward fashion and we (implicitly) build a prefix of $\min_u$, at each step it is still true that we must consider the states with minimum value $\lambda(v)$ and, among those, the states with minimum value $\tau(v)$. This time, there is no chance that we include in $G_i(u)$ a state already included before: at every step, $\lambda(G_i(u))$ can only increase (because $\tau(G_i(u)) = 3$), so if there were a cycle, then all edges of the cycles would have the same label and by Lemma 7.12 we would conclude $\tau(G_i(u)) \leq 2$ for some $2 \leq i \leq \ell_u - 1$, a contradiction. The γ_u's and the $\mathtt{t}_u$'s are defined as before, and Lemma 7.20 still implies that Σ' and $\preceq'$ must be defined as before. When we define $\bar{\mathcal{D}} = (\bar{Q}, \bar{E})$, we define $\bar{Q} = \{\bar{u} \mid \tau(u) = 1\}$ and we define $\bar{E}$ as before. It is easy to check that we can build $\bar{\mathcal{D}}$ in $O(|Q|^2)$ time as before, and if we have the min-partition of $\bar{Q}$ (with respect to $\bar{\mathcal{D}}$), then we also have the min-partition of $\{u \in Q \mid \tau(u) = 1\}$ (with respect to $\bar{\mathcal{D}}$).

7.9 The Complementary Case: Merging

Let $u \in Q$ such that $\tau(u) = 3$. By Lemma 7.5, there exist $k \geq 1$, $c \in \Sigma$ and $\gamma' \in \Sigma^\omega$ such that $\min_u = \lambda(u)^k c\gamma'$ and $\lambda(u) \prec c$. Then, we define $\psi(u) = k$.

In the following, we will often use the following observation. Let $v \in Q$ such that $\tau(v) = 3$. Let $u \in Q$ such that $(u, v) \in E$ and $\min_v = \lambda(v)\min_u$. Then, (i) $\lambda(v) \preceq \lambda(u)$ and (ii) if $\lambda(v) = \lambda(u)$, then $\tau(u) = 3$ and $\psi(v) = \psi(u) + 1$.

Note that, if $u, v \in Q$ are such that $\tau(u) = \tau(v) = 3$, $\lambda(u) = \lambda(v)$ and $\psi(u) \neq \psi(v)$, then $\min_u \prec \min_v$ if and only if $\psi(v) < \psi(u)$.

It is easy to compute all $\psi(u)$'s in $O(|E|)$ time. Start from each $u \in Q$ such that $\tau(u) = 3$, and explore the DFA in a backward fashion by only following edges labeled $\lambda(u)$, until either we encounter a state for which we have already determined $\psi(u)$, or we cannot explore the graph any longer. For all the states u' that we encounter it must be $\tau(u') = 3$ (otherwise it would be $\tau(u) \neq 3$), and if we cannot explore any longer from some u' that we encounter, it must be $\psi(u') = 1$. We cannot encounter the same state twice because $\mathcal{D}$ is a DFA, and we cannot have cycles otherwise it would be $\tau(u) \neq 3$ by Lemma 7.12. As a consequence, we have built a tree (where the root u has no *outgoing* edges), and computing the $\psi(u)$'s is equivalent to computing the height of each state.

Let us show how we can use the $\psi(u)$'s to determine the min-partition $\mathcal{A}$ of Q, assuming that we already have the min-partition $\mathcal{B}$ of $\{u \in Q \mid \tau(u) = 1\}$. As before we have the min-partition $\mathcal{B}'$ of $\{u \in Q \mid \tau(u) = 2\}$. For every $c \in \Sigma$ and for every $t \in \{1, 2, 3\}$, we define $Q_{c,t}$, $\mathcal{A}_{c,t}$ and $A_{c,t}$ as before, and our problem reduces to compute each $\mathcal{A}_{c,t}$.

By using $\mathcal{B}$ and $\mathcal{B}'$, we can compute the $\mathcal{A}_{c,1}$'s and the $\mathcal{A}_{c,2}$'s as before, so the challenging part is to compute the $\mathcal{A}_{c,3}$'s. Let $\Sigma = \{c_1, c_2, \dots, c_\sigma\}$, with $c_1 \prec c_2 \prec \dots \prec c_\sigma$. During the algorithm, we will assign a number $\psi(I) \geq 1$ to every element I being in some $A_{c,3}$ at some point.

Notice that $\mathcal{A}_{c_\sigma,3}$ must be empty because otherwise we would conclude that c_σ is not largest character. This suggest that this time the order in which we process the $\mathcal{A}_{c,t}$'s must be $\mathcal{A}_{c_\sigma,2}$, $\mathcal{A}_{c_\sigma,1}$, $\mathcal{A}_{c_{\sigma-1},3}$, $\mathcal{A}_{c_{\sigma-1},2}$, $\mathcal{A}_{c_{\sigma-1},1}$, $\mathcal{A}_{c_{\sigma-2},3}$ and so on. Moreover, we will build each $\mathcal{A}_{c,t}$ incrementally from its largest element to its smallest element (so we will consider *suffixes* of min-partitions, not prefixes). This time we will not mark states, but we will mark entries of the $A_{c,t}$'s to indicate that a state has been removed from an element in $A_{c,t}$. Intuitively, this time we need to remove states because it will be true that when we process I in $A_{c_i,t}$ then we have already built the suffix (not the prefix) of $\mathcal{A}$ whose *largest* element is I, but we are now building $\mathcal{A}$ from its largest element to its smallest element, so a state reached by an edge leaving a state u in I may also be reached by an edge leaving a state u' that we have not processed, and for which it holds $\min_{u'} \prec \min_u$.

Assume that we process I in $A_{c_i,t}$. Note that if $v \in Q$ is such that $\tau(v) = 3$, $\lambda(v) = c_k$, and $(u, v) \in E$ for some $u \in I$, then it must be $k \leq i$ otherwise $\tau(v) = 1$; if I is an element in the $\mathcal{A}_{c,1}$'s or in the $\mathcal{A}_{c,2}$'s, it must always be $k < i$ because, if it were $k = i$, then we would conclude $\tau(v) \neq 3$. For every $k \leq i$, define $\psi_{I,k} = 1$ if $k < i$, and $\psi_{I,k} = \psi(I) + 1$ if $k = i$. We compute the set J_k of all states $v \in Q$ such that $\tau(v) = 3$, $\lambda(v) = c_k$, $\psi(v) = \psi_{I,k}$ and $(u, v) \in E$ for some $u \in I$ and, if $J_k \neq \emptyset$, then (i) we mark the entries of $A_{c_k,3}$ containing an element in J_k and (ii) we add J_k to $A_{c_k,3}$, letting $\psi(J_k) = \psi_{I,k}$. We assume that we maintain an additional array that for every state in $\{u \in Q \mid \tau(u) = 3\}$ already occurring in some $A_{c,3}$ stores its (unique) current position, so that operation (i) can be performed in constant time without affecting the running time of the algorithm (which is still $O(|E|)$).

Notice that at any time the following will be true in each $A_{c,3}$: (a) if K is in $A_{c,3}$, then $\psi(K) \geq 1$; (b) if $A_{c,3}$ is nonempty, then the first K that we have added is such that $\psi(K) = 1$; (c) If K' has been added to $A_{c,3}$ immediately after K, then $\psi(K) \leq \psi(K') \leq \psi(K) + 1$; (d) If $v \in K$ for some K in $A_{c,3}$, then $\psi(v) = \psi(K)$.

Let us prove that, when we consider I in $A_{c_i,t}$ (and before computing the J_k's), then I is an element of $\mathcal{A}$ and we have already built the suffix of $\mathcal{A}$ whose largest element is I.

At the beginning of the algorithm we consider I in $A_{c_\sigma,t}$, with $t \in \{1, 2\}$, so our claim is true because we have already built $\mathcal{A}_{c_\sigma,2}$ and $\mathcal{A}_{c_\sigma,1}$.

Now, assume that our claim is true when we consider I in $A_{c_i,t}$. Let us prove that we can consider the next element I' in $A_{c'_i,t'}$ (if it exists), then I' is an element of $\mathcal{A}$ and we have already built the suffix of $\mathcal{A}$ whose largest element is I'. Notice that either $i' = i$ or $i' = i - 1$.

1. Assume that $i' = i$ and $t' = t$. If $t' = t \neq 3$ we are done because we have already built the $\mathcal{A}_{c,2}$'s and the $\mathcal{A}_{c,1}$'s. Now assume that $t' = t = 3$. This implies that $\psi(I) + 1 \leq \psi(I') \leq \psi(I)$.

   (a) Suppose that $\psi(I') = \psi(I)$. First, let us prove that, if $v_1, v_2 \in I'$, then $\min_{v_1} = \min_{v_2}$. Let $u_1, u_2 \in Q$ be such that $(u_1, v_1), (u_2, v_2) \in E$, $\min_{v_1} = c_i \min_{u_1}$ and $\min_{v_2} = c_i \min_{u_2}$. Since $\tau(v) = 3$ we obtain that either (i) $c_i \prec \lambda(u_1)$, or (ii) $\lambda(u_1) = c_i$, $\tau(u_1) = 3$ and $\psi(u_1) = \psi(I') - 1 = \psi(I) - 1$. In both cases, we conclude that u_1 is an element K of the suffix of $\mathcal{A}$ whose largest element is I, which by the inductive hypothesis has been correctly built, and so v_1 is added to an element of $A_{c_i,t}$; v_1 is never removed from this element otherwise in I_{v_1} there would be a string smaller that $\min_{v_1}$. As a consequence, it must also be $u_2 \in K$, so by the inductive hypothesis $\min_{u_1} = \min_{u_2}$ and we conclude $\min_{v_1} = \min_{v_2}$.

We are only left with proving that, if v_1 is neither in I', nor in the suffix of $\mathcal{A}$ whose largest element is I, and if $v_2 \in I'$, then $\min_{v_1} \prec \min_{v_2}$. The conclusion is immediate if $\tau(v_i) \neq 3$, so we can assume $\tau(v_i) = 3$. It cannot be $c_i = \lambda(v_2) \prec \lambda(v_1)$ otherwise v_1 would be in the suffix of $\mathcal{A}$ whose largest element is I. Since $\tau(v_i) = 3$, it cannot be $\lambda(v_1) = c_i$ and $\psi(v_1) < \psi(I)$, otherwise again v_1 would be in the suffix of $\mathcal{A}$ whose largest element is I. As a consequence, it must $\lambda(v_1) \preceq c_i$ and, if $\lambda(v_1) = c_i$, then $\psi(I') = \psi(I) \leq \psi(v_1)$. If $\lambda(v_1) \prec c_i$, or $\lambda(v_1) = c_i$ and $\psi(I') < \psi(v_1)$, we immediately conclude $\min_{v_1} \prec \min_{v_2}$. Hence, we can assume $\lambda(v_1) = c_i$, $\tau(v_1) = 3$ and $\psi(v_1) = \psi(I')$. Let $u_1, u_2 \in Q$ be such that $(u_1, v_1), (u_2, v_2) \in E$, $\min_{v_1} = c_i \min_{u_1}$ and $\min_{v_2} = c_i \min_{u_2}$. As before, we must have already considered u_1 and u_2, but since v_1 is not in I', by construction it must be $\min_{u_1} \prec \min_{u_2}$ and so we conclude $\min_{v_1} \prec \min_{v_2}$.

(b) Suppose that $\psi(I') = \psi(I) + 1$. Arguing as we did above we infer that all $v \in V_{c_i,3}$ such that $\psi(v) = \psi(I)$ are already in some element of the suffix of $\mathcal{A}$ whose largest element is I. By using this information, the same proof of the previous case shows that I' is correct.

2. Assume that $i' = i$ and $t' \leq 2$. Since we have already built the $\mathcal{A}_{c,2}$'s and the $\mathcal{A}_{c,1}$'s, we only have to prove that, if $t = 3$, then all $v \in Q_{c_i,3}$ are already in some element of the suffix of $\mathcal{A}$ whose largest element is I. Assume for sake of contradiction that this is not true for some $v \in Q_{c_i,3}$, and choose v such that $\psi(v)$ is as small as possible. By arguing as before, we conclude (by the minimality of $\psi(v)$) that v must be in some $A_{c_i,3}$; but since $t' \leq 2$, we conclude that v must be in some element of the suffix of $\mathcal{A}$ whose largest element is I, a contradiction.

3. Assume that $i' = i - 1$ and $t' = 3$. In particular, $\psi(I') = 1$. As in the previous

case, we obtain that if $t = 3$, then all $v \in Q_{c_i,3}$ are already in some element of the suffix of $\mathcal{A}$ whose largest element is I. Moreover, $\mathcal{A}_{c_i,2}$ and $\mathcal{A}_{c_i,1}$ must necessarily empty because we have already built them, and I' is not contained in any of them. First, let us prove that, if $v_1, v_2 \in I'$, then $\min_{v_1} = \min_{v_2}$. Let $u_1, u_2 \in Q$ be such that $(u_1, v_1), (u_2, v_2) \in E$, $\min_{v_1} = c_{i-1} \min_{u_1}$ and $\min_{v_2} = c_{i-1} \min_{u_2}$. Since $\tau(v) = 3$ we obtain that either (i) $c_{i-1} \prec \lambda(u_1)$, or (ii) $\lambda(u_1) = c_{i-1}$, $\tau(u_1) = 3$ and $\psi(u_1) = \psi(I') - 1$. However, case (ii) cannot occur because $\psi(I') = 1$, so it must be $c_{i-1} \prec \lambda(u_1)$, and we conclude that u_1 is an element K of the suffix of $\mathcal{A}$ whose largest element is I. As before, we conclude that also u_2 is in K, $\min_{u_1} = \min_{u_2}$ and $\min_{v_1} = \min_{v_2}$.

We are only left with proving that, if v_1 is neither in I', nor in the suffix of $\mathcal{A}$ whose largest element is I, and if $v_2 \in I'$, then $\min_{v_1} \prec \min_{v_2}$. The conclusion is immediate if $\tau(v_i) \neq 3$, so we can assume $\tau(v_i) = 3$. It cannot be $c_{i-1} = \lambda(v_2) \prec \lambda(v_1)$ otherwise v_1 would be in the suffix of $\mathcal{A}$ whose largest element is I. It cannot be $\lambda(v_1) = c_{i-1}$ and $\psi(v_1) < \psi(I')$, because $\psi(I') = 1$. As a consequence, it must hold $\lambda(v_1) \preceq c_{i-1}$ and, if $\lambda(v_1) = c_{i-1}$, then $1 = \psi(I') \leq \psi(v_1)$. If $\lambda(v_1) \prec c_{i-1}$, or $\lambda(v_1) = c_{i-1}$ and $1 = \psi(I') < \psi(v_1)$, we immediately conclude $\min_{v_1} \prec \min_{v_2}$. Hence, we can assume $\lambda(v_1) = c_{i-1}$, $\tau(v_1) = 3$ and $\psi(v_1) = \psi(I') = 1$. Let $u_1, u_2 \in Q$ be such that $(u_1, v_1), (u_2, v_2) \in E$, $\min_{v_1} = c_{i-1} \min_{u_1}$ and $\min_{v_2} = c_{i-1} \min_{u_2}$. As before, we must have already considered u_1 and u_2, but since v_1 is not in I', by construction it must be $\min_{u_1} \prec \min_{u_2}$ and so we conclude $\min_{v_1} \prec \min_{v_2}$.

4. Assume that $i' = i - 1$ and $t' \leq 2$. As before, we obtain that if $t = 3$, then all $v \in V_{c_i,3}$ are already in some element of the suffix of $\mathcal{A}$ whose largest element is I. Moreover, $\mathcal{A}_{c_i,2}$ and $\mathcal{A}_{c_i,1}$ must necessarily be empty because we have already built them, and I' is not contained in any of them. Since we have already built $\mathcal{A}_{c_{i-1},2}$ and $\mathcal{A}_{c_{i-1},1}$, we only have to prove that $\mathcal{A}_{c_{i-1},3}$ is empty. If it were not empty, in particular there would exists $v \in V_{c_{i-1},3}$ with $\psi(v) = 1$. If $u \in Q$ is such that $(u, v) \in E$ and $\min_v = c_{i-1} \min_u$, then we conclude that it must be $c_{i-1} = \lambda(v) \prec \lambda(u)$. Then, u must be in some element of the the suffix of $\mathcal{A}$ whose largest element is I, so v would be in some element of $A_{c_{i-1},3}$; but this is a contradiction, because $t' \leq 2$.

Lastly, if after considering I in $A_{c_i,t}$ there is no element I' to consider, we can check that every $v \in Q$ is in some element of the suffix of $\mathcal{A}$ whose largest element is I (and so have built $\mathcal{A}$). Indeed, assume for the sake of contradiction that this is not true. Consider the set S of all states v that do not satisfy these properties (it must be $\tau(v) = 3$); let $S' \subseteq S$ be the set of all states v in S for which $\lambda(v)$ is maximal, and let $S'' \subseteq S'$ be the set of all states v in S' for which $\psi(v)$ is minimal. Pick any $v \in S''$, and let $u \in Q$ be such that $(u, v) \in E$ and $\min_v = \lambda(v) \min_u$. As before, we obtain that u must be in some element of the suffix of $\mathcal{A}$ whose largest element is I, so v should be in some $A_{c,3}$ and there would be an element I' to consider, a contradiction.

7.10 The Final Algorithm

In the previous sections, given a DFA $\mathcal{D} = (Q, E)$, we have shown how to build the min-partition of Q in $O(n^2)$ time. It is easy to check that, in $O(n^2)$ time, we can also build the *max-partition* of Q. Indeed, we can build the max-partition of Q by simply considering the transpose total order $\preceq^*$ of $\preceq$ (the one for which $a \preceq^* b$ if and only if $b \preceq a$) and building the min-partition. As a consequence, the algorithm to build the max-partition is entirely symmetrical to the algorithm to build the min-partition.

We are now ready to prove Theorem 7.3. Let $\mathcal{D}_1 = (Q_1, E_1)$ and $\mathcal{D}_2 = (Q_2, E_2)$ be two DFAs on the same alphabet $(\Sigma, \preceq)$, with $Q_1 \cap Q_2 = \emptyset$. We must prove that we can build the min/max partition of (Q_1, Q_2) in $O((|Q_1| + |Q_2|)^2)$ time. We compute $\tau(\min_u)$ for every $u \in Q_1$ and we compute $\tau(\max_u)$ for every $u \in Q_2$. If the number of values equal to 3 is smaller than the number of values equal to 1, then (in time $O(|Q_1|^2 + |Q_2|^2) = O((|Q_1| + |Q_2|)^2)$) we build the DFAs $\bar{\mathcal{D}}_1 = (\bar{Q}_1, \bar{E}_1)$ and $\bar{\mathcal{D}}_2 = (\bar{Q}_2, \bar{E}_2)$ as defined before, where $\bar{Q}_1 = \{\bar{u} \mid u \in Q_1, \tau(\min_u) = 3\}$ and $\bar{Q}_2 = \{\bar{u} \mid u \in Q_2, \tau(\max_u) = 3\}$, otherwise we consider the complementary case (which is symmetrical). When building $\bar{\mathcal{D}}_1 = (\bar{Q}_1, \bar{E}_1)$ and $\bar{\mathcal{D}}_2 = (\bar{Q}_2, \bar{E}_2)$, we define a *unique* alphabet $(\Sigma', \preceq')$ obtained by jointly sorting the $(\gamma_{\min_u}, \mathtt{t}_{\min_u})$'s and the $(\gamma_{\max_u}, \mathtt{t}_{\max_u})$'s, which is possible because Lemma 7.20 also applies to maxima. Note that $|\bar{Q}_1| + |\bar{Q}_2| \leq (|Q_1| + |Q_2|)/2$.

Assume that we have recursively obtained the min/max-partition of $(\bar{Q}_1, \bar{Q}_2)$ with respect to $\bar{\mathcal{D}}_1$ and $\bar{\mathcal{D}}_2$. This yields the min/max-partition of $(\{u \in Q_1 \mid \tau(\min_u) = 3\}, \{u \in Q_2 \mid \tau(\max_u) = 3\})$. Then, we can build the min/max-partition of (Q_1, Q_2)

by jointly applying the merging step, which is possible because both the merging step for minima and the merging step for maxima require to build the $\mathcal{A}_{c,1}$'s by processing $A_{c_1,2}A_{c_1,3}$, $A_{c_2,1}$, $A_{c_2,2}$, $A_{c_2,3}$, $A_{c_3,1}$ and so on in this order.

Since we obtain the same recursion as before, we conclude that we can compute the min/max partition of (Q_1, Q_2) in $O((|Q_1| + |Q_2|)^2)$ time.

Chapter 8

Matching Statistics

In Chapter 4 we showed that the co-lex orders can be as a generalization of suffix arrays from strings to automata. The suffix array of a string was introduced in 1990 by Manber and Myers [88] as a replacement of the *suffix tree* of a string, a data structure introduced by Weiner in 1973 [118]. Suffix arrays do not have the full functionality of suffix trees, but they are more space-efficient, both in theory and in practice. The natural question is whether it is possible to extend the suffix tree from strings to automata. The "suffix tree" of an automaton is expected to store all the paths of the automaton, so it is not clear how the suffix tree should be defined. Notably, in this chapter we provide a first step towards extending suffix tree functionality to automata.

The suffix array of a string can be augmented with some additional data structures — notably, the longest common prefix (LCP) array — so that it is possible to retrieve the full functionalities of a suffix tree [1]. All these components can be successfully compressed, leading to the so-called *compressed suffix trees* [108]. As a consequence, we may define the suffix tree of an automaton by extending the each component from strings to automata. We have already extended the suffix array from strings to automata, so here we will focus on introducing a notion of longest common prefix (LCP) array automata. As a first step, we consider Wheeler DFAs, thus we will define the LCP array of a Wheeler DFA (Definition 8.3).

A typical problem that can be efficiently solved by using the suffix tree of a string is the problem of computing *matching statistics*. Indeed, in bioinformatics we are not only interested in exact pattern matching, but also in a myriad of variations of the pattern matching problem [71], and matching statistics were introduced to solve the approximate pattern matching problem [31]. In particular, Ohlebusch et al. [98] proposed a time- and space-efficient algorithm for computing matching statistics that relies on some components of a compressed suffix tree, notably, the suffix array and the LCP array. As a consequence, we can use matching statistics to test our definition of LCP array for Wheeler DFAs. In fact, we will show that the LCP array of a Wheeler DFA allows efficiently computing matching statistics on a Wheeler DFA (Theorem 8.1).

We have seen that the compressed suffix tree of a string contains a *compressed* version of the LCP array. Storing explicitly the LCP array of a string would require too much space; as expected, the same is true for the LCP array of a Wheeler DFA. We will show that it is possible to *sample* some entries of the LCP array of a Wheeler DFA in such a way that it is possible to quickly retrieve each entry, thus providing a space-time tradeoff (Theorem 8.5), which in particular applies to matching statistics (Theorem 8.6). We also show that our space-time tradeoff improves the navigation of *variable-order de Bruijm graphs* [20], which are used in bioformatics for Eulerian sequence assembly [75, 101] (Theorem 8.8).

8.1 Matching Statistics on Strings

The *matching statistics* of a pattern $\pi = \pi[1..m]$ with respect to a string $S = S[1, n]$ are defined as follows. Assume that $S[n] = \$ \notin \Sigma$, where $\$ \prec a$ for every $a \in \Sigma$. Determining the matching statistics of π with respect to S means determining, for $1 \leq i \leq m$, (i) the longest prefix π' of $\pi[i..m]$ which occurs in S, and (ii) the interval corresponding to the set of all strings starting with π' in the list of all lexicographically sorted suffixes. We can describe (i) and (ii) by means of three values: the length ℓ_i of π', and the endpoints l_i and r_i of the interval considered in (ii). For example, let $S = mississippi\$$ (see Figure 8.1), and $\pi = stpissi$. For $i = 1$, we have $\pi' = s$, so $\ell_1 = 1$ and $[l_1, r_1] = [9, 12]$ (suffixes starting with s). For $i = 2$, we have $\pi' = \varepsilon$, so $\ell_2 = 0$ and $[l_2, r_2] = [1, n] = [1, 12]$ (all suffixes start with the empty string). For $i = 3$, we have $\pi' = pi$, so $\ell_3 = 2$, and $[l_3, r_3] = [7, 7]$ (suffixes starting with pi). For $i = 4$, we have $\pi' = issi$, so $\ell_4 = 4$, and $[l_4, r_4] = [4, 5]$ (suffixes starting with $issi$). One can proceed analogously for $i = 5, 6, 7$.

Let us show how to find the matching statistics of a pattern π with respect to a string S. We describe an algorithm by Ohlebusch et al. [98] based on the backward search (see Chapter 3). The algorithms computes the matching statistics using a number of iterations linear in the length m of the pattern. We start from the end of

i	Sorted suffixes	LCP_S	SA_S
1	$		12
2	i$	0	11
3	ippi$	1	8
4	issippi$	1	5
5	ississippi$	4	2
6	mississippi$	0	1
7	pi$	0	10
8	ppi$	1	9
9	sippi$	0	7
10	sissippi$	2	4
11	ssippi$	1	6
12	ssissippi$	3	3

Figure 8.1: The sorted suffixes of "mississippi$" and the arrays LCP_S and SA_S. We assume that $ is the smallest character.

π, and we use the backward search (starting from the interval $[1, n]$ which corresponds to the set of suffixes prefixed by the empty string) to find the interval of all occurrences of the last character of π in S (if any). Then, starting from the new interval, we use the backward search to find all the occurrences of the suffix of length 2 of π in S (if any), and so on. At some point, it may happen that for some $i \leq m+1$ we have that $\pi[i..m]$ occurs in S, but the next application of the backward search returns the empty interval, so that $\pi[i-1..m]$ does not occur in S (the case $i = m+1$ corresponds to the initial setting when $\pi[i..m]$ is the empty string). We distinguish two cases:

- (Case 1) If $l_i = 1$ and $r_i = n$, this means that all suffixes of S are prefixed by $\pi[i..m]$. This may happen in particular if $i = m+1$, because then $\pi[i..m]$ is the empty string: this means that the first backward search has been unsuccessful. We immediately conclude that character $\pi[i-1]$ does not occur in S, so $\ell_{i-1} = 0$ and $[l_{i-1}, r_{i-1}] = [1, n]$ (because all suffixes start with the empty string). In this case, in the following iterations of the algorithm, we can simply discard $\pi[i-1, m]$: when for $i' \leq i-2$ we will be searching for the longest prefix of $\pi[i', m]$ occurring in S, it will suffice to search for the longest prefix of $\pi[i', i-2]$ occurring in S.

- (Case 2) If $l_i > 1$ or $r_i < n$, this means that the number of suffixes of S starting with $\pi[i..m]$ is less than n. Now, every suffix starting with $\pi[i..m]$ also starts with $\pi[i..m-1]$. If the number of suffixes starting with $\pi[i..m-1]$ is equal to the number of suffixes starting with $\pi[i..m]$, then also $\pi[i-1..m-1]$ does not occur in S. More in general, for $j \leq m-1$ we can have that $\pi[i-1..j]$ occurs in S only if the number of suffixes starting with $\pi[i..j]$ is larger than the number of suffixes starting with $\pi[i..m]$. Since we are interested in maximal matches, we want j to be as large as possible: we will show later how to compute the largest integer j such that the number of suffixes starting with $\pi[i..j]$ is larger than the number of suffixes starting with $\pi[i..m]$. Notice that j always exists, because all n suffixes start with the empty string, but less than n suffixes start with $\pi[i..m]$. After determining j we discard $\pi[j+1..m]$ (so in the following iterations of the algorithm we will simply consider $\pi[1..j]$), and we recursively apply the backward search starting from the interval associated with the occurrences of $\pi[i..j]$ — we will also see how to compute this interval.

Let us apply the above algorithm to $T = mississippi\$$ and $\pi = stpissi$. We start with the interval $[1, n] = [1, 12]$, corresponding to the empty pattern, and character $\pi[7] = i$. A backward step yields the interval $[l_7, r_7] = [2, 5]$ (suffixes starting with i), so $\ell_7 = 1$. Now, we apply a backward step from $[2, 5]$ and $\pi[6] = s$, obtaining $[l_6, r_6] = [9, 10]$ (suffixes starting with si), so $\ell_6 = 2$. Again, we apply a backward step from $[9, 10]$ and $\pi[5] = s$, obtaining $[l_5, r_5] = [11, 12]$ (suffixes starting with ssi), so $\ell_5 = 3$. Again, we apply a backward step from $[11, 12]$ and $\pi[4] = i$, obtaining $[l_4, r_4] = [4, 5]$ (suffixes starting with $issi$), so $\ell_4 = 4$. We now apply a backward step from $[4, 5]$ and $\pi[3] = p$, and we obtain the empty interval. This means that no suffix starts with $pissi$. Notice in Figure 8.1 that the number of suffixes starting with $issi$ is equal to the number of suffixes starting with iss or is, but the number of suffixes starting with i is bigger. As a consequence, we consider the interval of all suffixes starting with i — which is $[2, 5]$ — and we apply a backward step with $\pi[3] = p$. This time the backward step is successful, and we obtain $[l_3, r_3] = [7, 7]$ (suffixes starting with pi), and $\ell_3 = 2$. We now apply a backward step from $[7, 7]$ and $\pi[2] = t$, obtaining the empty interval. This means that no suffix starts with tpi. Notice in Figure 8.1 that the number of suffixes starting with p is bigger than the number of suffixes starting with pi. The corresponding interval is $[7, 8]$, but a backward step with $\pi[2] = t$ is still unsuccessful (so no suffix starts with tp). The number of suffixes starting with p is smaller than the number of suffixes starting with the empty string (which is equal to $n = 12$), so we apply a backward step with $[1, 12]$ and $\pi[2] = t$. Since the backward step is still unsuccessful, we conclude that $\pi[2] = t$ does not occur in S, so $[l_2, r_2] = [1, n] = [1, 12]$ and $\ell_2 = 0$. Finally, we start again from the whole interval $[1, 12]$, and a backward step with $\pi[1] = s$ returns $[l_1, r_1] = [9, 12]$ (suffixes starting with s), so $\ell_1 = 1$.

It is easy to see that the number of iterations is linear in m. Indeed, every time we apply a backward step, either we move to the left across π to compute a new matching statistic, or we increase by at least 1 the length of the suffix of π which is forever discarded. This implies that the number of iterations is bounded by $2|\pi| = 2m$.

We are only left with showing (i) how to compute j and (ii) the interval of all suffixes starting with $\pi[i..j]$ in Case 2 of the algorithm. To this end, we introduce the longest common prefix (LCP) array $\mathsf{LCP}_S = \mathsf{LCP}_S[2, n]$ of S. We define $\mathsf{LCP}_S[i]$ to be the length of the longest common prefix of the $(i-1)$-st lexicographically smallest suffix of S and the i-th lexicographically smallest suffix of S. In Figure 8.1 we have $\mathsf{LCP}_S[5] = 4$ because the fourth lexicographically smallest suffix of S is *issippi*\$, the fifth lexicographically smallest suffix of S is *ississippi*\$, and the longest common prefix of *issippi*\$ and *ississippi*\$ is *issi*, which has length 4. Remember that in the example the backward search starting from $[4, 5]$ (suffixes starting with *issi*) and p was unsuccessful, so computing j means determining the longest prefix of *issi* such that the the number of suffixes starting with such a prefix is bigger than 2. This is easy to compute by using the LCP array: the longest such prefix is the one of length $\max\{\mathsf{LCP}_S[4], \mathsf{LCP}_S[6]\} = \max\{1, 0\} = 1$, so that the desired prefix is i. As a consequence, we are only left with showing how to compute the interval of all suffixes starting with the prefix i — which is $[2, 5]$. Notice that in order to compute this interval, it is enough to expand the interval $[4, 6]$ in both directions as long as the LCP value does not go below 1. Since $\mathsf{LCP}_S[4] = 1$, $\mathsf{LCP}_S[3] = 1$, and $\mathsf{LCP}_S[2] = 0$, and we already know that $\mathsf{LCP}_S[6] = 0$, we conclude that the desired interval is $[2, 5]$. In other words, given a position t, we must be able to compute the biggest integer k less than t such that $\mathsf{LCP}_S[k] < \mathsf{LCP}_S[t]$, and the smallest integer k bigger than t such that $\mathsf{LCP}_S[k] < \mathsf{LCP}_S[t]$ (in our case, $t = 4$). These queries are called PSV (”previous smaller value”) and NSV (”next smaller value”) queries. The LCP array

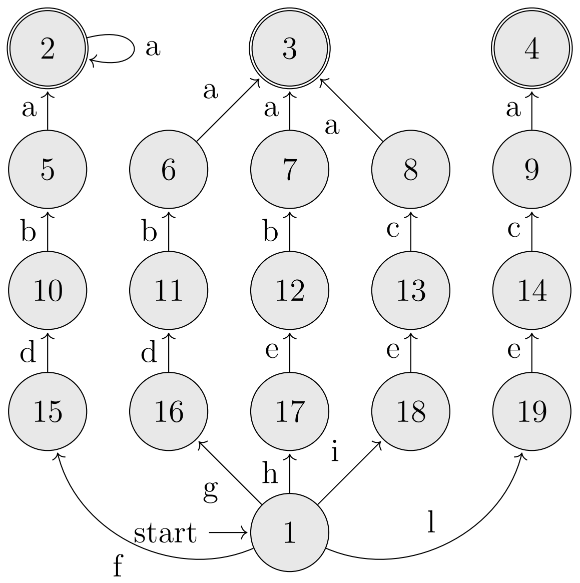


Figure 8.2: A Wheeler DFA. States are numbered according to their positions in the Wheeler order.

can be augmented in such a way that PSV and NSV queries can be solved efficiently: different space-time trade-offs are possible, we refer the reader to [98] for details.

8.2 The LCP Array and Matching Statistics for Wheeler DFAs

In the following, we fix a Wheeler DFA $\mathcal{D} = (Q, \delta, s, F)$, where we assume $Q = \{u_1, \ldots, u_n\}$, with $u_1 < u_2 < \cdots < u_n$ in the Wheeler order (in particular, u_1 coincides with the initial state s). See Figure 8.2 for an example. Given a string $\pi \in \Sigma^*$, if we start from the whole set of states and repeatedly apply the forward search (see Section 4.4) we reach the set of all states u_i for which there exists $\alpha \in I_{u_i}$ prefixed by π^R; this is an interval with respect to the Wheeler order: in the following we call this interval $T(\pi)$.

In view of Section 3.4, the problem of matching statistics will be defined in a symmetrical way on Wheeler DFAs. Given a pattern $\pi = \pi[1..m]$, for every $1 \leq i \leq m$ we want to determine (i) the longest suffix π' of $\pi[1..i]$ which occurs in the Wheeler DFA $\mathcal{D}$ (that is, that can be read somewhere on $\mathcal{D}$ by concatenating edges), and (*ii*) the endpoints of the interval $T(\pi')$.

We will prove the following theorem.

Theorem 8.1. *We can augment the compact representation of a Wheeler DFA $\mathcal{D}$ with $O(n \log n)$ bits, where n is the number of states, in such a way that we can compute the matching statistics of a pattern of length m with respect to the Wheeler DFA in $O(m \log n)$ time.*

Broadly speaking, we can apply the same idea of the algorithm for strings, but in a symmetrical way. We start from the *beginning* of π (not from the end of π), and initially we consider the whole set of states. We repeatedly apply the *forward* search (not the backward search), until the forward search returns the empty interval for some $i \geq 0$. This means that $\pi[1..i+1]$ does not occur in $\mathcal{D}$. Then, if $T(\pi[1..i])$ is the whole set of states, we conclude that the character $\pi[i+1]$ labels no edge in the graph. Otherwise, we must find the smallest j such that $T(\pi[1..i])$ is strictly contained in $T(\pi[j..i])$ (that is, we must determine the longest suffix $\pi[j..i]$ of $\pi[1..i]$ which reaches more states than $\pi[1..i]$). Then we must determine the endpoints of the interval $T(\pi[j..i])$ so that we can go on with the forward search.

The challenge now is to find a way to solve the same subproblems that we identified in Case 2 of the algorithm for strings. In other words, we must find a way to determine j and find the endpoints of the interval $T(\pi[j..i])$. We will show that the solution is not as simple as the one for the algorithm on strings. In particular, we need to define a longest common prefix array of Wheeler DFAs. To this end, let us start with a variant of Corollary 4.10 for Wheeler DFAs, bearing in mind that we consider *infinite* string sorted in *lexicographic* order (see Section 2.1 and Section 2.2).

Lemma 8.2. *Let $i, j \in [1, n]$, with $i < j$. Let $\alpha \in I_{u_i}$ and $\beta \in I_{u_j}$. Then, $\alpha \preceq \beta$.*

Proof. Let $f_1, f_2, \dots$ in $[1, n]$ be such that (i) $f_1 = i$, (ii) $u_{f_k} \in \delta(u_{f_{k+1}}, \alpha[k])$ for every $k \geq 1$. Analogously, let $g_1, g_2, \dots$ in $[1, n]$ be such that (i) $g_1 = j$, (ii) $u_{g_k} \in \delta(u_{g_{k+1}}, \beta[k])$ for every $k \geq 1$. Since $f_1 < i < j = g_1$, from Axiom 1 we obtain $\alpha[1] \preceq \max_{\lambda(u_{f_1})} \preceq \min_{\lambda(u_{g_1})} \preceq \beta[1]$. If $\alpha[1] \prec \beta[1]$, we immediately conclude that $\alpha \prec \beta$. If $\alpha[1] = \beta[1]$, then from $u_{f_1} \in \delta(u_{f_2}, \alpha[1])$, $u_{g_1} \in \delta(u_{g_2}, \beta[1])$, $f_1 < g_1$ and Axiom 2 we conclude $f_2 < g_2$. Analogously, from Axiom 1 we obtain $\alpha[2] \preceq \max_{\lambda(u_{f_2})} \preceq \min_{\lambda(u_{g_2})} \preceq \beta[2]$. If $\alpha[2] \prec \beta[2]$, we immediately conclude that $\alpha \prec \beta$, otherwise by proceeding as before we obtain $f_3 < g_3$. By arguing inductively, either at some point we conclude $\alpha \prec \beta$, or it must be $\alpha[k] = \beta[k]$ for every $k \geq 1$ and so $\alpha = \beta$. $\square$

Lemma 8.2 implies that:

$$\min_1 \preceq \max_1 \preceq \min_2 \preceq \max_2 \preceq \cdots \preceq \max_{n-1} \preceq \min_n \preceq \max_n.$$

This suggests to generalize the LCP array as follows. Given $\alpha, \beta \in \Sigma^* \cup \Sigma^\omega$, let $\mathsf{lcp}(\alpha, \beta)$ be the length of the longest common prefix of α and β (if $\alpha = \beta \in \Sigma^\omega$, define $\mathsf{lcp}(\alpha, \beta) = \infty$).

Definition 8.3. The *longest common prefix (LCP) array* of a Wheeler DFA $\mathcal{D}$ is the array $\mathsf{LCP}_\mathcal{D} = \mathsf{LCP}_\mathcal{D}[2, 2n]$ which contains the following $2n-1$ values as follows in this order: $\mathsf{lcp}(\min_1, \max_1)$, $\mathsf{lcp}(\max_1, \min_2)$, $\mathsf{lcp}(\min_2, \max_2)$, $\dots$, $\mathsf{lcp}(\max_{n-1}, \min_n)$, $\mathsf{lcp}(\min_n, \max_n)$.

From the above characterization of $\min_i$ and $\max_i$, one can prove that for every entry either $\mathsf{LCP}_\mathcal{D}[i] = \infty$ or $\mathsf{LCP}_\mathcal{D}[i] < 3n$ (it follows from Fine and Wilf Theorem [59, 89]), and one can design a polynomial time algorithm to compute $\mathsf{LCP}_\mathcal{D}$, but we will not pursue this further here. In particular, each entry of $\mathsf{LCP}_\mathcal{D}$ can be stored by using $O(\log n)$ bits, and so $\mathsf{LCP}_\mathcal{D}$ can be stored by using $O(n \log n)$ bits.

Unfortunately, the array $\mathsf{LCP}_\mathcal{D}$ alone is not sufficient for computing matching statistics. Assume that $T(\pi) = \{u_r, u_{r+1}, \dots, u_{s-1}, u_s\}$, and that when we apply the forward search by adding a character c, we obtain $T(\pi c) = \emptyset$. We must then determine the largest suffix π' of $T(\pi)$ such that $T(\pi)$ is *strictly* contained in $T(\pi')$. Suppose that *every* string in I_{u_r} is prefixed by π^R, and *every* string in I_{u_s} is prefixed by π^R. In particular, both $\min_r$ and $\max_s$ are prefixed by π^R. In this case, we can proceed like in the algorithm for strings: the desired suffix π' is the one having length $\max\{\mathsf{lcp}(\max_{r-1}, \min_r), \mathsf{lcp}(\max_s, \min_{s+1})\}$, which can be determined using $\mathsf{LCP}_\mathcal{D}$. However, in general, even if *some* string in I_{u_r} must be prefixed by π^R, the string $\min_r$ need not be prefixed by π^R, and similarly $\max_s$ need not be prefixed by π^R. The worst-case scenario occurs when $r = s$. Consider Figure 8.2, and assume that $\pi = heba$. Then, we have $r = s = 3$ (note that $abeh\#\#\#\dots$ is a string in I_{u_3} prefixed by π^R). However, both $\min_3 = abdg\#\#\#\dots$, and $\max_3 = acei\#\#\#\dots$, are not prefixed by π^R. Notice that $\mathsf{lcp}(\max_2, \min_3) = 3$ and $\mathsf{lcp}(\max_3, \min_4) = 3$, but π' is not the suffix of length 3 of π. Indeed, since $\min_3$ is only prefixed by the prefix of π^R of length 2, and $\max_3$ is only prefixed by the prefix of π^R of length 1, we conclude that it must be $|\pi'| = 2$. In general, the desired suffix π' is the one having length $|\pi'|$ given by:

$$\max\Big\{ \min\{\mathsf{lcp}(\max_{r-1}, \min_r), \mathsf{lcp}(\min_r, \pi^R)\}, \min\{\mathsf{lcp}(\pi^R, \max_s), \mathsf{lcp}(\max_s, \min_{s+1})\} \Big\}. \tag{8.1}$$

The above formula shows that, in order to compute π', in addition to $\mathsf{LCP}_\mathcal{D}$ it suffices to know the values $\mathsf{lcp}(\min_r, \pi^R)$ and $\mathsf{lcp}(\pi^R, \max_s)$ (π' is a suffix of π, so it is determined by its length). We now show how our algorithm can efficiently maintain the current pattern π, the set $T(\pi) = \{u_r, u_{r+1}, \dots, u_{s-1}, u_s\}$ and the values $\mathsf{lcp}(\min_r, \pi^R)$ and $\mathsf{lcp}(\pi^R, \max_s)$ during the computation of the matching statistics. We assume that the input DFA is encoded with the rank/select data structures supporting the execution of a step of forward search in $O(\log\log|\Sigma|)$ time (the encoding is simply a special case of Theorem 4.47, see [61] for details). In addition, we will use the following result.

Lemma 8.4. *Let $A[1, n]$ be a sequence of values over an ordered alphabet Σ. Consider the following queries: (i) given $i, j \in [1..n]$, compute the minimum value in $S[i..j]$, and (ii) given $t \in [1..n]$ and $c \in \Sigma$, determine the biggest $k < t$ (or the smallest $k > t$) such that $A[k] < c$. Then, A can be augumented with a data structure of $2n + o(n)$ bits such that query (i) can be answered in constant time and query (ii) can be answered in $O(\log n)$ time.*

Proof. There exists a data structure of $2n + o(n)$ bits that allows to solve range minimum queries in constant time [60], so using A we can solve queries (i) in constant time. Now, let us show how to solve queries (ii). Let f_1 be the answer of query (i) on input $i = \lceil t/2 \rceil$ and $j = t - 1$. If $f_1 < c$, then we must keep searching in the interval $[\lceil t/2 \rceil, t - 1]$, otherwise, we must keep searching in the interval $[1, \lceil t/2 \rceil - 1]$. In other words, we can answer a query (ii) by means of a binary search on $[1, t - 1]$, which takes $O(\log t)$ (and so $O(\log n)$) time. □

Notice that query (ii) can be seen as a variant of PSV and NSV queries. In the following, we assume that the array $\mathsf{LCP}_\mathcal{D}$ has been augmented with the data structure of Lemma 8.4.

At the beginning we have $\pi = \varepsilon$, so $T(\varepsilon) = \{1, 2, \dots, n\}$ and trivially $\mathsf{lcp}(\min_r, \pi^R) = \mathsf{lcp}(\pi^R, \max_s) = 0$. At each iteration we perform a step of forward search computing

$T(\pi c)$ given $T(\pi)$; then we distinguish two cases according to whether $T(\pi c)$ is empty or not.

Case 1. $T(\pi c) = \{u_{r'}, u_{r'+1}, \dots, u_{s'-1}, u_{s'}\}$ is not empty. In that case πc will become the pattern at the next iteration. Since we already have $T(\pi c)$ we are left with the task of computing $\mathsf{lcp}(\min_{r'}, c\pi^R)$ and $\mathsf{lcp}(c\pi^R, \max_{s'})$. We only show how to compute $\mathsf{lcp}(\min_{r'}, c\pi^R)$, the latter computation being analogous. Let k be the smallest integer in $[1, n]$ such that $(u_k, u_{r'}) \in E$. Notice that we can easily compute k by means of standard rank/select operations on the compact data structure used to encode $\mathcal{D}$. Since $u_{r'} \in T(\pi c)$, it must be $k \leq s$. Moreover, the characterization of $\min_{r'}$ that we described above implies that $\min_{r'} = c \min_k$, hence $\mathsf{lcp}(\min_{r'}, c\pi^R) = \mathsf{lcp}(c \min_k, c\pi^R) = 1 + \mathsf{lcp}(\min_k, \pi^R)$. To compute $\mathsf{lcp}(\min_k, \pi^R)$ we distinguish two subcases:

a) $k > r$, hence $r < k \leq s$. Since $u_r, u_s \in T(\pi)$, there exist $\alpha \in I_{u_r}$ and $\beta \in I_{u_s}$ both prefixed by π^R. But $\alpha \preceq \max_r \preceq \min_k \preceq \min_s \preceq \beta$, so $\min_k$ is also prefixed by π^R, and we conclude $\mathsf{lcp}(\min_k, \pi^R) = |\pi|$.

b) $k \leq r$. In this case, we have $\min_k \preceq \max_k \preceq \min_{k+1} \prec \max_{k+1} \preceq \cdots \preceq \min_r \prec \pi^R$, and therefore $\mathsf{lcp}(\min_k, \pi^R)$ is equal to

$$\min\{\mathsf{lcp}(\min_k, \max_k), \mathsf{lcp}(\max_k, \min_{k+1}), \mathsf{lcp}(\min_{k+1}, \max_{k+1}), \dots, \mathsf{lcp}(\min_r, \pi^R)\}.$$

With the above formula we can compute $\mathsf{lcp}(\min_k, \pi^R)$ using query (i) of Lemma 8.4 over the range $\mathsf{LCP}_{\mathcal{D}}[2k, 2r-1]$ and the value $\mathsf{lcp}(\min_r, \pi^R)$.

Case 2. $T(\pi c)$ is empty. In this case at the next iteration the pattern will be largest suffix π' of π such that $T(\pi)$ is *strictly* contained in $T(\pi') = \{u_{r''}, \dots, u_{s''}\}$. We compute $|\pi'|$ using (8.1); if $|\pi'| > \mathsf{lcp}(\min_r, \pi^R)$ we set $r'' = r$, otherwise we apply query (ii) of Lemma 8.4 to find the rightmost entry r'' in $\mathsf{LCP}_{\mathcal{D}}[2, 2r-1]$ smaller than $|\pi'|$. Computing s'' is analogous.

Given $T(\pi') = \{u_{r''}, u_{r''+1}, \dots, u_{s''-1}, u_{s''}\}$, where $r'' \leq r$, $s \leq s''$, and at least one inequality is strict, we want to compute $\mathsf{lcp}(\min_{r''}, (\pi')^R)$ and $\mathsf{lcp}((\pi')^R, \max_{s''})$. We only consider $\mathsf{lcp}(\min_{r''}, (\pi')^R)$, the latter computation being analogous. We distinguish two subcases:

a) $r'' = r$. Then $\mathsf{lcp}(\min_{r''}, (\pi')^R) = \mathsf{lcp}(\min_r, (\pi')^R) = \min\{\mathsf{lcp}(\min_r, \pi^R), |\pi'|\}$.

b) $r'' < r$. In particular, since $u_{r''}$ is the left endpoint of $T(\pi')$ and $|T(\pi')| \geq 2$, one can prove like in Case 1*a*) that $\max_{r''}$ is prefixed by $(\pi')^R$. We immediately conclude that $\mathsf{lcp}(\min_{r''}, (\pi')^R) = \min\{\mathsf{lcp}(\min_{r''}, \max_{r''}), |\pi'|\}$, which can be immediately computed since $\mathsf{lcp}(\min_{r''}, \max_{r''})$ is a value stored in $\mathsf{LCP}_{\mathcal{D}}$.

8.3 A Space-Time Trade-Off for the LCP Array

Theorem 8.1 states that, in order to efficiently compute matching statistics, we need to augment the compact representation of a Wheeler DFA with the LCP array, which requires $O(n \log n)$ bits. One drawback is that storing the LCP array often requires less space. In a DFA, the number e of edges is at most σn, so from Theorem 4.47 we obtain that storing a Wheeler DFA requires $O(e \log \sigma) = O(n\sigma \log \sigma)$ bits. If the alphabet is small — that is, if $\sigma \log \sigma = o(\log n)$ — then the number of required bits is $o(n \log n)$; if $\sigma = O(1)$, then the number of required bits is $O(n)$. The latter case is especially relevant in practice, because de Bruijn graphs [21] are the prototypes of Wheeler graphs, and in bioinformatics de Bruijn graphs are defined over the constant-size alphabet $\Sigma = \{A, C, G, T\}$.

Let us show that we can *sample* entries of the LCP array in such a way that, by storing only a linear number of additional bits on top of the Wheeler graph, we can compute each entry of the LCP array in logarithmic time, thus providing a space-time trade-off. More precisely:

Theorem 8.5. *We can augment the compact representation of a Wheeler DFA $\mathcal{A}$ with $O(n)$ bits ($O(n \log\log \sigma)$ bits, respectively), where n is the number of states and σ is the size of the alphabet, in such a way that we can compute each entry of the LCP array of $\mathcal{A}$ in $O(\log n \log\log \sigma)$ time ($O(\log n)$ time, respectively).*

From Theorem 8.1 and Theorem 8.5 we obtain the following space-time tradeoff for computing matching statistics.

Theorem 8.6. *We can augment the compact representation of a Wheeler DFA $\mathcal{A}$ with $O(n \log\log \sigma)$ bits, where n is the number of states and σ is the size of the alphabet, in such a way that we can compute the matching statistics of a pattern of length m with respect to the Wheeler DFA in $O(m \log^2 n)$ time.*

Proof. Let us recall how the bounds in Theorem 8.1 are obtained. The space bound is $O(n \log n)$ bits because we need to store $\mathsf{LCP}_{\mathcal{D}}$. We also store a data structure to solve range minimum queries on $\mathsf{LCP}_{\mathcal{D}}$, which only takes $O(n)$ bits. The time bound $O(m \log n)$ follows from performing $O(m)$ steps to compute all matching statistics. In each of these $O(m)$ steps, we may need to perform a binary search on $\mathsf{LCP}_{\mathcal{D}}$. In each step of the binary search, we need to solve a range minimum query once and we need to access $\mathsf{LCP}_{\mathcal{D}}$ once, so the binary search takes $O(\log n)$ time per step. By Theorem 8.5, if we store only $O(n \log\log \sigma)$ bits, we can access $\mathsf{LCP}_{\mathcal{D}}$ in $O(\log n)$ time, so the time for the binary search becomes $O(\log^2 n)$ per step and Theorem 8.6 follows. $\square$

In the remainder of the section we prove Theorem 8.5. The sampling mechanism is obtained by conveniently defining an auxiliary graph from the entries of the LCP array.

Let $G = (V, H)$ be a finite (unlabeled) directed graph such that every node has at most one incoming edge. For every $v \in V$ and for every $i \geq 0$, there exists at most one node $v' \in V$ such that there exists a directed path from v' to v having i edges; if v' exists, we denote it by $v(i)$. Fix a parameter $h \geq 1$. Let us prove that there exists $V(h) \subseteq V$ such that (i) $|V(h)| \leq \frac{|V|}{h}$ and (ii) for every $v \in V$ there exists $0 \leq i \leq 2h-2$ such that $v(i)$ is defined and either $v(i) \in V(h)$ or $v(i)$ has no incoming edges or $v(i) = v(j)$ for some $0 \leq j < i$. We build $V(h)$ incrementally following Algorithm 2. Let us prove that, at the end of the algorithm, properties (i) and (ii) are true. For every $v \in V(h)$, define $S_v = \{v, v(1), v(2), \dots, v(h-1)\}$, which is possible because by construction if $v \in V(h)$, then $v(i)$ is defined for every $0 \leq i \leq h-1$. It must be $v(i) \neq v(j)$ for $0 \leq i < j \leq h-1$, so $|S_v| = h$. If $v, v' \in V(h)$ and $v \neq v'$, then by construction S_v and $S_{v'}$ are disjoint. As a consequence, $|V| \geq \sum_{v \in V(h)} |S_v| = \sum_{v \in V(h)} h = h|V_h|$ and so $|V_h| \leq \frac{|V|}{h}$, which proves property (i). Let us prove property (ii). Pick $v \in V$; we must prove that there exists $0 \leq i \leq 2h-2$ such that $v(i)$ is defined and either $v(i) \in V(h)$ or $v(i)$ has no incoming edges or $v(i) = v(j)$ for some $0 \leq j < i$. We distinguish three cases:

1. there exists i with $1 \leq i \leq h-1$ such that $v(i-1)$ is defined but $v(i)$ is not defined. Then, $v(i-1)$ has no incoming edges.

2. there exist i, j with $0 \leq j < i \leq h-1$ such that $v(j)$ and $v(i)$ are defined and

Algorithm 2 Building $V(h)$

```
$V(h) \leftarrow \emptyset$
$U \leftarrow \emptyset$
while there exists $v \in V$ such that (a) $v(i)$ is defined for $0 \leq i \leq h-1$, (b) $v(i) \neq v(j)$ for $0 \leq j < i \leq h-1$, (c) $v(i) \notin U$ for $0 \leq i \leq h-1$ do
    Pick such a $v$, add $v(h-1)$ to $V(h)$ and add $v(i)$ to $U$ for every $0 \leq i \leq h-1$
end while
```

$v(i) = v(j)$. In this case, the conclusion is immediate.

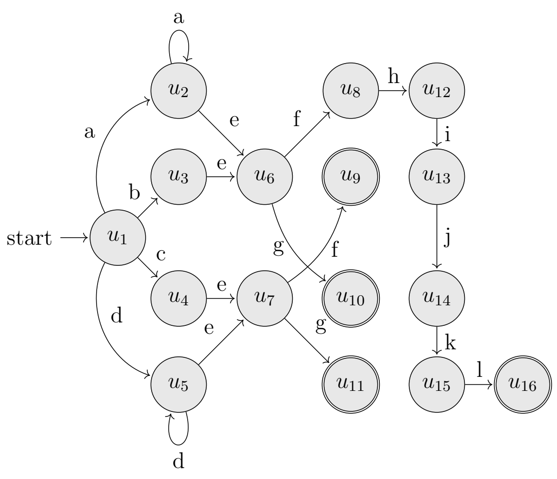
3. $v(i)$ is defined for every $0 \leq i \leq h$ and $v(i) \neq v(j)$ for $0 \leq j < i \leq h-1$. Since Algorithm 2 has terminated, then there exists $0 \leq j \leq h-1$ such that $v(j) \in U$. The construction of U implies that there exists $v' \in V$ and $0 \leq j' \leq h-1$ such that $v(j) = v'(j')$ and $v'(h-1) \in V(h)$. As a consequence $v(h-1+j-j') = v(j)(h-1-j') = (v'(j'))(h-1-j') = v'(h-1) \in V(h)$. Since $j \leq h-1$ and $j' \geq 0$, we conclude $h-1+j-j' \leq 2h-2$ and we are done.

Now, let us prove Theorem 8.5. For every $2 \leq i \leq j \leq 2n$, let $RMQ_{\mathsf{LCP}_{\mathcal{D}}}(i,j)$ be the position of a minimum in $\mathsf{LCP}_{\mathcal{D}}[i,j]$ (a range minimum query); we can store a data structure of $O(n)$ bits that solves queries $RMQ_{\mathsf{LCP}_{\mathcal{D}}}(i,j)$ *without accessing* $\mathsf{LCP}_{\mathcal{D}}$ [60]. Moreover, for every $1 \leq i \leq n$, let $p_{\min}(i)$ be the smallest $1 \leq i' \leq n$ such that $u_i \in \delta(u_{i'}, \min_{\lambda(u_i)})$ and let $p_{\max}(i)$ be the largest $1 \leq i'' \leq n$ such that $u_i \in \delta(u_{i''}, \max_{\lambda(u_i)})$. The compact data structure storing the Wheeler DFA allows computing $p_{\min}(i)$ and $p_{\max}(i)$ in $O(\log\log\sigma)$ time, for every $1 \leq i \leq n$.

Consider the entry $\mathsf{LCP}_{\mathcal{D}}[2i-1] = \mathsf{lcp}(\max_{i-1}, \min_i)$, for $2 \leq i \leq n$, and assume that $\mathsf{LCP}_{\mathcal{D}}[2i-1] \geq 1$. Let $k = p_{\max}(i-1)$ and $k' = p_{\min}(i)$. Since $\mathsf{LCP}_{\mathcal{D}}[2i-1] \geq 1$, then there exists $a \in \Sigma$ such that $\max_{i-1} = a\max_k$ and $\min_{i-1} = a\min_{k'}$. In particular, $(u_k, u_{i-1}, a) \in E$ and $(u_{k'}, u_i, a) \in E$, so from Axiom 2 we obtain $k < k'$. Moreover, we have $\mathsf{LCP}_{\mathcal{D}}[2i-1] = \mathsf{lcp}(\max_{i-1}, \min_i) = \mathsf{lcp}(a\max_k, a\min_{k'}) = 1 + \mathsf{lcp}(\max_k, \min_{k'})$. Notice that:

$$\begin{aligned}
&\mathsf{lcp}(\max_k, \min_{k'}) = \min\{\mathsf{lcp}(\max_k, \min_{k+1}), \mathsf{lcp}(\min_{k+1}, \max_{k+1}), \dots, \\
&= \mathsf{lcp}(\min_{k'-1}, \max_{k'-1}), \mathsf{lcp}(\max_{k'-1}, \min_{k'})\} = \\
&= \min\{\mathsf{LCP}_{\mathcal{D}}[2k+1], \mathsf{LCP}_{\mathcal{D}}[2k+2], \dots, \mathsf{LCP}_{\mathcal{D}}[2k'-2], \mathsf{LCP}_{\mathcal{D}}[2k'-1]\}.
\end{aligned}$$

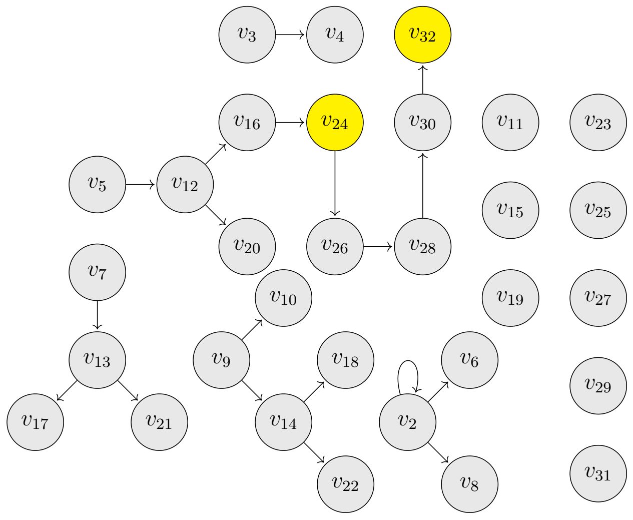
Let $j = RMQ_{\mathsf{LCP}_{\mathcal{D}}}(2k+1, 2k'-1)$. Then, $\mathsf{LCP}_{\mathcal{D}}[j] = \min\{\mathsf{LCP}_{\mathcal{D}}[2k+1], \mathsf{LCP}_{\mathcal{D}}[2k+2], \dots, \mathsf{LCP}_{\mathcal{D}}[2k'-2], \mathsf{LCP}_{\mathcal{D}}[2k'-1]\}$, so $\mathsf{LCP}_{\mathcal{D}}[2i-1] = 1 + \mathsf{LCP}_{\mathcal{D}}[j]$ (we assume $t + \infty =$



(a)

State	i	$\mathsf{LCP}_{\mathcal{D}}[i]$	k	k'	$\mathcal{R}(i)$
1	1				
	2	∞	1	1	2
2	3	0	-	-	-
	4	1	1	2	3
3	5	0	-	-	-
	6	∞	1	1	2
4	7	0	-	-	-
	8	∞	1	1	2
5	9	0	-	-	-
	10	1	1	5	9
6	11	0	-	-	-
	12	1	2	3	5
7	13	1	3	4	7
	14	1	4	5	9
8	15	0	-	-	-
	16	2	6	6	12
9	17	2	6	7	13
	18	2	7	7	14
10	19	0	-	-	-
	20	2	6	6	12
11	21	2	6	7	13
	22	2	7	7	14
12	23	0	-	-	-
	24	3	8	8	16
13	25	0	-	-	-
	26	4	12	12	24
14	27	0	-	-	-
	28	5	13	13	26
15	29	0	-	-	-
	30	6	14	14	28
16	31	0	-	-	-
	32	7	15	15	30

(b)



(c)

i	$C[i]$	$\mathsf{LCP}^*_{\mathcal{D}}$
1		3
2	0	7
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	
16	0	
17	0	
18	0	
19	0	
20	0	
21	0	
22	0	
23	0	
24	1	
25	0	
26	0	
27	0	
28	0	
29	0	
30	0	
31	0	
32	1	

(d)

Figure 8.3: (a) A Wheeler DFA. States are numbered according to the Wheeler order. (b) The array $\mathsf{LCP}_{\mathcal{D}}$, and the values needed to compute $G = (V, H)$. We assume that a range minimum query returns the *largest* position of a minimum value. (c) The graph $G = (V, H)$, with $V(\lceil \log n \rceil) = V(4) = \{v_{24}, v_{32}\}$ (yellow states). (d) The data structures that we store.

Algorithm 3 Input: $h \in [2, 2n]$. Output: $\mathsf{LCP}_{\mathcal{D}}[h]$.

```
procedure MAIN_FUNCTION($h$)
    Initialize a global bit array $D[2, 2n]$ to zero        ▷ $D[2, 2n]$ marks the entries already considered
    return LCP($h$)
end procedure

procedure LCP($h$)
    $D[h] \leftarrow 1$
    if $C[h] = 1$ then        ▷ The desired value has been sampled
        return $\mathsf{LCP}^*_{\mathcal{A}}[rank(C, h)]$
    else if $h$ is odd then
        $i \leftarrow \lceil h/2 \rceil$
        if $\max_{\lambda(u_{i-1})} \prec \min_{\lambda(u_i)}$ then
            return 0
        else
            $k \leftarrow p_{\max}(i-1)$
            $k' \leftarrow p_{\min}(i)$
            $j \leftarrow RMQ_{\mathsf{LCP}_{\mathcal{D}}}(2k+1, 2k'-1)$
            if $D[j] = 1$ then        ▷ We have already considered this entry before, so there is a cycle
                return $\infty$
            else
                return $1 +$ LCP($j$)
            end if
        end if
    else
        $i \leftarrow h/2$
        if $\min_{\lambda(u_i)} \prec \max_{\lambda(u_i)}$ then
            return 0
        else
            $k \leftarrow p_{\min}(i)$
            $k' \leftarrow p_{\max}(i)$
            $j \leftarrow RMQ_{\mathsf{LCP}_{\mathcal{D}}}(2k, 2k')$
            if $D[j] = 1$ then        ▷ We have already considered this entry before, so there is a cycle
                return $\infty$
            else
                return $1 +$ LCP($j$)
            end if
        end if
    end if
end procedure
```

∞ for every $t \geq 0$), and we have reduced the problem of computing $\mathsf{LCP}_{\mathcal{D}}[2i-1]$ to the problem of computing $\mathsf{LCP}_{\mathcal{D}}[j]$. In the following, let $\mathcal{R}(2i-1) = j$. Given $2 \leq i \leq n$, we can compute $j = \mathcal{R}(2i-1)$ in $O(\log\log\sigma)$ time, because we can compute $k = p_{\max}(i-1)$ and $k' = p_{\min}(i)$ in $O(\log\log\sigma)$ time and we can compute j

in $O(1)$ time by means of a range minimum query.

We proceed analogously with the entries $\mathsf{LCP}_{\mathcal{D}}[2i] = \mathsf{lcp}(\min_i, \max_i)$, for $1 \leq i \leq n$ (assuming that $\mathsf{LCP}_{\mathcal{D}}[2i] \geq 1$). Let $k = p_{\min}(i)$ and $k' = p_{\max}(i)$; by the definitions of $p_{\min}$ and $p_{\max}$ it must be $k \leq k'$. Hence, $\mathsf{LCP}_{\mathcal{D}}[2i] = 1 + \mathsf{lcp}(\min_k, \max_{k'})$ and similarly $\mathsf{lcp}(\min_k, \max_{k'}) = \min\{\mathsf{LCP}_{\mathcal{D}}[2k], \mathsf{LCP}_{\mathcal{D}}[2k+1], \dots, \mathsf{LCP}_{\mathcal{D}}[2k' - 1], \mathsf{LCP}_{\mathcal{D}}[2k']\}$. Let $j = RMQ_{\mathsf{LCP}_{\mathcal{D}}}(2k, 2k')$. In the following, let $\mathcal{R}(2i) = j$. Given $1 \leq i \leq n$, we can compute $j = \mathcal{R}(2i)$ in $O(\log\log\sigma)$ time. See Figure 8.3 for an example.

Now, consider the (unlabeled) directed graph $G = (V, H)$ defined as follows. Let V be a set of $2n-1$ nodes v_2, v_3, $\dots$, v_{2n}. Moreover, $v_i \in V$ has no incoming edge in G if $\mathcal{R}(i)$ is not defined, which happens if $\mathsf{LCP}_{\mathcal{D}}[i] = 0$ (and so i is odd and $\lambda(u_{i-1}) \neq \lambda(u_i)$); $v_i \in V$ has exactly one incoming edge if $\mathcal{R}(i)$ is defined, namely, $(v_{\mathcal{R}(i)}, v_i)$. Note that v_{2i} has an incoming edge for every $1 \leq i \leq n$. Let $h \geq 1$ be a parameter. We know that there exists $V(h) \subseteq V$ such that (i) $|V(h)| \leq \frac{|V|}{h}$ and (ii) for every $v_i \in V$ there exists $0 \leq k \leq 2h-2$ such that $v_i(k)$ is defined and either $v_i(k) \in V(h)$ or $v_i(k)$ has no incoming edges or $v_i(k) = v_i(l)$ for some $0 \leq l < k$. Notice that if $v_i(k) = v_i(l)$ for some $0 \leq l < k$, then $\mathsf{LCP}_{\mathcal{D}}[i] = \infty$ (because there is a cycle and so $v_i(k')$ is defined for every $k' \geq 0$). Let $n' = |V(h)|$, and let $\mathsf{LCP}^*_{\mathcal{A}}[1, n']$ an array storing the value $\mathsf{LCP}_{\mathcal{D}}[i]$ for each $v_i \in V(h)$, sorted by increasing i. Since $n' \leq \frac{|V|}{h} = \frac{2n-1}{h}$, storing $\mathsf{LCP}^*_{\mathcal{A}}[1, n']$ takes $n'O(\log n) = O(\frac{n\log n}{h})$ bits. We store a bitvector $C[2, 2n]$ such that $C[i] = 1$ if and only if $v_i \in V(h)$ for every $2 \leq i \leq 2n$; we augment C with $o(n)$ bits so that it supports rank queries in $O(1)$ time. For every $2 \leq i \leq 2n$, in $O(1)$ time we can check whether $\mathsf{LCP}_{\mathcal{D}}[i]$ has been stored in $\mathsf{LCP}^*_{\mathcal{A}}$ by checking whether $C[i] = 1$, and if $C[i] = 1$ it must be $\mathsf{LCP}_{\mathcal{D}}[i] = \mathsf{LCP}^*_{\mathcal{A}}[rank(C, i)]$.

From our discussion, it follows that Algorithm 3 correctly computes $\mathsf{LCP}_{\mathcal{D}}[i]$ for every $2 \leq i \leq n$. Property (ii) ensures that the function lcp is called at most h times. Every call requires $O(\log\log\sigma)$ time, so the running time of our algorithm is $O(h\log\log\sigma)$ (the initialization of $D[2, 2n]$ in Algorithm 3 can be simulated in $O(1)$ time [95]). We conclude that we store $O(n + \frac{n\log n}{h})$ bits, and in $O(h\log\log\sigma)$ time we can compute $\mathsf{LCP}_{\mathcal{D}}[i]$ for every $2 \leq i \leq n$.

By choosing $h = \lceil\frac{\log n}{\log\log\sigma}\rceil$, we conclude that our data structure can be stored using $O(n\log\log\sigma)$ bits and it allows to compute $\mathsf{LCP}_{\mathcal{D}}[i]$ for every $2 \leq i \leq n$ in $O(\log n)$ time. By choosing $h = \lceil\log n\rceil$ we conclude that our data structure can be stored using $O(n)$ bits and it allows to compute $\mathsf{LCP}_{\mathcal{D}}[i]$ for every $2 \leq i \leq n$ in $O(\log n \log\log \sigma)$ time, thus proving Theorem 8.5.

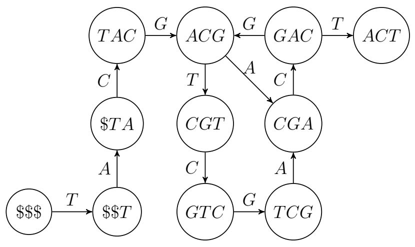
8.4 An Application: Variable-order de Bruijn Graphs

Let us show an application of our sampling mechanism for the LCP array (Theorem 8.5). Let $k \geq 0$ be a parameter, and let $\mathcal{S}$ be a set of strings on the alphabet $\Sigma = \{A, C, G, T\}$ (in this application we always assume $\sigma = O(1)$). The k-th order de Bruijn graph of $\mathcal{S}$ is defined as follows. The set of nodes is the set of all strings of Σ of length k that occur as a substring of some string in $\mathcal{S}$. There is an edge from node α to node β labeled $c \in \Sigma$ if and only if (i) the suffix of α of length $k-1$ is equal to the prefix of β of length $k-1$ and (ii) the last character of β is c. If some node α has no incoming edges, then we add nodes $\$^i\alpha_{k-i}$ for $1 \leq i \leq k$, where α_j is the prefix of α of length j and \$ is a special character, and we add edges as above; see Figure 8.4 for an example.

Wheeler DFAs are a generalization of de Bruijn graphs (we do not need to define an initial state and a set of final states, because here we are not interested in studying the applications of de Bruijn graphs and Wheeler automata to automata theory); the Wheeler order is the one such that node α comes before node β if and only if the string α^R is lexicographically smaller than the string β^R [61]. In particular, the compact representation of a Wheeler graph is a generalization of the BOSS representation of a de Bruijn graph [21], and our results on the LCP array also apply to a de Bruijn graph.

Notice that, in a k-th order de Bruijn graph G, all strings that can be read from node α by following edges in a backward fashion start with α^R (as usual, we assume that node \$\$\$ has a self-loop labeled \$). As a consequence, it holds $\mathsf{LCP}_G[2i] \geq k$ for every $1 \leq i \leq n$ and $\mathsf{LCP}_G[2i-1] \leq k-1$ for every $2 \leq i \leq n$ (so any value in an odd entry is smaller than any value in an even entry).

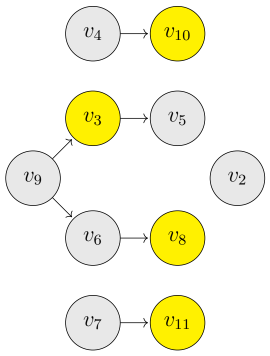
Many assemblers [11, 100, 84, 113] consider all k-mers occurring in a set of reads and build a k-th order de Bruijn graph (on the alphabet $\Sigma = \{A, C, G, T\}$) to perform Eulerian sequence assembly [75, 101]. However, the choice of the parameter k impacts the assembly quality, so some assemblers try several choices for k [11, 100], which slows down the process because several de Bruijn graphs need to be built. In [20]



(a)

i	Node	$\overline{\mathsf{LCP}}_G[i]$	k	k'	$\mathcal{R}(i)$
1	$$$				
2	CGA	0	-	-	-
3	$TA	1	9	9	9
4	GAC	0	-	-	-
5	TAC	2	3	3	3
6	GTC	1	4	11	9
7	ACG	0	-	-	-
8	TCG	2	6	6	6
9	$$T	0	-	-	-
10	ACT	1	2	4	4
11	CGT	1	6	7	7

(b)



(c)

i	$C[i]$	$\overline{\mathsf{LCP}}^*_G$
1		1
2	0	2
3	1	1
4	0	1
5	0	
6	0	
7	0	
8	1	
9	0	
10	1	
11	1	

(d)

Figure 8.4: The 3-rd order de Bruijn graph for the set $\mathcal{S} = \{CGAC, GACG, GACT, TACG, GTCG, ACGA, ACGT, TCGA, CGTC\}$ from [20]. We proceed like in Figure 8.3 (now we only consider odd entries of LCP_G, and $h = \lceil \log k \rceil = 2$).

it was shown that the k-order de Bruijn graph of $\mathcal{S}$ can be used to *implicitly* store the k'-th order de Bruijn graph of $\mathcal{S}$ for *every* $k' \leq k$, thus leading to a *variable-order de Bruijn graph*. The challenge is to navigate this implicit representation (that is, how to follow edges in a forward or backward fashion). In [20], it was shown that the navigation is possible by storing or by simulating an array $\overline{\mathsf{LCP}}_G$ which can be seen as a simplification of the LCP array of the Wheeler graph G. More precisely, the navigation of a variable-order de Bruijn graph is possible by storing or by simulating the values in the odd entries of the LCP array. Formally, we define $\overline{\mathsf{LCP}}_G[i] = \mathsf{LCP}_G[2i-1]$ for every $2 \leq i \leq n$; see Figure 8.4. Note that $\overline{\mathsf{LCP}}_G[i] \leq k-1$ for every $2 \leq i \leq n$, so $\overline{\mathsf{LCP}}_G$ can be stored by using $O(n \log k)$ bits. Notice that Theorem 8.5 also applies to $\overline{\mathsf{LCP}}_G[i]$ (we do not need to store values in the even entries because a value in an odd entry is smaller than a value in an even entry, so even entries are never selected in the sampling process when answering a range minimum query on LCP_G). Then, we have the following result (see [20]; we assume $\sigma = O(1)$).

Theorem 8.7. 1. *We can augment the BOSS representation of a k-th order de Bruijn graph with $O(n \log k)$ bits, where n is the number of nodes, so that the underlying variable-order de Bruijn graph can be navigated in $O(\log k)$ time per visited node.*

2. *We can augment the BOSS representation of a k-th order de Bruijn graph with $O(n)$ bits, where n is the number of nodes, so that the underlying variable-order de Bruijn graph can be navigated in $O(k \log n)$ time per visited node.*

Essentially, the first solution in Theorem 8.7 explicitly stores $\overline{\mathsf{LCP}}_G$, while the second solution in Theorem 8.7 computes the entries of $\overline{\mathsf{LCP}}_G$ by exploiting the BOSS representation. In general, a big k (close to the size of the reads) allows to retrieve the expressive power on an overlap graph [44], so in Theorem 8.7 we cannot assume that k is small. On the one hand, the *space* required for the first solution can be too large, because a de Bruijn graph can be stored by using only $O(n)$ bits. On the other hand, the *time* bound in the second solution increases substantially. We can now improve the second solution by providing a data structure that achieves the best of both worlds.

Theorem 8.8. *We can augment the BOSS representation of a k-th order de Bruijn graph with $O(n)$ bits, where n is the number of nodes, so that the underlying variable-order de Bruijn graph can be navigated in $O(\log k \log n)$ time per visited node.*

As in the proof of Theorem 8.5, we can conveniently sample some entries of $\overline{\mathsf{LCP}}_G$. However, we can now choose a better parameter $h \geq 1$ in our sampling process. Indeed, each entry of $\overline{\mathsf{LCP}}_G$ can be stored by using $O(\log k)$ bits (not $O(\log n)$ bits), so if we choose $h = \lceil \log k \rceil$, we conclude that we can augment the BOSS representation of a de Bruijn graph with $O(n)$ bits such that for every $2 \leq i \leq n$ we can compute $\overline{\mathsf{LCP}}_G[i]$ in $O(\log k)$ time.

The first solution in Theorem 8.7 consists in storing a wavelet tree on $\overline{\mathsf{LCP}}_G$, which requires $O(n \log k)$ bits and allows to navigate the graph in $O(\log k)$ time per visited node. The second solution in Theorem 8.7 does not store $\overline{\mathsf{LCP}}_G$ at all; whenever needed, an entry of $\overline{\mathsf{LCP}}_G$ is computed in $O(k)$ time by exploiting the

BOSS representation of the de Bruijn graph. The second solution only stores a data structures of $O(n)$ bits to solve range minimum queries. The details can be found in [20]. Essentially, the time bound $O(k \log n)$ comes from performing binary searches on $\overline{\mathsf{LCP}}_G$ while explicitly computing an entry of $\overline{\mathsf{LCP}}_G$ at each step in $O(k)$ time. However, we have seen that, while staying within the $O(n)$ space bound, we can augment the BOSS representation so that we can compute the entries of $\overline{\mathsf{LCP}}_G$ in $O(\log k)$ time, so the time bound $O(k \log n)$ becomes $O(\log k \log n)$, which implies Theorem 8.8.

Chapter 9

Conclusion

In the previous chapter, we mentioned some open problems. For example, in Chapter 7 we described a recursive algorithm for building the maximum co-lex order more efficiently, but we still do not know whether is possible to reach linear time. It looks even more challenging to understand whether the maximum co-lex *relation* can be determined in subquadratic time (see Theorem 6.17). From a theoretical perspective, we do not whether the Hasse automaton can be effectively built (see Section 5.4), and we do not know whether the paradigm that we presented in this thesis can be extended to other formalisms (such as ω-regular languages and tree languages). Our results can also have didactic implications: Chapter 3 suggests that the Burrows-Wheeler Transform of a string can be introduced more naturally if one interprets a string as a graph.

In the next sections, we briefly sketch how our results can find applications in different areas, and we outline some partial results. We remark that before the PhD defense the results sketched in Section 9.1 were accepted for publication [37].

9.1 Generalized Automata

In his monumental work [47] on automata theory (which dates back to 1974), Eilenberg proposed a natural generalization of NFAs where edges can be labeled not only with characters but with (possibly empty) finite strings, the so-called *generalized non-deterministic finite automata (GNFAs)*, which can represent regular languages more concisely than classical automata. The problem of studying the notion of determinism in the setting of generalized automata was approached by Giammaresi and Montalbano [64, 63], who defined *generalized deterministic finite automata (GDFAs)* who showed that there can exist two or more non-isomorphic state-minimal GDFAs recognizing a given regular language. The non-uniqueness of a state-minimal GDFA seems to imply a major difference in the behavior of generalized automata compared to conventional automata, so it looks like there is no hope of deriving a structural result like the Myhill-Nerode theorem in the model of the generalized automata. We have some results showing that, in fact, the lack of uniqueness can be explained by introducing a set of strings $\mathcal{W}$. Once we fix a language $\mathcal{L}$ and a set $\mathcal{W}$, we can retrieve a full Myhill-Nerode theorem; in particular, our results contain the textbook Myhill-Nerode theorem as a degenerate case, because $\mathcal{W}$ must necessarily be equal to Pref($\mathcal{L}$) if we consider conventional automata. The set $\mathcal{W}$ is also the starting point for extending the results in Chapter 4 (Suffix array, Burrows-Wheeler Transform and FM-index) from conventional automata to generalized automata. In fact, generalized automata also play a role in data compression. It is common to consider edge-labeled graphs where one compresses unary paths in the graph to save space and the path is replaced by a single edge labeled with the concatenation of all labels. For example, some common data structures that are stored using this mechanism are Patricia trees, suffix trees and pangenomes [96, 8, 93].

9.2 Suffix Trees of Automata

In Chapter 8, we presented some results that go in the direction of extending the full functionality of suffix trees to automata. To this end, we are expected to face three challenges:

1. We need to extend the results from Wheeler DFAs to arbitrary automata (all results in Chapter 8 only apply to Wheeler DFAs)

2. We need to improve running times (the algorithm for finding matching statistics on Wheeler DFA is less efficient than the algorithm for finding matching statistics with respect to a string).

3. We need to support all suffix trees query (finding matching statistics on Wheeler DFAs only requires supporting the *parent* operation in a suffix tree, but a fully-functional suffix tree should also support other operations such as *child*, *suffix link*, *Weiner link* and so on [96]).

We have some preliminary results. In Theorem 8.1, we saw that we can find matching statistics on a Wheeler DFA in $O(m \log n)$ time, where n is the number of states and

m is the length of the pattern. Our results show that, in fact, it is possible to find matching statistics on *arbitrary Wheeler NFAs* in $O(m \log\log \sigma)$ time, where σ is the size of the alphabet. In particular, we can extend the notion of LCP array to arbitrary Wheeler NFAs. We believe that, by suitably defining the LCP array for an arbitrary NFA, it is possible to compute matching statistics on arbitrary NFAs within the same time bound of Theorem 4.47. This is similar to what happens on strings: the suffix array of a string supports exact pattern matching, and the suffix trees of a string supports several variants of the pattern matching problems (including computing matching statistics) as efficiently as exact pattern matching.

9.3 Regular Expressions

The class of Wheeler languages admits a number of remarkable properties: (i) non-determinism and determinism have the same expressive power, (ii) every Wheeler language is recognized by a minimum DFA and (iii) there exists an algebraic characterization of Wheeler languages in terms of convex equivalence relations [5]. However, we still do not have a characterization in terms of regular expressions: in other words, we do not have a Kleene theorem for Wheeler languages [74]. The main challenge seems to be that Wheeler languages are not closed under concatenation due to their co-lexicographic structure [5]. More generally, we do not know how to define regular expressions for each level of the hierarchy of regular languages that we introduced in this thesis. Since our hierarchy captures all regular languages (and we saw in the introduction that in the literature there are only few parameterizations of the class of regular languages), a characterization in terms of regular expressions could shed new light on the generalized star-height problem, which is probably the most famous open problem in formal language theory.

Appendix A

Partitions and Orders

This chapter is devoted to the proof of Theorem 5.14 and to other useful properties of entangled convex sets. All these results follow from general results valid for arbitrary total orders and partitions. From now on, we fix a total order $(Z, \leq)$ and a finite partition $\mathcal{P} = \{P_1, \ldots, P_m\}$ of Z. We first give a notion of entanglement, with respect to $\mathcal{P}$, for subsets $X \subseteq Z$. The main result of this section, Theorem A.5, states that there always exists a finite, *ordered* partition $\mathcal{V}$ of Z composed of entangled convex sets.

It is convenient to think of the elements of $\mathcal{P}$ as letters of an alphabet, forming finite or infinite strings while labelling element of Z. A finite string $P_1 \ldots P_k \in \mathcal{P}^*$ is said to be *generated* by $X \subseteq Z$, if there exists a sequence $x_1 \leq \cdots \leq x_k$ of elements in X such that $x_j \in P_j$, for all $j = 1, \ldots, k$. We also say that $P_1 \ldots P_k$ *occurs in* X *at* $x_1, \ldots, x_k$. Similarly, an infinite string $P_1 \ldots P_k \cdots \in \mathcal{P}^\omega$ is generated by $X \subseteq Z$ if there exists a monotone sequence $(x_i)_{i \in \mathbb{N}}$ of elements in X such that $x_j \in P_j$, for all $j \in \mathbb{N}$. Notice that if X is a finite set, then there exists an index i_0 such that for every $i \geq i_0$ it holds $P_i = P_{i_0}$. We can now re-state the notion of entanglement in this, more general, context.

Definition A.1. Let $(Z, \leq)$ be a total order, let $\mathcal{P}$ be a partition of Z, and let $X \subseteq Z$.

1. We define $\mathcal{P}_X = \{P \in \mathcal{P} : P \cap X \neq \emptyset\}$.

2. If $\mathcal{P}' = \{P_1, \ldots, P_m\} \subseteq \mathcal{P}$, we say that $\mathcal{P}'$ is *entangled* in X if the infinite string $(P_1 \ldots P_m)^\omega$ is generated by X.

3. We say that X is *entangled* if $\mathcal{P}_X$ is entangled in X.

The property of X being entangled is captured by the occurrence of an infinite string $(P_1 \ldots P_m)^\omega$. In fact, as proved in the following lemma, finding $(P_1 \ldots P_m)^k$ for arbitrarily big k is sufficient to guarantee the existence of $(P_1 \ldots P_m)^\omega$.

Lemma A.2. *Let $(Z, \leq)$ be a total order, let $\mathcal{P}$ be a partition of Z, let $X \subseteq Z$, and let $\mathcal{P}' = \{P_1, \dots, P_m\} \subseteq \mathcal{P}$. The following are equivalent:*

1. *For every $k \in \mathbb{N}$, the string $(P_1 \dots P_m)^k$ is generated by X.*

2. *$(P_1 \dots P_m)^\omega$ is generated by X.*

Proof. The nontrivial implication is (1) $\Rightarrow$ (2). If $m = 1$, then by choosing $k = 1$ we obtain that there exists $x \in X$ such that $x \in P_1$, so $(P_1)^\omega$ occurs in X, as witnessed by the monotone sequence $(x_i)_{i \in \mathbb{N}}$ such that $x_i = x$ for every $i \in \mathbb{N}$. Thus, in the following we can assume $m \geq 2$. This implies that for every k, if $(P_1 \dots P_m)^k$ occurs in X at $x_1, x_2, \dots, x_{m_k}$, then $x_1 < x_2 < \dots < x_{m_k}$, that is, the inequalities are strict. If (1) holds, we prove that we can find an infinite family $(Y_i)_{i \geq 1}$ of pairwise disjoint subsets of X, each containing an occurrence of $(P_1 \dots P_m)$, such that for every pair of distinct integers i, j it holds either $Y_i < Y_j$ (that is, each element in Y_i is smaller than each element in Y_j) or $Y_j < Y_i$. This will imply (2), because if the set $\{i \geq 1 | (\forall j > i)(Y_i < Y_j)\}$ is infinite, then (2) is witnessed by an increasing sequence, and if $\{i \geq 1 | (\forall j > i)(Y_i < Y_j)\}$ is finite, then (2) is witnessed by a decreasing sequence.
Let us show a recursive construction of $(Y_i)_{i \geq 1}$. We say that $\langle X_1, X_2 \rangle$ is a *split* of X if $\{X_1, X_2\}$ is a partition of X and $X_1 < X_2$. Given a split $\langle X_1, X_2 \rangle$ of X, we claim that (1) must hold for either X_1 or X_2 (or both). In fact, reasoning by contradiction, assume there exists $\bar{k}$ such that the string $(P_1 \dots P_m)^{\bar{k}}$ is neither generated by X_1 nor by X_2. This promptly leads to a contradiction, since $(P_1 \dots P_m)^{2\bar{k}}$ is generated by X and hence,if $(P_1 \dots P_m)^{\bar{k}}$ is not generated by X_1, then it must be generated by X_2.
Now consider an occurrence of $(P_1 \dots P_m)^2$ generated by X and a split $\langle X_1, X_2 \rangle$ such that $(P_1 \dots P_m)$ is generated by X_1 and $(P_1 \dots P_m)$ is generated by X_2. Now, if (1) holds for X_1, then define $Y_1 = X_2$ and repeat the construction using X_1 instead of X. If (1) holds for X_2, then define $Y_1 = X_1$ and repeat the construction using X_2 instead of X. We can then recursively define a family $(Y_i)_{i \geq 1}$ with the desired properties. $\square$

We now introduce the notion of an *entangled convex decomposition*, whose aim is to identify entangled regions of $(Z, \leq)$ with respect to a partition $\mathcal{P}$.

Definition A.3. Let $(Z, \leq)$ be a total order and let $\mathcal{P}$ be a partition of Z. We say that a partition $\mathcal{V}$ of Z is an *entangled, convex decomposition of $\mathcal{P}$ in $(Z, \leq)$ (e.c.*

decomposition, for short) if all the elements of $\mathcal{V}$ are entangled (w.r.t. the partition $\mathcal{P}$) convex sets in $(Z, \leq)$.

Example A.4. Consider the total order $(\mathbb{Z}, \leq)$, where $\mathbb{Z}$ is the set of all integers and $\leq$ is the usual order on $\mathbb{Z}$. Let $\mathcal{P} = \{P_1, P_2, P_3\}$ be the partition of $\mathbb{Z}$ defined as follows:

$$\begin{aligned}
P_1 &= \{n \leq 0 : n \text{ is odd}\} \cup \{n > 0 : n \equiv 1 \bmod 3\}\\
P_2 &= \{n \leq 0 : n \text{ is even}\} \cup \{n > 0 : n \equiv 2 \bmod 3\},\\
P_3 &= \{n > 0 : n \equiv 0 \bmod 3\}
\end{aligned}$$

The partition $\mathcal{P}$ generates the following *trace* over $\mathbb{Z}$:

$$\dots P_1P_2P_1P_2 \dots P_1P_2P_1P_2P_3P_1P_2P_3 \dots$$

Now define $\mathcal{V} = \{V_1, V_2\}$, where $V_1 = \{n \in \mathbb{Z} : n \leq 0\}$, $V_2 = \{n \in \mathbb{Z} : n > 0\}$. It is immediate to check that $\mathcal{V}$ is an e.c. decomposition of $\mathcal{P}$ in $(\mathbb{Z}, \leq)$. More trivially, even $\mathcal{V}' = \{\mathbb{Z}\}$ is an e.c. decomposition of $\mathcal{P}$ in $(\mathbb{Z}, \leq)$.

Below we prove that if $\mathcal{P}$ is a finite partition, then there always exists a *finite* e.c. decomposition of $\mathcal{P}$.

Theorem A.5. *Let $(Z, \leq)$ be a total order, and let $\mathcal{P} = \{P_1, \dots, P_m\}$ be a finite partition of Z. Then, $\mathcal{P}$ admits a finite e.c. decomposition in $(Z, \leq)$.*

Proof. We proceed by induction on $m = |\mathcal{P}|$. If $m = 1$, then $\mathcal{P} = \{Z\}$, so $\{Z\}$ is an e.c. decomposition of $\mathcal{P}$ in $(Z, \leq)$. Assume $m \geq 2$ and notice that we may also assume that the sequence $(P_1 \dots P_m)^\omega$ is *not* generated by Z, otherwise the partition $\mathcal{V} = \{Z\}$ is a finite e.c. decomposition of $\mathcal{P}$ in $(Z, \leq)$ and we are done. Since $(P_1 \dots P_m)^\omega$ is not generated by Z, for any permutation π of the set $\{1, \dots, m\}$ the sequence $(P_{\pi(1)}, \dots, P_{\pi(m)})^\omega$ is not generated by Z and therefore, by Lemma A.2, for any π there exists an integer s_π such that $(P_{\pi(1)}, \dots, P_{\pi(m)})^{s_\pi}$ is not generated by Z. Using this property we prove that there exists a finite partition $\mathcal{V}$ of Z into convex sets such that for every $V \in \mathcal{V}$ and every π, the string $(P_{\pi(1)}, \dots, P_{\pi(m)})^2$ does not occur in V.

Consider Algorithm 4. The algorithm starts with $\mathcal{V} = \{Z\}$ and recursively partitions a V in $\mathcal{V}$ into two nonempty convex sets as long as $(P_{\pi(1)}, \dots, P_{\pi(m)})^2$ occurs in

Algorithm 4

```
1: $\mathcal{V} \leftarrow \{Z\}$;                                        ▷ initialise the partition
2: for $\pi$ permutation of $\{1,\dots,m\}$ do
3:     while exists an element in $\mathcal{V}$ generating $(P_{\pi(1)} \dots P_{\pi(m)})^2$ do
4:         let $V \in \mathcal{V}$ generating $(P_{\pi(1)} \dots P_{\pi(m)})^2$;
5:         $\mathcal{V} \leftarrow \mathcal{V} \setminus V$;
6:         let $\alpha_1 < \dots < \alpha_m < \alpha'_1 < \dots < \alpha'_m$ in $V$ be such that $\alpha_j, \alpha'_j \in P_{\pi(j)}$, for $j = 1,\dots,m$;
7:         $\mathcal{V} \leftarrow \mathcal{V} \cup \{\{\alpha \in V \mid \alpha \leq \alpha_m\}, \{\alpha \in V \mid \alpha > \alpha_m\}\}$;
8:     end while
9: end for
10: return $\mathcal{V}$
```

V. Notice that the algorithm ends after at most $\sum_\pi s_\pi$ iterations, returning a finite partition $\mathcal{V}$ of Z into convex sets.

Now fix $V \in \mathcal{V}$ and consider the partition $\mathcal{P}_{|V} = \{P \cap V \mid P \in \mathcal{P} \wedge P \cap V \neq \emptyset\} = \{P \cap V \mid P \in \mathcal{P}_V\}$ of V, where $\mathcal{P}_V$ is as in Def. A.1 and $|\mathcal{P}_{|V}| \leq m$. To complete the proof it will be enough to prove that $\mathcal{P}_{|V}$ admits a finite e.c. decomposition in $(V, \leq)$ because then a finite e.c. decomposition of $\mathcal{P}$ in $(Z, \leq)$ can be obtained by merging all the decompositions of $\mathcal{P}_{|V}$, for $V \in \mathcal{V}$.

If $|\mathcal{P}_{|V}| < m$, then $\mathcal{P}_{|V}$ admits an e.c. decomposition by inductive hypothesis. Otherwise, $|\mathcal{P}_{|V}| = m$, say $\mathcal{P}_{|V} = \{P'_1, \dots, P'_m\}$. By construction, we know that for any permutation π of $\{1,\dots,m\}$ the string $(P'_{\pi(1)}, \dots, P'_{\pi(m)})^2$ does not occur in V. Example A.6 below may help with an intuition for the rest of the argument. Let $k \geq 1$ be the number of distinct permutations π of $\{1,\dots,m\}$ such that $(P'_{\pi(1)}, \dots, P'_{\pi(m)})$ occurs in V. We proceed by induction on k. If $k = 1$, then each set P'_j is convex and trivially entangled, so $\{P'_j \mid 1 \leq j \leq m\}$ is an e.c. decomposition of $\mathcal{P}_{|V}$ in $(V, \leq)$. Now assume $k \geq 2$ and fix a permutation π such that $(P'_{\pi(1)}, \dots, P'_{\pi(m)})$ occurs in V. Define:

$$V_1 = \{\alpha \in V \mid \exists\ \alpha_1, \dots, \alpha_m \text{ with } \alpha \leq \alpha_1 < \dots < \alpha_m \text{ and } \alpha_i \in P'_{\pi(i)}\ \}$$

and $V_2 = V \setminus V_1$. Let us prove that V_1 and V_2 are nonempty. Just observe that if $\alpha_1, \dots, \alpha_m$ is a witness for $(P'_{\pi(1)}, \dots, P'_{\pi(m)})$ in V, then $\alpha_1 \in V_1$. Moreover, $\alpha_m \in V \setminus V_1 = V_2$, otherwise, since $m > 1$ and $P'_{\pi(1)} \neq P'_{\pi(m)}$, the sequence $(P'_{\pi(1)}, \dots, P'_{\pi(m)})^2$ would occur in V. The previous observation implies also that $(P'_{\pi(1)}, \dots, P'_{\pi(m)})$ does not occur in V_1, nor in V_2. Moreover V_1 and V_2 are clearly convex. To conclude it will suffice to prove that the V_i-partition $\mathcal{P}_{|V_i} = \{P \cap V_i \mid P \in \mathcal{P}_{|V}, P \cap V_i \neq$

$\emptyset\}$ admits a finite e.c. decomposition in $(V_i, \leq)$, for $i = 1, 2$. For any given i, if $|\mathcal{P}_{|V_i}| < m$, we conclude by the inductive hypothesis on m. If, instead, $|\mathcal{P}_{|V_i}| = m$, say $\mathcal{P}_{|V_i} = \{P''_1, \ldots, P''_m\}$, we use the inductive hypothesis on k: the number of distinct permutations π' of $\{1, \ldots, m\}$ such that $(P''_{\pi'(1)}, \ldots, P''_{\pi'(m)})$ occurs in V_i is less than k, since $(P'_{\pi(1)}, \ldots, P'_{\pi(m)})$ occurs in V while $(P''_{\pi(1)}, \ldots, P''_{\pi(m)})$ does not occur in V_i.

□

Example A.6. We give an example of the final part of the construction described in Theorem A.5. Suppose $\mathcal{P}_{|V} = \{P'_1, P'_2, P'_3\}$ leaves the following trace over $(V, \leq)$:

$$(P'_1 P'_2)^\omega (P'_3 P'_2)^\omega (P'_1)^\omega (P'_2)^\omega$$

Notice that, as assumed in the last part of the above proof, for any permutation π of $\{1, 2, 3\}$ the sequence $(P'_{\pi(1)} P'_{\pi(2)} P'_{\pi(3)})^2$ does not appear in V; however, the sequence $(P'_{\pi(1)} P'_{\pi(2)} P'_{\pi(3)})$ appears in V for $\pi = id$. If we fix $\pi = id$ and consider the sets V_1, V_2 as in the above proof, then the partition $\mathcal{P}_{|V}$ leaves the following traces on the sets V_1, V_2:

$$(P'_1 P'_2)^\omega \qquad \text{and} \qquad (P'_3 P'_2)^\omega (P'_1)^\omega (P'_2)^\omega,$$

respectively. Notice that P'_3 does not appear in V_1, while the sequence $(P'_1 P'_2 P'_3)$ does not appear in V_2, so that the inductive hypothesis can be applied.

We say that an e.c. decomposition of $\mathcal{P}$ in $(Z, \leq)$ is a *minimum-size* e.c. decomposition if it has minimum cardinality. As shown in the following remark, minimum-size e.c. decompositions ensure additional interesting properties.

Remark A.7. Let $(Z, \leq)$ be a total order, let $\mathcal{P}$ be a finite partition of Z, and let $\mathcal{V} = \{V_1, \ldots, V_r\}$ be a minimum-size e.c. decomposition of $\mathcal{P}$ in $(Z, \leq)$, where $V_1 < \cdots < V_r$. Then, for every $1 \leq i < r$, we have $\mathcal{P}_{V_i} \not\subseteq \mathcal{P}_{V_{i+1}}$, where $\mathcal{P}_{V_i} = \{P \in \mathcal{P} \mid P \cap V_i \neq \emptyset\}$ (see Definition A.1). In fact, if this were not the case, $\mathcal{V}' = \{V_1, \ldots, V_{i-1}, V_i \cup V_{i+1}, \ldots V_r\}$ would be a smaller size e.c. decomposition of $\mathcal{P}$ in $(Z, \leq)$. Similarly, for every $1 < i \leq r$, it must be $\mathcal{P}_{V_i} \not\subseteq \mathcal{P}_{V_{i-1}}$. In conclusion, for every $i = 1, \ldots, r$, there exist $R_i \in \mathcal{P}_{V_i} \setminus \mathcal{P}_{V_{i+1}}$ and $L_i \in \mathcal{P}_{V_i} \setminus \mathcal{P}_{V_{i-1}}$, where we assume $V_0 = V_{r+1} = \emptyset$.

In general, a minimum-size e.c. decomposition is not unique.

Example A.8. Let us show that even in the special case when $(Z, \leq) = (\text{Pref}(\mathcal{L}(\mathcal{D})), \preceq)$ and $\mathcal{P} = \{I_u \mid u \in Q\}$ we can have more than one minimum-size e.c. decomposition.

Consider the DFA $\mathcal{D}$ in Figure A.1. Notice that in every e.c. decomposition of $\mathcal{D}$ one element is $\{\varepsilon\}$, because $I_0 = \{\varepsilon\}$. Moreover, every e.c. decomposition of $\mathcal{D}$ must have cardinality at least three, because, since $1 <_{\mathcal{D}} 3$, states 1 and 3 are not entangled. It is easy to check that:

$$\mathcal{V} = \{\{\varepsilon\}, \{ac^* \cup bc^*\}, \{[b(c+d)^* \setminus bc^*] \cup f(c+d)^* \cup gd^*\}\}$$

and:

$$\mathcal{V}' = \{\{\varepsilon\}, \{ac^* \cup b(c+d)^* \cup [f(c+d)^* \setminus fd^*]\}, \{fd^* \cup gd^*\}\}$$

are two distinct minimum-size e.c. decompositions of $\mathcal{D}$.

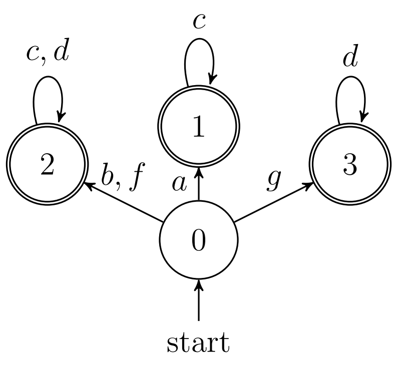


Figure A.1: An automaton $\mathcal{D}$ admitting two distinct minimum-size e.c. decompositions.

We need the following lemmas on entangled convex sets in Section 5.4. Assume that $(Z, \leq)$ is a total order and $\mathcal{P}$ is a partition of Z.

Lemma A.9. *Let $\mathcal{V}$ be a minimum-size e.c. decomposition of $\mathcal{P}$. Assume that $V \in \mathcal{V}$ is such that there exist $C_1 < \cdots < C_n$ entangled convex sets with $V \subseteq \bigcup_{i=1}^n C_i$. Then, for all $P \in \mathcal{P}$ it holds:*

$$\forall i \in \{1, \ldots, n\}(C_i \cap P \neq \emptyset) \;\; \rightarrow \;\; V \cap P \neq \emptyset.$$

Proof. If $C_j \subseteq V$ for some j, then $V \cap P \neq \emptyset$ since $V \cap P \supseteq C_j \cap P \neq \emptyset$. Otherwise, consider the smallest i such that $C_i \cap V \neq \emptyset$. Since $C_1 < \cdots < C_n$, V is convex and

V does not contain any C_j, it must be $V \subseteq C_i \cup C_{i+1}$, (where we assume $C_{i+1} = \emptyset$ if $i = n$). Let $\mathcal{V} = \{V_1, \ldots, V_r\}$ with $V_1 < \cdots < V_r$. For all $j = 1, \ldots, r$, consider the elements $R_j \in \mathcal{P}_{V_j} \setminus \mathcal{P}_{V_{j+1}}$ and $L_j \in \mathcal{P}_{V_j} \setminus \mathcal{P}_{V_{j-1}}$ (where we assume $V_0 = V_{r+1} = \emptyset$), as in Remark A.7 . Let s be such that $V = V_s$. We distinguish three cases.

1. $V_s \cap C_{i+1} = \emptyset$. In this case, it must be $V_s \subseteq C_i$. Let V_{s-h}, V_{s-h+1}, $\ldots$, V_s, $\ldots$, V_{s+k-1}, V_{s+k} $(h, k \geq 0)$ be all the elements of $\mathcal{V}$ contained in C_i. Since $V_1 < \cdots < V_r$, we conclude:

   $$V_{s-h} \cup \cdots \cup V_s \cup \cdots \cup V_{s+k} \subseteq C_i \subseteq V_{s-h-1} \cup V_{s-h} \cup \cdots \cup V_s \cup \cdots \cup V_{s+k} \cup V_{s+k+1}.$$

   We know that $L_{s-h}, \ldots, L_s, R_s, \ldots R_{s+k}$ occur in $V_{s-h} \cup \cdots \cup V_{s+k}$, so they also occur in C_i. Moreover, we also know that P occurs in C_i. Since C_i is entangled, there exists a sequence $\alpha_{s-h} \leq \cdots \leq \alpha_s \leq \beta \leq \gamma_s \leq \cdots \leq \gamma_{s+k}$ of elements in C_i witnessing that the ordered sequence $L_{s-h}, \ldots, L_s, P, R_s, \ldots R_{s+k}$ occurs in C_i; it follows that $L_{s-h}, \ldots, L_s, P, R_s, \ldots R_{s+k}$ occurs in $V_{s-h-1} \cup V_{s-h} \cup \cdots \cup V_s \cup \cdots \cup V_{s+k} \cup V_{s+k+1}$ as well. Since L_{s-h} does not occur in V_{s-h-1}, then the ordered sequence $L_{s-h+1}, \ldots, L_s, P, R_s, \ldots R_{s+k}$ occurs in $V_{s-h} \cup \cdots \cup V_s \cup \cdots \cup V_{s+k} \cup V_{s+k+1}$. Now, L_{s-h+1} does not occur in V_{s-h}, so the ordered sequence $L_{s-h+2}, \ldots, L_s, P, R_s, \ldots R_{s+k}$ occurs in $V_{s-h+1} \cup \cdots \cup V_s \cup \cdots \cup V_{s+k} \cup V_{s+k+1}$. Proceeding in this way, we obtain that the sequence $P, R_s, \ldots R_{s+k}$ occurs in this order in $V_s \cup \cdots \cup V_{s+k} \cup V_{s+k+1}$. Now suppose for sake of a contradiction that P does not occur in V_s. As before we obtain that $R_s, \ldots R_{s+k}$ occurs in this order in $V_{s+1} \cup \cdots \cup V_{s+k} \cup V_{s+k+1}$, $R_{s+1}, \ldots R_{s+k}$ occurs in this order in $V_{s+2} \cup \cdots \cup V_{s+k} \cup V_{s+k+1}$, and so on. We finally conclude that R_{s+k} occurs in V_{s+k+1}, a contradiction.

2. $V_s \cap C_i = \emptyset$. In this case, it must be $V_s \subseteq C_{i+1}$ and one concludes as in the previous case.

3. $V_s \cap C_i \neq \emptyset$ and $V_s \cap C_{i+1} \neq \emptyset$. In this case, let $V_{s-h}, \ldots, V_{s-1}$ $(h \geq 0)$ be all elements of $\mathcal{V}$ contained in C_i, and let $V_{s+1}, \ldots, V_{s+k}$ $(k \geq 0)$ be all elements of $\mathcal{V}$ contained in C_{i+1}. As before:

   $$V_{s-h} \cup \cdots \cup V_{s-1} \subseteq C_i \subseteq V_{s-h-1} \cup V_{s-h} \cup \cdots \cup V_{s-1} \cup V_s$$

and:

$$V_{s+1} \cup \cdots \cup V_{s+k} \subseteq C_{i+1} \subseteq V_s \cup V_{s+1} \cup \cdots \cup V_{s+k} \cup V_{s+k+1}.$$

Now, assume by contradiction that P does not occur in V_s. First, let us prove that L_s does not occur in C_i. Suppose by contradiction that L_s occurs in C_i. We know that $L_{s-h}, \dots, L_{s-1}$ occurs in C_i, and we also know that P occurs in C_i. Since C_i is entangled, then $L_{s-h}, \dots, L_{s-1}, L_s, P$ should occur in this order in C_i and so also in $V_{s-h-1} \cup V_{s-h} \cup \cdots \cup V_{s-1} \cup V_s$; however, reasoning as in case 1, this would imply that P occurs in V_s, a contradiction. Analogously, one shows that R_s does not occur in C_{i+1}.

Since R_s and L_s occur in V_s, then there exists a monotone sequence in V_s whose trace consists of alternating values of R_s and L_s. But $V_s \subseteq C_i \cup C_{i+1}$ and $C_i \prec C_{i+1}$, so the monotone sequence is definitely contained in C_i or C_{i+1}. In the first case we would obtain that L_s occurs in C_i, and in the second case we would obtain that R_s occurs in C_{i+1}, so in both cases we reach a contradiction.

□

Lemma A.10. *Let $C \subseteq Z$ be an entangled convex set and consider any pair of convex sets C_1, C_2 such that $C = C_1 \cup C_2$. Then, there exists $i \in \{1, 2\}$ such that C_i is entangled and $\mathcal{P}_{C_i} = \mathcal{P}_C$.*

Proof. Let $(z_i)_{i \geq 1}$ be a monotone sequence witnessing the entanglement of C. Then, infinitely many z_j's appear in either C_1 or C_2 (or both). In the former case C_1 is entangled: since C_1 is convex, if $j_0 \geq 1$ is such that $z_{j_0} \in C_1$, then the subsequence $(z_j)_{j \geq j_0}$ is in C_1 and, clearly, we have $\mathcal{P}_{C_1} = \mathcal{P}_C$. In the latter case, analogously, C_2 is entangled and $\mathcal{P}_{C_2} = \mathcal{P}_C$. □

Lemma A.11. *Let $C_1, C_2 \subseteq Z$ be entangled convex sets. Then, at least one the following holds true:*

1. *$C_1 \setminus C_2$ is entangled and convex and $\mathcal{P}_{C_1 \setminus C_2} = \mathcal{P}_{C_1}$;*
2. *$C_2 \setminus C_1$ is entangled and convex and $\mathcal{P}_{C_2 \setminus C_1} = \mathcal{P}_{C_2}$;*
3. *$C_1 \cup C_2$ is entangled and convex and $\mathcal{P}_{C_1 \cup C_2} = \mathcal{P}_{C_1}$ or $\mathcal{P}_{C_1 \cup C_2} = \mathcal{P}_{C_2}$.*

Proof. If $C_1 \cap C_2 = \emptyset$, (1) and (2) hold. If $C_2 \subseteq C_1$ or $C_1 \subseteq C_2$, (3) holds. In the remaining cases observe that $C_1 \setminus C_2$, $C_2 \setminus C_1$, $C_1 \cap C_2$ and $C_1 \cup C_2$ are convex. Since $C_1 = (C_1 \setminus C_2) \cup (C_1 \cap C_2)$ and $C_2 = (C_2 \setminus C_1) \cup (C_1 \cap C_2)$, by Lemma A.10 we conclude that at least one the following holds true:

1. $C_1 \setminus C_2$ is entangled and convex and $\mathcal{P}_{C_1 \setminus C_2} = \mathcal{P}_{C_1}$, or $C_2 \setminus C_1$ is entangled and convex and $\mathcal{P}_{C_2 \setminus C_1} = \mathcal{P}_{C_2}$;

2. the intersection $C_1 \cap C_2$ is entangled and convex and $\mathcal{P}_{C_1 \cap C_2} = \mathcal{P}_{C_1} = \mathcal{P}_{C_2}$.

In the first case we are done, while in the second case we have $\mathcal{P}_{C_1 \cap C_2} = \mathcal{P}_{C_1} = \mathcal{P}_{C_2} = \mathcal{P}_{C_1} \cup \mathcal{P}_{C_2} = \mathcal{P}_{C_1 \cup C_2}$. Since $C_1 \cap C_2 \subseteq C_1 \cup C_2$ and $C_1 \cap C_2$ is entangled, we conclude that $C_1 \cup C_2$ is entangled. □

Lemma A.12. *Let $C_1, \ldots, C_n \subseteq Z$ be entangled convex sets. Then, there exist $m \leq n$ pairwise disjoint, entangled convex sets $C'_1, \ldots, C'_m \subseteq Z$, such that:*

- *$C'_1 < \cdots < C'_m$ and $\bigcup_{i=1}^{n} C_i = \bigcup_{i=1}^{m} C'_i$;*

- *if $P \in \mathcal{P}$ occurs in all C_i's, then it occurs in all C'_i's as well.*

Proof. We can suppose without loss of generality that the C_i's are non-empty and we proceed by induction on the number of intersections $r = |\{(i, j) | i < j \wedge C_i \cap C_j \neq \emptyset\}|$.

If $r = 0$, then the C_i's are pairwise disjoint and, since they are convex, they are comparable. Hence, it is sufficient to take $C'_1, \ldots, C'_n$ as the permutation of the C_i's such that $C'_1 < \cdots < C'_n$.

Now assume $r \geq 1$ and let, without loss of generality, $C_1 \cap C_2 \neq \emptyset$. We now produce a new sequence of at most n entangled convex sets to which we can apply the inductive hypothesis. By Lemma A.11 at least one among $C_1 \setminus C_2$, $C_2 \setminus C_1$ and $C_1 \cup C_2$, is an entangled convex set. If $C_1 \cup C_2$ is an entangled convex set, then let $C_1 \cup C_2, C_3, \ldots, C_n$ be the new sequence. Otherwise, if $C_1 \setminus C_2$ (the case $C_2 \setminus C_1$ analogous) is entangled, let the new sequence be $C_1 \setminus C_2, C_2, C_3, \ldots, C_n$. In both cases the number of intersections decreases: this is clear in the first case, while in the second case $C_1 \setminus C_2 \subseteq C_1$ and $(C_1 \setminus C_2) \cap C_2 = \emptyset$.

In all the above cases Lemma A.11 implies that if a $P \in \mathcal{P}$ occurs in all C_i's, then it occurs in all elements of the new family and we can conclude by the inductive hypothesis. □

Below we prove a fairly intuitive result on the intersection of convex sets.

Lemma A.13. *Let $(Z, \leq)$ be a total order. If $C_1, \ldots, C_n \subseteq Z$ are non-empty, convex sets such that $C_i \cap C_j \neq \emptyset$ for all $i, j \in \{1, \ldots, n\}$, then $\bigcap_{i=1}^n C_i \neq \emptyset$.*

Proof. We proceed by induction on n. Cases $n = 1, 2$ are trivial, so assume $n \geq 3$. For every $i \in \{1, \ldots, n\}$, the set:

$$\bigcap_{\substack{k \in \{1,\ldots,n\} \\ k \neq i}} C_k$$

is nonempty by the inductive hypothesis, so we can pick an element d_i. If for some distinct i and j we have $d_i = d_j$, then such an element witnesses that $\bigcap_{i=1}^n C_i \neq \emptyset$. Otherwise, assume without loss of generality that $d_1 < \cdots < d_n$. Fix any integer j such that $1 < j < n$, and let us prove that d_j witnesses that $\bigcap_{i=1}^n C_i \neq \emptyset$. We only have to prove that $d_j \in C_j$. This follows from $d_1, d_n \in C_j$ and the fact that C_j is convex. □

Bibliography

[1] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *J. of Discrete Algorithms*, 2(1):53–86, 2004.

[2] Donald Adjeroh, Timothy Bell, and Amar Mukherjee. *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching.* Springer Publishing Company, Incorporated, 1 edition, 2008.

[3] Jarno Alanko, Nicola Cotumaccio, and Nicola Prezza. Linear-time minimization of Wheeler DFAs. In *2022 Data Compression Conference (DCC)*, pages 53–62. IEEE, 2022.

[4] Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 911–930. SIAM, 2020.

[5] Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Wheeler languages. *Inf. Comput.*, 281:104820, 2021.

[6] Jarno N. Alanko, Travis Gagie, Gonzalo Navarro, and Louisa Seelbach Benkner. Tunneling on Wheeler graphs. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2019, Snowbird, UT, USA, March 26-29, 2019*, pages 122–131. IEEE, 2019.

[7] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), feb 2008.

[8] Jasmijn A. Baaijens, Paola Bonizzoni, Christina Boucher, Gianluca Della Vedova, Yuri Pirola, Raffaella Rizzi, and Jouni Sirén. Computational graph pangenomics: a tutorial on data structures and their applications. *Nat. Comput.*, 21(1):81–108, 2022.

[9] Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 457–466. IEEE, 2016.

[10] Uwe Baier, Timo Beller, and Enno Ohlebusch. Graphical pan-genome analysis with compressed suffix trees and the Burrows–Wheeler transform. *Bioinformatics*, 32(4):497–504, 10 2015.

[11] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012. PMID: 22506599.

[12] Ruben Becker, Manuel Cáceres, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Francisco Olivares, and Nicola Prezza. Sorting Finite Automata via Partition Refinement. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[13] D. Belazzougui and G. Navarro. Optimal lower and upper bounds for representing sequences. *ACM Transactions on Algorithms*, 11(4):article 31, 2015.

[14] Jason W. Bentley, Daniel Gibney, and Sharma V. Thankachan. On the Complexity of BWT-Runs Minimization via Alphabet Reordering. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[15] Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon Pissis, Giovanna Rosone, et al. Even faster elastic-degenerate string matching via fast matrix multiplication. In *The ICALP 2019 Proceedings*, volume 132, pages 1–15. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2019.

[16] Maciej Besta and Torsten Hoefler. Survey and taxonomy of lossless graph compression and space-efficient graph representations. *arXiv preprint arXiv:1806.01799*, abs/1806.01799, 2019.

[17] Philip Bille and Mikkel Thorup. Faster regular expression matching. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, pages 171–182, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[18] Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In Amihood Amir and Laxmi Parida, editors, *Combinatorial Pattern Matching*, pages 138–150, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[19] Henning Bordihn, Markus Holzer, and Martin Kutrib. Determination of finite automata accepting subregular languages. *Theoretical Computer Science*, 410(35):3209–3222, 2009. Descriptional Complexity of Formal Systems.

[20] Christina Boucher, Alex Bowe, Travis Gagie, Simon J. Puglisi, and Kunihiko Sadakane. Variable-order de Bruijn graphs. In *2015 Data Compression Conference*, pages 383–392, 2015.

[21] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*, pages 225–235, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[22] Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 307–318. IEEE, 2017.

[23] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 79–97. IEEE, 2015.

[24] Nieves R Brisaboa, Susana Ladra, and Gonzalo Navarro. k2-trees for compact web graph representation. In *SPIRE*, volume 9, pages 18–30. Springer, 2009.

[25] Janusz A Brzozowski and Rina Cohen. On decompositions of regular events. *Journal of the ACM (JACM)*, 16(1):132–144, 1969.

[26] Janusz A. Brzozowski and Faith E. Fich. Languages of R-trivial monoids. *J. Comput. Syst. Sci.*, 20(1):32–49, 1980.

[27] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[28] Manuel Caceres. Parameterized algorithms for string matching to dags: Funnels and beyond. *arXiv preprint arXiv:2212.07870*, 2022.

[29] Sankardeep Chakraborty, Roberto Grossi, Kunihiko Sadakane, and Srinivasa Rao Satti. Succinct representations for (non) deterministic finite automata. In *LATA*, pages 55–67. Elsevier, 2021.

[30] Sankardeep Chakraborty and Seungbum Jo. Compact representation of interval graphs and circular-arc graphs of bounded degree and chromatic number. *Theoretical Computer Science*, 941:156–166, 2023.

[31] W. I. Chang and E. L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12:327–344, 2005.

[32] Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.

[33] Francisco Claude and Gonzalo Navarro. A fast and compact web graph representation. In *String Processing and Information Retrieval: 14th International Symposium, SPIRE 2007 Santiago, Chile, October 29-31, 2007 Proceedings 14*, pages 118–129. Springer, 2007.

[34] Alessio Conte, Nicola Cotumaccio, Travis Gagie, Giovanni Manzini, Nicola Prezza, and Marinella Sciortino. Computing matching statistics on wheeler dfas. In *2023 Data Compression Conference (DCC)*, pages 150–159, 2023.

[35] Nicola Cotumaccio. Graphs can be succinctly indexed for pattern matching in $O(|E|^2 + |V|^{5/2})$ time. In *2022 Data Compression Conference (DCC)*, pages 272–281. IEEE, 2022.

[36] Nicola Cotumaccio. Prefix Sorting DFAs: A Recursive Algorithm. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[37] Nicola Cotumaccio. A myhill-nerode theorem for generalized automata, with applications to pattern matching and compression, 2024 *(to appear in the proceedings of STACS 2024)*.

[38] Nicola Cotumaccio, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Co-lexicographically ordering automata and regular languages - part i. *J. ACM*, 70(4), aug 2023.

[39] Nicola Cotumaccio, Travis Gagie, Dominik Köppl, and Nicola Prezza. Space-time trade-offs for the lcp array of wheeler dfas. In *String Processing and Information Retrieval: 30th International Symposium, SPIRE 2023, Pisa, Italy, September 26–28, 2023, Proceedings*, page 143–156, Berlin, Heidelberg, 2023. Springer-Verlag.

[40] Nicola Cotumaccio and Nicola Prezza. On indexing and compressing finite automata. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2585–2599. SIAM, 2021.

[41] Isabel F Cruz, Alberto O Mendelzon, and Peter T Wood. A graphical query language supporting recursion. *ACM SIGMOD Record*, 16(3):323–330, 1987.

[42] Giovanna D'Agostino, Davide Martincigh, and Alberto Policriti. Ordering regular languages: a danger zone. In Claudio Sacerdoti Coen and Ivano Salvo, editors, *Proceedings of the 22nd Italian Conference on Theoretical Computer Science, Bologna, Italy, September 13-15, 2021*, volume 3072 of *CEUR Workshop Proceedings*, pages 46–69. CEUR-WS.org, 2021.

[43] Narsingh Deo and Bruce Litow. A structural approach to graph compression. In *Proc. of the 23th MFCS Workshop on Communications*, pages 91–101, 1998.

[44] Diego Díaz-Domínguez, Travis Gagie, and Gonzalo Navarro. Simulating the DNA Overlap Graph in Succinct Space. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*, volume 128 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[45] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[46] Lawrence C Eggan. Transition graphs and the star-height of regular events. *Michigan Mathematical Journal*, 10(4):385–397, 1963.

[47] Samuel Eilenberg. *Automata, Languages, and Machines.* Academic Press, Inc., USA, 1974.

[48] Joost Engelfriet. *Context-Free Graph Grammars*, pages 125–213. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[49] Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[50] Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In Tomás Bures, Riccardo Dondi, Johann Gamper, Giovanna Guerrini, Tomasz Jurdzinski, Claus Pahl, Florian Sikora, and Prudence W. H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science - 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25-29, 2021, Proceedings*, volume 12607 of *Lecture Notes in Computer Science*, pages 608–622. Springer, 2021.

[51] M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 137–143, 1997.

[52] Arash Farzan and Shahin Kamali. Compact navigation and distance oracles for graphs with small treewidth. *Algorithmica*, 69:92–116, 2014.

[53] Guy Feigenblat, Ely Porat, and Ariel Shiftan. Linear time succinct indexable dictionary construction with applications. In *2016 Data Compression Conference (DCC)*, pages 13–22. IEEE, 2016.

[54] Stefan Felsner, Vijay Raghavan, and Jeremy P. Spinrad. Recognition algorithms for orders of small width and graphs of small dilworth number. *Order*, 20(4):351–364, 2003.

[55] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 184–193. IEEE Computer Society, 2005.

[56] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398. IEEE Computer Society, 2000.

[57] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, jul 2005.

[58] L. Ferres, J. Fuentes-Sepúlveda, T. Gagie, M. He, and G. Navarro. Fast and compact planar embeddings. *Computational Geometry Theory and Applications*, page article 101630, 2020.

[59] N. J. Fine and H. S. Wilf. Uniqueness theorem for periodic functions. *Proc. Amer. Math. Soc.*, (16):109–114, 1965.

[60] J. Fischer. Optimal succinctness for range minimum queries. In *LATIN 2010: Theoretical Informatics*, pages 158–169, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[61] Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: a framework for BWT-based data structures. *Theoretical Computer Science*, 698:67 – 78, 2017. Algorithms, Strings and Theoretical Approaches in the Big Data Era (In Honor of the 60th Birthday of Professor Raffaele Giancarlo).

[62] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully Functional Suffix Trees and Optimal Text Searching in BWT-Runs Bounded Space. *J. ACM*, 67(1), jan 2020.

[63] Dora Giammarresi and Rosa Montalbano. Deterministic generalized automata. In Ernst W. Mayr and Claude Puech, editors, *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, volume 900 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 1995.

[64] Dora Giammarresi and Rosa Montalbano. Deterministic generalized automata. *Theor. Comput. Sci.*, 215(1-2):191–208, 1999.

[65] Daniel Gibney. An efficient elastic-degenerate text index? not likely. In *String Processing and Information Retrieval: 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13–15, 2020, Proceedings*, pages 76–88. Springer, 2020.

[66] Daniel Gibney, Gary Hoppenworth, and Sharma V. Thankachan. Simple reductions from formula-SAT to pattern matching on labeled graphs and subtree isomorphism. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 232–242. SIAM, 2021.

[67] Daniel Gibney, Gary Hoppenworth, and Sharma V. Thankachan. Simple reductions from formula-sat to pattern matching on labeled graphs and subtree isomorphism. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 232–242. SIAM, 2021.

[68] Daniel Gibney and Sharma V Thankachan. Text indexing for regular expression matching. *Algorithms*, 14(5):133, 2021.

[69] Daniel Gibney and Sharma V Thankachan. On the complexity of recognizing Wheeler graphs. *Algorithmica*, 84(3):784–814, 2022.

[70] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.

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

[72] Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 317–326. Springer, 2001.

[73] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

[74] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

[75] Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2 2:291–306, 1995.

[76] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[77] Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Ultra-succinct representation of ordered trees with applications. *Journal of Computer and System Sciences*, 78(2):619–631, 2012.

[78] Shahin Kamali. Compact representation of graphs of small clique-width. *Algorithmica*, 80:2106–2131, 2018.

[79] Shahin Kamali. Compact representation of graphs with bounded bandwidth or treedepth. *Information and Computation*, 285:104867, 2022.

[80] Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, nov 2006.

[81] Dong Kyue Kim, Jeong Seop Sim, Heejin Park, and Kunsoo Park. Constructing suffix arrays in linear time. *Journal of Discrete Algorithms*, 3(2):126–142, 2005. Combinatorial Pattern Matching (CPM) Special Issue.

[82] Sung-Hwan Kim, Francisco Olivares, and Nicola Prezza. Faster prefix-sorting algorithms for deterministic finite automata. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[83] Pang Ko and Srinivas Aluru. Space efficient linear time construction of suffix arrays. *Journal of Discrete Algorithms*, 3(2):143–156, 2005. Combinatorial Pattern Matching (CPM) Special Issue.

[84] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen, S.Z. Li, Huanming Yang, Jian Wang, and Jun Wang. De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20:265–72, 12 2009.

[85] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[86] Anirban Majumdar and Denis Kuperberg. Computing the width of non-deterministic automata. *Logical Methods in Computer Science*, 15, 2019.

[87] Veli Mäkinen, Niko Välimäki, and Jouni Sirén. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11:375–388, 2014.

[88] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, page 319–327, USA, 1990. Society for Industrial and Applied Mathematics.

[89] S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. An extension of the Burrows–Wheeler transform. *Theoretical Computer Science*, 387(3):298–312, 2007. The Burrows-Wheeler Transform.

[90] Tomás Masopust and Markus Krötzsch. Partially ordered automata and piecewise testability. *Log. Methods Comput. Sci.*, 17(2), 2021.

[91] Robert McNaughton and Saymour A. Papert. *Counter-Free Automata.* The MIT Press, USA, 1971.

[92] Robert McNaughton and Seymour A Papert. *Counter-Free Automata (MIT research monograph no. 65).* The MIT Press, 1971.

[93] Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Bioinformatics in the Era of High-Throughput Sequencing.* Cambridge University Press, 2 edition, 2023.

[94] Joong Chae Na. Linear-time construction of compressed suffix arrays using o(n log n)-bit working space for large alphabets. In Alberto Apostolico, Maxime Crochemore, and Kunsoo Park, editors, *Combinatorial Pattern Matching*, pages 57–67, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[95] Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4), mar 2014.

[96] Gonzalo Navarro. *Compact data structures: A practical approach.* Cambridge University Press, 2016.

[97] Abhinav Nellore, Austin Nguyen, and Reid F. Thompson. An Invertible Transform for Efficient String Matching in Labeled Digraphs. In Paweł Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*, volume 191 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[98] Enno Ohlebusch, Simon Gog, and Adrian Kügel. Computing matching statistics and maximal exact matches on compressed full-text indexes. In Edgar Chavez and Stefano Lonardi, editors, *String Processing and Information Retrieval*, pages 347–358, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[99] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[100] Yu Peng, Henry C. M. Leung, S. M. Yiu, and Francis Y. L. Chin. IDBA – a practical iterative de Bruijn graph de novo assembler. In Bonnie Berger, editor, *Research in Computational Molecular Biology*, pages 426–440, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[101] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[102] Aaron Potechin and Jeffrey O. Shallit. Lengths of words accepted by nondeterministic finite automata. *Inf. Process. Lett.*, 162:105993, 2020.

[103] Nicola Prezza. On locating paths in compressed tries. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 744–760. SIAM, 2021.

[104] Simon J. Puglisi, W. F. Smyth, and Andrew H. Turpin. A taxonomy of suffix array construction algorithms. *ACM Comput. Surv.*, 39(2):4–es, jul 2007.

[105] Rajeev Raman and S. Srinivasa Rao. *Succinct Representations of Ordinal Trees*, pages 319–332. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[106] Nicola Rizzo, Alexandru I Tomescu, and Alberto Policriti. Solving string problems on graphs using the labeled direct product. *Algorithmica*, 84(10):3008–3033, 2022.

[107] Fred S Roberts. On the boxicity and cubicity of a graph. *Recent progress in combinatorics*, 1(1):301–310, 1969.

[108] K. Sadakane. Compressed suffix trees with full functionality. *Theor. Comp. Sys.*, 41(4):589–607, 2007.

[109] Kai Salomaa and Sheng Yu. NFA to DFA transformation for finite languages over arbitrary languages. *Journal of Automata, Languages and Combinatorics*, 2:177–186, 1997.

[110] Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: a new characterization of DA. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa, editors, *Developments in Language Theory, 5th International Conference, DLT 2001, Vienna, Austria, July 16-21, 2001, Revised Papers*, volume 2295 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2001.

[111] M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[112] H.-J. Shyr and G. Thierrin. Ordered automata and associated languages. *Tamkang J. Math*, 5:9–20, 1974.

[113] Jared Simpson, Kim Wong, Shaun Jackman, Jacqueline Schein, Steven Jones, and Inanç Birol. ABySS: A parallel assembler for short read sequence data. *Genome research*, 19:1117–23, 02 2009.

[114] Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 11(2):375–388, mar 2014.

[115] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, page 1–9, New York, NY, USA, 1973. Association for Computing Machinery.

[116] Howard Straubing. A generalization of the schützenberger product of finite monoids. *Theoretical Computer Science*, 13(2):137–150, 1981.

[117] Denis Thérien. Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14(2):195–208, 1981.

[118] P. Weiner. Linear pattern matching algorithms. In *Proc. 14th IEEE Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[119] Chengcheng Xu, Shuhui Chen, Jinshu Su, S. M. Yiu, and Lucas C. K. Hui. A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms. *IEEE Communications Surveys & Tutorials*, 18(4):2991–3029, 2016.

[120] Tatsuya Yanagita, Sankardeep Chakraborty, Kunihiko Sadakane, and Srinivasa Rao Satti. Space-Efficient Data Structure for Posets with Applications. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*, volume 227 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.