



Wheeler Graphs and Wheeler Languages

Nicola Cotumaccio  

University of Helsinki, Finland

Giovanna D'Agostino  

University of Udine, Italy

Daniel Gibney  

University of Texas at Dallas, TX, USA

Alberto Policriti  

University of Udine, Italy

Nicola Prezza  

DAIS, Ca' Foscari University of Venice, Italy

Sharma V. Thankachan  

North Carolina State University, Raleigh, NC, USA

Abstract

Suffix sorting stands at the core of the most efficient solutions for indexed pattern matching: the suffix tree, the suffix array, compressed indexes based on the Burrows-Wheeler transform, and so on. In [Gagie, Manzini, Sirén, TCS 2017] this concept was extended to labeled graphs, obtaining the rich class of *Wheeler graphs*. This work opened a very fruitful line of research, ultimately generating results able to bridge the fields of compressed data structures, graph theory, and regular language theory. In a Wheeler graph, nodes are sorted according to the alphabetic order of their incoming labels, propagating this order through pairs of equally-labeled edges. This apparently-simple definition makes it possible to solve on Wheeler graphs problems (including, but not limited to: compression, subpath queries, NFA equivalence, determinization, minimization) that on general labeled graphs are extremely hard to solve, and induces a rich structure in the class of regular languages (*Wheeler languages*) recognized by automata whose state transition is a Wheeler graph. The goal of this survey is to provide a summary of (and intuitions behind) the results on Wheeler graphs that appeared in the literature since their introduction, in addition to a discussion of interesting problems that are still open in the field.

2012 ACM Subject Classification Theory of computation → Sorting and searching; Theory of computation → Graph algorithms analysis; Theory of computation → Pattern matching; Theory of computation → Formal languages and automata theory

Keywords and phrases Wheeler languages, Wheeler graphs, pattern matching, indexing, compressed data structures

Digital Object Identifier 10.4230/OASICS.Manzini.2025.12

Funding *Nicola Cotumaccio*: Funded by the Helsinki Institute for Information Technology (HIIT). *Nicola Prezza*: Funded by the European Union (ERC, REGINDEX, 101039208). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Sharma V. Thankachan: Partially supported by NSF under grants CCF-2316691 and CCF-2315822.

1 Notation

We refer the reader to [52] for basics on automata theory.



© Nicola Cotumaccio, Giovanna D'Agostino, Daniel Gibney, Alberto Policriti, Nicola Prezza, and Sharma V. Thankachan;

licensed under Creative Commons License CC-BY 4.0

The Expanding World of Compressed Data: A Festschrift for Giovanni Manzini's 60th Birthday.

Editors: Paolo Ferragina, Travis Gagie, and Gonzalo Navarro; Article No. 12; pp. 12:1–12:28



OpenAccess Series in Informatics

OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Sequences

A sequence (or string) $\alpha \in \Sigma^*$ is a concatenation of characters from a finite alphabet Σ of size σ . Indices start from 1: $\alpha[i]$, with $i \geq 1$ is the i -th character of the sequence. Notation $\alpha[i : j]$, with $j \geq i$, denotes $\alpha[i]\alpha[i+1]\dots\alpha[j]$.

1.2 Automata and Labeled Graphs

A *finite (nondeterministic) automaton* (NFA for brevity) is a tuple $\mathcal{A} = (Q, s, \Sigma, \delta, F)$ where Q is a finite, non empty set of states, s is the initial state, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function, and $F \subseteq Q$ is the set of final states. We sometimes omit the alphabet Σ and simply write an NFA as a tuple $\mathcal{A} = (Q, s, \delta, F)$. Sometimes we shall describe the transition function δ using triples (edges), where (q, q', a) stands for $q' \in \delta(q, a)$.

A *semi-automaton* is an automaton that does not specify the initial state nor the set of final states. More formally, a semi-automaton is a triple $\mathcal{A} = (Q, \Sigma, \delta)$, where Q, Σ, δ are defined as above. Equivalently, one can view a semi-automaton as an edge-labeled graph $G = (Q, \Sigma, E)$ where $E = \{(q, q', a) \in Q \times Q \times \Sigma : q' \in \delta(q, a)\}$. When clear from the context, sometimes we will drop the alphabet Σ and simply write (Q, E) to denote a labeled graph. When talking about labeled graphs, we will refer to the elements of Q as *vertices* or *nodes*. Due to their equivalence, in this manuscript we will treat labeled graphs and semi-automata interchangeably. We denote with $|\delta|$ and $|Q|$ the number of transitions and states of a (semi-) automaton $\mathcal{A} = (Q, \Sigma, \delta)$, and with $|\mathcal{A}| = |Q| + |\delta|$ its total size. Similarly, $|G|$ denotes the size of graph $G = (Q, E)$, that is, the number $|Q|$ of its nodes plus the number $|E|$ of its edges.

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

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

The same definitions hold for semi-automata. Given a finite automaton \mathcal{A} , we denote by $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^* \mid \delta(s, \alpha) \cap F \neq \emptyset\}$ the language accepted (or recognized) by \mathcal{A} . We say that two automata are *equivalent* if they accept the same language.

Given a NFA $\mathcal{A} = (Q, s, \Sigma, \delta, F)$ and any of its states $u \in Q$, we denote with symbol I_u the set of all strings labeling paths starting from the source of \mathcal{A} and ending in u : $I_u = \{\alpha \in \Sigma^* : u \in \delta(s, \alpha)\}$.

A finite automaton is *deterministic* (for brevity, DFA) if $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. In the deterministic case we consider the transition function as a (possibly partial) function $\delta : Q \times \Sigma \rightarrow Q$ (hence, our DFA do not need to be complete). When δ is not defined on a pair (q, a) we write $\delta(q, a) = \perp$.

Unless otherwise stated, we assume that every finite automaton is *trimmed*, that is, every state is reachable from the initial state and can reach at least one final state. This assumption is not restrictive: every automaton can be put in an equivalent trimmed form in linear time.

For trimmed automata the following hold:

- there is at most one state without incoming edges, namely s ;
- every string that can be read starting from s belongs to the set of prefixes, $\text{Pref}(\mathcal{L})$, of the language \mathcal{L} .

The languages accepted by automata form the class of *regular languages* and are closed under boolean operations (union, intersection, complementation), concatenation, and the Kleene star.

Let $\mathcal{A} = (Q, s, \delta, F)$ be a NFA and $q \in Q$. The *input language* of q , denoted by I_q , is the set of strings leaving s and entering q , that is, I_q is the language recognized by the automaton $(Q, s, \delta, \{q\})$. We will refer to I_q also as the *set of strings reaching* q . Similarly, the *output language* of q , denoted by O_q is the language recognized by the automaton (Q, q, δ, F) .

Given a regular language, there exists a unique, up to isomorphism, state-wise minimum DFA that recognizes such language. The states of this minimum DFA correspond to the classes of the Myhill-Nerode equivalence relation, defined as follows.

► **Definition 1** (Myhill-Nerode equivalence [48]). *Let $\mathcal{L} \subseteq \Sigma^*$ be a language. Given a string $\alpha \in \Sigma^*$, we define the right context of α as*

$$\alpha^{-1}\mathcal{L} = \{\gamma \in \Sigma^* \mid \alpha\gamma \in \mathcal{L}\},$$

and we denote by $\equiv_{\mathcal{L}}$ the Myhill-Nerode equivalence on $\text{Pref}(\mathcal{L})$ defined as

$$\alpha \equiv_{\mathcal{L}} \beta \iff \alpha^{-1}\mathcal{L} = \beta^{-1}\mathcal{L}.$$

If \mathcal{L} is a regular language we will denote by $\mathcal{D}_{\mathcal{L}} = (Q, s, \delta, F)$ its minimum automaton, having as set of states Q the classes $[\alpha]_{\mathcal{L}}$ of the Myhill-Nerode equivalence, $s = [\varepsilon]_{\mathcal{L}}$, $\delta([\alpha]_{\mathcal{L}}, a) = [\alpha \cdot a]_{\mathcal{L}}$, and $F = \{[\alpha]_{\mathcal{L}}^c \mid \alpha \in \mathcal{L}\}$.

It is customary to define a version of the Myhill-Nerode equivalence relation on the states of a DFA by putting into the same class states u, v whose strings in I_u, I_v are Myhill-Nerode equivalent. More formally, one can define the following relation:

► **Definition 2** (Myhill-Nerode equivalence on DFA). *Let $\mathcal{D} = (Q, s, \delta, F)$ be a DFA. We denote by $\equiv_{\mathcal{D}}$ the Myhill-Nerode equivalence on Q defined as*

$$u \equiv_{\mathcal{D}} v \iff (\forall \alpha \in \Sigma^*)(\delta(u, \alpha) \in F \iff \delta(v, \alpha) \in F).$$

The minimum DFA for a regular language \mathcal{L} can be equivalently obtained by collapsing $\equiv_{\mathcal{D}}$ -equivalent states of any DFA \mathcal{D} recognizing \mathcal{L} . Hopcroft's algorithm [41] achieves this goal efficiently by computing $\equiv_{\mathcal{D}}$ in $O(|\mathcal{D}| \log |\mathcal{D}|)$ time.

1.3 Orders

We assume that there is a fixed total order \preceq on the alphabet Σ . We extend \preceq to strings in Σ^* *co-lexicographically*, that is, for $\alpha, \beta \in \Sigma^*$, we have $\alpha \preceq \beta$ if and only if either α is a suffix of β (denoted by $\alpha \dashv \beta$), or there exist $\alpha', \beta', \gamma \in \Sigma^*$ and $a, b \in \Sigma$, such that $\alpha = \alpha'a\gamma$ and $\beta = \beta'b\gamma$ and $a \prec b$.

If (Z, \leq) is a partial order, we denote by $(Z, <)$ its corresponding strict partial order. Given a partial order (Z, \leq) we say that a subset $I \subseteq Z$ is *convex* if, for any $x, y, z \in Z$ with $x < y < z$, if $x, z \in I$ then $y \in I$.

2 Wheeler Graphs

Gagie, Manzini, and Sirén in [35] showed a natural way to extend the co-lexicographic order of strings to labeled graphs. The idea is simple: let us focus on the class of labeled graphs for which there exists a total order of the nodes that is propagated along pairs of equally-labeled transitions, further requiring that pairs of nodes whose incoming labels differ are sorted consistently with the underlying total order of Σ . Due to the fact (as shown next) that these graphs generalize the celebrated *Burrows-Wheeler transform* [16] (*BWT* for short), the authors of [35] decided to call such a class *Wheeler graphs*. More formally:

► **Definition 3** (Wheeler Graph). A Wheeler graph is a labeled graph $G = (Q, \Sigma, E)$ endowed with a binary relation $<$ such that: $(Q, <)$ is a linear order where all nodes with in-degree equal to zero are smaller than nodes with in-degree strictly larger than zero, and for which the following two Wheeler properties (sometimes also called Wheeler axioms) are satisfied. Let $(u, u', a), (v, v', b) \in E$:

(W1) $a < b \rightarrow u' < v'$

(W2) $(a = b \wedge u < v \wedge u' \neq v') \rightarrow u' < v'$.

In this manuscript it will be useful to adapt the above definition to finite automata, in order to study the class of regular languages recognized by automata whose transition relation is a Wheeler graph.

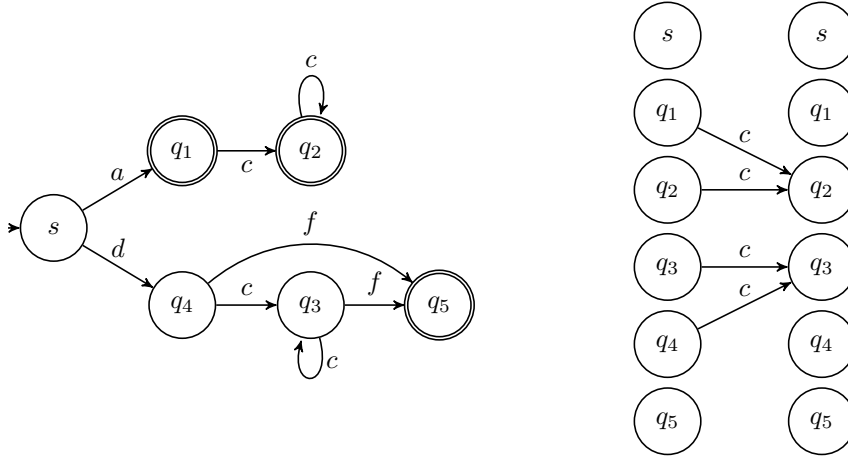
► **Definition 4** (Wheeler Automaton). A Wheeler automaton \mathcal{A} (WNFA for brevity) is a NFA (Q, s, δ, F) endowed with a binary relation $<$ such that: $(Q, <)$ is a linear order having the initial state s as minimum, s has no in-going edges, and the following two Wheeler properties (or Wheeler axioms) are satisfied. Let $p' \in \delta(p, a)$ and $q' \in \delta(q, b)$:

(W1) $a < b \rightarrow p' < q'$

(W2) $(a = b \wedge p < q \wedge p' \neq q') \rightarrow p' < q'$.

A Wheeler DFA (WDFA) is a WNFA in which $|\delta(q, a)| \leq 1$, for all $q \in Q$ and $a \in \Sigma$.

An example of a WDFA is presented in Figure 1 (left). In the same figure (right) we show a very insightful representation of Wheeler automata, called the *bipartite representation*: we duplicate states in two columns following any Wheeler order of the automaton, and we draw edges from left to right (in Figure 1, for clarity only edges labeled with c are shown). In this representation, Wheeler Axiom W1 translates to the fact that nodes are sorted by increasing incoming letter, and Wheeler Axiom W2 to the fact that no same-letter edges cross. Most properties, algorithms, and data structures on Wheeler graphs can be easily understood by keeping in mind this representation.



■ **Figure 1** Left: A WDFA \mathcal{D} recognizing the language $\mathcal{L}_d = ac^* \cup dc^*f$. The only order that makes \mathcal{D} Wheeler is $s < q_1 < q_2 < q_3 < q_4 < q_5$. Right: Bipartite representation of the Wheeler automaton \mathcal{D} , showing only edges labeled with c for clarity. In this representation, Axiom W2 translates to the fact that no same-letter edges cross.

Of particular interest for this survey is the class of Wheeler languages:

► **Definition 5** ([3]). *A regular language is said to be Wheeler if there exists a WNFA recognizing it.*

As a matter of fact, nondeterminism is not important in Definition 5. As proved in [3] (we will come back to this in Section 4), WDFA and WNFA have the same expressive power: both such classes of finite automata recognize precisely the class of Wheeler languages.

The following considerations hold both for Wheeler graph and Wheeler automata, so we state them only for the latter class of combinatorial objects.

First, we observe that the automata for which there exists a total order satisfying Axiom W2 correspond to the class of totally-sortable *Ordered Automata*, studied by Shyr and Thierrin in 1974 [53]. In that article, the authors showed that the languages recognized by such automata are star-free; in particular, it follows that all Wheeler languages are star-free. As discussed in this manuscript, by imposing the additional constraint on nodes with in-degree equal to zero and the additional Axiom W1, the work [35] unveiled a restricted class of Ordered Automata (the Wheeler automata) possessing a large number of extremely interesting properties.

Also observe that a consequence of Wheeler Axiom (W1) is that \mathcal{A} is *input-consistent*, that is all transitions entering a given state $q' \in Q$ must have the same label: if $q' \in \delta(q, a)$ and $q' \in \delta(p, b)$, then $a = b$. Therefore a function $\lambda : Q \rightarrow \Sigma$ that labels each state with the unique label of its incoming edges, can be introduced. For the initial state s , the only one without incoming edges, we set $\lambda(s) = \# \notin \Sigma$, stipulating that $\# \prec a$, for all $a \in \Sigma$. Observe that this simplifies the definition of Wheeler automata in that, by replacing a and b with $\lambda(p')$ and $\lambda(q')$ in Definition 4, we no longer need to require that states with in-degree equal to zero are smaller than states with in-degree strictly larger than zero. We decided to adopt the formulation of Definitions 3 and 4 since it is closer to the one originally defined in [35].

For a fixed-sized (constant) alphabet, requiring an automaton to be *input-consistent* is not computationally demanding. In fact, given an NFA $\mathcal{A} = (Q, s, \delta, F)$ we can build an equivalent, input-consistent one just by creating, for each state $q \in Q$, at most $|\Sigma|$ copies of q , that is, one for each different incoming label of q . This operation can be performed in $O(|Q| \cdot |\Sigma|)$ time.

3 Compact Data Structures on Wheeler Graphs

Wheeler graphs extend the suffix array [45], the Burrows-Wheeler Transform [16] and Ferragina and Manzini's FM-index [34] from strings to graphs. Following [35], we now show how to encode Wheeler graphs using a small number of bits. Consider the automaton in Figure 1. We store the following strings:

- The string **LAB** = *adccfcf* is obtained by concatenating the labels of all edges leaving s , the labels of all edges leaving q_1 , the labels of all edges leaving q_2 , and so on. If a node has multiple outgoing edges, the corresponding labels are sorted in alphabetic order.
- The string **OUT** = 10010101001001 is obtained by storing in unary the outdegree of s , the outdegree of q_1 , the outdegree of q_2 , and so on. The outdegrees are 2, 1, 1, 2, 2, 0.
- The string **IN** = 11010010010100 is obtained by storing in unary the indegree of s , the indegree of q_1 , the indegree of q_2 , and so on. The indegrees are 0, 1, 2, 2, 1, 2.
- The bit array **FIN** = 011001 remembers which states are final: s is not final, q_1 is final, q_2 is final, and so on.

Let $e = |\delta|$ be the number of transitions in the NFA, $n = |Q|$ be the number of nodes and $\sigma = |\Sigma|$ be the size of the alphabet. We can store **LAB** in $e \log \sigma$ bits, **OUT** in $e + n$ bits, **IN** in $e + n$ bits and **FIN** in n bits. It turns out that these sequences provide a compact encoding

of the Wheeler graphs that extends the Burrows-Wheeler transform from strings to Wheeler automata. Let us show how we can use these sequences to retrieve the automaton. By using LAB and OUT, we infer that s has two outgoing edges labeled a and d , q_1 has an outgoing edge labeled c , q_2 has an outgoing edge labeled c , q_3 has two outgoing edges labeled c and f , q_4 has two outgoing edges labeled c and f , and q_5 has no outgoing edges. Now we sort the character in LAB, obtaining $acccdf f$. From Wheeler property (W1) and IN we infer that s has no incoming edges, q_1 has an incoming edge labeled a , q_2 has two incoming edges both labeled c , q_3 has two incoming edges both labeled c , q_4 has an incoming edge labeled d , and q_5 has two incoming edges labeled f . By using the bipartite representation of Figure 1, we can retrieve the whole set of edges. For example, consider character c . We know that there are outgoing edges labeled c from q_1 , q_2 , q_3 , q_4 , and we know that there are incoming edges labeled c from q_2 (two edges) and q_3 (two edges), so since same-letter edges cannot cross we conclude that the edges labeled c are (q_1, q_2, c) , (q_2, q_2, c) , (q_3, q_3, c) , (q_4, q_3, c) . Lastly, we retrieve the set of all final states by using FIN.

Not only Wheeler graphs can be encoded in a small number of bits as described above, but they also support efficient pattern matching queries on the automaton's paths (equivalently, on the language recognized by the automaton). Consider the following pattern matching problems (where we assume, without loss of generality, that every state can reach a final state):

1. Given $\alpha \in \Sigma^*$, decide whether α is accepted by the automaton \mathcal{A} (that is, decide, whether $\alpha \in \mathcal{L}(\mathcal{A})$).
2. Given $\alpha \in \Sigma^*$, decide whether α occurs on the paths of the automaton \mathcal{A} (that is, decide whether there exist $\gamma_1, \gamma_2 \in \Sigma^*$ such that $\gamma_1 \alpha \gamma_2 \in \mathcal{L}(\mathcal{A})$).

For example, in Figure 1, the string $\alpha = cccf$ is not accepted by the automaton, but it occurs in the automaton (there is a path from q_4 to q_5 labeled with α).

The compressed representation of a Wheeler graph allow solving both problems efficiently. First, as observed in [35], from the bipartite representation of a graph we can deduce *path-coherence*: if we start from an interval of consecutive nodes in Wheeler order, we consider a character c , and we follow all edges labeled with c from those nodes, we end up in another (possibly empty) *interval* of consecutive nodes. For example, if we start from $\{q_1, q_2, q_3, q_4\}$ and we follow all edges labeled c , we end up in $\{q_2, q_3\}$.

This property implies that we can solve both pattern matching queries above listed in a linear number of steps. For the first problem, we start from the interval of states $\{s\}$, and for the second problem we start from the whole set of states Q . To update an interval by following the edges labeled with c , we augment the compressed representation of a Wheeler automaton with compressed data structures that support efficient rank/select queries, thus extending the FM-index from strings to Wheeler automata. This leads to the following result:

► **Theorem 6** (adapted from [35]). *Let \mathcal{A} be a Wheeler NFA with e edges on an alphabet of size σ . Then, \mathcal{A} can be encoded by using a data structure of $e \log \sigma(1 + o(1)) + O(e)$ bits that supports pattern matching queries in $O(m \log \log \sigma)$ time, where m is the length of the string to be matched.*

Wheeler graphs are therefore a very convenient class of graphs, because they support pattern matching in linear time (for constant alphabets) within compressed space, while on arbitrary graphs pattern matching cannot be solved in subquadratic time (assuming that the orthogonal vector hypothesis is true) [32]. At the same time, the class of Wheeler graphs subsumes all previous extensions of the Burrows-Wheeler transform and the FM-index, including labeled trees [33], circular strings [46, 40] and de Bruijn graphs [14]. Wheeler

graphs also inherit compressibility properties of the Burrows-Wheeler transform. For example, *tunneling*, a method for identifying blocks that capture some redundancy of the BWT [8], can be extended to Wheeler graphs [4].

Some more complicated pattern matching problems require more advanced data structures. A typical example is the problem of computing *matching statistics*: given an automaton \mathcal{A} , and a string α , determine for every $1 \leq i \leq |\alpha|$ the longest suffix α_i of $\alpha[1, i]$ that occurs in \mathcal{A} . For example, in Figure 1, if $\alpha = accfdf$, we have $\alpha_1 = a$, $\alpha_2 = ac$, $\alpha_3 = acc$, $\alpha_4 = cc$, $\alpha_5 = d$, $\alpha_6 = df$. On strings, this problem can be solved by exploiting the FM-index and the *longest common prefix array* [49]. In [19], the notion of longest common prefix array was extended to Wheeler DFA. The idea is to consider the smallest and the largest string reaching each state. For example, in Figure 1, the smallest string reaching q_2 is ac and the largest string reaching q_2 is $\dots ccccc$ (these strings can be read by starting from q_2 , following edges in a backward fashion until either reaching the source s or a loop, and finally reversing the obtained string). More generally, we proceed as follows. Let \mathcal{A} be a Wheeler DFA with n nodes, in which the i -th state in the Wheeler order is q_i . We define $2n$ strings: γ_1 is the smallest string reaching q_1 , γ_2 is the largest string reaching q_1 , γ_3 is the smallest string reaching q_2 , γ_4 is the largest string reaching q_2 , and so on. We define the array $\text{LCS}[2, 2n]$ such that $\text{LCS}[i]$ is the length of the longest common suffix between γ_{i-1} and γ_i . Then, it can be shown that, for every $2 \leq i \leq 2n$, either $\text{LCS}[i]$ is infinite or its value is less than $2n$ [19, 5]. This implies that LCS can be stored using at most $O(n \log n)$ bits.

The LCS array of a Wheeler DFA supports computing matching statistics efficiently:

► **Theorem 7** ([19]). *Let \mathcal{A} be a Wheeler DFA with n states. By storing the LCS array of \mathcal{A} using $O(n \log n)$ bits, we can compute matching statistics in $O(m \log n)$ time, where m is the length of the string to be matched.*

Storing a Wheeler DFA requires only $e \log \sigma(1 + o(1)) + O(e)$ bits (Theorem 6), so storing the LCS array using $O(n \log n)$ bits may require significantly more space than storing the automaton itself, if the alphabet and the numbers of edges are small. Similarly to the string case, it is possible to design a sampling mechanism for the LCS array that allows better space-time trade-offs. The key idea is to store a range minimum query data structure on the LCS array (which only requires $O(n)$ bits) and sample $O(n/\log n)$ entries of the LCS array in such a way that, by solving range minimum queries, it is possible to retrieve each entry of the LCS array in $O(\log n)$ steps. This leads to the following result.

► **Theorem 8** ([26]). *Let \mathcal{A} be a Wheeler DFA with n states. By storing a data structure of $O(n \log \log \sigma)$ bits, we can compute matching statistics in $O(m \log^2 n)$ time, where m is the length of the string to be matched.*

4 Wheeler Languages

Given the interesting properties of Wheeler automata discussed in the previous section, a natural language-theoretic question is: which languages are recognized by Wheeler DFA (WDFA for brevity) and Wheeler NFA (WNFA for brevity)? In this section we consider the class of *Wheeler Languages*, that is the class of languages recognized by a WNFA or, equivalently as we shall see in Theorem 10, by a WDFA.

A first observation is that the Wheeler *local* conditions of Definition 4 are equivalent, on DFA, to a more global condition expressed in terms of the sets I_q of words reaching a state q . If \mathcal{D} is a DFA and \prec is a (strict) order of the alphabet, we can define a (strict) partial

order $<_{\mathcal{D}}$ on its states by stipulating, for $q \neq q'$, that

$$q <_{\mathcal{D}} q' \Leftrightarrow \forall \alpha \in I_q \forall \beta \in I_{q'} (\alpha \prec \beta)$$

The (in general) partial order $<_{\mathcal{D}}$ indicates whether \mathcal{D} is Wheeler in the following sense:

► **Lemma 9** ([25]). *Let \mathcal{D} be an input-consistent DFA in which the initial state s has no in-going edges. Then*

$$\mathcal{D} \text{ is a WDFA if and only if the order } <_{\mathcal{D}} \text{ is total.} \quad (1)$$

Using the previous lemma we easily see that we can enlarge the class of WDFA to non input-consistent automata, without changing the class of recognized languages, by defining a WDFA as a DFA in which the initial state s has no in-going edges and the order \mathcal{D} is total. In the following we will freely use the new enlarged class of WDFA.

Notice that if $<_{\mathcal{D}}$ is total, then we can decide whether $q <_{\mathcal{D}} q'$ by simply checking the relative co-lexicographical order of any two strings $\alpha \in I_q$ and $\beta \in I_{q'}$. An easy consequence of this is that, if \mathcal{D} is a WDFA and $(\alpha_n)_{n \in \omega}$ is a co-lexicographically (strictly) monotone sequence of words in $\text{Pref}(\mathcal{L}(\mathcal{D}))$, then for some state q of \mathcal{D} , $(\alpha_n)_{n \in \omega}$ will eventually belong to I_q . This fact helps us to locate Wheeler languages among sub-regular languages by testing it against a very well-known class: the star-free languages. Languages in this class can be described by regular expressions constructed from the letters of the alphabet, the empty word, the empty set symbol, all boolean operators – including complementation – and concatenation but no Kleene star. We can indeed prove that a Wheeler language is always star-free. To see this we recall that a characterization of the star-free languages says that a language \mathcal{L} is star-free if and only if there exists n such that for all $\alpha, \beta, \gamma \in \Sigma^*$ it holds:

$$\alpha\beta^n\gamma \in \mathcal{L} \Rightarrow \forall m \geq n \quad \alpha\beta^m\gamma \in \mathcal{L}. \quad (2)$$

Suppose now that a language \mathcal{L} is Wheeler, that is, there exists a WDFA \mathcal{D} recognizing \mathcal{L} . Consider a sequence of words of the form $(\alpha\beta^k)_{k \in \omega}$: this sequence is always monotone (increasing or decreasing depending on whether $\alpha \preceq \alpha\beta$ or $\alpha \succeq \alpha\beta$). Hence, by the Wheelerness of \mathcal{D} , there must exist a \mathcal{D} -state q and $n \in \mathbb{N}$ such that, for all $m \geq n$, $\alpha\beta^m$ belongs to I_q . Condition (2) easily follows.

As a natural next step we present below a characterization of the class of Wheeler languages, both from the language as well as from the accepting-automata view points.

The former can be given by making an appeal to the celebrated Myhill-Nerode equivalence $\equiv_{\mathcal{L}}$. To this end we say that an equivalence relation \equiv over an ordered set (A, \leq) is *convex* if whenever $a \leq b \leq c$ and $a \equiv c$, we also have $b \equiv a$. Consider the following “convex refinement” $\equiv_{\mathcal{L}}^c$ of $\equiv_{\mathcal{L}}$. For all $\alpha, \beta \in \text{Pref}(\mathcal{L})$ we say that $\alpha \equiv_{\mathcal{L}}^c \beta$ if and only if

$$\text{end}(\alpha) = \text{end}(\beta) \wedge (\forall \gamma \in \text{Pref}(\mathcal{L})) (\min\{\alpha, \beta\} \preceq \gamma \preceq \max\{\alpha, \beta\} \rightarrow \gamma \equiv_{\mathcal{L}} \alpha),$$

where $\text{end}(\alpha)$ is the final character of α when $\alpha \neq \epsilon$, and ϵ otherwise.

The following theorem shows that the convex equivalence $\equiv_{\mathcal{L}}^c$ plays exactly the role of the Myhill-Nerode equivalence when referred to Wheeler Languages.

► **Theorem 10** (Myhill-Nerode for Wheeler Languages [2, 3]). *Given a language $\mathcal{L} \subseteq \Sigma^*$, the following are equivalent:*

1. \mathcal{L} is a Wheeler language (i.e. \mathcal{L} is recognized by a WNFA).
2. $\equiv_{\mathcal{L}}^c$ has finite index.

3. \mathcal{L} is a union of classes of a convex, input-consistent, right invariant, finite index equivalence relation over $(\text{Pref}(\mathcal{L}), \prec)$.
4. \mathcal{L} is recognized by a WDFA.

As a consequence of Theorem 10 we get an elegant further characterization of Wheeler languages by considering, again, monotone sequences. The condition expresses the fact that they turn out being eventually trapped in a single state, this time of the minimum DFA for the language:

► **Corollary 11** ([2, 3]). *A regular language \mathcal{L} is Wheeler if and only if all monotone sequences in $(\text{Pref}(\mathcal{L}), \preceq)$ become eventually constant modulo $\equiv_{\mathcal{L}}$. In other words, for all sequences $(\alpha_i)_{i \in \omega}$ in $\text{Pref}(\mathcal{L})$ such that:*

$$\alpha_1 \preceq \alpha_2 \preceq \dots \preceq \alpha_i \preceq \dots \quad \text{or} \quad \alpha_1 \succeq \alpha_2 \succeq \dots \succeq \alpha_i \succeq \dots,$$

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

Proof Sketch: to see that the condition is sufficient we use Theorem 10 and suppose, by contraposition, $\equiv_{\mathcal{L}}^c$ has an infinite number of classes. Then, there is a single $\equiv_{\mathcal{L}}$ -class which is partitioned into an infinite number of $\equiv_{\mathcal{L}}^c$ -classes and we can single out a monotone sequence $(\beta_i)_{i \in \omega}$ (say, increasing) whose elements are pairwise non $\equiv_{\mathcal{L}}^c$ -equivalent words. By discarding at most a finite number of them, we may suppose that all β_i are not only $\equiv_{\mathcal{L}}$ -equivalent, but also end with the same letter. For any $i \in \omega$, since $\beta_i \not\equiv_{\mathcal{L}}^c \beta_{i+1}$, by definition of $\equiv_{\mathcal{L}}^c$, there must exist η_i with $\beta_i \prec \eta_i \prec \beta_{i+1}$ and $\eta_i \not\equiv_{\mathcal{L}} \beta_i$. But then the monotone sequence

$$\beta_1 \prec \eta_1 \prec \beta_2 \prec \eta_2 \prec \dots$$

is never trapped in a single $\equiv_{\mathcal{L}}$ class.

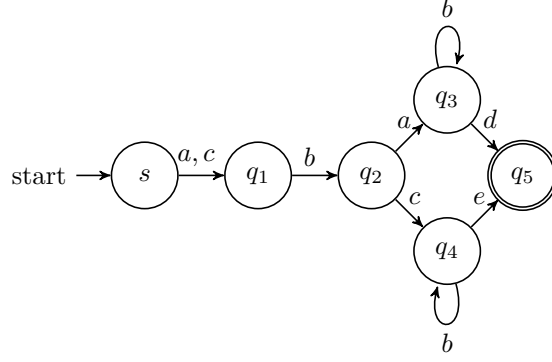
The condition is also necessary, since a monotone sequence gets trapped in any WDFA recognizing \mathcal{L} , it gets a fortiori trapped in the minimum DFA $\mathcal{D}_{\mathcal{L}}$ recognizing the language (even if $\mathcal{D}_{\mathcal{L}}$ is not Wheeler).

Using Corollary 11 one can show that Wheeler languages are closed under a few classical operations while other (even very simple) compositions of Wheeler languages do not maintain Wheelerness.

► **Lemma 12** ([3]). *The following hold:*

1. *Finite and co-finite languages are Wheeler.*
2. *The intersection of two Wheeler languages is Wheeler.*
3. *Wheeler languages are not closed under union, concatenation, Kleene star.*
4. *The union of a Wheeler language with a finite set is Wheeler.*
5. *The right-concatenation of a Wheeler language with a finite set is Wheeler.*

We now consider the problem of characterizing Wheeler languages using automata. For star-free languages we know that a combinatorial property of the minimum DFA – i.e. being *counter-free* – characterizes membership in that class and we can give a similar result also for the Wheeler class. However, let us first notice that being Wheeler for a language \mathcal{L} does not necessarily imply Wheelerness for its minimum DFA $\mathcal{D}_{\mathcal{L}}$. Consider the DFA \mathcal{D} depicted in Figure 2. If \preceq is the alphabetical order on Σ , since $a \prec ab \prec ac$ and since $a, ac \in I_{q_1}$, $ab \in I_{q_2}$, it follows that states q_1, q_2 are $<_{\mathcal{D}}$ -incomparable. Hence \mathcal{D} is not Wheeler, even though $\mathcal{L}(\mathcal{D})$ is Wheeler. To see this just consider the DFA \mathcal{D}' obtained from \mathcal{D} by duplicating state q_1 into $q_{1,a}$ and $q_{1,c}$ and by defining $\delta(s, a) = q_{1,a}$, $\delta(s, c) = q_{1,c}$, and $\delta(q_{1,a}, b) = \delta(q_{1,c}, b) = q_2$. In \mathcal{D}' the order $<_{\mathcal{D}'}$ is total so that \mathcal{D}' is a WDFA recognizing $\mathcal{L}(\mathcal{D})$.



■ **Figure 2** The depicted DFA \mathcal{D} is the minimum DFA accepting the Wheeler language $\mathcal{L}(\mathcal{D})$. However, \mathcal{D} is not Wheeler since states q_1, q_2 are incomparable with respect to $<_{\mathcal{D}}$.

Hence, in order to decide whether a language \mathcal{L} is Wheeler we cannot simply rely on the Wheelerness of its minimum DFA $D_{\mathcal{L}}$. Yet, a simple combinatorial property of $<_{D_{\mathcal{L}}}$ characterizes Wheelerness. Even though there are Wheeler languages in which $<_{D_{\mathcal{L}}}$ is not total, in such cases $<_{D_{\mathcal{L}}}$ -incomparability must be limited to special kind of pairs. To illustrate this consider again the minimum DFA in Figure 2, this time with respect to the following order on the alphabet: $a \prec c \prec b \prec d \prec e$. The following theorem tells us that the language recognized is not Wheeler, because the two nodes q_3, q_4 are not only $<_{\mathcal{D}}$ -incomparable but are also infinitely *entangled*, since they are the starting point of two equally labeled cycles – this was not so using the alphabetical order since $q_3 \prec_{\mathcal{D}} q_4$ in that case.

Intuitively speaking, \mathcal{L} is *not* Wheeler if and only if there is no *finite* splitting of the states of $\mathcal{D}_{\mathcal{L}}$ that can possibly make $\mathcal{D}_{\mathcal{L}}$ Wheeler.

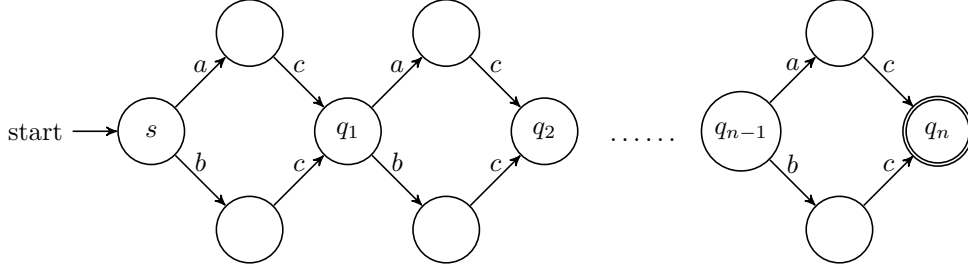
► **Theorem 13** ([3, 11]). *Let \mathcal{L} be a regular language and let $D_{\mathcal{L}}$ be the minimum trimmed DFA accepting \mathcal{L} . Then, \mathcal{L} is not Wheeler if and only if $D_{\mathcal{L}}$ contains two $<_{D_{\mathcal{L}}}$ -incomparable nodes $u \neq v$ and two equally labeled cycles, one starting from u and the other from v .*

As an easy corollary of the previous theorem we see at once that if the minimum trimmed DFA $D_{\mathcal{L}}$ accepting a regular language \mathcal{L} contains three equally labeled cycles starting from three different states q_1, q_2, q_3 , then \mathcal{L} cannot be Wheeler. Indeed, suppose w.l.o.g. that the cycles are labeled by the word γ and $\alpha_1 \prec \alpha_2 \prec \alpha_3$ are non empty words, different from γ , reaching q_1, q_2, q_3 , respectively. Then two out of the three among $\alpha_1, \alpha_2, \alpha_3$ must lie on the same side of γ , say $\alpha_1 \prec \alpha_2 \prec \gamma$. Then $\alpha_1\gamma \prec \alpha_2\gamma \prec \alpha_1\gamma^2$ with $\alpha_1\gamma, \alpha_1\gamma^2 \in I_{q_1}$ and $\alpha_2\gamma \in I_{q_2}$, showing that q_1, q_2 are $<_{D_{\mathcal{L}}}$ -incomparable and \mathcal{L} is not Wheeler by Theorem 13.

A similar combinatorial condition on the minimum DFA will be used in the following to find efficient algorithms for analyzing the “level of Wheelerness” of regular languages.

Finally, we consider the problem of constructing the minimum WDFA for a Wheeler language. Consider, for any $n \in \mathbb{N}$, the language whose minimum (input-consistent) DFA D_n is depicted in Figure 3. This language is finite, hence it is Wheeler – in fact, for *any* order of the alphabet. However, if we consider the standard alphabetical order, D_n is not Wheeler, because states q_1, q_2 cannot be ordered by $<_{D_n}$ since $acac \prec bc \prec bcbc$ with $acac, bcbc \in I_{q_2}$, $bc \in I_{q_1}$. However, Theorem 10 says that every Wheeler language has a unique state-minimum WDFA, whose set of states is the set of equivalence classes of $\equiv_{\mathcal{L}}^c$, while initial and final states are defined as in the regular case. For the language in Figure 3 it can be proved that the state-minimum WDFA has a number of states which is exponential

in the number n of states in the minimum (input-consistent) DFA D_n (see [47] for further consideration on the relative size of minimum DFA and WDFA for the same language). We will discuss algorithmic solutions to the problem of computing such minimum WDFA in Section 7.4.



■ **Figure 3** The depicted DFA is accepting a Wheeler (finite) language but the minimum Wheeler DFA accepting the same language has size exponential in n .

4.1 The *rational* embedding

A different “reading” of the effect of co-lexicographic ordering on the collection of strings accepted by a given Wheeler automaton can be given mapping strings on rational numbers in between zero and one (see [47]). To this end it is sufficient to use characters as digits whose positional value is decreasing right-to-left. The connection that this view will provide is based on the fact that the co-lex ordering of strings turns out to be the (familiar) ordering of the rationals onto which they map.

Consider the following definition:

► **Definition 14 (The Rational Embedding of Σ^*).** *The Rational Embedding of Σ^* is the map $\mathfrak{q} : \Sigma^* \rightarrow \mathbb{Q}[0, 1)$ defined as follows. If $\sigma = |\Sigma|$, for any $\alpha = \alpha_1 \dots \alpha_m \in \Sigma^*$:*

$$\mathfrak{q}(\alpha) = \sum_{i=1}^m \alpha_i \cdot (\sigma + 2)^{-(m-i+1)}.$$

The function \mathfrak{q} maps characters in digits and strings in rational numbers in $[0, 1)$ written in base $(|\Sigma| + 2)$, without using the smallest and the largest digit.

The mapping has the following elegant property:

$$\alpha \preceq \beta \text{ if and only if } \mathfrak{q}(\alpha) \leq \mathfrak{q}(\beta).$$

Letting $I_{\mathbb{Q}[0,1)}$ be the collection of non-empty convex sets of rationals in $\mathbb{Q}[0, 1)$:

$$I_{\mathbb{Q}[0,1)} = \{J \subseteq \mathbb{Q}[0, 1) \mid J \neq \emptyset \wedge (\forall a, c \in J)(\forall b \in \mathbb{Q})(a \leq b \leq c \Rightarrow b \in J)\},$$

we can map any collection of strings reaching a given state into a convex subset of rationals in $[0, 1)$.

► **Definition 15 (The Rational Embedding of an automaton).** *The Rational Embedding of $\mathcal{A} = (Q, s, \delta, F)$ is the map $I^{\mathfrak{q}} : Q \rightarrow I_{\mathbb{Q}[0,1)}$ defined as follows: for any $q \in Q$,*

$$I^{\mathfrak{q}}(q) = \bigcap \{J \in I_{\mathbb{Q}[0,1)} \mid (\forall \alpha \in I_q)(\mathfrak{q}(\alpha) \in J)\}.$$

In other words, $I^{\mathfrak{q}}(q)$ is the convex closure (convex hull) of $\{\mathfrak{q}(\alpha) \mid \alpha \in I_q\}$.

12:12 Wheeler Graphs and Wheeler Languages

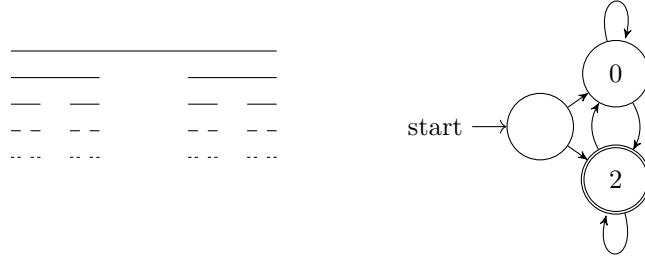
Denoting $I^q(q)$ by I_q^q we can restate Wheelerness as follows: for any pair of states $q_1, q_2 \in Q$, either the sup (right limit) of $I_{q_1}^q$ is smaller than the inf (left limit) of $I_{q_2}^q$, or vice versa.

Moreover, many further questions relating numbers and accepted strings arise. For example, are the inf and sup of I_q^q s always rational numbers? How many accumulation points can we observe by analyzing the embeddings of strings belonging to a regular language? Are accumulation points useful in recognizing Wheelerness?

The first of the above question has been tackled in [47], where the following has been proved:

► **Theorem 16.** *If $\mathcal{L} = \mathcal{L}(\mathcal{D})$, with \mathcal{L} Wheeler and \mathcal{D} either minimum or Wheeler, then for all $q \in Q$, we have that the inf and the sup of I_q^q are rationals.*

The above result was essentially already present in [38] where, however, no co-lexicographic ordering of strings was (explicitly) taken into account and where a real number was considered *defined* by an automaton when all its approximations are accepted by the automaton. Among other things, Hartmanis and Stearns observe that the measure zero Cantor set¹ can be defined by an automaton (see Figure 4).



■ **Figure 4** Cantor set and the automaton accepting it. Notice that the automaton on the right is trimmed: the underlying alphabet is $\Sigma = \{0, 1, 2\}$.

For distinct states $q, q' \in Q$, the presence of accumulation points reached by rationals in I_q^q and $I_{q'}^q$, respectively, can imply *non*-Wheelerness of a language (consider Figure 2 above and see Theorem 17 below). However, the mere presence of accumulation points can easily comply with Wheelerness: Cantor set, for example, can be accepted by a Wheeler automaton using Theorem 13 and the approach outlined here. Fixing $\Sigma = \{0, 1, 2, 3, 4\}$, the automaton \mathcal{C} obtained from the one in Figure 4 replacing 0 by 1 and 2 by 3 and making both its non-starting states final, is Wheeler and complying with Definition 14. Strictly speaking, since we are not using the first and last digits of Σ , none of the rational embeddings of strings in $\mathcal{L}(\mathcal{C})$, that is $I^q(\mathcal{L}(\mathcal{C}))$, is an accumulation point², however $I^q(\mathcal{L}(\mathcal{C}))$ retains most of the properties of the classical Cantor set. More general *Cantor-like* sets are, for example, the rational embeddings of *reverse definite* languages, that is languages $\mathcal{L} = F \cup G\Sigma^*$, for finite $F, G \subseteq \Sigma^*$, or *strictly locally testable* languages, that is languages $\mathcal{L} = F \cup (H\Sigma^* \cap \Sigma^*K) \setminus \Sigma^*W\Sigma^*$ for finite $H, K, W, F \subseteq \Sigma^*$.

In general, the rational embedding of a Wheeler language can have many accumulation points – think of the trivial case of $\mathcal{L} = \Sigma^*$ – and the analysis of their position within different I_q^q s, can provide useful insights. Consider, for example, the following sufficient condition for non-Wheelerness.

¹ The celebrated Cantor set can be seen as the set of rationals in $[0, 1]$ written in base 3 without using the digit 1. Is an example of *perfect* set (all its points are accumulation points) that is also *nowhere dense* (the interior of its closure is empty).

² Hence, $I^q(\mathcal{L}(\mathcal{C}))$ is not perfect.

► **Theorem 17** ([47]). *If \mathcal{D} is the minimum DFA accepting \mathcal{L} and for distinct $q, q' \in Q$,*

- *$x = \inf(I_q^{\mathcal{Q}}) = \inf(I_{q'}^{\mathcal{Q}})$ or $x = \sup(I_q^{\mathcal{Q}}) = \sup(I_{q'}^{\mathcal{Q}})$ and*
- *x is an accumulation point for both $I_q^{\mathcal{Q}}$ and $I_{q'}^{\mathcal{Q}}$,*

then \mathcal{L} is not Wheeler.

4.2 State Complexity

The *state complexity* (also known as *quotient complexity*) of a regular language is naturally defined as the number of states of the minimum DFA $\mathcal{D}_{\mathcal{L}}$ recognizing \mathcal{L} . State complexity is also used to measure the complexity of operations on regular languages: the state complexity of an operation is a function that, starting from the state complexities of the operands, associate the worst-case state complexity of the language resulting from the operation. For instance, on regular languages the state complexity of the intersection of \mathcal{L}_1 and \mathcal{L}_2 is mn , where m and n are the number of states of $\mathcal{D}_{\mathcal{L}_1}$ and $\mathcal{D}_{\mathcal{L}_2}$ respectively. This can be seen using the state-product construction for $\mathcal{D}_{\mathcal{L}_1}$ and $\mathcal{D}_{\mathcal{L}_2}$ and in [55] it is proved that this bound is tight. Moreover, as proved in [15], the same complexity is met for the intersection of star-free languages.

If we restrict to Wheeler languages, it is natural to define the Wheeler state complexity of a Wheeler language \mathcal{L} as the number of states of the minimum WDFA $\mathcal{D}_{\mathcal{L}}^W$ recognizing \mathcal{L} . The following lemma shows that the convex property of a Wheeler DFA can be exploited to prove that the Wheeler state complexity of the intersection of Wheeler languages is significantly better than the state complexity of the intersection for regular or star-free languages.

► **Lemma 18** ([30]). *Let \mathcal{D}_1 and \mathcal{D}_2 be two WDFA recognizing the languages \mathcal{L}_1 and \mathcal{L}_2 respectively. Then, the direct product $\mathcal{D} := \mathcal{D}_1 \times \mathcal{D}_2$ recognizing the language $\mathcal{L} := \mathcal{L}_1 \cap \mathcal{L}_2$ is Wheeler and, in its trimmed form, it has at most $|Q_1| + |Q_2| - 1$ states, where $|Q_i|$ is the cardinality of the set of states of \mathcal{D}_i .*

Here is an intuition for the proof of the previous lemma. The product of two DFA $\mathcal{D}_i = (Q_i, s_i, \delta_i, F_i)$ is the DFA $\mathcal{D}_1 \times \mathcal{D}_2 = (Q_1 \times Q_2, (s_1, s_2), \delta_1 \times \delta_2, F_1 \times F_2)$, where $\delta_1 \times \delta_2((q, r), \sigma) = (\delta_1(q, \sigma), \delta_2(r, \sigma))$. If the \mathcal{D}_i 's are Wheeler for $i = 1, 2$, say w.r.t. the Wheeler orders (Q_i, \leq_i) , then we can consider the colexicographic order on $Q_1 \times Q_2$:

$$(q_1, p_1) <_{\text{colex}} (q_2, p_2) \Leftrightarrow (p_1 <_2 p_2) \vee [(p_1 = p_2) \wedge (q_1 <_1 q_2)].$$

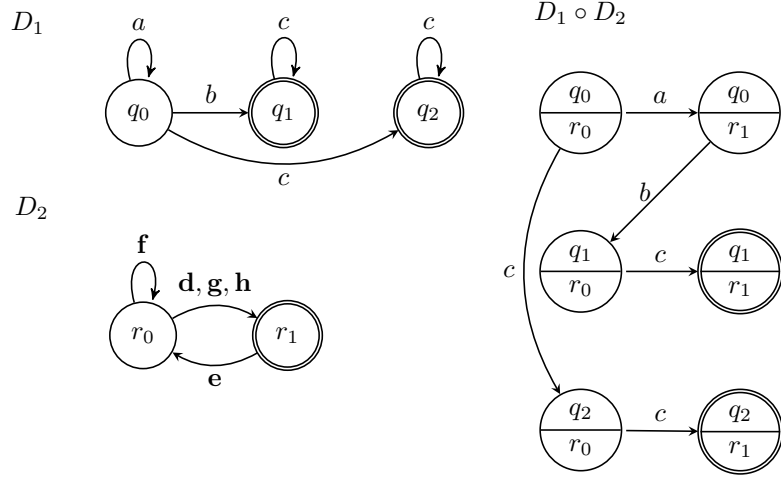
It is then possible to prove that the set of reachable states R in the product $\mathcal{D}_1 \times \mathcal{D}_2$ is such that if $(q_1, p_1) <_{\text{colex}} (q_2, p_2)$ with $(q_1, p_1), (q_2, p_2) \in R$ and $p_1 <_2 p_2$ then $q_1 \leq_1 q_2$ holds as well, and this allows to prove that $|R| \leq |Q_1| + |Q_2|$.

It is natural to ask whether there are more operations, beside intersection, preserving the Wheeler properties. As already mentioned, Wheeler DFA do not behave well when classic operations on DFA are concerned (booleans, concatenation, and Kleene star), so we have to look somewhere else. DFA being special cases of semi-groups, it is there where we can find an interesting operation, the so called *cascade product* of DFA's, and prove that it gets along well with Wheelerness. The cascade product is a generalization of the direct product, in which the parallelism of the computation between the two DFA in the product is substituted by a more hierarchical interaction. Indeed, the second DFA in the cascade does not only read the input string, but also read, moving a step later, the run that the first DFA makes on the input string. To do so, if the alphabet of the first DFA $\mathcal{D}_1 = (Q_1, s_1, \delta_1, F_1)$ is Σ , the alphabet of the second DFA must be $Q_1 \times \Sigma$. More formally, we define the *cascade product* as follows.

► **Definition 19** (Cascade product). Let $\mathcal{D}_1 = (Q_1, s_1, \delta_1, F_1 \times F_2)$ be a DFA over the alphabet Σ and let $\mathcal{D}_2 = (Q_2, Q_1 \times \Sigma, \delta_2, F_2)$ be a second DFA whose alphabet is the cartesian product of $Q_1 \times \Sigma$. The cascade product $\mathcal{D}_1 \circ \mathcal{D}_2 = (Q_1 \times Q_2, (s_1, s_2), \delta, F_1 \times F_2)$ is the automaton over the alphabet Σ with transition function defined by

$$\delta((q, r), a) = (\delta_1(q, a), \delta_2(r, (q, a))).$$

Figure 5 shows an example of cascade product between two automata.



■ **Figure 5** On the left the automata \mathcal{D}_1 and \mathcal{D}_2 , where the \mathcal{D}_2 -transitions are: $\mathbf{d} = (q_0, a)$, $\mathbf{e} = (q_0, b)$, $\mathbf{f} = (q_0, c)$, $\mathbf{g} = (q_1, c)$, $\mathbf{h} = (q_2, c)$. On the right, the cascade product $\mathcal{D}_1 \circ \mathcal{D}_2$.

Notice that the direct product between automata can be seen as a particular case of the cascade product: the direct product $\mathcal{D}_1 \times \mathcal{D}_2$ is equal to the cascade product $\mathcal{D}_1 \circ \mathcal{D}'_2$, where \mathcal{D}'_2 is obtained from \mathcal{D}_2 by replacing each transition $\delta_2(r, a) = r'$ with $|Q_1|$ transitions $\delta'_2(r, (q, a)) = r'$, one for each $q \in Q_1$.

The cascade product plays a fundamental role in the theory of automata (or semi-groups): the celebrated Krohn-Rhodes Decomposition Theorem ([44]) is in fact for semi-groups the analogous of the Jordan-Holder Decomposition Theorem ([6]), allowing to decompose any semi-group (automaton) as a cascade product of “basic” semi-groups (automata).

It is possible to prove that Wheelerness gets along very well with the cascade product. Indeed the following generalization of Lemma 18 holds:

► **Lemma 20** ([28]). If $\mathcal{D}_1 = (Q_1, \Sigma, s_1, \delta_1)$ is Wheeler with respect to the orders $(\Sigma, <)$ and $(Q_1, <_1)$ and $\mathcal{D}_2 = (Q_2, Q_1 \times \Sigma, s_2, \delta_2)$ is Wheeler with respect to the co-lexicographic order on the set of pairs $Q_1 \times \Sigma$ and the order $(Q_2, <_2)$ then $\mathcal{D}_1 \circ \mathcal{D}_2$ (restricted to accessible states) is a WDFA with respect to $(\Sigma, <)$ and the co-lexicographic order over $Q_1 \times Q_2$ (restricted to accessible states).

Moreover, if \mathcal{D}_1 has n_1 states and \mathcal{D}_2 has n_2 states then the cascade product $\mathcal{D}_1 \circ \mathcal{D}_2$ has at most $n_1 + n_2 - 1$ states.

In Sec 7.1 we shall see that this good behavior of the Wheeler class is not restricted to the intersection/cascade operation, as we find a significant improvement in the determinization of Wheeler NFA as well.

5 Hardness and Lower Bounds

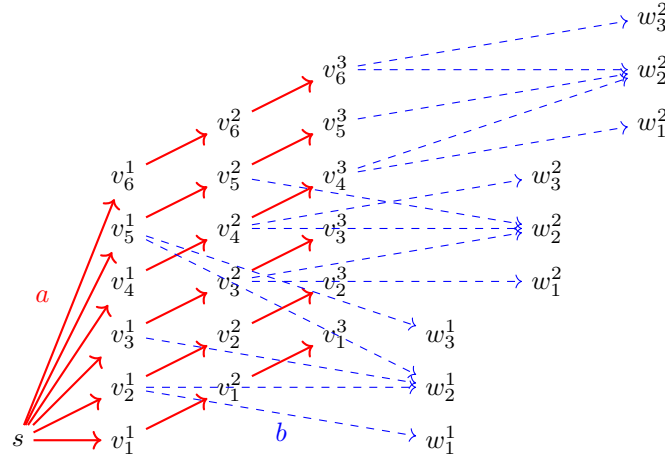
We now move to algorithmic problems on Wheeler graphs. We first discuss the computational complexity of the natural problem of determining if a given edge labeled graph is a Wheeler graph. The following hardness result by Gibney and Thankachan implies the computational intractability of this problem.

► **Theorem 21** ([36]). *Determining if a given labeled graph $G = (Q, \Sigma, E)$ is a Wheeler graph is NP-complete. This holds even when restricted to the class of graphs over binary edge label alphabets having a single source and with vertices having total degree (in-degree + out-degree) at most five.*

We describe a simple reduction that requires a source vertex with a large degree, but demonstrates many of the same techniques used to prove the full statement of Theorem 21. We first introduce the Betweenness Problem, proven NP-complete by Opatrny [50].

► **Problem 1** (Betweenness Problem). *Given an integer n and a set of triples $X \subseteq \{(a, b, c) \mid a, b, c \in \{1, 2, \dots, n\}\}$, determine if there exists a permutation π of $\{1, 2, \dots, n\}$ such that for every $(a, b, c) \in X$, either $\pi(a) < \pi(b) < \pi(c)$ or $\pi(c) < \pi(b) < \pi(a)$.*

We call the set of triples X *betweenness constraints*. For example, an instance of the Betweenness problem is $(n = 6, X = \{(3, 2, 5), (1, 5, 2), (4, 5, 6), (2, 6, 4)\})$. The permutation resulting in the ordering 1, 4, 5, 6, 2, 3 would satisfy every triple in X . The permutation resulting in the ordering 1, 2, 3, 4, 5, 6 would violate the triples $(3, 2, 5)$, $(1, 5, 2)$ and $(2, 6, 4)$.



■ **Figure 6** The labeled graph constructed from the Betweenness Problem instance $(n = 6, X = \{(2, 3, 5), (3, 5, 4), (4, 5, 6)\})$. This graph is a Wheeler graph if and only if there exists a permutation of $\{1, 2, \dots, n\}$ satisfying all betweenness constraints.

Given an instance (n, X) of the Betweenness Problem we construct a labeled graph $G = (Q, \Sigma, E)$ as follows:

- We create a single source vertex s .
- We next create a set of n vertices, denoted v_i^1 for $i \in [1, n]$ and the labeled edges (s, v_i^1, a) for $i \in [1, n]$.
- Next, for $j \in [2, |X|]$, $i \in [1, n]$, we create the vertex set v_i^j and the labeled edges (v_i^{j-1}, v_i^j, a) .

12:16 Wheeler Graphs and Wheeler Languages

Observe that in the definition of a total vertex order $<$ for a Wheeler graph, the vertex s must be ordered first, followed by some ordering of $v_1^1, v_2^1, \dots, v_n^1$, immediately followed by the same order applied to $v_1^2, v_2^2, \dots, v_n^2$. In fact, the vertices described above must have an ordering of the form

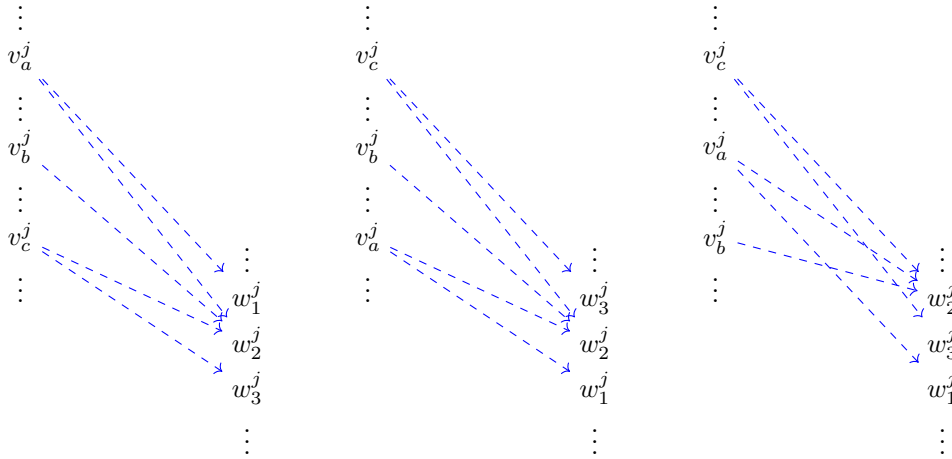
$$s < v_{\pi(1)}^1 < \dots < v_{\pi(n)}^1 < v_{\pi(1)}^2 < \dots < v_{\pi(n)}^2 < \dots < v_{\pi(1)}^{|X|} < \dots < v_{\pi(n)}^{|X|}$$

for some permutation π . Otherwise, there would exist some i' and i such that $v_{i'}^{j-1} < v_i^{j-1}$ but $v_i^j < v_{i'}^j$, contradicting Wheeler graph axiom W2.

Next, we add the following betweenness constraint gadgets. We order X arbitrarily. Iterating through X , consider (a, b, c) as the j^{th} betweenness constraint. We add the new vertices w_1^j, w_2^j , and w_3^j and the labeled edges (v_a^j, w_1^j, b) , (v_a^j, w_2^j, b) , (v_b^j, w_2^j, b) , (v_c^j, w_2^j, b) , and (v_c^j, w_3^j, b) . See Figure 6 for an example with $n = 6$ and three betweenness constraint gadgets.

The key property enforced by the betweenness constraint gadget is as follows. If the j^{th} constraint is (a, b, c) , then, for the total order $<$ on the vertices to satisfy Wheeler Axiom W2, the vertex v_b^j must be ordered between v_a^j and v_c^j . For non-example, if $v_b^j < v_a^j < v_c^j$ we must have $w_2^j < w_1^j < w_3^j$ due to the edges (v_b^j, w_2^j, b) , (v_a^j, w_1^j, b) , and (v_c^j, w_3^j, b) . However, we also have the edges (v_a^j, w_1^j, b) and (v_c^j, w_2^j, b) where $v_a^j < v_c^j$ and $w_2^j < w_1^j$, violating Wheeler axiom W2. A similar argument holds in all cases where v_b^j is not ordered between v_a^j and v_c^j . Taking the bipartite view of the graph from Section 2, this is equivalent to the only ordering of the vertices allowing for non-crossing dashed edges labeled 2 being one where the v_b^j is positioned between v_a^j and v_c^j . See Figure 7. By drawing similar figures, one can confirm that the only possible orderings for the vertices in which blue dashed lines do not cross are $v_a^j < v_b^j < v_c^j, w_1^j < w_2^j < w_3^j$ and $v_c^j < v_b^j < v_a^j, w_3^j < w_2^j < w_1^j$.

We note here that if the single source restriction is lifted, then by making each v_i^1 for $i \in [1, n]$ a source vertex, the reduction implies NP-completeness for the case where the total degree of each vertex is at most three.

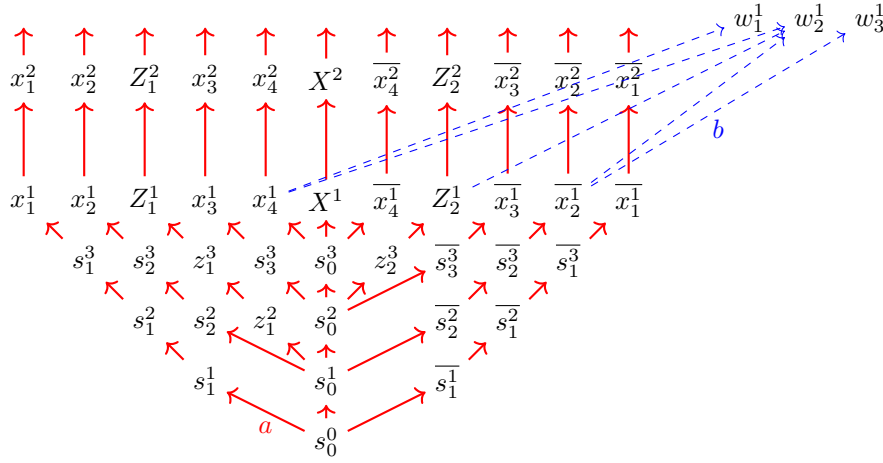


■ **Figure 7** The orderings represented in the left and middle figures are the only vertex orderings that result in no blue dashed edges crossing. The right figure shows an invalid ordering that violates this condition.

We next discuss how the complete statement of Theorem 21 is obtained in the single source case. An examination of Opatrný's NP-completeness proof of the Betweenness Problem shows that not all permutations of $\{1, 2, \dots, n\}$ have to be under consideration for NP-hardness

of the Betweenness problem to hold. Only a subset of permutations allowing for pivoting of elements around some central value X , along with some potential placement of some additional elements have to be under consideration for the problem to be NP-hard. This observation guides a reduction from Not-All-Equal SAT (NAESAT) that utilizes a tree structure rather than a single source with degree n , which previously had allowed for all permutations of v_1^1, \dots, v_n^1 . An example of this tree structure is shown in Figure 8.

In more detail, in the Not-All-Equal Boolean Satisfiability (NAESAT) Problem, one needs to find a satisfying Boolean assignment in which each clause has both a true literal and a false literal. Like in Opatrny's reduction from NAESAT to the Betweenness Problem, in the reduction from NAESAT to Wheeler graph recognition, each variable x_i is represented by two elements (or vertices) x_i and \bar{x}_i . Betweenness constraints force these to be in one of two orderings $x_i < X < \bar{x}_i$ or $\bar{x}_i < X < x_i$ corresponding to either a true or false assignment to x_i . Each clause requires one element (represented with a vertex) and two additional betweenness constraints. After constructing the tree structure, all betweenness constraints can be enforced in the same manner as in the reduction from the Betweenness Problem to Wheeler graph recognition. We refer the readers to [36] for details.



the assumption that $P \neq NP$. The proof is through a reduction from the Feedback Arc Set Problem, that is, given a directed graph $G = (V, E)$, determine a minimum cardinality subset of edges $E' \subset E$ such that $(V, E \setminus E')$ is a directed acyclic graph. This same reduction implies that, under the Unique Games Conjecture, it is NP-hard to find a solution to WGV that is within a constant factor of optimal for any constant $C > 0$ [37].

Further hardness results in this vein include work by D'Agostino et al. [30] that proves that deciding whether a language of an NFA is a Wheeler language is PSPACE-complete. The same authors demonstrate that the problem of determining whether there exists an alphabet order that causes a DFA to become Wheeler is NP-complete [29]. As a fine-grained complexity result, a quadratic lower bound condition on the Strong Exponential Time Hypothesis (SETH) for the problem of determining if a DFA recognizes a Wheeler language is provided by Becker et al. [11], along with an algorithm having a time complexity matching this lower bound. Section 7.3 provides more details on this last result.

6 Sorting Algorithms

As discussed in Section 5, the problems of (i) deciding whether a given NFA is a WNFA and (ii) finding a Wheeler order for a given WNFA are NP-complete. Subsequent research, however, has shown that this is not the end of the story: as we show next, problems (i) and (ii) can still be solved in polynomial time for a strict superclass of the DFA, and can actually be solved in polynomial time for all NFA by slightly changing the Wheeler order definition (i.e. moving to preorders, a solution which preserves the ability to index the WNFA).

6.1 Sorting WDFA and 2-WNFA

Alanko et al. in [2] showed that in the deterministic case (WDFA) the recognition and sorting problems are easy. This result is easily explained using Lemma 9 (Section 4). Intuitively, this Lemma states that a DFA \mathcal{A} is Wheeler if and only if, for any pair of states $u \neq v$ of \mathcal{A} , all strings labeling paths from the source of \mathcal{A} to u are co-lexicographically smaller than all strings labeling paths from the source of \mathcal{A} to v (or the other way round). If this is the case, then one can sort the states of \mathcal{A} according to any representative of I_u , for all $u \in Q$. In particular, one can build a spanning tree (rooted in the source) of \mathcal{A} and sort it using existing algorithms for sorting labeled trees (for example, [33]); if \mathcal{A} is Wheeler, then the resulting node order is a Wheeler order. If on the other hand, \mathcal{A} is not Wheeler, then one can easily check in linear time that the resulting order does not satisfy Definition 4. This yields:

► **Theorem 22** ([2]). *Let \mathcal{A} be a DFA. In $O(|\mathcal{A}|)$ time one can:*

1. *Decide whether \mathcal{A} is a Wheeler DFA, and*
2. *If \mathcal{A} is a Wheeler DFA, return its (unique) Wheeler order.*

Importantly, note that Lemma 9 does not hold for NFA: the nondeterministic case is, in fact, much more complicated.

Another interesting case covered in [2] is that of acyclic WDFA, which can be sorted *online* following any topological order of the edges (that is, after the i -th edge is received, the algorithm maintains internally the Wheeler order of all the nodes incident to the i edges received so far). The algorithm is a generalization of a folklore online algorithm for building the Burrows-Wheeler transform in an online fashion [39], and runs with $O(\log |\mathcal{A}|)$ delay per edge.

What about nondeterministic finite automata? While Gibney and Thankachan [36] established that the problem is NP-complete in general (i.e. NFA), a closer look reveals that every node of the labeled graph used in the reduction of [36] has at most five outgoing transitions labeled with the same character. Let us formalize this concept:

► **Definition 23** ([2]). *A d -NFA is a NFA such that, for every state u and every character $a \in \Sigma$, it holds $|\delta(u, a)| \leq d$.*

In other words, the work [36] established that 5-WNFA are NP-hard to recognize and sort. Note that 1-NFA correspond to DFA (recognizable and sortable in linear time, as discussed above), which leaves open the cases $2 \leq d \leq 4$. Surprisingly, [2] showed that also the case $d = 2$ admits a polynomial-time (quadratic) solution. The idea is to encode the candidate Wheeler order as a set of $|Q|^2$ boolean variables (each encoding the statement “ $u < v$ ” for every possible choice of $u, v \in Q$), and to enforce the Wheeler axioms of Definition 4 via a 2-SAT formula of size $O(|\mathcal{A}|^2)$ (solvable in time $O(|\mathcal{A}|^2)$). While Axioms 1-2 are easily expressible using only 2-SAT clauses, the tricky part turns out to be the totality requirement for the Wheeler order and, in particular, transitivity (which requires 3-SAT clauses in the general case). The core of the reduction shown in [2] is to prove that the Wheeler axioms, together with antisymmetry and totality of the order (also easily expressible via 2-SAT clauses), imply transitivity on 2-NFA. As a result, transitivity does not need to be enforced at all and a 2-SAT formula can be used to find a Wheeler order, if one exists.

2-SAT formulas can also be used to attack the sorting problem for the union of two Wheeler automata (see [31] for details).

6.2 Sorting Arbitrary WNFA: Preorders and Relations

As it turns out, the NP-completeness of the problem of recognizing and sorting WNFA with the Wheeler order defined in Definition 4, does not mean that such automata cannot be indexed for pattern matching queries in polynomial time. Another closer look at the NP-completeness reduction of Gibney and Thankachan [36] reveals that the labeled graph (in fact, a semiautomaton with a single source) of their reduction has another peculiarity: several distinct node pairs $u \neq v$ satisfy $I_u = I_v$, i.e. they are reached by exactly the same strings from the source. Observe that such pairs of nodes can safely be merged without affecting the strings that can be read on the graph’s paths; in particular, the answers to pattern matching existential queries (does a given query string α label any path in the graph?) are not affected by this transformation. One idea could therefore be to not take any decision on the relative order of such equivalent nodes, simply deeming them as being equivalent in the order. In other words, we may slightly change our goal: rather than looking for a *strict* Wheeler order, we may look for a *preorder*. As shown by Alanko et al. in [3], this road is indeed the right way to go. A particular case is the one where no equivalent states exist. In this case, Alanko et al. [3] proved:

► **Theorem 24** ([3]). *Let us call reduced a semi-automaton $\mathcal{A} = (Q, \Sigma, \delta)$ such that $I_u \neq I_v$ for all pairs of states $u, v \in Q$ with $u \neq v$. Then, in $O(|\delta| \cdot |Q|^2)$ time we can decide if \mathcal{A} is Wheeler and, if so, compute a Wheeler order for it.*

The algorithm behind Theorem 24 was named *Forward algorithm* due to the fact that it is a partition refinement algorithm “propagating forward” (in the direction of the automaton’s transitions) Wheeler axiom W2. At any point of the algorithm’s execution, a total order $<$ is maintained among the classes of the current partition of the states. The algorithm starts by partitioning the automaton’s states into $|\Sigma|$ classes, according to the labels of

their incoming transitions (recall that we require *input-consistency*: all incoming transitions of a given state bear the same label); classes are sorted according to the total order of Σ . This step corresponds to enforcing Wheeler Axiom W1. At this point, the algorithm implements iteratively the following observation. Fix a given character a and a class C . Let moreover S (the *splitter*) be the smallest class (in the total order of the classes that we are maintaining) such that $C \cap \delta(S, a) \neq \emptyset$. Then, states in $C' = C \cap \delta(S, a)$ must precede those in $C'' = C \setminus \delta(S, a)$ in *any* Wheeler order for \mathcal{A} (if \mathcal{A} is Wheeler). Split C into C' and C'' and record the order $C' < C''$. A careful implementation of this observation results in a partition refinement algorithm – the Forward algorithm – satisfying Theorem 24.

As a matter of fact, the Forward algorithm achieves much more than what is stated in Theorem 24: it can sort *any* WNFA (actually, even a strict superclass of the WNFA) with a preorder that still allows indexing the NFA for pattern matching queries. First, it is worth noting that the partition output by the Forward algorithm is the *coarsest forward stable partition* of the NFA, i.e. the coarsest partition such that, for every pair of classes S, C and every $a \in \Sigma$, either $C \cap \delta(S, a) = \emptyset$ or $C \subseteq \delta(S, a)$ hold. Furthermore, the following properties, shown in [3] and later studied more in detail in [9, 13], hold:

► **Lemma 25** ([3]). *Let \equiv be the equivalence relation induced by the coarsest forward-stable partition of \mathcal{A} 's states. Let moreover $\mathcal{A}_{/\equiv}$ denote the quotient automaton obtained by collapsing states of \mathcal{A} being \equiv -equivalent. Then, the following properties hold:*

1. *If $u \equiv v$, then $I_u = I_v$.*
2. *If $\mathcal{A}_{/\equiv}$ is Wheeler, then the Forward algorithm finds a Wheeler order for $\mathcal{A}_{/\equiv}$.*
3. *If \mathcal{A} is Wheeler, then $\mathcal{A}_{/\equiv}$ is Wheeler.*
4. *The converse of property 3 does not hold: there exist non-Wheeler \mathcal{A} such that $\mathcal{A}_{/\equiv}$ is Wheeler.*

Recalling that states such that $I_u = I_v$ can be safely collapsed for pattern matching purposes, the above Lemma states that any Wheeler NFA can be indexed for pattern matching queries in polynomial time, despite the fact that recognizing and sorting WNFA is an NP-complete problem. This apparently counterintuitive result is made possible by Property 4 of Lemma 25: there do exist non-Wheeler NFA \mathcal{A} such that $\mathcal{A}_{/\equiv}$ is Wheeler (see [3] for a concrete example of such a NFA). In other words, the Forward algorithm cannot decide if the input NFA \mathcal{A} is Wheeler, but if it is it can index an equivalent (Wheeler) automaton $\mathcal{A}_{/\equiv}$. What's more, there exist non-Wheeler NFA \mathcal{A} that can still be indexed by the Forward algorithm, because they are such that $\mathcal{A}_{/\equiv}$ is Wheeler.

Another partition-refinement strategy for propagating forward the Wheeler Axiom W2 was later independently proposed in [18], although in that work the authors did not prove that their partition-refinement algorithm yields the coarsest forward-stable partition. In that paper, the goal was however slightly different than that of the previously-mentioned results: once obtained an ordered partition consistent with any Wheeler order of the input NFA, the algorithm proposed in [18] goes on to find in worst-case exponential time a total order also among states still belonging to the same partition's class (for which an order has not been established yet) by exhaustive search or by using a Satisfiability Modulo Theory (SMT) solver. The output of this tool is therefore a Wheeler order for the original NFA \mathcal{A} (NP-complete to find), rather than a Wheeler order for an equivalent quotient automaton as done in [3, 9] (computable in polynomial time). As shown in [18], this heuristic is in fact very efficient since in practice the equivalence classes output by the first partition refinement step are small, thus even a brute-force (or solver-based) enumeration of the remaining permutations to be tested (those of equivalent nodes) is fast.

The cubic complexity of the Forward algorithm was finally reduced to nearly-linear $O(|\delta| \log |Q|)$ in [9], who proved:

► **Theorem 26** ([9]). *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a (semi-) NFA and \equiv be the equivalence relation corresponding to the coarsest forward-stable partition of \mathcal{A} 's states. Then, in $O(|\delta| \log |Q|)$ time one can compute a total order among the states of $\mathcal{A}_{/\equiv}$ that is a valid Wheeler order for $\mathcal{A}_{/\equiv}$, if $\mathcal{A}_{/\equiv}$ is Wheeler.*

The idea behind the result in Theorem 26 is to use the “smallest-splitter” rule of the classic relational coarsest partition refinement algorithm of Paige and Tarjan [51], opportunely adapted to work with ordered partitions. Intuitively, consider a class C that has been split into the (totally-ordered) classes $C_1 < \dots < C_k$ using some splitter class S (while in the Forward algorithm each class is split into $k = 2$ sub-classes as mentioned above, the algorithm from [9] uses a three-way split and thus $k \leq 3$; the initial class Q is an exception since it is initially split into $k = |\Sigma| + 1$ classes, according to the states' incoming labels). In the next steps, we will choose the smallest between C_1 and C_k as splitter (Paige and Tarjan's algorithm, instead, chooses the smallest between C_1 and C_2). This choice guarantees that (i) we can still propagate Wheeler Axiom W2, and (ii) each state ultimately belongs to at most $\log |Q|$ splitters, leading to the claimed linearithmic complexity.

The NP-completeness of the problem of recognizing Wheeler NFA can also be attacked using a more general method based on *arbitrary relations* [20]. The key idea is that the antisymmetry and transitivity of a Wheeler order are not required to extend the Burrows-Wheeler transform and the FM-index to automata and so can be dropped. A “Wheeler relation” still satisfies axioms (W1) and (W2), but there may exist states u and v for which both $u < v$ and $v < u$ hold. The existence of a Wheeler relation can be checked in polynomial time [20], and this powerful approach can be extended to arbitrary automata (see Section 7.4).

7 Algorithms on Wheeler Automata and Languages

The wonders of Wheeler automata are not limited to their language-theoretic and indexing properties. In this section, we show that Wheeler automata and languages possess several additional remarkable properties that imply efficient algorithms for problems which are typically hard on arbitrary automata.

7.1 Determinization via Powerset Construction

The first such remarkable property is determinization. Consider a Wheeler NFA $\mathcal{A} = (Q, s, \Sigma, \delta, F)$. Consider moreover running the classic powerset construction algorithm on \mathcal{A} , yielding an equivalent DFA \mathcal{D} . A classic result is that \mathcal{D} has at most $2^{|Q|}$ states when \mathcal{A} is an arbitrary NFA. As shown in [2], in the Wheeler case the situation is radically different: if \mathcal{A} is a WNFA then \mathcal{D} is a WDFA with at most $2|Q| - 1 - |\Sigma|$ states. In other words, determinization of a WNFA maintains the Wheeler property and causes a blow-up in the number of states by a factor of (at most) 2, rather than exponential as in the general case. This result has several interesting implications. First of all, we deduce that WNFA and WDFA have the same expressive power: a language \mathcal{L} is Wheeler if and only if it is recognized by a WNFA, if and only if it is recognized by a WDFA. Second, hard problems on NFA such as equivalence (do two given NFA recognize the same language?) and universality (does a given NFA recognize Σ^* ?) are easy on WNFA (via determinization). Third, in polynomial time we can compute a 2-approximation of the smallest WNFA for a Wheeler language: this

is just the smallest WDFA for the language (computable in polynomial time from any WDFA, as discussed in the following subsection). Note that all such problems are PSPACE-hard on general NFA.

We give an intuition over the above result. Denote by \mathcal{L} the language recognized by the input WNFA. Consider the bipartite representation of the WNFA (see Figure 1, right; even though the figure refers to a WDFA, the following reasoning holds on WNFA as well). The figure should make it clear that, when following all edges labeled with a given character from states belonging to an *interval* (example: in the figure, consider following all edges labeled c from states $q_1 \dots q_4$), we reach another *interval* of states (in the figure, $q_2 \dots q_3$). Now, powerset construction can be equivalently described as the process of creating a DFA state corresponding to set $\delta(s, \alpha) \subseteq Q$, for every possible $\alpha \in \text{Pref}(\mathcal{L})$. Since the singleton set $\{s\}$ is an interval, the above consideration implies that all sets $\delta(s, \alpha)$ are intervals (on any Wheeler order for the WNFA). As there can be at most $(|Q| + 1)|Q|/2 \leq |Q|^2$ distinct intervals on any total order of Q , this already implies that powerset construction can create at most $|Q|^2$ DFA states (which is already a considerable improvement over the $2^{|Q|}$ states of the general case). A more accurate analysis allows to reduce this bound from quadratic to linear. The crucial property that can be proved is that, by Wheeler Axiom W2, no interval $\delta(s, \alpha)$ can be strictly contained inside another interval $\delta(s, \beta)$ for $\alpha, \beta \in \text{Pref}(\mathcal{L})$, except for the cases where $\delta(s, \alpha)$ is either a prefix or a suffix of $\delta(s, \beta)$ on the total order of Q . As it turns out, a family of intervals with this property over a total order with N elements can contain at most $2N - 1$ distinct intervals [2, Lem. 2.1]. Let N_a be the number of nodes with incoming label $a \in \Sigma$, and note that (by input-consistency) any interval $\delta(s, \alpha)$ contains only nodes whose incoming label is $\alpha[|\alpha|]$. We can therefore apply the above bound separately for every $a \in \Sigma$, and obtain that there are at most $\sum_a (2N_a - 1) = 2|Q| - |\Sigma|$ distinct intervals of the form $\delta(s, \alpha)$, for $\alpha \in \text{Pref}(\mathcal{L})$. The final bound $2|Q| - 1 - |\Sigma|$ comes from the observation that only interval $\delta(s, \epsilon) = \{s\}$ contains the source state s . To conclude, one can show that the following order is a Wheeler order among the states of the powerset DFA: for all strings $\alpha < \beta$, with $\alpha, \beta \in \text{Pref}(\mathcal{L})$, if $\delta(s, \alpha) \neq \delta(s, \beta)$ then we deem $\delta(s, \alpha) < \delta(s, \beta)$.

7.2 Linear-Time Minimization of WDFA

A classic algorithmic result in automata theory is that any DFA \mathcal{D} can be minimized in $O(|\mathcal{D}| \log |\mathcal{D}|)$ time via Hopcroft's algorithm [41], outputting a minimum-size (i.e. minimum number of states) DFA \mathcal{D}' equivalent to \mathcal{D} . In [2], the authors considered the analogous problem on the restricted space of Wheeler DFA:

► **Problem 2.** *Given a WDFA \mathcal{D} , compute the minimum-size equivalent WDFA \mathcal{W} .*

This problem can be solved by characterizing the smallest WDFA of a given Wheeler language, either in terms of the Myhill-Nerode equivalence relation [48] (see Theorem 10) or, equivalently, in terms a process merging states of any WDFA \mathcal{D} recognizing it [2, 3]. The latter characterization, which we sketch below, yields an algorithm solving Problem 2 in $O(|\mathcal{D}| \log |\mathcal{D}|)$ time. Later, this time was improved by [1] to optimal $O(|\mathcal{D}|)$.

The authors of [2] proved the following:

► **Theorem 27.** *Let \mathcal{D} be a WDFA. Consider the process of merging all maximal intervals of states $u_1 < \dots < u_t$ such that (i) $\lambda(u_1) = \dots = \lambda(u_t)$ and (ii) the states u_1, \dots, u_t are Myhill-Nerode equivalent (see Definition 2). Then, the resulting automaton \mathcal{W} is the smallest WDFA recognizing $\mathcal{L}(\mathcal{D})$.*

Theorem 27 can be turned into an algorithm by simply observing that the Myhill-Nerode equivalence classes of \mathcal{D} 's states can be computed in $O(|\mathcal{D}| \log |\mathcal{D}|)$ time with Hopcroft's algorithm, and the (unique) Wheeler order of \mathcal{D} 's states can be computed in linear time using the algorithms of Section 6.1. Why does this algorithm work? First, recalling the bipartite representation of Figure 1, observe that merging an interval of states with the same incoming letter preserves the Wheeler properties (in particular, merging such an interval of states cannot introduce same-letter arc crossings in the bipartite representation). As a result, the automaton \mathcal{W} generated by this process is certainly a WNFA. Moreover, \mathcal{W} is equivalent to \mathcal{D} , since we only merge Myhill-Nerode equivalent states. To show that \mathcal{W} is actually deterministic, consider a collapsed interval of states $u_1 < \dots < u_t$, and let $a \in \Sigma$. As previously observed, the image $\delta(\{u_1, \dots, u_t\}, a) = \{v_1, \dots, v_t\}$ of those states through letter a , must also be an interval of states, all reached by letter a . Additionally, v_1, \dots, v_t must be Myhill-Nerode equivalent, otherwise u_1, \dots, u_t would not be Myhill-Nerode equivalent. We conclude that v_1, \dots, v_t must be collapsed into a single state of the output \mathcal{W} by the algorithm, which implies that \mathcal{W} is deterministic. To prove minimality of \mathcal{W} , let $u < w < v$ be states of \mathcal{D} sorted according to its Wheeler order $<$ (in particular, u and v are not adjacent in the order), such that u and v are Myhill-Nerode equivalent, but they are not Myhill-Nerode equivalent to w . Then, one can easily observe that merging u and v will violate a Wheeler Axiom (W1 if $\lambda(u) \neq \lambda(v)$, or W2 otherwise). It follows that, when minimizing \mathcal{D} while maintaining the Wheeler properties, the best we can do is to collapse maximal intervals of states according to Theorem 27.

7.3 Recognizing Wheeler Languages

A very natural problem, considered for the first time in [3], is that of recognizing Wheeler languages:

► **Problem 3.** *Given a DFA \mathcal{D} , is the language $\mathcal{L}(\mathcal{D})$ Wheeler?*

The solution to Problem 3 provided in [3] is based on Corollary 11 (Section 4). Consider a minimum DFA $\mathcal{D} = (Q, s, \delta, F)$ recognizing a language $\mathcal{L}(\mathcal{D})$. Corollary 11 states that this language is Wheeler if and only if the image through $\delta(s, \cdot)$ of any co-lexicographically monotone sequence of strings $\alpha_1, \alpha_2, \dots$ (that is, this sequence is either increasing or decreasing in co-lexicographic order) belonging to $\text{Pref}(\mathcal{L})$, ultimately stabilizes in a single state of \mathcal{D} . In other words, there exists $N \in \mathbb{N}$ and $q \in Q$ such that, for all $i \geq N$, $\delta(s, \alpha_i) = q$. This characterization can be also understood in terms of Theorem 27: if every such sequence stabilizes, then the sequence of states obtained by mapping $\text{Pref}(\mathcal{L})$ (in co-lexicographic order) through $\delta(s, \cdot)$ is formed by a *finite* number of maximal intervals containing just one distinct state. Those intervals essentially correspond to the states of the minimum WDFA for the language. If, on the other hand, there exists such a non-stabilizing monotone sequence, then the sequence of states $\delta(s, \text{Pref}(\mathcal{L}))$ is formed by an *infinite* number of maximal intervals containing just one distinct state, thus no WDFA for the language can exist since, by Theorem 27, we can only merge Myhill-Nerode equivalent states being adjacent in co-lexicographic order.

Using this characterization, [3] devised a dynamic-programming polynomial-time algorithm solving Problem 3. Since the smallest WDFA for a Wheeler language \mathcal{L} could be *exponentially larger* than the smallest DFA for \mathcal{L} [3], the fact that Problem 3 can be solved in polynomial time at all is not trivial. Additionally, [30] showed that Problem 3 becomes PSPACE-complete when the input is an arbitrary NFA, rather than a DFA.

The polynomial-time solution of [3] is based on the observation that a non-stabilizing monotone sequence $\alpha_1, \alpha_2, \dots$ can be identified by searching for particular *cycles* in the smallest DFA for the language. More in detail, [3] proved that such a non-stabilizing monotone sequence exists if and only if we can identify two disjoint cycles C_1 and C_2 in the smallest DFA for the language, such that there exist two nodes u, v belonging to C_1 and C_2 , respectively, satisfying: (i) there exists a string γ such that $\delta(u, \gamma) = u$ and $\delta(v, \gamma) = v$, and (ii) there exist strings α, β with $\delta(s, \alpha) = u$ and $\delta(s, \beta) = v$ such that either $\alpha, \beta \prec \gamma$ or $\gamma \prec \alpha, \beta$ hold.

The work [3] showed that such three sequences α, β, γ (and corresponding cycles) can be identified via dynamic programming. The drawback of this solution was the degree of the polynomial running time: $\Omega(|\mathcal{D}|^{13})$. Later, this polynomial was reduced to $\Omega(|\mathcal{D}|^2)$ by [11], who also proved a conditional lower bound stating that, unless the Strong Exponential Time Hypothesis [42] (SETH) fails, Problem 3 cannot be solved in strongly sub-quadratic time. Thus, this work essentially settled the complexity of the problem. The intuition behind this result is the following. On the upper-bound side, [11] provided a simpler cycle-based characterization of Wheeler languages: a language \mathcal{L} is Wheeler if and only if its minimum DFA \mathcal{D} does not have two distinct states $u \neq v$ such that (i) the open intervals (in co-lexicographic order) $(\inf I_u, \sup I_u)$ and $(\inf I_v, \sup I_v)$ overlap, where $\inf X$ and $\sup X$ denote the infimum and supremum of set X , respectively, and (ii) $\delta(u, \alpha) = u$ and $\delta(v, \alpha) = v$ for some string α (i.e. u and v belong to two disjoint equally-labeled cycles). Condition (i) can be checked in $O(|\mathcal{D}|^2)$ time simultaneously on all state pairs u, v by computing $\inf I_u, \sup I_u$ for all states u using the partition-refinement algorithm of [9]. Condition (ii) can then be verified in $O(|\mathcal{D}|^2)$ time by checking if the square automaton $\mathcal{D} \times \mathcal{D}$ contains cycles involving state pairs (u, v) satisfying condition (i). On the lower-bound side, [11] observed that one can convert any instance of the *Orthogonal Vectors (OV) problem* (decide if there exist orthogonal vectors in two sets of binary vectors) into a (minimum) DFA \mathcal{D} of the same asymptotic size of the OV instance having two disjoint equally-labeled cycles if and only if the original OV instance has two orthogonal vectors. By building \mathcal{D} in such a way that condition (i) holds on all node pairs, one then obtains that there exist two orthogonal vectors in the OV instance if and only if the language of \mathcal{D} is Wheeler. By a classic reduction [54], OV cannot be solved in strongly subquadratic time unless the Strong Exponential Time Hypothesis [42] (SETH) fails. As a result, Wheeler languages cannot be recognized in strongly subquadratic time (in the input DFA's size), unless SETH fails.

7.4 Other Algorithmic Results

We conclude the section by mentioning other interesting lines of algorithmic research related with Wheeler automata.

DFA to minimum WDFA

The following variant of Problem 2 was first considered in [2]:

► **Problem 4.** *Given a minimum DFA \mathcal{D} recognizing a Wheeler language, compute the minimum-size equivalent WDFA \mathcal{W} .*

Since \mathcal{W} can be exponentially-larger than \mathcal{D} [3], Problem 4 cannot be solved in sub-exponential time. The goal is then to solve the problem in time as close as possible to the output size $|\mathcal{W}|$. Alanko et al. solved Problem 4 in near-optimal time $O(|\mathcal{W}| \log |\mathcal{W}|)$ in the particular case where \mathcal{D} is acyclic. The idea is to process the transitions of \mathcal{D} in any

topological order, inserting them in \mathcal{W} (initially empty). Whenever inserting a transition in \mathcal{W} , if this breaks Wheeler Axiom W2, i.e. if this transition produces a crossing in the bipartite representation of \mathcal{W} (see Figure 1), then a state of \mathcal{W} is split into two states (partitioning the incoming transitions of the state into two parts) in order to remove the axiom violation. This operation is supported by encoding \mathcal{W} with a dynamic data structure essentially storing the bipartite representation of \mathcal{W} . The $\log |\mathcal{W}|$ multiplicative factor in the running time comes from the query time of this dynamic structure. Later, [30] solved Problem 4 in the general case (i.e. \mathcal{D} is any DFA recognizing a Wheeler language) in time $O(|\mathcal{D}|^3 |\mathcal{W}| \log |\mathcal{W}|)$.

Random WDFA generation

Another interesting algorithmic problem, considered in [10], is that of generating Wheeler DFA according to a uniform distribution over all Wheeler DFA with a given number of states, transitions, and alphabet size. Interestingly, the paper shows that the Wheeler properties allow solving the problem in expected linear time (in the output size) and *constant* working space, provided that the transitions of the WDFA are streamed to output following the Wheeler order of their destination states. Such a result is not known to be possible for arbitrary DFA.

Computing the union and complementation of Wheeler languages

In [31], the authors study the problem of computing, given two Wheeler automata, a Wheeler automaton recognizing the union of their languages, provided that the union itself is Wheeler (as mentioned in Section 4, Wheeler languages are not closed under union). Their solution runs in quadratic time and linear (succinct) space. An alternative solution to the problem was later provided by [17], by providing an algorithm for *completing* WDFA (when such a completion is possible at all), and then applying the classical DFA complementation/union algorithms working on complete DFA.

Generalization to arbitrary automata

To conclude, a fruitful line of research initiated by [25, 27] and continued in [20, 13, 9, 21, 43, 5, 22, 24, 12, 23] later showed that almost all the results on Wheeler automata described in this survey can be generalized to *arbitrary* automata by simply not requiring the Wheeler order of Definition 4 to be total. As shown in [25, 27], any automaton admits a *partial* order satisfying the axioms of Definition 4. Interestingly, the *width* p of such a partial order – that is, the size of its largest antichain (a parameter being equal to 1 on total orders, i.e. Wheeler automata) – parameterizes several problems on automata being hard in the general case. Examples include: pattern matching on graphs (See Section 3) can be performed in time proportional to p^2 per character in the input pattern, automata can be encoded using $O(\log p)$ bits per edge, and the powerset construction algorithm for turning NFA into equivalent DFA runs in time exponential in p (the width of the input NFA's order), rather than in the size of the input NFA (as a classical result shows).

References

- 1 Jarno Alanko, Nicola Cotumaccio, and Nicola Prezza. Linear-time minimization of wheeler dfas. In *2022 Data Compression Conference (DCC)*, pages 53–62, 2022. doi:10.1109/DCC52660.2022.00013.

- 2 Jarno Alanko, Giovanna D’Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 911–930. SIAM, 2020. doi:10.1137/1.9781611975994.55.
- 3 Jarno Alanko, Giovanna D’Agostino, Alberto Policriti, and Nicola Prezza. Wheeler languages. *Information and Computation*, 281:104820, 2021. doi:10.1016/J.IC.2021.104820.
- 4 Jarno Alanko, Travis Gagie, Gonzalo Navarro, and Louisa Seelbach Benkner. Tunneling on wheeler graphs. In *2019 Data Compression Conference (DCC)*, pages 122–131, 2019. doi:10.1109/DCC.2019.00020.
- 5 Jarno N. Alanko, Davide Cenzato, Nicola Cotumaccio, Sung-Hwan Kim, Giovanni Manzini, and Nicola Prezza. Computing the LCP Array of a Labeled Graph. In *35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024)*, volume 296 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:15, 2024. doi:10.4230/LIPIcs.CPM.2024.1.
- 6 Michael Aschbacher. *Finite Group Theory*. Cambridge studies in advanced mathematics. Cambridge University Press, 1986.
- 7 László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183. ACM, 1983. doi:10.1145/800061.808746.
- 8 Uwe Baier. On Undetected Redundancy in the Burrows-Wheeler Transform. In *29th Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*, volume 105 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2018.3.
- 9 Ruben Becker, Manuel Cáceres, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Francisco Olivares, and Nicola Prezza. Sorting Finite Automata via Partition Refinement. In *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.15.
- 10 Ruben Becker, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Riccardo Maso, and Nicola Prezza. Random Wheeler Automata. In *35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024)*, volume 296 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2024.5.
- 11 Ruben Becker, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Alberto Policriti, and Nicola Prezza. Optimal wheeler language recognition. In *String Processing and Information Retrieval*, pages 62–74, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-43980-3_6.
- 12 Ruben Becker, Nicola Cotumaccio, Sung-Hwan Kim, Nicola Prezza, and Carlo Tsoni. Encoding co-lex orders of finite-state automata in linear space. In *36th Annual Symposium on Combinatorial Pattern Matching*, page 17, 2025.
- 13 Ruben Becker, Sung-Hwan Kim, Nicola Prezza, and Carlo Tsoni. Indexing finite-state automata using forward-stable partitions. In *International Symposium on String Processing and Information Retrieval*, pages 26–40. Springer, 2024. doi:10.1007/978-3-031-72200-4_3.
- 14 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de bruijn graphs. In *Algorithms in Bioinformatics*, pages 225–235, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-33122-0_18.
- 15 Janusz A. Brzozowski and Bo Liu. Quotient complexity of star-free languages. *Int. J. Found. Comput. Sci.*, 23(6):1261–1276, 2012. doi:10.1142/S0129054112400515.
- 16 Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. Technical report, DIGITAL SRC RESEARCH REPORT, 1994.
- 17 Giuseppa Castiglione, Antonio Restivo, et al. Completing wheeler automata. In *CEUR Workshop proceedings*, volume 3811, pages 120–132. CEUR-WS, 2024. URL: <https://ceur-ws.org/Vol-3811/paper060.pdf>.
- 18 Kuan-Hao Chao, Pei-Wei Chen, Sanjit A Seshia, and Ben Langmead. Wgt: Tools and algorithms for recognizing, visualizing, and generating wheeler graphs. *Iscience*, 26(8), 2023.

- 19 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. IEEE, 2023. doi:10.1109/DCC55655.2023.00023.
- 20 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.
- 21 Nicola Cotumaccio. Prefix sorting dfas: A recursive algorithm. In *34th International Symposium on Algorithms and Computation (ISAAC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.
- 22 Nicola Cotumaccio. A Myhill-Nerode Theorem for Generalized Automata, with Applications to Pattern Matching and Compression. In *41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)*, volume 289 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2024.26.
- 23 Nicola Cotumaccio. Fast pattern matching with epsilon transitions. In *International Workshop on Combinatorial Algorithms*, pages 242–255. Springer, 2025.
- 24 Nicola Cotumaccio. Improved circular dictionary matching. In *36th Annual Symposium on Combinatorial Pattern Matching (CPM 2025)*, pages 18–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025.
- 25 Nicola Cotumaccio, Giovanna D’Agostino, Alberto Policriti, and Nicola Prezza. Co-lexicographically ordering automata and regular languages-part i. *Journal of the ACM*, 70(4):1–73, 2023. doi:10.1145/3607471.
- 26 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*, pages 143–156, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-43980-3_12.
- 27 Nicola Cotumaccio and Nicola Prezza. On indexing and compressing finite automata. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2585–2599. SIAM, 2021. doi:10.1137/1.9781611976465.153.
- 28 Giovanna D’Agostino, Luca Geatti, Davide Martincigh, and Alberto Policriti. Cascade products and wheeler automata. *Theor. Comput. Sci.*, 1013:114754, 2024. doi:10.1016/J.TCS.2024.114754.
- 29 Giovanna D’Agostino, Davide Martincigh, and Alberto Policriti. Ordering regular languages: a danger zone. In *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. URL: <https://ceur-ws.org/Vol-3072/paper5.pdf>.
- 30 Giovanna D’Agostino, Davide Martincigh, and Alberto Policriti. Ordering regular languages and automata: Complexity. *Theoretical Computer Science*, 949:113709, 2023. doi:10.1016/j.tcs.2023.113709.
- 31 Lavinia Egidi, Felipe A Louza, and Giovanni Manzini. Space efficient merging of de bruijn graphs and wheeler graphs. *Algorithmica*, 84(3):639–669, 2022. doi:10.1007/S00453-021-00855-2.
- 32 Massimo Equi, Veli Mäkinen, Alexandru I. Tomescu, and Roberto Grossi. On the complexity of string matching for graphs. *ACM Trans. Algorithms*, 19(3), April 2023. doi:10.1145/3588334.
- 33 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1), November 2009. doi:10.1145/1613676.1613680.
- 34 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, July 2005. doi:10.1145/1082036.1082039.
- 35 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theoretical computer science*, 698:67–78, 2017. doi:10.1016/J.TCS.2017.06.016.

- 36 Daniel Gibney and Sharma V Thankachan. On the complexity of recognizing wheeler graphs. *Algorithmica*, 84(3):784–814, 2022. doi:10.1007/S00453-021-00917-5.
- 37 Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 573–582. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.51.
- 38 Julius Hartmanis and Richard E. Stearns. Sets of numbers defined by finite automata. *The American Mathematical Monthly*, 74(5), 1967.
- 39 Wing-Kai Hon, Tak-Wah Lam, Kunihiro Sadakane, Wing-Kin Sung, and Siu-Ming Yiu. A space and time efficient algorithm for constructing compressed suffix arrays. *Algorithmica*, 48:23–36, 2007. doi:10.1007/S00453-006-1228-8.
- 40 Wing-Kai Hon, Chen-Hua Lu, Rahul Shah, and Sharma V. Thankachan. Succinct indexes for circular patterns. In *Algorithms and Computation*, pages 673–682, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-25591-5_69.
- 41 John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.
- 42 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 43 Sung-Hwan Kim, Francisco Olivares, and Nicola Prezza. Faster Prefix-Sorting Algorithms for Deterministic Finite Automata. In *34th Annual Symposium on Combinatorial Pattern Matching (CPM 2023)*, volume 259 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2023.16.
- 44 Kenneth Krohn and John Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965.
- 45 Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. doi:10.1137/0222058.
- 46 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An extension of the burrows–wheeler transform. *Theoretical Computer Science*, 387(3):298–312, 2007. The Burrows-Wheeler Transform. doi:10.1016/j.tcs.2007.07.014.
- 47 Giovanni Manzini, Alberto Policriti, Nicola Prezza, and Brian Riccardi. The rational construction of a wheeler DFA. In *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024, June 25-27, 2024, Fukuoka, Japan*, volume 296 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CPM.2024.23.
- 48 Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.
- 49 Enno Ohlebusch, Simon Gog, and Adrian Kügel. Computing matching statistics and maximal exact matches on compressed full-text indexes. In *String Processing and Information Retrieval*, pages 347–358, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-16321-0_36.
- 50 Jaroslav Opatrny. Total ordering problem. *SIAM J. Comput.*, 8(1):111–114, 1979. doi:10.1137/0208008.
- 51 Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. doi:10.1137/0216062.
- 52 Jacques Sakarovitch. *Elements of automata theory*. Cambridge university press, 2009.
- 53 Huei Jang Shyr and Gabriel Thierrin. Ordered automata and associated languages. *Tamkang J. Math*, 5:9–20, 1974.
- 54 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.
- 55 Sheng Yu, Qingyu Zhuang, and Kai Salomaa. The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.*, 125:315–328, 1994. doi:10.1016/0304-3975(92)00011-F.