Encoding Co-Lex Orders of Finite-State Automata in Linear Space

Ruben Becker
DAIS, Ca' Foscari University of Venice, Italy

Nicola Cotumaccio
University of Helsinki, Finland

Sung-Hwan Kim
DAIS, Ca' Foscari University of Venice, Italy

Nicola Prezza
DAIS, Ca' Foscari University of Venice, Italy

Carlo Tosoni
DAIS, Ca' Foscari University of Venice, Italy

Abstract

The Burrows-Wheeler transform (BWT) is a string transformation that enhances string indexing and compressibility. Cotumaccio and Prezza [SODA '21] extended this transformation to nondeterministic finite automata (NFAs) through co-lexicographic partial orders, i.e., by sorting the states of an NFA according to the co-lexicographic order of the strings reaching them. As the BWT of an NFA shares many properties with its original string variant, the transformation can be used to implement indices for locating specific patterns on the NFA itself. The efficiency of the resulting index is influenced by the width of the partial order on the states: the smaller the width, the faster the index. The most efficient index for *arbitrary* NFAs currently known in the literature is based on the coarsest forward-stable co-lex (CFS) order of Becker et al. [SPIRE '24]. In this paper, we prove that this CFS order can be encoded within linear space in the number of states in the automaton. The importance of this result stems from the fact that encoding such an order in linear space represents a big first step in the direction of building the index based on this order in near-linear time – the biggest open research question in this context. The currently most efficient known algorithm for this task run in quadratic time in the number of transitions in the NFA and are thus infeasible to run on very large graphs (e.g., pangenome graphs). At this point, a near-linear time algorithm is solely known for the simpler case of deterministic automata [Becker et al., ESA '23] and, in fact, this algorithmic result was enabled by a linear space encoding for deterministic automata [Kim et al., CPM '23].

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Data structures design and analysis; Theory of computation → Pattern matching; Theory of computation → Graph algorithms analysis

Keywords and phrases Burrows-Wheeler Transform, Co-Lexicographic Orders, Nondeterministic Finite Automata, Graph Walks

Digital Object Identifier 10.4230/LIPIcs.CPM.2025.15

Funding *Ruben Becker, Sung-Hwan Kim, Nicola Prezza, Carlo Tosoni*: funded by the European Union (ERC, REGINDEX, 101039208). Views and opinions expressed are however those of the authors 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. *Nicola Cotumaccio*: funded by the Helsinki Institute for Information Technology.

1 Introduction

The *Burrows-Wheeler transform (BWT)* [4] is a renowned reversible string transformation that rearranges a string's characters to improve compressibility, while at the same time allowing the implementation of efficient indices. Although the original BWT was designed for strings, Gagie et al. extended this transformation to a particular class of nondeterministic finite automata (NFAs), which they termed *Wheeler NFAs* [11]. Subsequently, Cotumaccio et al. [10, 8] managed to extend the transformation to arbitrary NFAs through the concept of *co-lexicographic orders* (abbreviated to co-lex orders), yielding a natural extension of the BWT to NFAs. More precisely, co-lex orders are particular partial orders $\leq$ on an NFA's states such that, if $u \leq v$, with u, v being two states, then the strings reaching u and not reaching v are smaller than or equal to the strings reaching v and not reaching u. Such co-lex orders exist for every NFA and can be used to implement indices on the recognized regular language. The efficiency of the index depends on the width of the used co-lex order (a parameter being equal to 1 on Wheeler NFAs and always upper-bounded by the number of states). Specifically, the smaller the width of the co-lex order, the faster and smaller the resulting index. Computing the co-lex order of minimal width is however an NP-hard problem [12]. This issue has been addressed by Becker et al. [2, 3], who introduced coarsest forward-stable co-lex (CFS) orders, a new category of partial preorders that are as useful as co-lex orders for indexing purposes. Such CFS orders are guaranteed to exist for every NFA and, furthermore, are unique and can be computed in polynomial time. Moreover, the width of the CFS order is never larger than that of any co-lex order and, in some cases, is asymptotically smaller than the minimum-width co-lex order. As a result, CFS orders enable the implementation of indices in polynomial time, which are never slower than those based on co-lex orders, and that in some cases are asymptotically faster and smaller. However, the state-of-the-art algorithm for computing such CFS orders has quadratic time complexity with respect to the number of transitions in the automaton [3, Corollary 1]. This quadratic time complexity makes the application of such CFS orders infeasible in practice, e.g., in bioinformatics, where pangenome graphs (i.e., graphs encoding the DNA of a population) are used more and more frequently [16]. Such pangenome graphs fall within the category of big data for which only near-linear time algorithms can be considered feasible [15]. For this reason, the current main open research problem in this realm is to find an efficient, i.e., near-linear time, algorithm for computing co-lex orders of small width for arbitrary NFAs. For the special case of deterministic finite automata such a near-linear time algorithm is known [2, Algorithm 2] and its discovery was preceded by an encoding of this order that takes linear space with respect to the number of states of the automaton [13]. A similar linear space representation for the general case of nondeterministic finite automata is however not known for any of the candidate co-lex orders in the literature. In this paper, we resolve this main problem that hinders us to find an efficient algorithm for computing co-lex orders for arbitrary NFAs. We do so by giving an efficient data structure for the following problem.

► **Problem 1.** *Given a forward-stable NFA, find a data structure for its maximum co-lex order $\leq_{FS}$ that supports queries of the form: given two states u and v, determine if $u \leq_{FS} v$.*

Here a *forward-stable NFA* is an NFA for which the coarsest forward stable partition [14] is equal to the partition consisting of all singleton sets. The CFS order on an arbitrary NFA [3, Definition 6] is defined as the maximum co-lex order on the corresponding forward-stable NFA. The forward stable NFA is a quotient automaton of the original automaton (thus its size is at most the size of the original automaton). As a result, a data structure for Problem 1 permits to represent the CFS order of an arbitrary NFA and is thus general enough to represent a co-lex order for an arbitrary NFA.

Contribution and Main Techniques. In this article we give a data structure for Problem 1 stated above. More precisely, we prove the following theorem. In what follows, we denote with n the number of states and with m the number of transitions of the NFA at hand.

▶ **Theorem 2.** *Given a forward-stable NFA with n states, there exists a data structure for Problem 1 taking $O(n)$ space and supporting queries in $O(n)$ time.*

Here space is measured in RAM words of $\Theta(\log n)$ bits. In order to better put our result into context we observe that there are two trivial solutions to Problem 1. (1) Explicitly storing the n^2 pairs of the co-lex order yields a data structure for Problem 1 that takes $O(n^2)$ bits and supports queries in $O(1)$ time. (2) Storing the input NFA takes $O(m)$ space and the NFA inherently represents its maximum co-lex order $\leq_{FS}$. It is however unclear how to support queries efficiently in this case. Both of these approaches require $\Omega(n^2)$ bits to be stored in the worst-case as the number of transitions may be $\Theta(n^2)$.

Our data structure that satisfies the properties in Theorem 2 relies on three main techniques. (i) We assume to have computed a *co-lex extension* $\leq$, i.e., a total order that is a superset of the maximum co-lex order of a forward-stable NFA (see Definition 11 for the formal definition). Such a co-lex extension can be obtained by running the ordered partition refinement algorithm of Becker et al. [2]. (ii) For each state, we store a *left-minimal infimum walk* $P_u^{\inf}$ and a *right-maximal supremum walk* $P_u^{\sup}$ to u. An infimum (supremum) walk to a state u is a walk encoding the lexicographic smallest (largest) string reaching u from the initial state. An infimum (supremum) walk P_u to a state u is left-minimal (right-maximal) if, whenever it intersects with another infimum (supremum) walk P'_u, then the predecessor in P_u is smaller (larger) than or equal to the predecessor in P'_u according to the co-lex extension. We study this type of walks in Sections 3 and 6. (iii) For each state u, we furthermore store two integers $\phi(u, P_u^{\inf})$ and $\gamma(u, P_u^{\sup})$ that we use in order to encode the deepest states on the infimum walk $P_u^{\inf}$ (supremum walk $P_u^{\sup}$) that is in *infimum* (*supremum*) *conflict* with $P_u^{\inf}$ ($P_u^{\sup}$). We introduce this concept of infimum and supremum conflicts in Section 4. Intuitively, a state $\bar{u}$ in $P_u^{\inf}$ is in infimum conflict with $P_u^{\inf}$ if there exists a state $\hat{u}$ that is incomparable with $\bar{u}$ according to the maximum co-lex order and both $\bar{u}$ and $\hat{u}$ can reach u with the same string α.

Figure 1 shows the decision tree used by our data structure in order to determine comparability with respect to the CFS order $\leq_{FS}$ between two states u and v based on the above three concepts. Let u and v be two states. If $u = v$, then $u \leq_{FS} v$ holds by reflexivity, case (a) in Figure 1. Otherwise, if $v < u$ (here $<$ means $v \leq u$ with respect to the co-lex extension and $v \neq u$), we can conclude $\neg(u \leq_{FS} v)$, case (b) in Figure 1. Otherwise, let $P_u^{\sup}$ be the right-maximal supremum walk to u and let $P_v^{\inf}$ be the left-minimal infimum walk to v. Imagine traversing the two walks from u and v backwards yielding a sequence of pairs $(u_i, v_i)_{i \geq 1}$. While traversing the walks we can construct the strings $\sup I_u$ and $\inf I_v$. If $\sup I_u \leq \inf I_v$, we know that $u \leq_{FS} v$, case (c) in Figure 1. Otherwise, when comparing the pairs u_i, v_i with respect to the co-lex extension $\leq$, we are guaranteed to find a pair u_j, v_j such that $v_j < u_j$. We distinguish two cases, if for each $i \in [j-1]$, we have $u_i < v_i$ as in case (d) of Figure 1, then $\neg(u \leq_{FS} v)$. Otherwise, there exists a minimal integer $j' \leq j$ such that $u_{j'} = v_{j'}$. In this case, the comparability of u and v can be decided using the infimum/supremum conflicts. Consider the maximum integer h such that either u_h is in sup conflict with $P_u^{\sup}$ or v_h is in inf conflict with $P_v^{\inf}$. If $h \geq j'$ (case (e) in Figure 1), we know $\neg(u \leq_{FS} v)$. In fact, the opposite would imply that either $P_u^{\sup}$ is not right-maximal or $P_u^{\inf}$ is not left-minimal. Finally, if $h < j'$ (case (f) in Figure 1), we conclude that $u \leq_{FS} v$. The details of the data structure are presented in Section 5. We remark that the existence of our left-minimal infimum and right-maximal supremum walks is actually proved independently of the labels in the graph through an unlabeled analogue that we call leftmost/rightmost walks. These walks represent a combinatorial object in unlabeled directed graphs that may be of independent interest. As a central ingredient of our proof, we show constructively (i.e., through an algorithm) that such leftmost/rightmost walks are guaranteed to exist for any (unlabeled) directed graph and can be represented by a linear space function that encodes the predecessor of each node in their leftmost/rightmost walk. This is shown in Section 6.

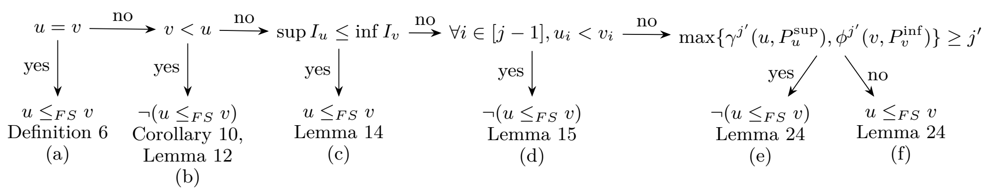


Figure 1 Let $\leq_{FS}$ and $\leq$ be the maximum co-lex order (see Definition 6) and a co-lex extension (see Definition 11) of a forward-stable NFA $\mathcal{A}$, respectively. Let u and v be any two states in $\mathcal{A}$. Denote with $P_u^{\sup} = (u_i)_{i\geq 1}$ a supremum right-maximal walk to the state u and with $P_v^{\inf} = (v_i)_{i\geq 1}$ an infimum left-minimal walk to v (see Definitions 5 and 19). The figure shows the decision tree representing all possible cases that may arise when determining whether $u \leq_{FS} v$. Here, j is the smallest integer such that $v_j < u_j$, while j' is the smallest integer such that $u_{j'} = v_{j'}$. Functions $\phi^{j'}$ and $\gamma^{j'}$ represent the *deepest* states in infimum/supremum conflict with the walks $P_u^{\sup}$ and $P_v^{\inf}$.

2 Preliminaries

Strings and NFAs. Given an alphabet Σ, we denote by Σ^* the set of all finite strings over Σ, where $\varepsilon \in \Sigma^*$ is the empty string. Moreover, we define Σ^ω as the set containing all strings formed by an infinite enumerable concatenation of characters from Σ (i.e., strings of infinite length). In particular, we consider *right-infinite* strings, meaning that $\alpha \in \Sigma^\omega$ is constructed from ε by appending an infinite sequence of characters to its end. Therefore, the operation of prepending a character $a \in \Sigma$ of $\alpha \in \Sigma^\omega$ is well defined and yields the string $a\alpha$. The notation α^ω, with $\alpha \in \Sigma^*$, denotes the concatenation of an infinite (enumerable) number of copies of α. In this paper, we assume to have a fixed total order $\preceq$ over Σ. We extend $\preceq$ to $\Sigma^* \cup \Sigma^\omega$ in order to obtain the *lexicographic order* on strings. For each $\alpha \in \Sigma^* \cup \Sigma^\omega$, $|\alpha| = l$ denotes the length of α, where $l = \infty$ if $\alpha \in \Sigma^\omega$. In addition, for each integer i with $1 \leq i < l+1$, α_i denotes the i-th character of α, starting from the left. Finally, we denote with $\alpha[i,j]$, where i,j are two integers such that $1 \leq i \leq j < l+1$, the string of Σ^* formed by the concatenation of characters $\alpha_i\,\alpha_{i+1}\ldots\alpha_j$.

A nondeterministic finite automaton (NFA) is a 4-tuple (Q, δ, Σ, s), where Q represents the set of the states, $\delta : Q \times \Sigma \to 2^Q$ is the automaton's transition function, Σ is the alphabet, and $s \in Q$ is the initial state. The standard definition of NFAs also includes a set of final states; however, we omitted them since we are not interested in distinguishing between final states and non-final states. Given an NFA $\mathcal{A} = (Q, \delta, \Sigma, s)$, a state $u \in Q$, and a character $a \in \Sigma$, we may use the shortcut $\delta_a(u)$ for $\delta(u, a)$. We make the following assumptions on NFAs: (i) We assume the alphabet Σ to be effective; each character of Σ labels at least one edge of the transition function. (ii) We assume that every state is reachable from the initial state. (iii) We assume that s has only one incoming edge, $s \in \delta(s, \#)$, where $\# \in \Sigma$ does not label any other edge of $\mathcal{A}$. (iv) We do not require each state to have an outgoing edge for all possible characters of Σ. (v) We assume that our NFAs are *input-consistent*; an NFA is said to be *input-consistent* if all edges reaching the same state have the same label of Σ. This assumption is not restrictive since any NFA can be transformed into an input-consistent NFA by replacing each state with at most $|\Sigma|$ copies of itself, without changing its regular language. Given an NFA $\mathcal{A} = (Q,\ \delta,\ \Sigma,\ s)$, and a state $u \in Q$, with $u \neq s$, we denote with $\lambda(u)$ the (unique) character of Σ that labels the incoming edges of u, thus $\lambda(s) = \#$.

▶ **Definition 3** (Forward-Stability). *Given an NFA $\mathcal{A} = (Q, \delta, \Sigma, s)$ and two sets of states $S, T \subseteq Q$, we say that S is forward-stable with respect to T, if, for all $a \in \Sigma$, one of the following conditions holds: (i) for every $u \in S$, there exists $v \in T$ such that $u \in \delta_a(v)$. (ii) For every $u \in S$ and $v \in T$, it holds that $u \notin \delta_a(v)$. A partition $\mathcal{Q}$ of $\mathcal{A}$'s states is forward-stable for $\mathcal{A}$, if, for any two parts $S, T \in \mathcal{Q}$, it holds that S is forward-stable with respect to T.*

For each NFA there exists a unique *coarsest* forward-stable partition, i.e., the forward-stable partition with the lowest cardinality [2]. Henceforth, we call an NFA *forward-stable* if its coarsest forward-stable partition is formed by singleton sets.

Infimum and supremum walks. Given an integer j, we denote by $[j]$ the set $\{1, \dots, j\}$. Given an NFA $\mathcal{A} = (Q, \delta, \Sigma, s)$, and a state $u \in Q$, we say that a *walk to* u, denoted as $P_u = (u_i)_{i=1}^{l}$, is a non-empty sequence of l states from Q that satisfies the following conditions: (i) $u_1 = u$, and (ii) for each $i \in [l-1]$, $u_i \in \delta_a(u_{i+1})$ where $a = \lambda(u_i)$. Moreover, we denote by $P_u = (u_i)_{i\geq 1}$ a walk of infinite length to $u \in Q$. We define I_u as the set of all strings $\alpha \in \Sigma^\omega$, for which there exists a walk $P_u = (u_i)_{i\geq 1}$ such that $\alpha = \lambda(u_1)\lambda(u_2)\lambda(u_3)\dots$. We now provide the definition of supremum and infimum strings of an NFA's state, which was introduced by Conte et al. [5] and Kim et al. [13, Definition 7].

▶ **Definition 4** (Infimum and supremum strings). *Let $\mathcal{A}$ be an NFA, and let u be a state of $\mathcal{A}$. Then, the* infimum string *of u, denoted* $\inf I_u$*, is the lexicographically smallest string in I_u. The supremum string of u, denoted* $\sup I_u$*, instead is the lexicographically largest string I_u.*

It is easy to demonstrate that for each $u \in Q$, the strings $\inf I_u$ and $\sup I_u$ exist (for instance, it follows from Observation 8 in the work of Kim et al. [13], see also [7]). We now provide the definition of infimum and supremum walks.

▶ **Definition 5** (infimum and supremum walks). *Let $\mathcal{A}$ be an NFA, and let u be a state in $\mathcal{A}$. Consider $\alpha = \inf I_u$ and $\beta = \sup I_u$ and a walk $P_u = (u_i)_{i\geq 1}$, then we say that:*
- *P_u is an* infimum walk to u, *denoted as $P_u^{\inf}$, if for each integer $i \geq 1$, $\lambda(u_i) = \alpha_i$.*
- *P_u is a* supremum walk to u, *denoted as $P_u^{\sup}$, if for each integer $i \geq 1$, $\lambda(u_i) = \beta_i$.*

In other words, a walk to u is an infimum (supremum) walk, if it is labeled with the infimum (supremum) string of u.

Co-lex orders. A partial order $\leq$ on a set U is a reflexive, antisymmetric and transitive relation on U. Given a partial order $\leq$ on a set U, for each $u, v \in U$, we write $u < v$ if $u \leq v$ and $u \neq v$. We now report the formal definition of co-lex order of an NFA, which was first introduced by Cotumaccio and Prezza [10, Definition 3.1].

▶ **Definition 6** (Co-lex order). *Let $\mathcal{A}$ be an NFA. A* co-lex order *of $\mathcal{A}$ is a partial order $\leq$ over Q that satisfies the following two axioms:*
1. *For each $u, v \in Q$, if $u \leq v$, then $\lambda(u) \leq \lambda(v)$.*
2. *For each pair $u \in \delta_a(u')$ and $v \in \delta_a(v')$, if $u < v$, then $u' \leq v'$.*

Note that every NFA $\mathcal{A} = (Q, \delta, \Sigma, s)$ admits a co-lex order; in fact, the partial order $\leq := \{(u, u) : u \in Q\}$ trivially satisfies the two axioms of Definition 6. We say that a co-lex order $\leq$ of an automaton $\mathcal{A}$ is the *maximum* co-lex order of $\mathcal{A}$ if $\leq$ is equal to the union of all co-lex orders of $\mathcal{A}$. Becker et al. [3, Lemma 3] showed that every forward-stable NFA admits a maximum co-lex order. Hereafter, we will denote by $\leq_{FS}$ the maximum co-lex order of an NFA.

▶ **Observation 7.** *Let $\mathcal{A}$ be a forward-stable NFA, and let u, v be two states of $\mathcal{A}$ such that $\lambda(u) < \lambda(v)$. Then $u <_{FS} v$.*

Proof. Consider the partial order $\leq$ defined as follows; $\leq := \{(z,z) : z \in Q\} \cup \{(u,v)\}$. $\leq$ is a co-lex order of $\mathcal{A}$, as it does not violate the axioms of Definition 6. Therefore, since $\leq_{FS}$ is defined as the union of every co-lex order of $\mathcal{A}$, it follows that $u <_{FS} v$ holds. ◀

Preceding pairs. We start with the definition of preceding pairs [6, Definition 6].

▶ **Definition 8** (Preceding pairs)**.** *Let $\mathcal{A}$ be an NFA and let $(\bar{u}, \bar{v}), (u, v) \in Q \times Q$ be pairs of distinct states. We say that $(\bar{u}, \bar{v})$* precedes *(u, v), denoted by $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$, if there exist two walks, $P_u = (u_i)_{i=1}^{l}$ and $P_v = (v_i)_{i=1}^{l}$, such that (i) $u_l = \bar{u}$ and $v_l = \bar{v}$, (ii) for each $i \in [l]$, $u_i \neq v_i$, and (iii) for each $i \in [l-1]$, $\lambda(u_i) = \lambda(v_i) = a$, for some $a \in \Sigma$.*

Note that if $u, v \in Q$ are distinct states, then the pair (u, v) trivially precedes itself. The following observation directly follows from Definition 8.

▶ **Observation 9.** *The relation $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ is transitive.*

Preceding pairs characterize the maximum co-lex order of a forward-stable NFA as follows.

▶ **Corollary 10.** *Let $\mathcal{A}$ be a forward-stable NFA and let $\leq_{FS}$ be its maximum co-lex order. Then $u <_{FS} v$ holds for two distinct states u, v in of $\mathcal{A}$ if and only if, for each pair $(\bar{u}, \bar{v})$ with $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$, it holds that $\lambda(\bar{u}) \leq \lambda(\bar{v})$.*

Proof. Let u and v be two distinct states. Following [6, Lemma 7], (u, v) is contained in the maximum co-lex relation (see [6, Definition 4]) if and only if $\lambda(\bar{u}) \leq \lambda(\bar{v})$ for every pair $(\bar{u}, \bar{v})$ with $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$. The maximum co-lex order $\leq_{FS}$ of a forward-stable NFA $\mathcal{A}$ is equal to its maximum co-lex relation according to [3, Lemma 3]. ◀

Note that for any distinct states u, v of $\mathcal{A}$, the statement $\neg(u <_{FS} v)$ holds if and only if there exists $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ with $\lambda(\bar{u}) > \lambda(\bar{v})$. We now define *co-lex extensions.*

▶ **Definition 11** (Co-lex extension)**.** *Let $\mathcal{A}$ be a forward-stable NFA and Q its set of states. Consider the maximum co-lex order $\leq_{FS}$ of $\mathcal{A}$. Then, a total order $\leq$ on Q is a* co-lex extension *of $\mathcal{A}$, if $\leq_{FS} \subseteq \leq$.*

Due to Lemma 11 of the work of Becker et al. [2], it follows that the *ordered partition refinement* [2, Algorithm 1] represents a feasible algorithm for computing a co-lex extension.

▶ **Lemma 12.** *Consider a forward-stable NFA $\mathcal{A} = (Q, \delta, \Sigma, s)$. Moreover, let $\leq_{FS}$ and $\leq$ be the maximum co-lex order and a co-lex extension of $\mathcal{A}$, respectively. Then, for every $u, v \in Q$ such that $u < v$, there exists a pair $(\bar{u}, \bar{v})$ with $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ such that $\lambda(\bar{u}) < \lambda(\bar{v})$.*

Proof. According to Definition 11 $u < v$ implies $\neg(v <_{FS} u)$. Corollary 10 then yields that there exists $(\bar{u}, \bar{v})$ with $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ such that $\lambda(\bar{u}) < \lambda(\bar{v})$. ◀

3 Infimum and Supremum Walks

In this section we prove structural properties of infimum and supremum walks in a forward-stable NFA $\mathcal{A}$. At the end of the section, we define left-minimal infimum and right-maximal supremum walks, the special type of infimum/supremum walks that our data structure is based on. As before, we let $\leq_{FS}$ be the maximum co-lex order of $\mathcal{A}$, and $\leq$ be a co-lex extension of $\mathcal{A}$. We further denote by n the number of states of $\mathcal{A}$. The following lemma states that infimum and supremum can be compared using their first $2n-1$ characters (see [5, 1, 9] for similar results).

▶ **Lemma 13.** *Let $\mathcal{A}$ be an NFA with set of states Q, where $|Q| = n$, and let $u, v \in Q$. If* $\sup I_u \neq \inf I_v$, *then* $\sup I_u[1, 2n-1] \neq \inf I_v[1, 2n-1]$.

Proof. Let C be the set consisting of all strings $\sup I_u$'s and all strings $\inf I_u$'s, and write $C = \{\gamma_1, \gamma_2, \dots, \gamma_{n'}\}$ where $\gamma_1 < \gamma_2 < \dots < \gamma_{n'}$. Then, $n' \leq 2n$. Notice that for every $u \in Q$, by the maximality of $\sup I_u$ there exist $c_1 \in \Sigma$ and $u_1 \in Q$ such that $\sup I_u = c_1 \sup I_{u_1}$, and by the minimality of $\inf I_v$ there exist $c_2 \in \Sigma$ and $u_2 \in Q$ such that $\inf I_u = c_2 \inf I_{u_2}$. This implies that for every $2 \leq i \leq n'$ there exist $c \in \Sigma$ and $2 \leq i' \leq n'$ such that $\gamma_i = c\gamma_{i'}$. We define the array $\mathsf{LCP}[2, n']$ such that $\mathsf{LCP}[i] = \mathsf{lcp}(\gamma_{i-1}, \gamma_i)$, where $\mathsf{lcp}(\alpha, \beta)$ is the length of the longest common prefix between α and β. To prove the lemma it is sufficient to show that $\mathsf{LCP}[i] \leq n' - 2$ for every i with $2 \leq i \leq n'$. To prove this it is sufficient to show that if $\mathsf{LCP}[i] = d$ for some $d \geq 1$ and i with $2 \leq i \leq n'$, then there exists j with $2 \leq j \leq n'$ such that $\mathsf{LCP}[j] = d - 1$, because then we obtain that LCP has at least $d + 1$ distinct entries, so $d + 1 \leq n' - 1$, which implies $d \leq n' - 2$. Assume that $\mathsf{LCP}[i] = d$, with $d \geq 1$. Let $c_1, c_2 \in \Sigma$ and $2 \leq i', i'' \leq n'$ be such that $\gamma_i = c_1\gamma_{i'}$ and $\gamma_{i-1} = c_2\gamma_{i''}$. Since $1 \leq d = \mathsf{LCP}[i] = \mathsf{lcp}(\gamma_{i-1}, \gamma_i) = \mathsf{lcp}(c_1\gamma_{i'}, c_2\gamma_{i''})$, we have $c_1 = c_2$ and from $\gamma_{i-1} < \gamma_i$ we obtain $\gamma_{i''} < \gamma_{i'}$, which implies $i'' < i'$. Then, $\mathsf{LCP}[i] = 1 + \mathsf{lcp}(\gamma_{i'}, \gamma_{i''}) = 1 + \min_{i'+1 \leq j \leq i''} \mathsf{lcp}(\gamma_{j-1}, \gamma_j) = 1 + \min_{i'+1 \leq j \leq i''} \mathsf{LCP}[j]$, so there exists $i' + 1 \leq j \leq i''$ such that $d = \mathsf{LCP}[i] = 1 + \mathsf{LCP}[j]$, which implies $\mathsf{LCP}[j] = d - 1$. ◀

The following lemma treats the case in which the relation between the supremum and infimum of two states already implies their respective order with respect to $\leq_{FS}$.

▶ **Lemma 14.** *Let u, v be two distinct states. Then,* $\sup I_u \leq \inf I_v$ *implies $u <_{FS} v$.*

Proof. We show the contrapositive. Assume that $\neg(u <_{FS} v)$. By Corollary 10, this implies the existence of a pair $(\bar{u}, \bar{v})$ with $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ such that $\lambda(\bar{u}) > \lambda(\bar{v})$. By Definition 8, this implies that there exist two walks $P_u = (u_i)_{i=1}^{l}$ and $P_v = (v_i)_{i=1}^{l}$ with $u_1 = u, v_1 = v, u_l = \bar{u}, v_l = \bar{v}$ such that $\alpha' = \beta'$ where $\alpha' := \lambda(u_1)\lambda(u_2)\dots\lambda(u_{l-1})$ and $\beta' := \lambda(v_1)\lambda(v_2)\dots\lambda(v_{l-1})$. Now recall that every state is reachable from the initial state s and $s \in \delta(s, \#)$. Thus there exist infinite strings $\alpha'' \in I_{\bar{u}}$ and $\beta'' \in I_{\bar{v}}$. Note that $\alpha'' > \beta''$ because $\lambda(\bar{u}) > \lambda(\bar{v})$. Since $\alpha = \alpha'\alpha'' \in I_u$ and $\beta = \beta'\beta'' \in I_v$, we obtain $\sup I_u \geq \alpha = \alpha'\alpha'' = \beta'\alpha'' > \beta'\beta'' = \beta \geq \inf I_v$, completing the proof. ◀

With the next lemma we show a case in which we can determine if $\neg(u <_{FS} v)$ holds.

▶ **Lemma 15.** *Let u, v be two states with $u < v$ and* $\sup I_u > \inf I_v$. *Furthermore, let $P_u^{\sup} = (u_i)_{i \geq 1}$ and $P_v^{\inf} = (v_i)_{i \geq 1}$. Then there exists an integer j with $1 < j < 2n$ such that $v_j < u_j$ and $u_i \leq v_i$ as well as $\lambda(u_i) = \lambda(v_i)$ for each $i \in [j-1]$. In addition, if $u_i < v_i$ holds for all $i \in [j-1]$, then $\neg(u <_{FS} v)$.*

Proof. Consider $\alpha = \sup I_u$ and $\beta = \inf I_v$. Let k be the maximal integer such that $\alpha[1, k-1] = \beta[1, k-1]$. Since $\alpha > \beta$, it holds that $\lambda(u_k) > \lambda(v_k)$. By Lemma 13, $k < 2n$. Moreover, by Observation 7, $\lambda(u_k) > \lambda(v_k)$ implies $v_k <_{FS} u_k$ which, by Definition 11, in turn implies $v_k < u_k$. Therefore, if we define $j > 1$ as the smallest integer for which $v_j < u_j$, then clearly for each $i \in [j-1]$, $u_i \leq v_i$. Moreover, since $j \leq k$, and $\alpha[1, k-1] = \beta[1, k-1]$, for each $i \in [j-1]$, it holds that $\lambda(u_i) = \lambda(v_i)$. For the second part of the lemma, if $u_i < v_i$ (implying $u_i \neq v_i$) for each $i \in [j-1]$, then $(u_j, v_j) \rightrightarrows (u, v)$ by definition. Since $v_j < u_j$, by Lemma 12, there exists $(\bar{u}, \bar{v}) \rightrightarrows (u_j, v_j)$ with $\lambda(\bar{u}) > \lambda(\bar{v})$. Thus, by Observation 9 it holds that $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ which by Corollary 10 proves that $\neg(u <_{FS} v)$. ◀

We summarize what we achieved so far. Given two distinct states u, v, to understand whether or not $u <_{FS} v$ holds we can proceed as follows. If according to a co-lex extension $\leq$, the statement $v < u$ holds, by Definition 11, we know $\neg(u <_{FS} v)$. If $u < v$, then if $\sup I_u \leq \inf I_v$, by Lemma 14, we know that $u <_{FS} v$. Otherwise, consider a supremum walk to u, $(u_i)_{i\geq 1}$, and an infimum walk to v, $(v_i)_{i\geq 1}$. If $\sup I_u > \inf I_v$, and there exists j such that $v_j < u_j$ and $u_i < v_i$ for each $i \in [j-1]$, then by Lemma 15 $\neg(u <_{FS} v)$. See Figure 2 for an example. The only remaining case to address is the existence of an integer $j' \in [j-1]$ such that $u_{j'} = v_{j'}$. To study this case, we introduce left-minimal (right-maximal) walks.

Left-minimal/right-maximal walks. We proceed with the definition of *left-minimal infimum walks* and *right-maximal supremum walks*. We first define infimum and supremum graphs.

▶ **Definition 16** (Infimum and supremum graphs). *The* infimum (supremum) graph $G = (Q, E)$ *of $\mathcal{A}$ is the directed unlabeled graph defined as follows: (i) The node set of G is identical to the one of $\mathcal{A}$. (ii) For each $u, v \in Q$, we let $(u, v) \in E$ if and only if there exists an infimum (supremum) walk $(u_i)_{i\geq 1}$ to a state $u' \in Q$ and an integer j such that $u_{j+1} = u$ and $u_j = v$.*

We now prove a preliminary result concerning infimum/supremum graphs.

▶ **Observation 17.** *Let $\mathcal{A}$ be an NFA and let G be its infimum (supremum) graph. Then, every walk of infinite length in G is an infimum (supremum) walk in $\mathcal{A}$.*

Proof. Let u be a state of $\mathcal{A}$, and let $P_u = (u_i)_{i\geq 1}$ be an arbitrary walk of infinite length. We prove this result for infimum walks, the proof for the supremum walks is analogous. Suppose for the sake of a contradiction that P_u is not an infimum walk. We consider the largest integer j for which there exists an infimum walk $P_u^{\inf} = (\bar{u}_i)_{i\geq 1}$, such that, for each $i \in [j]$, $\bar{u}_i = u_i$. Thus, $\bar{u}_{j+1} \neq u_{j+1}$. By Definition 16, there exists a state v, an infimum walk $P_v^{\inf} = (v_i)_{i\geq 1}$, and an integer k for which $v_k = u_j$ and $v_{k+1} = u_{j+1}$. Consider $\alpha = \lambda(\bar{u}_1)\ldots\lambda(\bar{u}_j)$, $\gamma = \lambda(\bar{u}_{j+1})\lambda(\bar{u}_{j+2})\ldots$, $\beta = \lambda(v_1)\ldots\lambda(v_k)$, $\gamma' = \lambda(v_{k+1})\lambda(v_{k+2})\ldots$, we know $\inf I_u = \alpha\gamma$, $\inf I_v = \beta\gamma'$. Moreover, consider the walks $P'_u = (\hat{u}_i)_{i\geq 1}$, where for each $i \in [j]$, $\hat{u}_i = \bar{u}_i$, and for each $i \geq j$, $\hat{u}_i = v_{i+k-j}$, and $P'_v = (\hat{v}_i)_{i\geq 1}$, where, for each $i \in [k]$, $\hat{v}_i = v_i$, and, for each $i \geq k$, $\hat{v}_i = \bar{u}_{i+j-k}$. We know that $\alpha\gamma' \in I_u$ and $\beta\gamma \in I_v$. By Definition 5, $\alpha\gamma \leq \alpha\gamma'$ which implies $\gamma \leq \gamma'$, and $\beta\gamma' \leq \beta\gamma$ which implies $\gamma' \leq \gamma$. Thus $\gamma = \gamma'$. Consequently, P'_u is an infimum walk. However, $\hat{u}_j = u_j$ and $\hat{u}_{j+1} = u_{j+1}$, a contradiction. ◀

We now introduce leftmost/rightmost walks in directed (unlabeled) graphs.

▶ **Definition 18** (Leftmost/rightmost walk). *Let $G = (V, E)$ be a directed graph, let $\leq$ be a total order on V and let $u \in V$. A walk $P_u = (u_i)_{i\geq 1}$ is a* leftmost (rightmost) *walk to u, if for each walk $\bar{P}_u = (\bar{u}_i)_{i\geq 1}$ and integer $j > 1$, $u_j = \bar{u}_j$ implies $u_{j-1} \leq \bar{u}_{j-1}$ ($\bar{u}_{j-1} \leq u_{j-1}$).*

We are now ready to introduce left-minimal and right-maximal walks.

▶ **Definition 19** (Left-minimal and right-maximal walks). *Let u be a state of $\mathcal{A}$. We say that an infimum (supremum) walk $P_u = (u_i)_{i\geq 1}$ is* left-minimal (right-maximal) *if P_u is a leftmost (rightmost) walk to u in the infimum (supremum) graph of $\mathcal{A}$ according to $\leq$.*

The proof of existences of left-minimal and right-maximal walks in directed graphs is given in Theorem 25 in Section 6. The next corollary then follows together with Observation 17.

▶ **Corollary 20.** *Let Q be the states of $\mathcal{A}$. There exists $p : Q \to Q$, such that, for each $u \in Q$, the sequence $(p^i(u))_{i\geq 0}$ is a left-minimal infimum (right-maximal supremum) walk to u.*

4 Inf and Sup Conflicts

We still consider $\mathcal{A}$ to be a fixed forward-stable NFA and $\leq_{FS}$ and $\leq$ be the maximum co-lex order and a co-lex extension of $\mathcal{A}$, respectively. We now define inf/sup conflicts.

▶ **Definition 21** (inf sup conflicts). *Let u be a state of $\mathcal{A}$ and let $P_u = (u_i)_{i\geq 1}$ be an infimum (supremum) walk. For $j > 1$, we say that u_j is in* inf (sup) conflict *with P_u, denoted as $u_j \sqcap P_u$ ($u_j \sqcup P_u$), if there exists $\bar{P}_u = (\bar{u}_i)_{i=1}^{j}$ satisfying: (i) For each i with $1 < i \leq j$, it holds that $\bar{u}_i \neq u_i$ and $\lambda(\bar{u}_i) = \lambda(u_i)$. (ii) It holds that $\neg(u_j <_{FS} \bar{u}_j)$ ($\neg(\bar{u}_j <_{FS} u_j)$).*

At this point, we present a first result concerning inf and sup conflicts.

▶ **Lemma 22.** *Let u be a state of $\mathcal{A}$, and $P_u = (u_i)_{i\geq 1}$ be an infimum (supremum) walk. If for some $j > 1$, $u_j \sqcap P_u$ ($u_j \sqcup P_u$) holds, then for any integer j' with $1 < j' \leq j$, it holds that $u_{j'} \sqcap P_u$ ($u_{j'} \sqcup P_u$).*

Proof. We prove the lemma for infimum walks and inf conflicts; the proof for supremum walks and sup conflicts is analogous. Consider an integer j such that $u_j \sqcap P_u$ and an arbitrary integer j' with $1 < j' < j$. Moreover, let $(\bar{u}_i)_{i=1}^{j}$ be the walk for which state u_j is in inf conflict with P_u. Clearly, $(\bar{u}_i)_{i=1}^{j'}$ satisfies condition (i) of Definition 21 for $u_{j'} \sqcap P_u$. It remains to prove that $\neg(u_{j'} <_{FS} \bar{u}_{j'})$. Note that for each i with $j' \leq i \leq j$, we have $\bar{u}_i \neq u_i$ and $\lambda(\bar{u}_i) = \lambda(u_i)$. Hence every preceding pair of $(u_j, \bar{u}_j)$ is a preceding pair of $(u_{j'}, \bar{u}_{j'})$ and it follows from Corollary 10 that $\neg(u_j <_{FS} \bar{u}_j)$ implies $\neg(u_{j'} <_{FS} \bar{u}_{j'})$. This proves the lemma. ◀

We introduce now two functions ϕ and γ which will be at the basis of our data structure.

▶ **Definition 23** (Functions ϕ and γ). *Let u be a state of $\mathcal{A}$ and let $P_u^{\inf} = (u_i)_{i\geq 1}$ and $P_u^{\sup} = (u'_i)_{i\geq 1}$ be an infimum walk and a supremum walk to u, respectively. We define two functions ϕ and γ as*

$$\phi(u, P_u^{\inf}) := \max(\{i < 2n : u_i \sqcap P_u^{\inf}\} \cup \{1\}),\ \gamma(u, P_u^{\sup}) := \max(\{i < 2n : u'_i \sqcup P_u^{\sup}\} \cup \{1\}).$$

Furthermore, for every integer $j > 1$, we define two functions ϕ^j and γ^j as

$$\phi^j(u, P_u^{\inf}) := \max_{i\in[j-1]}\{\phi(u_i, P_{u_i}^{\inf}) + i - 1\},\ \gamma^j(u, P_u^{\sup}) := \max_{i\in[j-1]}\{\gamma(u'_i, P_{u'_i}^{\sup}) + i - 1\},$$

where $P_{u_i}^{\inf} = (u_{i'})_{i'\geq i}$ and $P_{u'_i}^{\sup} = (u'_{i'})_{i'\geq i}$.

In other words, the functions $\phi^j(u, P_u^{\inf})$ and $\gamma^j(u, P_u^{\sup})$ intuitively represent the largest integer k for which either $u_k \sqcap P_{u_i}^{\inf}$ or $u'_k \sqcup P_{u'_i}^{\sup}$ holds for an integer i with $i < j$.

▶ **Lemma 24.** *Let u, v be two states of $\mathcal{A}$ with $u < v$ and $\sup I_u > \inf I_v$ and let $P_u^{\sup} = (u_i)_{i\geq 1}$ and $P_v^{\inf} = (v_i)_{i\geq 1}$ be a right-maximal supremum walk and a left-minimal infimum walk according to the co-lex extension $\leq$, respectively. Furthermore, assume there exists an integer $j > 1$ such that $u_j = v_j$ and $u_i < v_i$ for each $i \in [j-1]$. Then $\neg(u <_{FS} v)$ if and only if $\max\{\gamma^j(u, P_u^{\sup}), \phi^j(v, P_v^{\inf})\} \geq j$.*

Proof. ($\Rightarrow$) By Corollary 10, $\neg(u <_{FS} v)$ implies the existence of a pair $(\bar{u}, \bar{v})$ with $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ such that $\lambda(\bar{u}) > \lambda(\bar{v})$. Let $(\bar{u}_i)_{i=1}^{j'}$ and $(\bar{v}_i)_{i=1}^{j'}$ be the walks for which $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ holds, respectively. Lemma 15 implies that $\lambda(u_i) = \lambda(v_i)$ for each $i \in [j]$ and $j < 2n$. Furthermore, we have $\lambda(\bar{u}_i) \leq \lambda(u_i)$ and $\lambda(v_i) \leq \lambda(\bar{v}_i)$ for each $i \in [j']$ by the properties of supremum and infimum walks. By the properties of preceding pairs (see Definition 8), for each $i \in [j]$, it holds that $\lambda(\bar{u}_i) = \lambda(\bar{v}_i)$ and thus $\lambda(\bar{u}_i) = \lambda(u_i)$ and $\lambda(v_i) = \lambda(\bar{v}_i)$. Moreover, since $\lambda(\bar{u}) > \lambda(\bar{v})$, it follows that $j' > j$. Due to the walks $(\bar{u}_i)_{i=j}^{j'}$ and $(\bar{u}_i)_{i=j}^{j'}$, it holds that $(\bar{u}, \bar{v}) \rightrightarrows (\bar{u}_j, \bar{v}_j)$, which by Corollary 10 and $\lambda(\bar{u}) > \lambda(\bar{v})$ implies $\neg(\bar{u}_j <_{FS} \bar{v}_j)$. Now, define $h := \max\{i \in [j-1] : \bar{u}_i = u_i\}$ and $h' := \max\{i \in [j-1] : \bar{v}_i = v_i\}$ and consider $P_{u_h}^{\sup} = (u_i)_{i \geq h}$ and $P_{v_{h'}}^{\inf} = (v_i)_{i \geq h'}$. In order to prove $\max\{\gamma^j(u, P_u^{\sup}), \phi^j(v, P_v^{\inf})\} \geq j$, since $j < 2n$, it is thus sufficient to prove that $A := u_j \sqcup P_{u_h}^{\sup} \vee v_j \sqcap P_{v_{h'}}^{\inf}$ holds. Due to the previous considerations, $(\bar{u}_i)_{i=h}^{j}$ satisfies condition (i) of Definition 21 for $u_j \sqcup P_{u_h}^{\sup}$, while $(\bar{v}_i)_{i=h'}^{j}$ satisfies condition (i) of Definition 21 for $v_j \sqcap P_{v_{h'}}^{\inf}$. Hence, A holds if and only if $\neg(\bar{u}_j <_{FS} u_j) \vee \neg(v_j <_{FS} \bar{v}_j)$ or equivalently $\neg(\bar{u}_j <_{FS} u_j \wedge v_j <_{FS} \bar{v}_j)$. Since $u_j = v_j$ and $\leq_{FS}$ is transitive, the conclusion is implied by $\neg(\bar{u}_j <_{FS} \bar{v}_j)$, which we argued above.

($\Leftarrow$) Consider the case where the maximum is attained by $\gamma^j(u, P_u^{\sup})$ (the other case is analogous) and call that value h. We know $h \geq j$ by hypothesis. By Corollary 10, we have to prove the existence of a pair $(\bar{u}, \bar{v})$ with $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$ such that $\lambda(\bar{u}) > \lambda(\bar{v})$. Let $k \in [j-1]$ be such that $\gamma(u_k, P_{u_k}^{\sup}) + k - 1 = h$. Due to Observation 17, the walk $P_{u_k}^{\sup} = (u_i)_{i \geq k}$ is a supremum walk to u_k. Since $u_h \sqcup P_{u_k}^{\sup}$ and $j \leq h$, by Lemma 22 it follows that $u_j \sqcup P_{u_k}^{\sup}$. Therefore, by Definition 21 there exists a walk $(\bar{u}_i)_{i=k}^{j}$ such that $\bar{u}_i \neq u_i$ and $\lambda(\bar{u}_i) = \lambda(u_i)$ for each i with $k < i \leq j$ as well as $\neg(\bar{u}_j <_{FS} u_j)$. Now define $\bar{u}_i := u_i$ for $i \in [k]$ and consider the two walks $(\bar{u}_i)_{i=1}^{j}$ and $(v_i)_{i=1}^{j}$. Note that $\bar{u}_k = u_k$. From Definition 21 and Lemma 15 it follows that $\lambda(\bar{u}_i) = \lambda(v_i)$ for each $i \in [j]$. Suppose now that $\bar{u}_{i'} = v_{i'}$ for some i' with $k < i' < j$. We can then define a new supremum walk $(\hat{u}_i)_{i \geq 1}$ to u as follows. We define (i) $\hat{u}_i := \bar{u}_i$ for $i \in [i']$, (ii) $\hat{u}_i := v_i$ for i with $i' < i < j$, and (iii) $\hat{u}_i := u_i$ for $i \geq j$. Since $u_j = \hat{u}_j$ and $u_{j-1} < \hat{u}_{j-1}$, we conclude that $P_u^{\sup}$ is not a right-maximal supremum walk to u, a contradiction. Hence, $\bar{u}_i \neq v_i$ for all $i \in [j]$ and consequently $(\bar{u}_j, v_j) \rightrightarrows (u, v)$. Note that $u_j = v_j$ holds by hypothesis. Finally, since $\neg(\bar{u}_j <_{FS} u_j)$, by Corollary 10, there exists $(\bar{u}, \bar{v})$ with $(\bar{u}, \bar{v}) \rightrightarrows (\bar{u}_j, u_j)$ such that $\lambda(\bar{u}) > \lambda(\bar{v})$. Observation 9 then implies $(\bar{u}, \bar{v}) \rightrightarrows (u, v)$. ◀

5 Data Structure

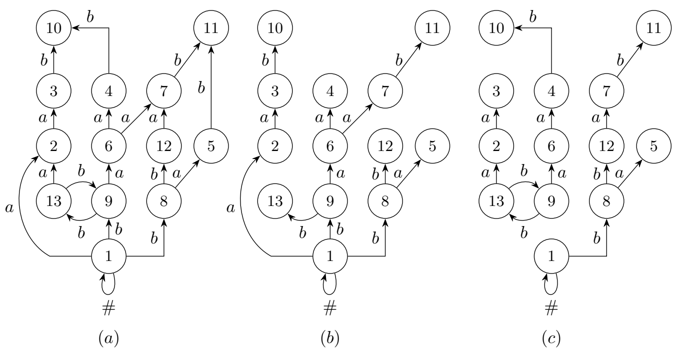
In this section, we present our data structure and finally prove Theorem 2, the main theorem of this article. Motivated by the above lemma and our previous observations, the data structure that enjoys the properties promised in Theorem 2 can be defined as follows.

Store. For each state u of $\mathcal{A}$, we store (i) a left-minimal infimum walk $P_u^{\inf}$ to u, (ii) a right-maximal supremum walk $P_u^{\sup}$ to u, and (iii) the integers $\phi(u, P_u^{\inf})$ and $\gamma(u, P_u^{\sup})$. Furthermore, we store a co-lex extension $\leq$ of $\mathcal{A}$. A possible co-lex extension of $\mathcal{A}$ can be computed in $O(m \log n)$ time using the *ordered partition refinement algorithm* [2, Algorithm 1], where n is the number of states of $\mathcal{A}$, and m the number of transitions.

Query. The procedure of a query on two arbitrary states u, v is illustrated in Figure 1. If $u = v$, then $u \leq_{FS} v$ by reflexivity. If $v < u$, then $\neg(u <_{FS} v)$ by Definition 11. If $u < v$, then four possible cases may arise. (i) Let us consider $\sup I_u$ and $\inf I_v$, which can be reconstructed from $P_u^{\sup}$ and $P_v^{\inf}$. If $\sup I_u \leq \inf I_v$ by Lemma 14 $u <_{FS} v$. (ii) Otherwise, $\sup I_u > \inf I_v$ holds. We then check, if there exists j such that $v_j < u_j$ and $u_i < v_i$ for all $i \in [j-1]$. If this is the case, then $\neg(u <_{FS} v)$ according to Lemma 15. Otherwise, we let j' with $1 < j' < j$ be such that $u_{j'} = v_{j'}$ and $u_i < v_i$ for all $i \in [j'-1]$. (iii) By Lemma 24, if $h := \max\{\gamma^{j'}(u, P_u^{\sup}), \phi^{j'}(v, P_v^{\inf})\} < j'$, we know that $u <_{FS} v$, (iv) while if $h \geq j'$, we know that $\neg(u <_{FS} v)$.

We refer the reader to Figure 2 for an example of this data structure. The correctness of our data structure follows immediately from the description above. In order to establish Theorem 2, it remains to argue why the space and query time bounds hold.

Proof of Theorem 2. By Corollary 20, we know that this information can be stored in $O(n)$ space. By Lemma 13, we can check $\sup I_u \leq \inf I_v$ in $O(n)$ time, and if this is the case by Lemma 14 $u <_{FS} v$. Otherwise, by Lemma 15, we can identify in $O(n)$ time the integer $j < 2n$ such that $v_j < u_j$ and $u_i \leq v_i$ for all $i \in [j-1]$. If $u_i < v_j$ for all $i \in [j-1]$, then $\neg(u <_{FS} v)$ by Lemma 15. On the other hand, if there exists j' with (i) $j' < j$, (ii) $u_{j'} = v_{j'}$, and (iii) $u_i < v_i$ for all $i \in [j'-1]$, by Definition 23, since $j' < 2n$, we can compute $\gamma^{j'}(u, P_u^{\sup})$ and $\phi^{j'}(v, P_v^{\inf})$ in $O(n)$ time. Hence the total time complexity is $O(n)$. ◀



u	$\inf I_u$	$\sup I_u$	ϕ	γ
1	$\#^\omega$	$\#^\omega$	1	1
2	$a\#^\omega$	ab^ω	1	1
3	$aa\#^\omega$	aab^ω	1	1
4	$aab\#^\omega$	aab^ω	1	1
5	$ab\#^\omega$	$ab\#^\omega$	1	1
6	$ab\#^\omega$	ab^ω	1	1
7	$aab\#^\omega$	$abb\#^\omega$	1	1
8	$b\#^\omega$	$b\#^\omega$	1	1
9	$b\#^\omega$	b^ω	1	1
10	$baa\#^\omega$	$baab^\omega$	1	25
11	$baab\#^\omega$	$babb\#^\omega$	2	1
12	$bb\#^\omega$	$bb\#^\omega$	1	1
13	$bb\#^\omega$	b^ω	1	1

Figure 2 Consider the forward-stable NFA $\mathcal{A}$ in Figure (a). Each state is assigned an integer i indicating its position in the co-lex extension $\leq$. We denote by u_i the i-th state according to $\leq$. Figures (b) and (c) show the NFAs encoding a left-minimal infimum walk and a right-maximal supremum walk, respectively, for each state. The table on the right shows for each state u the values of $\inf I_u$, $\sup I_u$, $\phi(u, P_u^{\inf})$, and $\gamma(u, P_u^{\sup})$, where $P_u^{\inf}$ and $P_u^{\sup}$ are the walks shown in Figures (b) and (c). Our data structure comprises $\leq$, the walks in Figures (b) and (c), and the two columns ϕ, γ from the table. We sketch the four cases that arise when determining whether $u <_{FS} v$ holds, assuming $u < v$. (i) By Lemma 14, since $\sup I_{u_3} \leq \inf I_{u_5}$, it follows that $u_3 <_{FS} u_5$. (ii) Consider $P_{u_2}^{\sup} = u_2, u_{13} \ldots$ and $P_{u_6}^{\inf} = u_6, u_9 \ldots$, since $\sup I_{u_2} > \inf I_{u_6}$, and $u_2 < u_6, u_{13} > u_9$, by Lemma 15, $\neg(u_2 <_{FS} u_6)$. (iii) Consider now $P_{u_4}^{\sup} = u_4, u_6 \ldots$ and $P_{u_7}^{\inf} = u_7, u_6 \ldots$. Since, $\sup I_{u_4} > \inf I_{u_7}$, $u_4 < u_7, u_6 = u_6$, and $\max\{\gamma^2(u_4, P_{u_4}^{\sup}), \phi^2(u_7, P_{u_7}^{\inf})\} = 1 < 2$, by Lemma 24, we can conclude $u_4 <_{FS} u_7$. (iv) Finally, consider $P_{u_{10}}^{\sup} = u_{10}, u_4, u_6 \ldots$ and $P_{11}^{\inf} = u_{11}, u_7, u_6 \ldots$, due to the fact that $\sup I_{u_{10}} > \inf I_{u_{11}}$, $u_{10} < u_{11}$, $u_4 < u_7$, $u_6 = u_6$, and $\max\{\gamma^3(u_{10}, P_{u_{10}}^{\sup}), \phi^3(u_{11}, P_{u_{11}}^{\inf})\} = 26 \geq 3$, by Lemma 24, we conclude that $\neg(u_{10} <_{FS} u_{11})$.

6 Existence of a Leftmost Walk

In this section, we consider an (unlabeled) directed graph $G = (V, E)$, and a total order $\leq$ on V. We always assume that every node of G has at least one incoming edge. Moreover, a walk to a node $u \in V$, denoted by $P_u = (u_i)_{i=1}^{l}$, is a sequence of l nodes such that: (i) $u_1 = u$, and (ii) $(u_{i+1}, u_i) \in E$ for each $i \in [l-1]$. We denote by $P_u = (u_i)_{i \geq 1}$ a walk to u of infinite length. We fix some further graph-related notation. The *induced subgraph of G on $V' \subseteq V$* is the graph $G[V'] := (V', E \cap (V' \times V'))$. The *subgraph of G reachable from a subset $S \subseteq V$*, denoted by $\delta_G(S)$, is the induced subgraph $G[V']$ on the nodes $V' = \{v \in V : \exists P_v = (v_i)_{i=1}^{\ell}$ with $v_\ell \in S\}$. For two sets of nodes $S, T \subseteq V$, we call $N(S) := \{v \in V : \exists u \in S$ and $(u, v) \in E\}$ the neighbors of S, $E(S) := E \cap (S \times V)$ the edges from S to $N(S)$, and $E(S, T) := E \cap (S \times T)$ the edges from S to T. For two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ their union is defined as $G_1 \cup G_2 := (V_1 \cup V_2, E_1 \cup E_2)$.

The rest of the section is devoted to proving the following theorem.

▶ **Theorem 25.** *Let $G = (V, E)$ be a directed graph, and let $\leq$ be a total order on V. Then, there exists a function $p : V \to V$ such that $P_u = (p^i(u))_{i\geq 0}$ is a leftmost (rightmost) walk to u for every $u \in V$.*

Hereafter, we only consider the leftmost case since the rightmost is analogous. We prove Theorem 25 constructively, i.e., we give an algorithm, termed *Forward Visit*, that computes the function p on input a directed graph $G = (V, E)$ and a total order $\leq$ on V. At a very high level, the algorithm consists of DFS and BFS visits that alternate with each other. Here the DFS visit has the purpose of computing a cycle C. Once this cycle C is computed, we start two BFS-like searches starting from C on two different subgraphs G_L and G_R of G. These two BFS-like searches each compute walks starting from nodes in C that when concatenated with C form the leftmost infinite walks for all nodes on those walks. These walks will then be represented by the function p. More precisely, the BFS-like searches work as follows: Let V' be nodes for which we have not computed a value for p yet. The graph G_L contains the nodes that can be reached in $G[V']$ from some node $u \in C$ through a neighbor node $v \notin C$ that is on the *left* of u's cycle successor with respect to the total order $\leq$. The subgraph G_R is defined symmetrically as reachable through *right* neighbors. The BFS-like searches are then a *multi-source shortest path* search in G_R and a *multi-source longest path* search in G_L. The intuition why we compute shortest paths for G_R and longest paths for G_L is as follows. Let $z \notin C$ be a node in G_L that is reachable by a length-d path from some cycle node $u \in C$ through a *left neighbor* $v \notin C$ of u. Now, consider some other length-d path from u to z that follows the cycle longer, i.e., goes through the successor of u in the cycle. Such a walk is not left-most by definition and in fact our algorithm will never output such a walk as it is constructed using a strictly shorter path from C to z than the one going through v. By a symmetric argument nodes in G_R should be connected to C via shortest path.

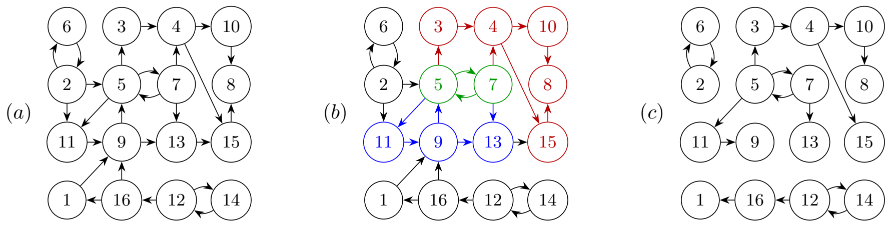


Figure 3 (a) A directed graph $G = (V, E)$ and a total order $\leq$ over V represented by the integer names of nodes. (b) In green the first cycle C that is found by the DFS of Algorithm 1, if we start a DFS from node 2; in red and blue the subgraphs G_L and G_R corresponding to C, respectively. Here, $L = \{3, 4\}$ and $R = \{11, 13\}$. (c) Leftmost walks represented by p (indicated by the shown edges).

The Forward Visit Algorithm. We proceed with a detailed description of the algorithm, a pseudocode implementation can be found in Algorithm 1. For each $u \in V$, the algorithm maintains three values, $p(u)$ (initially null), $u.color$ (initially white), and $u.\text{next}$ (initially null). The algorithm iterates over the nodes in V in an arbitrary order. For each $u \in V$ with $u.color = \text{white}$, the algorithm starts a DFS visit from u. This DFS first changes the color of u to gray, and eventually to black once the DFS from u terminated. The DFS visit from u is a classical DFS up to two caveats. (1) For a node $u \in V$, its neighbors $N(u)$ are processed in increasing order with respect to $\leq$. (2) The color $v.color$ of a neighbor $v \in N(u)$ determines our action when processing v as follows: (a) if $v.color = \text{black}$, the node v is ignored, (b) if $v.color = \text{white}$, we process the node normally, i.e., we launch a new DFS from v and set

$u.\text{next} = v$, and (c) if $v.color = \text{gray}$, we still set $u.\text{next} = v$ but we temporarily interrupt the DFS from u. Case (c) implies that the DFS from u found a cycle C in G that can be reconstructed from the next-values. We now call the function `ComputeWalks` with input C.

Algorithm 1 Forward Visit.

Input : directed graph $G = (V, E)$, total order $\leq$ over V
Output : function $p : V \to V$ s.t. $(p^i(u))_{i \geq 0}$ is a leftmost walk for every $u \in V$

Function `ComputeWalks(`C`)`:
  $V' \leftarrow \{v \in V : p(v) = \text{null}\}$
  $R \leftarrow \{v \in V' : \exists(u, v) \in E(C) \text{ and } v > u.\text{next}\}$ and $E_{C,R} \leftarrow E(C, R)$
  $L \leftarrow \{v \in V' : \exists(u, v) \in E(C) \text{ and } v < u.\text{next}\}$ and $E_{C,L} \leftarrow E(C, L)$
  $G_R \leftarrow (C, E_{C,R}) \cup \delta_{G[V']}(R)$ and $G_L \leftarrow (C, E_{C,L}) \cup \delta_{G[V']}(L)$
  `MultiSourceShortestPath(`G_R, C`)` // sets $p(v)$ for nodes v in G_R s.t. $v \notin C$
  `MultiSourceLongestPath(`G_L, C`)` // sets $p(v)$ for nodes v in G_L s.t. $v \notin C$
  foreach $v \in C$ **do** $p(v) \leftarrow u$ where $u \in C$ s.t. $u.\text{next} = v$
  foreach *node* u *in* G_L *and* G_R **do** $u.color \leftarrow \text{black}$

Function `DFS(`u`)`:
  $u.color \leftarrow \text{gray}$
  foreach $v \in N(u)$ *in increasing order w.r.t.* $\leq$ **do**
    if $v.color = white$ **then** $u.next \leftarrow v$, `DFS(`v`)`
    else if $v.color = gray$ **then**
      $u.next \leftarrow v$, $\bar{u} = u$, and $C \leftarrow \emptyset$
      do $C.\text{append}(u)$, $u \leftarrow u.next$ **while** $\bar{u} \neq u$ // build cycle C
      `ComputeWalks(`C`)`
  $u.color \leftarrow \text{black}$
$p(u) \leftarrow \text{null}$, $u.color \leftarrow \text{white}$, and $u.next \leftarrow \text{null}$ for all $u \in V$ // initialization
foreach $u \in V$ *such that* $u.color = white$ **do** `DFS(`u`)`

Let us denote with $V' := \{v \in V : p(v) = \text{null}\}$ all the nodes for which we have not yet constructed a leftmost walk. The function `ComputeWalks` computes two (possibly overlapping) subgraphs G_R and G_L of $G[V']$ and constructs walks from C to the nodes in these two graphs. The graphs G_R and G_L are defined as follows. First, we compute two (possibly overlapping) subsets R and L of $N := N(C) \cap V'$ as follows. We let $R := \{v \in V' : \exists(u, v) \in E(C) \text{ and } v > u.\text{next}\}$ and $L := \{v \in V' : \exists(u, v) \in E(C) \text{ and } v < u.\text{next}\}$, as well as $E_{C,R} := E(C, R)$ and $E_{C,L} := E(C, L)$. In other words, the set R (the set L) consists of those neighbors in V' of nodes $u \in C$ that are larger (smaller) than $u.\text{next}$ with respect to $\leq$. We then define $G_R = (V_R, E_R) := (C, E_{C,R}) \cup \delta_{G[V']}(R)$ and $G_L = (V_L, E_L) := (C, E_{C,L}) \cup \delta_{G[V']}(L)$. In order to compute the function $p(u)$ for nodes $u \in V_R \cup V_L \setminus C$, we now launch two BFS-like visits in a specific (and essential) order. We first launch a *multi-source shortest path* search in G_R and then a *multi-source longest path* search in G_L, in both cases from C. In both of these subroutines we set $p(v) = u$ whenever v is discovered to be the next node on a shortest (respectively longest) path starting from the cycle. In both of these BFS-like visits, we process a node's neighbors in increasing order with respect to $\leq$ (recall that we are searching for leftmost walks), we give more details in the paragraph below. It is essential that we first run the shortest path search on G_R and then the longest path search on G_L as the two graphs possibly overlap and we hence reset the p-values for nodes that are in both graphs, prioritizing paths in G_L. We then compute $p(u)$ also for the nodes $u \in C$ by setting $p(u) = v$ where $v \in C$ is such that $v.\mathrm{next} = u$. Lastly, we set $u.color = \text{black}$ for each $u \in V_L \cup V_R$. Then the function `ComputeWalks` terminates and the DFS is resumed. We refer the reader to Figure 3 for an example run of the algorithm.

Details on the BFS searches. The multi-source shortest path search is implemented as a classical BFS on G_R with a queue initially containing the nodes C. When a node u is dequeued, nodes $v \in N(u) \cap V_R$ are processed in increasing order with respect to $\leq$. If v is newly discovered by the BFS, we set $p(v) = u$ and enqueue v.

The multi-source longest path search is implemented as follows. It is essential that G_L is acyclic, see Lemma 27 (i). Hence, we can simply compute the maximum distance $d(C, u)$ from C to every node u in G_L by traversing G_L in a topological order. We then perform a BFS-like search on G_L with a queue initially containing the nodes C. When a node u is dequeued, nodes $v \in N(u) \cap V_L$ are processed in increasing order with respect to $\leq$. If v is newly discovered by the BFS and $d(C, v) = d(C, u) + 1$, we set $p(v) = u$ and enqueue v.

Analysis. We start with the following simple observation.

▶ **Observation 26.** (i) At termination of `ComputeWalks(`C`)` it holds that $p(u) \neq \text{null}$ for every $u \in V$ that can be reached from C. (ii) Before and after each run of `ComputeWalks`, if $(u_i)_{i=1}^{l}$ is a walk in G and $p(u_j) \neq \text{null}$ for some $j \in [l]$, then $p(u_1) \neq \text{null}$.

Proof. (i) Let $V' = \{u \in V : p(u) = \text{null}\}$ as in the algorithm. `ComputeWalks(`C`)` inserts every node $u \in V'$ that can be reached from C to the subgraphs G_L or G_R. Then it computes function p for all and only nodes in $G_L \cup G_R$. (ii) When the DFS outputs a cycle C, all nodes u that are reachable from C and that satisfy $p(u) = \text{null}$ are added to one of the subgraphs G_L or G_R. This holds as $R \cup L = N(C) \cap V'$, where V' were all nodes v with $p(v) = \text{null}$. Hence `ComputeWalks` always calculates $p(u)$ for all and only those nodes in G_L and G_R. ◀

We proceed with the lemma about the acyclicity of G_L.

▶ **Lemma 27.** *Let $G_L = (V_L, E_L)$ and C be the subgraph of $G = (V, E)$ and the cycle computed during an arbitrary execution of* `ComputeWalks` in Forward Visit. Then, (i) G_L is acyclic and (ii) it holds that $v \notin C$ for each v such that there exists $(u, v) \in E_L$.

Proof. Let $L \subseteq V_L$ be as in the algorithm. Suppose for the purpose of contradiction that (i) or (ii) does not hold. In both cases there exists $z \in V_L$ such that $z \in C'$ for some cycle C' in G different from C. Let $(u, v) \in E(C, L)$ be the edge such that v can reach z in G_L. As $z \in V_L$, we know that $p(z) = \text{null}$ and, by Observation 26 (ii), it follows that $p(\bar{z}) = \text{null}$ for each $\bar{z} \in C'$, and, for an analogous argument, also the nodes $\bar{v}$ that can reach C' must satisfy $p(\bar{v}) \neq \text{null}$. Thus, at this point of the algorithm execution, the function `ComputeWalks` cannot have colored those nodes black that can reach C', as it colors exactly the nodes for which it computes the function p (i.e., the nodes in $G_L \cup G_R$). The only remaining part in which Forward Visit may color a node $\bar{v}$ black that can reach C' is line 18, i.e., when `DFS(`$\bar{v}$`)` has terminated. Notice however that $\bar{v}$ can trivially reach C' and thus `DFS(`$\bar{v}$`)` has to find a cycle C'' that can reach the cycle C' in G before terminating (possibly $C'' = C'$). Thus, by Observation 26 (i), when `DFS(`u`)` is launched, `DFS(`$\bar{v}$`)` has not terminated yet, otherwise $p(z) \neq \text{null}$ would hold. When `DFS(`u`)` was started, the nodes in $N(u)$ were processed in increasing order with respect to $\leq$. In addition, as $v < u.\mathrm{next}$ by definition of L, it follows that v was visited by `DFS(`u`)` before $u.\mathrm{next}$. Finally, since v can reach z, it follows that

`DFS(`z`)` has to terminate before `DFS(`v`)`. As for each node $\bar{v}$ reaching C the function `DFS(`$\bar{v}$`)` has not terminated yet, it holds that `DFS(`v`)` has to output a cycle C'' (possibly $C'' = C'$) different from C before terminating, contradicting the assumption that C is the output cycle. ◀

We next observe that Forward Visit indeed computes a complete function p on V.

▶ **Observation 28.** *Upon termination Forward Visit has computed $p(u)$ for every node $u \in V$.*

Proof. Let $u \in V$. By assumption, every node has an incoming edge and thus there exists a cycle C_u in G containing u. If before or after any run of `ComputeWalks` there exists $v \in C_u$ with $p(v) \neq$ null, then Observation 26 (ii) shows that also $p(u) \neq$ null. Otherwise, before and after every run of `ComputeWalks` it holds that $p(v) =$ null for all $v \in C_u$. In this case, `ComputeWalks` has never colored the nodes in C_u black, as by line 9, it colors black all and only those nodes for which it has computed the function p. By line 20, the algorithm starts a `DFS` visit from every node in V (either in line 13 or in line 20). Therefore, let v be the first node of C_u for which a `DFS` visit is started. As for each $z \in C_u$ it holds that $p(v) =$ null, before `DFS(`v`)` starts $z.color =$ white for each $z \in C_u$. However, since $v \in C_u$ the `DFS` visit cannot terminate without finding a cycle C with $v \in C$. Thus, since v can reach u, by property (i) of Observation 26, we can conclude that the next run of `ComputeWalks` calculates $p(u)$. ◀

We are now ready to prove Theorem 25.

Proof of Theorem 25. By Observation 28, it holds that $p(v) \neq$ null for all $v \in V$ upon termination of the algorithm. It remains to prove that $P_u = (p(u)^i)_{i \geq 0}$ is a leftmost walk to u according to $\leq$ for any node $u \in V$. Let $u_{i+1} \coloneqq p(u)^i$ for $i \geq 0$ (i.e., $u_0 = u$) and suppose for the purpose of contradiction that P_u is not a leftmost walk to u. Then by Definition 18 there exists a walk $P'_u = (\bar{u}_i)_{i \geq 0}$ to u such that $\bar{u}_j = u_j$ for some $j > 1$ and $\bar{u}_{j-1} < u_{j-1}$. Let j be minimal with that property. Furthermore, let k with $0 \leq k < j-1$ be maximal such that $\bar{u}_k = u_k$ (such k exists as $\bar{u}_0 = u_0 = u$). Consider now the execution of `ComputeWalks` that has computed $p(u)$ and let the cycle C and the subgraphs $G_L = (V_L, E_L)$ and $G_R = (V_R, E_R)$ be the instances of those objects in that execution of the function. By Observation 26 (ii), at the beginning of this execution $p(v) =$ null for each v in P_u and P'_u. Consider the walk $\hat{P}_{u_k} = (\hat{u}_i)_{i \geq k}$ to u_k, where $\hat{u}_i = \bar{u}_i$ for i with $k \leq i \leq j$ and $\hat{u}_i = u_i$ for $i \geq j$. We distinguish three cases: (1) $u_k \in C$, (2) $u_k \in V_L \setminus C$, and (3) $u_k \in V_R \setminus (C \cup V_L)$.

(1) Let $u_k \in C$. Then, $u_i \in C$ for all $i \geq k$ from how the algorithm sets p in line 8. Hence, $\hat{u}_i \in C$ for all $i \geq j$. Now, consider the largest m with $k \leq m < j$ such that $\hat{u}_m = \hat{u}_{m'}$ for some $m < m'$. By assumption, this integer m exists since $k < j$ and $\hat{u}_k \in C$. Thus, if m' is the smallest integer satisfying this property, the sequence $C' = \hat{u}_m, \hat{u}_{m+1}, \ldots, \hat{u}_{m'}$ is a cycle different from C in G. Since $\hat{u}_{j-1} < u_{j-1}$, the DFS has to visit $\hat{u}_{j-1}$ before u_{j-1} and as $\hat{u}_{j-1}$ can reach $\hat{u}_m$ and consequently the cycle C', it contradicts that the DFS output C.

(2) Now, let $u_k \in V_L \setminus C$ and consider the smallest h with $h > k$ such that $u_h \in C$ and observe that then also $u_i \in C$ for $i \geq h$. Observe that $P'_{u_k} = (u_i)_{i=k}^{h}$ is a longest path from C to u_k in G_L. We distinguish two sub-cases. (a) Suppose that $h < j$. Then $u_{j-1} \in C$ and $\hat{u}_{j-1} \in L$ as $\hat{u}_{j-1} < u_{j-1}$. This implies $(\hat{u}_{i+1}, \hat{u}_i) \in E_L$ for each i with $k \leq i < j$ by the definition of E_L. By Lemma 27 (i), the graph G_L is acyclic and thus $(\hat{u}_i)_{i=k}^{j}$ is a path from C to $\hat{u}_k$ in G_L that is longer than P'_{u_k}, a contradiction. (b) Now assume that $h \geq j$. This implies $\hat{u}_{j-1} \in V_L \setminus C$ and trivially also $\hat{u}_i \in V_L$ for i with $k \leq i < j$. Lemma 27 (ii) implies $\hat{u}_i \in V_L \setminus C$ and hence h is also minimal such that $\hat{u}_h \in C$. Due to the previous

considerations $d(C, \hat{u}_i) = d(C, u_i)$ for each i with $k \leq i < j$, where d is as in the algorithm. As the multi-source longest path search processes the neighbors of u_j in increasing order with respect to $\leq$ and since $\hat{u}_{j-1} < u_{j-1}$, this contradicts $p(u_k) = u_{k+1}$.

(3) Now assume $u_k \in V_R \setminus (C \cup V_L)$. Note that $\hat{u}_{k'} \notin V_L \setminus C$ for all $k' > k$ as the opposite would imply $u_k \in V_L$. As in (2) let h be minimal with $h > k$ such that $u_h \in C$ and observe that then also $u_i \in C$ for each $i \geq h$. Observe that $P'_{u_k} = (u_i)_{i=k}^h$ is a shortest path from C to u_k in G_R. We again distinguish two sub-cases. (a) Suppose that $h < j$. Then $u_{j-1} \in C$ and $\hat{u}_{j-1} \in L$ as $\hat{u}_{j-1} < u_{j-1}$. This implies $\hat{u}_{j-1} \in V_L \setminus C$ as $p(u_{j-1}) = u_j$, a contradiction to our observation above. (b) Now assume that $h \geq j$. This implies $\hat{u}_{j-1} \in V_R \setminus C$ and $\hat{u}_i \in V_R$ for each i with $k \leq i < j$. Furthermore, $\hat{u}_i \notin C$ for each i with $k \leq i < j$ as $(u_i)_{i=k}^h$ is a shortest path from C to u_k. It follows that $(\hat{u}_i)_{i=k}^h$ is another shortest path from C to u_k in G_R. As the multi-source shortest path search processes the neighbors of u_j in increasing order with respect to $\leq$ and since $\hat{u}_{j-1} < u_{j-1}$, this contradicts $p(u_k) = u_{k+1}$. ◀

References

1 Jarno N. Alanko, Davide Cenzato, Nicola Cotumaccio, Sung-Hwan Kim, Giovanni Manzini, and Nicola Prezza. Computing the LCP array of a labeled graph. In Shunsuke Inenaga and Simon J. Puglisi, editors, *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024, June 25-27, 2024, Fukuoka, Japan*, volume 296 of *LIPIcs*, pages 1:1–1:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CPM.2024.1.

2 Ruben Becker, Manuel Cáceres, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Francisco Olivares, and Nicola Prezza. Sorting finite automata via partition refinement. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.15.

3 Ruben Becker, Sung-Hwan Kim, Nicola Prezza, and Carlo Tosoni. Indexing finite-state automata using forward-stable partitions. In Zsuzsanna Lipták, Edleno Silva de Moura, Karina Figueroa, and Ricardo Baeza-Yates, editors, *String Processing and Information Retrieval - 31st International Symposium, SPIRE 2024, Puerto Vallarta, Mexico, September 23-25, 2024, Proceedings*, volume 14899 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2024. doi:10.1007/978-3-031-72200-4_3.

4 Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. *SRS Research Report*, 124, 1994.

5 Alessio Conte, Nicola Cotumaccio, Travis Gagie, Giovanni Manzini, Nicola Prezza, and Marinella Sciortino. Computing matching statistics on wheeler dfas. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2023, Snowbird, UT, USA, March 21-24, 2023*, pages 150–159. IEEE, 2023. doi:10.1109/DCC55655.2023.00023.

6 Nicola Cotumaccio. Graphs can be succinctly indexed for pattern matching in $O(|E|^2 + |V|^{5/2})$ time. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2022, Snowbird, UT, USA, March 22-25, 2022*, pages 272–281. IEEE, 2022. doi:10.1109/DCC52660.2022.00035.

7 Nicola Cotumaccio. Prefix sorting dfas: A recursive algorithm. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023, December 3-6, 2023, Kyoto, Japan*, volume 283 of *LIPIcs*, pages 22:1–22:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.22.

8 Nicola Cotumaccio, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Co-lexicographically ordering automata and regular languages - part i. *J. ACM*, 70(4), August 2023. doi:10.1145/3607471.

9 Nicola Cotumaccio, Travis Gagie, Dominik Köppl, and Nicola Prezza. Space-time trade-offs for the LCP array of wheeler dfas. In Franco Maria Nardini, Nadia Pisanti, and Rossano Venturini, editors, *String Processing and Information Retrieval - 30th International Symposium, SPIRE 2023, Pisa, Italy, September 26-28, 2023, Proceedings*, volume 14240 of *Lecture Notes in Computer Science*, pages 143–156. Springer, 2023. doi:10.1007/978-3-031-43980-3_12.

10 Nicola Cotumaccio and Nicola Prezza. On indexing and compressing finite automata. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2585–2599. SIAM, 2021. doi:10.1137/1.9781611976465.153.

11 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017. doi:10.1016/J.TCS.2017.06.016.

12 Daniel Gibney and Sharma V. Thankachan. On the hardness and inapproximability of recognizing wheeler graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 51:1–51:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.51.

13 Sung-Hwan Kim, Francisco Olivares, and Nicola Prezza. Faster prefix-sorting algorithms for deterministic finite automata. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CPM.2023.16.

14 Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. doi:10.1137/0216062.

15 Shang-Hua Teng. Scalable algorithms for data and network analysis. *Found. Trends Theor. Comput. Sci.*, 12(1-2):1–274, 2016. doi:10.1561/0400000051.

16 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, October 2016. doi:10.1093/bib/bbw089.