MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Succinct Representation for (Non)Deterministic Finite Automata

Sankardeep CHAKRABORTY, Roberto GROSSI,
Kunihiko SADAKANE, and Srinivasa Rao SATTI

METR 2019–06 April 2019

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: https://www.keisu.t.u-tokyo.ac.jp/research/techrep/

Succinct Representation for (Non)Deterministic Finite Automata

Sankardeep Chakraborty
RIKEN Center for Advanced Intelligence Project, Japan
sankar.chakraborty@riken.jp

Roberto Grossi
Dipartimento di Informatica, Università di Pisa, Italy
grossi@di.unipi.it

Kunihiko Sadakane
The University of Tokyo, Japan
sada@mist.i.u-tokyo.ac.jp

Srinivasa Rao Satti
Seoul National University, South Korea
ssrao@cse.snu.ac.kr

Abstract

Deterministic finite automata are one of the simplest and most practical models of computation studied in automata theory. Their conceptual extension is the non-deterministic finite automata which also have plenty of applications. In this article, we study these models through the lens of succinct data structures where our ultimate goal is to encode these mathematical objects using information theoretically optimal number of bits along with supporting queries on them efficiently. Towards this goal, we first design a succinct data structure for representing any deterministic finite automaton $\mathcal{D}$ having n states over a σ-letter alphabet Σ using $(\sigma-1)n\log n + O(n\log\sigma)$ bits of space, which can determine, given an input string x over Σ, whether $\mathcal{D}$ accepts x optimally in time proportional to the length of x, using constant words of working space. When the input deterministic finite automaton is acyclic, we can improve the above space bound significantly to $(\sigma-1)(n-1)\log n + 3n + O(\log^2\sigma) + o(n)$ bits, without compromising the running time for string acceptance checking. Finally, we exhibit our succinct data structure for representing a non-deterministic finite automaton $\mathcal{N}$ having n states over a σ-letter alphabet Σ using $\sigma n^2 + n$ bits of space, such that given an input string x, we can decide whether $\mathcal{N}$ accepts x efficiently in polynomial time.

1 Introduction

Automata theory is a branch of theoretical computer science that deals exclusively with the definitions, properties and applications of different mathematical models of computation. These models play a major role in multiple applied areas of computer science. One of the most basic and fundamental models that is studied in automata theory since long time back is called the *finite automata*. It comes in two different types, *deterministic finite automata* (henceforth DFA) and *non-deterministic finite automata* (henceforth NFA). There exists more complex and sophisticated models as well, for example, *Context-free grammar, Turing machines* etc. In what follows, let us formally define DFA and NFA in a nutshell as these are our primary subjects of study in this article. A DFA $\mathcal{D}$ is a quintuple $\mathcal{D} = (\Sigma, Q, q_0, \delta, F)$ where:

- Σ is an *alphabet*; a finite set of letters,
- Q is the finite set of *states*,
- $q_0 \in Q$ is the *initial state*,
- $\delta : Q \times \Sigma \to Q$ is the *transition function* and

- $F \subseteq Q$ is the *set of final states*.

We often extend the transition function to $\delta : Q \times \Sigma^* \to Q$ which is defined recursively as follows: $\delta(q, \epsilon) = q$ for all $q \in Q$, where ϵ is the empty string; and $\delta(q, aw) = \delta(\delta(q, a), w)$ for all $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$. Given the above definition, we say that the DFA accepts a string x over the alphabet Σ if and only if $\delta(q, x) \in F$. The *language* $\mathcal{L}$ accepted by a DFA $\mathcal{D}$ is defined as the set of all strings accepted by the DFA $\mathcal{D}$, and is denoted by $\mathcal{L}(\mathcal{D})$. See Figure 1 for a simple example. In the rest of this paper, we assume that the alphabet Σ is $\{1, 2, \ldots, \sigma\}$, [1] and the state set Q is $\{q_0, q_1, \ldots, q_{n-1}\}$.

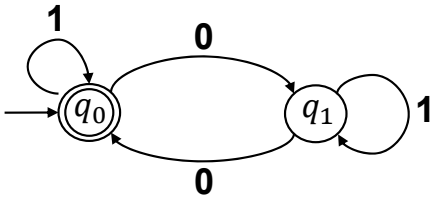

Figure 1 The *state transition diagram* for a DFA $\mathcal{D}$ where $\mathcal{D} = (\Sigma, Q, q_0, \delta, F)$ such that (i) $\Sigma = \{0, 1\}$, (ii) $Q = \{q_0, q_1\}$, (iii) $q_0 = q_0$ (marked with an incoming arrow coming from nowhere), (iv) $F = \{q_0\}$, and (v) the transition function is defined as the following set, $\{\delta(q_0, 1) = q_0, \delta(q_0, 0) = q_1, \delta(q_1, 1) = q_1, \delta(q_1, 0) = q_0\}$. Precisely the DFA $\mathcal{D}$ accepts all the strings containing an even number of zeros over the binary alphabet.

A deterministic automaton $\mathcal{A}$ is called *acyclic* [16] if it has a unique recurrent state where a state q is defined as *recurrent* if there exists a non-empty string x over Σ such that $\delta(q, x) = q$. Non-recurrent states are typically called *transient*, and the unique recurrent state (denoted by $q'' \in Q$) is classically called the *dead state* as $\delta(q'', \sigma) = q''$ for all $\sigma \in \Sigma$.

An NFA is a conceptual extension of DFAs where the definition of the transition function is mainly extended. More specifically, for DFA, the transition function is defined as δ: $Q \times \Sigma \to Q$ whereas for NFA, the same is defined as δ: $Q \times \Sigma \to \mathcal{P}(Q)$ where $\mathcal{P}(Q)$ denotes the power set of Q. Another extension, which is sometimes used in the literature, is to simply allow more than one initial state in an NFA, and in this case, the third item in the tuple becomes I denoting the set of initial states, instead of singleton $\{q_0\}$. The rest of above quintuple definition remains as it is for NFA. Thus, in the case of NFA $\mathcal{N}$, the language $\mathcal{L}(\mathcal{N})$ is defined as $\{x \mid \exists_{q \in I} \exists_{q' \in F} [q' \in \delta(q, x)]\}$. We refer the readers to the classic texts of [14, 23] for a thorough discussions on these mathematical models.

Even if a DFA is defined as an abstract mathematical concept, still it has got myriad of practical applications. More specifically, it is used in text processing, compilers, and hardware design [23]. Quite often it is implemented in small hardware and software tools for solving various specific tasks. For example, a DFA can model a software that can figure out whether or not online user input such as email addresses are valid. DFAs/NFAs are also used for network packet filtering. In some of these applications, the alphabet is large and there is a failure/exit state so that only a subset of transitions go to non-failure states; so we call the latter ones *non-failure* transitions.

Despite having so many applications in practically motivated problems, we are not aware of, to the best of our knowledge, any study of DFAs and NFAs from the point of view of

[1] We follow the standard assumption on Σ for succinct data structures, so the final time bounds in this paper will have an extra multiplicative factor of $O(\log \sigma)$ when σ is non-constant.

succinct data structures where the goal is to store an arbitrary element from a set Z of objects using the information theoretic minimum $\log(|Z|) + o(\log(|Z|))$ bits of space while still being able to support the relevant set of queries efficiently, which is what we focus on in this paper. We also assume the usual model of computation, namely a $\Theta(\log n)$-bit word RAM model where n is the size of the input.

1.1 Related Work

The field of *succinct data structures* originally started with the work of Jacobson [15], and by now it is a relatively mature field in terms of breadth of problems considered. To illustrate this further, there already exists a large body of work on representing various combinatorial objects succinctly. A partial list of such combinatorial objects would be trees [18, 21], various special graph classes like planar graphs [2], chordal graphs [19], partial k-trees [10], interval graphs [1] along with arbitrary general graphs [11], permutations [17], functions [17], bitvectors [22] among many others. We refer the reader to the recent book by Navarro [20] for a comprehensive treatment of this field. The study of succinct data structures is motivated by both theoretical curiosity and also by the practical needs as these combinatorial structures do arise quite often in various applications.

For DFA and NFA, other than the basic structure that is mentioned in the introduction, there exists many extensions/variations in the literature, for example, two-way finite automata, Büchi automata and many more. Researchers generally study the properties, limitations and applications of these mathematical structures. One such line of study that is particularly relevant to us for this paper is the research on counting DFAs and NFAs. Since the fifties there are plenty of attempts in exactly counting the number of DFAs and NFAs with n states over the alphabet Σ, and the state-of-the-art result is due to [3] for DFAs and [9] for NFAs respectively. We refer the readers to the survery (and the references therein) of Domaratzki [8] for more details. Basically, from these results, we can deduce the information theoretic lower bounds on the number of bits required to represent any DFA or NFA. Then we augment these lower bounds by designing data structures whose size matches the lower bounds, hence consuming optimal space, along with capable of executing algorithms efficiently using this succinct representation, and this is the main contribution of this paper.

1.2 DFA and NFA Enumeration

After a number of efforts by several authors, finally Bassino and Nicaud [3] found a matching upper and lower bound on the number of non-isomorphic initially-connected (i.e., all the states are reachable from the initial state) DFA's with n (including a fixed initial and one or possibly more final) states over an alphabet Σ (where $|\Sigma| = \sigma$) is $\Theta(n2^{2n}S_2(\sigma n, n))$ where $S_2(n, m)$ denotes the Stirling numbers of the second kind[2]. Using the approximation of the Stirling numbers of the second kind [13], which states that $S_2(n, m) \approx \frac{m^n}{m!}$, we can obtain the information theoretic lower bound for representing any DFA having n states and σ-sized alphabet is given by $\lg(n2^{2n}S_2(\sigma n, n)) = (\sigma - 1)n \lg n + O(n)$ bits. On the other hand, Domaratzki et al. [9] showed that there are asymptotically $2^{\sigma n^2 + n}$ initially connected NFAs on n states over a σ-letter alphabet with a fixed initial state and one or more final states. Thus, information theoretically, we need at least $\sigma n^2 + n$ bits to represent any NFA. In what follows later, we show that we can represent any given DFA/NFA using asymptotically

[2] It is defined recursively as $S_2(0, 0) = 1$, $S_2(n, 0) = 0$ for all $n \geq 1$ and for all $n, m \geq 1$, $S_2(n, m) = mS_2(n - 1, m) + S_2(n - 1, m - 1)$.

optimal number of bits as mentioned here. Throughout this paper, we assume that the input DFAs/NFAs that we want to encode succinctly are initially connected.

1.3 Our Main Results and Paper Organization

The classical representation of DFAs/NFAs consists of explicitly writing the transition function δ in a two dimensional array $J[0..n-1][1..\sigma]$ having n rows corresponding to the n states of the DFA/NFA and σ (where $|\Sigma| = \sigma$) columns corresponding to the alphabet Σ such that $J[i][j] = \delta(q_i, j)$ where $q_i \in Q, j \in \Sigma$. For DFA, the entry in $J[i][j]$ is a singleton set whereas for NFA it could possibly contain a set having more than one state. Thus, the space requirement for representing any given DFA (NFA respectively) is given by $O(n\sigma \log n)$ ($O(n^2\sigma \log n)$ respectively) bits. These space bounds are clearly not optimal – for the DFAs, it is off by an additive $n \log n$ term from the information theoretic minimum, while for the NFAs, it is off by a multiplicative factor of $\log n$ from the optimal bound. We alleviate this discrepancy in the space bounds by designing optimal succinct data structures for these objects.

Towards this goal, we start by listing all the preliminary data structures and graph theoretic terminologies that will be required in our paper in Section 2. Then, in Section 3.1 we first discuss the relevant prior work from [3], and show that, by using suitable data structures, their work already gives a succinct encoding of DFA. But the major drawback of this encoding is that it is not capable of handling the problem of checking whether a string is accepted by the DFA extremely efficiently. In Section 3.2, we overcome this problem by designing a succinct data structure for DFA, which can also check the string acceptance optimally. We summarize our main result in the following theorem.

Theorem 1. *Given an initially-connected deterministic finite automata $\mathcal{D}$ having n states and working over an alphabet Σ of size σ, there exists a succinct encoding for $\mathcal{D}$ taking $(\sigma-1)n \log n + O(n \log \sigma)$ bits of space, which can optimally determine, given an input string x over Σ, whether $\mathcal{D}$ accepts x in time proportional to the length of x, using constant words of working space. If the DFA has only $N < \sigma n$ non-failure transitions, then the space can be further reduced to $(N-n)\log n + O(N \log \sigma)$ bits.*

The upper bounds in Theorem 1 save roughly $n \log n$ bits with respect to the immediate representation of the DFA. The former upper bound is optimal as it matches the information-theoretical lower bound in Section 1.2, up to lower order terms. As for the latter upper bound, we do not know its optimality but it is smaller than the information-theoretical lower bound of $\lceil \log \binom{n^2}{N} \rceil + \Theta(N \log \sigma)$ bits derived for edge-labeled deterministic directed graphs [12]. Indeed, DFAs can be seen as a special case of these graphs where n is the number of nodes, $N \geq n-1$ is the number of arcs, and σ is the maximum node degree.[3]

We can improve the above space bound significantly if the given DFA is acyclic. More specifically, in Section 3.3, we obtain the following result in this case.

Theorem 2. *Given an initially-connected acyclic deterministic finite automata $\mathcal{A}$ having $n-1$ transient states, a unique dead state and working over an alphabet Σ of size σ, there exists a succinct encoding for $\mathcal{A}$ taking $(\sigma-1)(n-1)\log n + 3n + O(\log^2 \sigma) + o(n)$ bits of*

[3] A directed graph with labels on its arcs is deterministic if no two out-neighbor arcs have the same label. Since there are $\lceil \log \binom{n^2}{N} \rceil$ directed graphs [12] with n nodes and N arcs, each deterministic graph $G = (V, E)$ can have $L = \prod_{u \in V} d_u!$ label assignments for its arcs, where d_u s the out-degree of node u and $N = \sum_{u \in V} d_u$. Note that $\log L = \Theta(N \log \sigma)$ when labels are from Σ and thus $d_u \leq \sigma$.

space, which can optimally determine, given an input string x over Σ, whether $\mathcal{A}$ accepts x in time proportional to the length of x, using constant words of working space.

This is followed by the succinct data structure for NFA in Section 3.4 where we prove the following result.

Theorem 3. *Given an initially-connected non-deterministic finite automata $\mathcal{N}$ having n states and working over an alphabet Σ of size σ, there exists a succinct encoding for $\mathcal{N}$ taking $\sigma n^2 + n$ bits of space, which can determine, given an input string x over Σ, whether $\mathcal{N}$ accepts x, in polynomial time.*

Finally, we conclude in Section 4 with some concluding remarks.

2 Preliminaries

In this section we collect all the previous theorems and definitions that will be used throughout this paper.

2.1 Graph Terminology and Graph Algorithms

We will assume the knowledge of basic graph theoretic terminology (like trees, paths etc) as given in [6] and basic graph algorithms (mostly the depth first search (henceforth DFS) traversal of a graph and its related concepts) as given in [5]. Perhaps at this point it may seem slightly unusual that we are talking about graphs here when the focus of this paper is DFA/NFA and their succinct representations. Essentially in this paper we view DFA/NFA, more specifically their graphical representation i.e., *state transition diagram*, as a special case of an edge labeled directed graph G having n nodes corresponding to the $n = |Q|$ states of DFA/NFA, $m = \sigma n$ edges where $|\Sigma| = \sigma$ as each node has exactly σ outgoing edges, and each edge is labeled with some elements from Σ. It is with this point of view, we will design our succinct data structures for DFA/NFA in this paper.

2.2 Succinct Data Structures

Rank-Select. For a bit vector B and any $a \in \{0, 1\}$, the rank and select operations are defined as follows :

- $rank_a(B, i)$ = the number of occurrences of a in $B[1, i]$, for $1 \leq i \leq n$;
- $partial\_rank_1(B, i) = rank_1(B, i)$ if $B[i] = 1$, and -1 otherwise; and
- $select_a(B, i)$ = the position in B of the i-th occurrence of a, for $1 \leq i \leq n$.

We make use of the following theorems:

Theorem 4. *[4] We can store a bitstring B of length n with additional $o(n)$ bits such that rank and select operations can be supported in $O(1)$ time. Such a structure can also be constructed from the given bitstring in $O(n)$ time and space.*

Theorem 5. *[22] We can store a bitstring B of length n with m ones using $\log \binom{n}{m} + o(m) + O(\log \log n)$ bits such that $partial\_rank_1$ operations can be supported in $O(1)$ time. Such a structure can also be constructed from the given bitstring in $O(n)$ time and space.*

Succinct tree representation. We use following result from [18].

Theorem 6. *[18] Given a rooted ordered tree τ on n nodes, it can be succinctly represented as a sequence of balanced parenthesis of length $2n$ bits, such that given a node v, we can support subtree size and various navigational queries (such as parent and i-th child) on v in $O(1)$ time using an additional $o(n)$ bits. Such a structure can also be constructed in $O(n)$ time and space.*

Compact representation of increasing sequence. We use the following theorem from [24].

Theorem 7. *[24] Given an increasing integer sequence $a[\cdot]$ of length n such that $0 \leq a[1] \leq a[2] \leq \cdots \leq a[n] < u$, there exists a data structure to represent $a[\cdot]$ in compressed form using $O(\min\{\frac{1}{\epsilon}n^{\epsilon}u^{1-\epsilon}, \frac{1}{\epsilon}u^{\epsilon}n^{1-\epsilon}\})$ bits of space, where $\epsilon > 0$ is any fixed constant, such that any entry $a[i]$ and the value $\overline{a}[i] = |\{j \mid a[j] < i, 1 \leq j \leq n\}|$ can be still retrieved in constant time.*

We denote the above data structure by $D(n, u, \epsilon)$. If B denotes the characteristic vector for the sequence a, then computing $a[i]$ and $\overline{a}[i]$ correspond to computing select and rank on B.

Representation of a vector. We also make use of the following theorem from [7].

Theorem 8. *[7] There exists a data structure that can represent a vector $A[1..n]$ of elements from a finite alphabet Σ using $n \log |\Sigma| + O(\log^2 n)$ bits, such that any element of the vector can be read or written in constant time.*

3 Succinct Representations for DFA and NFA

In this section, we provide all the upper bound results of our paper dealing with DFA/NFA. Throughout this section, whenever we mention DFA (NFA resp.), it should refer to an initially-connected deterministic (non-deterministic resp.) finite automata having n states and working over an alphabet Σ of size σ. With this notation in mind, we start with the succinct encoding of DFA first.

3.1 Succinct Encoding of DFA

Bassino and Nicaud [3] proved a beautiful bijection between the state transition diagram of any DFA and pairs of integer sequences which can be represented by boxed diagrams (will be defined shortly) along with providing an efficient algorithm to perform this construction. We will refer the readers to [3] for complete details regarding the bijection, counting and many other details that we choose to not repeat here. However, we still need to provide some details/definitions (which basically follow their exposition) that are relevant to our own work and will also help to understand the results from their paper smoothly. Following [3], a *diagram* of width m and height n is defined as a sequence $(x_1, \ldots, x_m)$ of non-decreasing non-negative integers such that $x_m = n$, represented as a diagram of boxes. See Figure 2 for better visual description and understanding. A *boxed diagram* can be defined as a pair of sequences $((x_1, \ldots, x_m), (y_1, \ldots, y_m))$ where $(x_1, \ldots, x_m)$ is a diagram and for all i (such that $1 \leq i \leq m$), the y_i-th box of the column i of the diagram is marked. Note that $1 \leq y_i \leq x_i$. Thus, a diagram can lead to $\prod_{i=1}^{m} x_i$ boxed diagrams. A *k-Dyck diagram* of size n is defined as a diagram of width $m := (k-1)n + 1$ and height n such that $x_i \geq \lceil i/(k-1) \rceil$ for all $i \leq m - 1$. Finally, a *k-Dyck boxed diagram* of size n is boxed diagram where the first coordinate $(x_1, \ldots, x_{(k-1)n+1})$ is a k-Dyck diagram of size n. Given these definitions, Bassino and Nicaud [3] proved the following theorem.

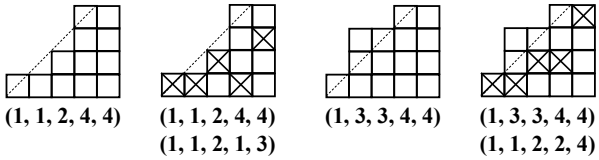

Figure 2 A diagram of width $m = 5$ and height $n = 4$, a boxed diagram, a k-Dyck diagram and a k-Dyck boxed diagram with $k = 2$. This example is borrowed from [3].

Theorem 9. *[3] The set $\mathcal{D}_n$ containing DFAs having n states and working over a σ-letter alphabet is in bijection with the set $\mathcal{B}_n$ of σ-Dyck boxed diagrams of size n. Moreover, the construction involving going from transition diagram of the DFA to k-Dyck boxed diagram and vice versa runs in linear time and space.*

Thus, by applying the above theorem, from any given DFA with n states and σ-letter alphabet, [3] produces a σ-Dyck boxed diagrams of size n, which can be in turn represented by two integer arrays $Max[1..m]$ and $Boxed[1..m]$ of length $m := (\sigma - 1)n + 1$ each. Furthermore, from these two arrays, it is possible to entirely reconstruct the DFA using the algorithm of Theorem 9. Thus, it is sufficient to store just these two arrays in order to encode any given DFA. For more details, readers are referred to [3]. For an example, see Figure 3 which will also serve as the working example for this part of our paper. In particular, the DFA of Figure 3 can be entirely encoded by the $Max[1..15] = \{3, 4, 4, 4, 4, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7\}$ and $Boxed[1..15] = \{1, 2, 3, 1, 4, 3, 4, 2, 3, 1, 4, 4, 5, 3, 6\}$ arrays of length $(\sigma - 1)n + 1 = 15$, and these can be computed using the algorithms of [3].

$\Sigma = \{a, b, c\}$
$Q = \{1, 2, 3, 4, 5, 6, 7\}$
$q_0 = \{1\}$
$F = \{7\}$

δ	a	b	c
1	2	7	6
2	3	5	4
3	1	4	4
4	2	3	1
5	3	6	1
6	4	2	3
7	4	5	3

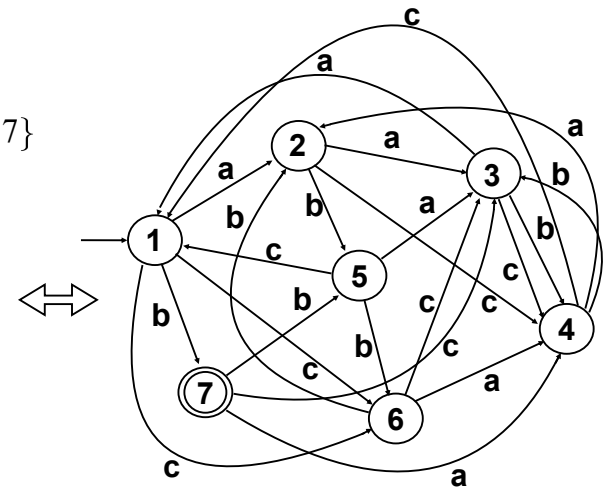

Figure 3 Two ways to define the same DFA. This DFA will serve as the working example for our discussion. By using the techniques of [3], this DFA can be entirely represented by the $Max[1..15] = \{3, 4, 4, 4, 4, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7\}$ and $Boxed[1..15] = \{1, 2, 3, 1, 4, 3, 4, 2, 3, 1, 4, 4, 5, 3, 6\}$ arrays of length $(\sigma - 1)n + 1 = 15$ each.

First, we observe that, by construction, the arrays satisfy $1 \leq Max[1] \leq Max[2] \leq \cdots \leq Max[m] \leq n$ and $1 \leq Boxed[i] \leq Max[i]$ for each $i = 1, 2, \ldots, m$. Now we consider the number

of bits needed to encode the array $Max[1..m]$. As it is an increasing integer sequence of length m and the range of the values is $[1, n]$, by using data structure $D(n, m, \epsilon)$ of Theorem 7, this array can be represented using $O(\frac{1}{\epsilon}m^{\epsilon}n^{1-\epsilon}) = O(\frac{1}{\epsilon}\{(\sigma-1)n+1\}^{\epsilon}n^{1-\epsilon})$ bits of space. By letting $\epsilon = 1/\log(\sigma-1)$, the size is $O(n\log\sigma)$ bits if $\sigma > 2$. If $\sigma = 2$, the space is obviously $O(n) = O(n\log\sigma)$ bits. Next we consider the number of bits required for array $Boxed[1..m]$. Because each entry of this array is an integer from 1 to n, we can use Theorem 8 to represent the $Boxed[1..m]$ array using $(\sigma-1)n\log n + O(\log^2 m)$ (recall $m = (\sigma-1)n+1$) bits. Thus, in total, the size of the representation using two integer arrays is $(\sigma-1)n\log n + O(n\log\sigma)$ bits. Because the information theoretic lower bound is $(\sigma-1)n\log n + O(n)$ bits for the representation of DFA, this representation is succinct.

We consider a special case when there is a failure/exit state labeled 0 and only N transitions among all the σn transitions go to non-failure states. Note that $Boxed$ has $N-n+1$ non-zero values. In this case we can reduce the space for $Boxed[1..m]$ by using a new bitvector $Z[1..m]$ which has $N-n+1$ ones. We use a new array $Boxed'[1..N-n+1]$ which stores non-zero values of $Boxed[1..m]$. Then $Boxed[i]$ is computed as follows. If $Z[i] = 0$, $Boxed[i] = 0$ (transition to the failure state). If $Z[i] = 1$, $Boxed[i] = Boxed'[partial\_rank_1(Z, i)]$. If we use the data structure of Theorem 4, Z is represented in $\sigma n + o(\sigma n)$ bits, which is asymptotically smaller than the space lower bound of $(\sigma-1)n\log n + O(n)$. But, by using the data structure of Theorem 5, the bitvector Z can be represented in $\log\binom{\sigma n}{N} + o(N) + O(\log\log(\sigma n)) = N\log\frac{\sigma n}{N} + O(N)$ bits to support $partial\_rank$ queries in $O(1)$ time. The space for $Boxed'$ is $(N-n+1)\log n$ bits. Therefore the total space for representing a DFA with N non-failure transitions is $(N-n)\log n + O(N\log\sigma)$ bits.

Even though this representation is optimal from the point of view of space occupancy, one major drawback of this representation is that, given a string x over Σ, it takes linear time (in the size of the DFA, i.e., $O(\sigma n)$ time where n is number of states of the DFA and σn is total number of transitions or edges in state transition diagram of the DFA) to decide whether the DFA accepts the string x, which is clearly not optimal as ideally it should be performed in time $O(|x|)$. This happens because the algorithm of Theorem 9 actually unravels the DFA from these two arrays $Max[1..m]$ and $Boxed[1..m]$, and then checks whether the input string can be accepted or not. Thus, from the point of view of string acceptance, this encoding of DFA is not optimal whereas space requirement point of view, this is optimal. This motivates the need of a succinct encoding of a given DFA, where the problem of string acceptance can be performed in optimal time (i.e., in time proportional to the string length). In what follows, we provide such an encoding.

3.2 Succinct Data Structure for DFA

Data structure: To design a succinct data structure for DFA, we need the following three bitvectors F, P and T in addition to an integer array $NewBoxed[1..m]$ (that can be obtained from the $Boxed[1..m]$ array of the previous section, as described later), which are defined as follows.

P is a balanced parentheses sequence of length $2n$ obtained from the lexicographic depth-first search (DFS) tree of the given input automaton $\mathcal{D}$. More specifically, given any DFA $\mathcal{D}$, we first perform the lexicographic DFS on $\mathcal{D}$ to generate the lexicographic DFS tree R of $\mathcal{D}$, i.e., while looking for a new edge to traverse during DFS, the algorithm always searches in lexicographic order of edge labels. For example, in Figure 3, from any vertex, lexicographic DFS first tries to traverse the edge labeled a, followed by b and finally c. The tree R is represented as a balanced parenthesis sequence P together with auxiliary structures to support the navigational queries on R, as mentioned in Theorem 6, using $2n + o(n)$ bits.

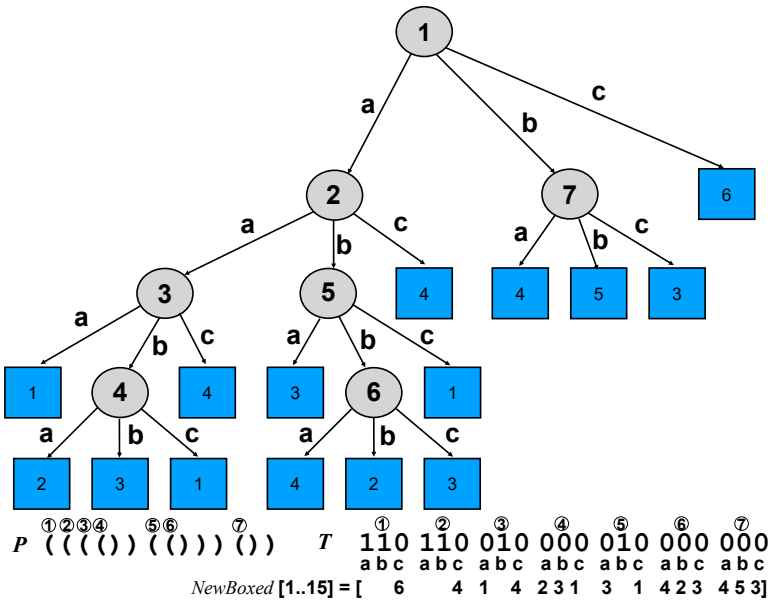

Figure 4 The extended lex-DFS tree S of the automaton of Figure 3 along with the corresponding bitvectors P, T, and the *NewBoxed*[1..15] array (the elements of this array are drawn exactly below the corresponding 0s with which they share one to one correspondence with). Note that, for the same automaton *Boxed*[1..15] array is given as $Boxed[1..15] = \{1, 2, 3, 1, 4, 3, 4, 2, 3, 1, 4, 4, 5, 3, 6\}$.

The bitvector F is used to mark all the final states of the input DFA, hence it takes n bits.

Before explaining the other bitvector, T, required for our succinct encoding, we want to explain the contents of Figure 4. The tree depicted in the figure is what we call an *extended lexicographic DFS tree* or *extended lex-DFS tree* (denoted by S) in short. If we delete the squared nodes and their incident edges (originating from the circled nodes), we obtain the lexicographic DFS tree of the automaton $\mathcal{D}$. Actually these edges represent the *back edges/cross edges/forward edges* [5] (i.e., non-tree edges) in the DFS tree of the automaton $\mathcal{D}$. Traditionally the vertices in the square are not drawn (as in our case of Figure 4), rather the edges point to the nodes in the circle only (hence all the nodes appear only once). We have chosen to draw and define the extended lex-DFS tree this way as it helps us to design and explain our succinct data structure well. Also note that, edges originating from a circled node and going to another circled node represents tree edges whereas edges from circled to squared nodes represent non-tree edges.

Now given the extended lex-DFS tree S, we visit the nodes of S in DFS order and append a bit string of length σ for each vertex v of S marking which of its children are attached to v via tree edges (marked with 1) and which are attached to v via non-tree edges (marked with 0) in the lexicographic order of the edge labels. The string obtained this way is referred to as T. Thus, T is a bit-vector of length σn which captures the information about the tree and non-tree edges of S. More specifically, it has exactly $n-1$ ones, which have one-to-one correspondence with the tree edges of the lexicographic DFS tree of DFA $\mathcal{D}$, and has exactly $(\sigma-1)n+1$ zeros, which correspond to non-tree edges of the lexicographic DFS tree of DFA $\mathcal{D}$. See Figure 4 for an example. We relabel all the states of $\mathcal{D}$ such that the i-th vertex (state) in R in preorder has label i, and also modify the transition function accordingly. Now it is easy to see that, for the state with label i $(1 \leq i \leq n)$, the corresponding node in the

lexicographic DFS tree has exactly σ outgoing edges, and we encode the tree edges among them using the bits in the range $T[\sigma(i-1)+1..\sigma i]$. More specifically, $T[\sigma(i-1)+c]=1$ if and only if the outgoing edge labeled c is a tree edge ($1 \leq c \leq \sigma$). Similarly, we can also find the j-th outgoing tree edge from the state i by $select_1(T, j + rank_1(T, \sigma(i-1)))$. Finally, we compress T by observing that the positions of 1s in the T array form an increasing sequence, hence by using the data structure $D(n-1, \sigma n, \epsilon)$ of Theorem 7, *access*, *rank* and *select* operations can be supported in constant time. By setting $\epsilon = 1/\log(\sigma-1)$, T can be encoded in $O(n \log \sigma)$ bits.

Now let us define the new integer array $NewBoxed[1..m]$. First, observe that elements of the array $Boxed[1..m]$ are nothing but the leaves (i.e., node labels in the squared nodes) of the extended lex-DFS tree S in the left to right order. More specifically, they are the node labels of the destinations of the non-tree edges emanating from the nodes of the lexicographic DFS tree of the automaton $\mathcal{D}$ in their preorder. Instead of this specific ordering (followed in the $Boxed[1..m]$ array), $NewBoxed[1..m]$ lists the same node labels in the order of their appearance in the T bitvector (from left to right). Note that, as mentioned previously, these node are marked by 0s in T and they are in one-to-one correspondence with all the non-tree edges of the lexicographic DFS tree of the automaton $\mathcal{D}$. Thus, the $NewBoxed[1..m]$ array contains the same node labels as the $Boxed[1..m]$ array, but in a different order. See Figure 4 for an example. This completes the description of our succinct data structure for DFA. Note that *Max* is no longer used in our data structure.

We now analyze the space complexity of our data structure. The array $NewBoxed[1..m]$ takes $(\sigma-1)n \log n + O(\log^2 m)$ bits (by similar analysis as before for the $Boxed[1..m]$ array). As mentioned previously, we store T using Theorem 7, hence it takes $O(n \log \sigma)$ bits. The bitvector F consumes n bits. Finally, the bitvector P is stored using Theorem 6, hence it occupies $2n + o(n)$ bits in total. Thus, overall our data structure uses $(\sigma-1)n \log n + O(n \log \sigma)$ bits. Hence, the data structure is succinct. It is easy to further reduce the size if the DFA has only $N < \sigma n$ non-failure transitions. Using the bitvector $Z[1..m]$ for indicating non-failure transitions, the array $NewBoxed[1..m]$ is compressed to $N - n + 1$ non-zero values, and the total space is $(N-n)\log n + O(N \log \sigma)$ bits. In what follows, we describe the string acceptance query algorithm using our data structures.

Query algorithm. Suppose we are given an input string x of length y over Σ, and we need to decide if the DFA $\mathcal{D}$ accepts x or not. We start the following procedure from the initial state (stored explicitly using $O(\log n)$ bits) and repeat until the end of the input string x. At any generic step, to figure out the transition function $\delta(q, c) := q'$ where $1 \leq q, q' \leq n$ are the states, we first look at the bit $T[\sigma(q-1)+c]$. If it is 1, the outgoing edge labeled c from state q is a tree edge. Let $j := rank_1(T, \sigma(q-1)+c) - rank_1(T, \sigma(q-1))$. Then the outgoing edge is the j-th tree edge of node q in the lex DFS tree. Therefore $q' = child(q, j)$ (supported using the Theorem 6). If the bit is 0, the outgoing edge labeled c from state q is a non-tree edge. Let $j := rank_0(T, \sigma(q-1)+c)$. Then the edge is the j-th non-tree edge in the DFA, and q' is obtained by $q' := NewBoxed[j]$. All of this can be done in constant time. Hence, when we reach the end of x, and if we are at an accepting/final states (can be figured out from the bitvector F), we say that the DFA $\mathcal{D}$ accepts x. It is easy to see that the whole procedure runs in time proportional to the length of the input string x along with using constant words of working space, hence our algorithm is optimal. This completes the proof of Theorem 1.

3.3 Succinct Data Structures for Acyclic DFA

As mentioned previously, an acyclic DFA $\mathcal{A}$ with total n states always has a unique dead state and $n-1$ transient (i.e., non dead) states. Another way to visualize $\mathcal{A}$ is to see that the state transition diagram of $\mathcal{A}$ does not have any cycles except at the unique dead state. Given such a setting, one can always use the succinct encoding (of the previous section) of an arbitrary DFA to represent them. In that case, we end up using $(\sigma-1)n\log n + O(n\log\sigma)$ bits of space. In what follows, we show that by exploiting the acyclic property, one can obtain improved space bound for representing $\mathcal{A}$.

We basically view the state transition diagram of $\mathcal{A}$ as a directed acyclic graph with a single source (i.e., the initial state), and a single sink i.e., the dead state (call it d). Given this, we first construct a spanning tree $W = (V, E)$ of $\mathcal{A}$ where $V = Q$ (i.e., the set of states of $\mathcal{A}$) and $E = \{(q_u, q_v) \mid \delta(q_v, \sigma) = q_u \text{ where } q_v \neq d\}$ by making the dead state d as the root of this tree. It is easy to see that such a spanning tree can always be constructed. By applying Theorem 6, we encode the structure of W using $2n + o(n)$ bits to support the navigational queries on W (in particular, the parent query) in $O(1)$ time. As done previously in Section 3.2 while constructing the succinct data structures for DFA, here also we relabel all the states of $\mathcal{A}$ such that the i-th vertex (state) in W in preorder has label i, and modify the transition function accordingly. Note that the dead state d is labeled with label 0 in this ordering, and we do not need to store the transition function for the dead state. We also mark in a bitvector of size n all the final states of $\mathcal{A}$, and we store the label of the start state. We then store a two dimensional array $L[1..n-1][1..\sigma-1]$ such that $L[q][i] = \delta(q, i)$ using data structure of Theorem 8. Thus, the overall space usage is $(\sigma-1)(n-1)\log n + 3n + O(\log^2\sigma) + o(n)$ bits.

In what follows, we explain how to check if $\mathcal{A}$ accepts any given string x over Σ. At any generic step, to compute $\delta(q, i)$, we simply output $L[q][i]$ if $i \in \{1, 2, \ldots, \sigma-1\}$; otherwise (i.e., if $i = \sigma$) the value of $\delta(q, \sigma)$ is given by the parent of q in W i.e., $\delta(q, i) = parent(q)$. Thus $\delta(q, i)$ can be computed in constant time, and hence we can optimally decide if $\mathcal{A}$ accepts x in time proportional to the length of x. This completes the proof of Theorem 2.

3.4 Succinct Encoding for NFA

As mentioned previously in Section 1.2, to encode an initially connected NFA on n states over a σ-letter alphabet Σ with a fixed initial state and one or more final states, we need at least $\sigma n^2 + n$ bits. In what follows, we show a very simple scheme achieving this bound.

We store a table H having n rows (corresponding to the n states of the input NFA) and σ columns (corresponding to each letter of the alphabet Σ). The entry $H[i][j]$ (where $0 \leq i \leq n-1$ and $1 \leq j \leq \sigma$) basically stores the corresponding transition function of the NFA i.e., $H[i][j] = \delta(q_i, j)$ where $q_i \in Q$ and $j \in \Sigma$. Now for an NFA, $\delta(i, j)$ is a subset of Q. If we store this subset explicitly, it might take $O(n\log n)$ bits in the worst case per transition of the NFA, leading to overall $\sigma n^2 \log n$ bits which is $O(\log n)$ multiplicative factor off from the optimal space requirement. Instead we simply store the charecteristic vector L of the subset (of length n, marking the corresponding states from the subset as 1, and rest of the bits in L are 0) where the state labeled i of the NFA moves to after reading the letter $j \in \Sigma$. Thus, the overall size of H is exactly σn^2 bits. Finally, we also mark in a separate bitvector (of length n) all the final states of the input NFA. Thus, in total the size of our encoding is given by $\sigma n^2 + n$ bits, which matches the lower bound. Hence, our encoding is succinct and optimal.

Now using our encoding, we can simply implement the classical algorithm (given in the

texts of [14, 23]) for checking if the NFA accepts a given input string or not, and this runs in polynomial time. Hence, we obtain the result mentioned in Theorem 3.

4 Concluding Remarks

We considered the problem of succinctly encoding any given DFA $\mathcal{D}$, acyclic DFA $\mathcal{A}$ or NFA $\mathcal{N}$ so as to check efficiently if they accept a given input string. To this end, we successfully designed succinct data structures for them that also support the string acceptance query optimally for DFA $\mathcal{D}$, acyclic DFA $\mathcal{A}$, and efficiently for NFA $\mathcal{N}$, matching the running times of the classical algorithms. To the best of our knowledge, our work is the first attempt to encode any mathematical models from the world of automata theory using the lens of succinct data structures, and we believe that our work will spur further interest in other similar problems in future.

References

1 H. Acan, S. Chakraborty, S. Jo, and S. R. Satti. Succinct data structures for families of interval graphs. In *WADS*, 2019.
2 L. C. Aleardi, O. Devillers, and G. Schaeffer. Succinct representations of planar maps. *Theor. Comput. Sci.*, 408(2-3):174–187, 2008.
3 F. Bassino and C. Nicaud. Enumeration and random generation of accessible automata. *Theor. Comput. Sci.*, 381(1-3):86–104, 2007.
4 D. R. Clark. *Compact Pat Trees*. PhD thesis. University of Waterloo, Canada, 1996.
5 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.
6 R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
7 Y. Dodis, M. Patrascu, and M. Thorup. Changing base without losing space. In *STOC*, pages 593–602, 2010.
8 M. Domaratzki. Enumeration of formal languages. *Bulletin of the EATCS*, 89:117–133, 2006.
9 M. Domaratzki, D. Kisman, and J. Shallit. On the number of distinct languages accepted by finite automata with n states. *Journal of Automata, Languages and Combinatorics*, 7(4):469–486, 2002.
10 A. Farzan and S. Kamali. Compact navigation and distance oracles for graphs with small treewidth. *Algorithmica*, 69(1):92–116, 2014.
11 A. Farzan and J. I. Munro. Succinct encoding of arbitrary graphs. *Theor. Comput. Sci.*, 513:38–52, 2013.
12 Arash Farzan and J. Ian Munro. Succinct encoding of arbitrary graphs. *Theor. Comput. Sci.*, 513:38–52, 2013. URL: `https://doi.org/10.1016/j.tcs.2013.09.031`, `doi:10.1016/j.tcs.2013.09.031`.
13 P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
14 J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.
15 G. J. Jacobson. *Succinct static data structures*. PhD thesis. Carnegie Mellon University, 1998.
16 V. A. Liskovets. Exact enumeration of acyclic deterministic automata. *Discrete Applied Mathematics*, 154(3):537–551, 2006.
17 J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations and functions. *Theor. Comput. Sci.*, 438:74–88, 2012.
18 J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001.
19 J. I. Munro and K. Wu. Succinct data structures for chordal graphs. In *ISAAC*, pages 67:1–67:12, 2018.

20 G. Navarro. *Compact Data Structures - A Practical Approach*. Cambridge University Press, 2016.

21 G. Navarro and K. Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3):16, 2014.

22 R. Raman, V. Raman, and S. R. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43, 2007.

23 M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

24 K. Sumigawa and K. Sadakane. An efficient representation of partitions of integers. In *IWOCA*, pages 361–373, 2018.