Languages Given by Finite Automata over the Unary Alphabet

Wojciech Czerwiński
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Maciej Dębski
Warsaw, Poland

Tomasz Gogasz
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Gordon Hoi
School of Informatics and IT, Temasek Polytechnic, Singapore, Singapore

Sanjay Jain
School of Computing, National University of Singapore, Singapore

Michał Skrzypczak
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Frank Stephan
Department of Mathematics and School of Computing, National University of Singapore, Singapore

Christopher Tan
Department of Mathematics, National University of Singapore, Singapore

Abstract

This paper studies the complexity of operations on finite automata and the complexity of their decision problems when the alphabet is unary and n the number of states of the finite automata considered. The following main results are obtained:

(1) Equality and inclusion of NFAs can be decided within time $2^{O((n \log n)^{1/3})}$; previous upper bound $2^{O((n \log n)^{1/2})}$ was by Chrobak (1986) via DFA conversion.

(2) The state complexity of operations of UFAs (unambiguous finite automata) increases for complementation and union at most by quasipolynomial; however, for concatenation of two n-state UFAs, the worst case is an UFA of at least $2^{\Omega(n^{1/6})}$ states. Previously the upper bounds for complementation and union were exponential-type and this lower bound for concatenation is new.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Nondeterministic Finite Automata, Unambiguous Finite Automata, Upper Bounds on Runtime, Conditional Lower Bounds, Languages over the Unary Alphabet

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2023.22

Related Version *Full Version*: `https://arxiv.org/abs/2302.06435`

Funding *Wojciech Czerwiński*: supported by the ERC grant INFSYS, agreement no. 950398.
Sanjay Jain: supported by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2019-T2-2-121 / R146-000-304-112 as well as NUS Provost Chair grant E-252-00-0021-01.
Michał Skrzypczak: supported by the National Science Centre, Poland (grant number 2021/41/B/ST6/03914).
Frank Stephan: supported by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2019-T2-2-121 / R146-000-304-112.

1 Introduction

This paper investigates the complexity and size-constraints related to languages over the unary alphabet – this is assumed everywhere throughout the paper – when these languages are given by a nondeterministic finite automaton (NFA) with a special emphasis on the case where this NFA is unambiguous. Unambiguous nondeterministic finite automata (UFAs) have many good algorithmic properties, even under regular operations with the languages, as long as no concatenation is involved. The study of unary languages is a theoretically important special case which allows for techniques and insights from number theory, as each word corresponds to the natural number which is its length. This sometimes gives techniques which can be extended to general case. Furthermore, the case is a bit special as the NFA-DFA trade-off is only $2^{O((n \log n)^{1/2})}$ and not 2^n as for other alphabets; Colcombet [3] published several influential conjectures which found much attention: Göös, Kiefer and Yuan [6] refuted his conjecture that any two n-state NFAs with disjoint languages can be separated by an UFA of size polynomially in n; this paper confirms a weak version of another conjecture by showing that unary UFAs have only a quasipolynomial blow-up for Boolean operations; Raskin [18] had refuted the original version of the conjecture.

In the following, a bound of type $2^{\Theta(n)}$ is called exponential and a bound of type $2^{n^{\Theta(1)}}$ is called exponential-type. The quasipolynomial functions are those in $O(n^{\log^{O(1)}(n)})$ which are not bounded by polynomials. In an expression of the form $2^{\alpha(n)}$, the function $\alpha(n)$ is called the exponent of the function. Let $f \in \Omega'(g)$ mean that there is a constant $c > 0$ such that, for infinitely many n, $f(n) \geq c \cdot g(n)$. Note that, under the Exponential Time Hypothesis (ETH), all **NP**-complete problems have, for infinitely many inputs, an exponential-type complexity and solving kSAT with $k \geq 3$ requires time $2^{\Omega'(n)}$. For this paper, for several results, the exponential of an exponential-type function from the upper bound is matched, assuming Exponential Time Hypothesis, by a lower bound whose logarithm differs from that of the upper bound only by a logarithmic or sublogarithmic expression.

Assume two languages L, H are given by n-state NFAs and one wants to know whether $L \subseteq H$. Chrobak [2] provided an algorithm by converting the NFAs into DFAs and then comparing these which runs, for the unary alphabet, in time $2^{O((n \log n)^{1/2})}$. Fernau and Krebs [5] proved, under the assumption of ETH, the conditional lower bound $2^{\Omega'(n^{1/3})}$ but did not resolve the remaining gap. Tan provides in an undergraduate thesis (UROP) [21] an alternative proof for the conditional lower bound by coding the variant of SAT where all clauses have at most length 3 and all variables occur at most 3 times; this variant is called 3-occur-3SAT. The present work narrows the gap significantly by providing a comparison algorithm which decides whether $L \subseteq H$ in time $2^{O((n \log n)^{1/3})}$, see Theorem 2 below.

Recall that for an unambiguous nondeterministic finite automata (UFA), every word outside the language has zero accepting runs and every word inside the language has exactly one accepting run – see below for more explanations for these technical terms. Prior research had established that the intersection of two n-state UFAs can be represented by an $O(n^2)$-state UFA and that, over the binary alphabet, the complexity of the Kleene star of an n-state UFA can be recognised (synonymously accepted) by an $O(n^2)$-state UFA [1, 3, 8, 14, 16, 17]. But the size-increase of the other regular operations (complement, union, concatenation) remained open; it was however known that disjoint union has linear complexity.

Unambiguous finite automata found much attention in recent research with a lower quasipolynomial lower bound for the blow-up of the size for complementation by Raskin [18] of the form $\Omega'(n^{(\log \log \log n)^q})$ where q is some positive rational constant. Thus it is impossible to achieve a polynomial sized complementation of UFAs for any alphabet. For comparison with the binary case, it should be mentioned that Göös, Kiefer and Yuan [6] proved that for binary languages of UFAs, the size of an NFA computing the complement is in $n^{\Omega'((\log n)/\mathrm{polylog}(\log n))}$. This lower bound also trivially applies to UFAs. However, this paper shows that, for unary alphabet, one can do all regular operations on UFAs with at most quasipolynomial size-increase except for concatenation where this paper achieves the lower bound of $2^{\Omega(n^{1/6})}$, confirming a weak version of a conjecture of Colcombet [3] – who predicted originally a polynomial size-increase for Boolean operations. For larger alphabets, there is a big gap between the lower bound $n^{\log n/\mathrm{polylog}(\log n)}$ by Göös, Kiefer and Yuan [6] and the upper bounds for complementing an n-state UFA of $2^{0.79n+\log n}$ by Jirásek, Jr., Jirásková and Šebej [13] and $\sqrt{n+1} \cdot 2^{n/2}$ by Indzhev and Kiefer [12].

For unary automata, Okhotin [15] determined the worst-case complexity on the determinisation of UFAs with $2^{\Theta((n\log^2(n))^{1/3})}$ and this is directly also an upper bound on the size-increase by complementation. In his master's thesis, author Dębski [4] constructed an algorithm to complement unambiguous automata over the unary alphabet while maintaining the upper size-bound of $n^{O(\log n)}$ for these automata. The present work tightens the analysis of the construction and gets the bound $n^{\log n+O(1)}$.

Note that it is not efficient to compare UFAs with respect to inclusion of the generated languages by constructing complementation of the second automaton and then taking intersection and checking for emptiness. Instead, one employs direct comparison algorithms; Stearns and Hunt [20] provided a polynomial time algorithm for this; this paper slightly improves it by providing a **LOGSPACE** algorithm in the case that the UFAs are in Chrobak Normal Form, and an **NLOGSPACE** algorithm without any assumption on a normal form. Note that the transformation into this normal form is a polynomial time algorithm which does not increase the number of states for UFAs; however, an **NLOGSPACE** algorithm might cause a polynomial size increase.

Tables 1, 2 and 3 provide the best known results by either this paper or prior work. Results of this paper refer to the corresponding theorem, proposition or remark while results of prior work give the corresponding reference. In particular, the tables summarise the results for UFAs. Previously published bounds are mentioned, where needed, near to the corresponding result. Note that $c(n) = n^{\log n+O(1)}$. Lower bounds on size imply lower bounds on computations; for concatenation a better lower bound is found (assuming ETH). In Table 1 the bounds on the size of UFAs are given. Table 2 (Table 3 respectively) give the bounds on the computation time (computation space respectively), needed for finding automata with the desired property or determining the truth of a formula are given. Note that in general for computing the complement, union, intersection and for comparing NFAs, the computation space $O(f(n))$ corresponds to the computation time $2^{O(f(n))}$ in the worst case. Thus every further improvement in the space usage will also have an improvement in the time usage. UFA comparison is in **NLOGSPACE**, which is more or less optimal and this corresponds to the polynomial time used for the comparison (though more concrete time bounds are given in the corresponding theorems).

2 Details of Technical Concepts and Methods

Finite Automata

A finite state automaton, see for example [9], is a tuple $(Q, \Sigma, Q_0, \delta, F)$, where Q is a finite set of states, Σ is a finite alphabet (unary for this paper), $Q_0 \subseteq Q$ is a set of initial states, δ is a transition function assigning to each pair in $Q \times \Sigma$ a subset of Q and $F \subseteq Q$ is a

Table 1 State complexity of operations with UFAs. Here $c(n) = n^{\log n + O(1)}$.

Operation	Lower Bound	Source	Upper Bound	Source
Intersection	$n^2 - n$	Holzer and Kutrib [7]	n^2	Holzer and Kutrib [7]
Complement	$n^{(\log\log\log n)^{\Omega(1)}}$	Raskin [18]	$c(n)$	Theorem 3
Disjoint union	$2n - 4$	–	$2n$	Jirásková and Okhotin [14]
Union	–	–	$n + n \cdot c(n)$	Remark 11
Symm. difference	$n^{(\log\log\log n)^{\Omega(1)}}$	Raskin [18]	$2n \cdot c(n)$	Theorem 3
Kleene Star	$(n-1)^2 + 1$	Čevorová [1]	$(n-1)^2 + 1$	Čevorová [1]
Concatenation	$2^{\Omega(n^{1/6})}$	Theorem 9	$2^{O((n \log^2 n)^{1/3})}$	Okhotin [15]

Table 2 Time bounds, except for the last, all refer to UFAs. The lower bounds using Ω' for computing concatenation, evaluating formulas using UFAs as inputs and comparing NFAs assume ETH and hold only infinitely often. The upper bound for universality in the first row assumes that the UFA is given in Chrobak Normal Form.

Operation	Lower Bound	Source	Upper Bound	Source
UFA Universality	–	–	$O(n \log^2 n)$	Theorem 5
UFA Comparison	–	–	Poly(n)	Stearns and Hunt III [20]
UFA Concatenation	$2^{\Omega'(n^{1/4})}$	Theorem 13	$2^{O((n \log^2 n)^{1/3})}$	Okhotin [15]
UFA Complement	$n^{(\log\log\log n)^{\Omega(1)}}$	Raskin [18]	$n^{O(\log n)}$	Theorem 3
UFA Formulas	$2^{\Omega'((n \log n)^{1/3})}$	Theorem 12	$2^{O((n \log^2 n)^{1/3})}$	Okhotin [15]
NFA Comparison	$2^{\Omega'(n^{1/3})}$	Fernau and Krebs [5]	$2^{O((n \log n)^{1/3})}$	Theorem 2

set of accepting states. A run of the automaton on input $a_1 a_2 \ldots a_n$ is a sequence of states $q_0, q_1, \ldots q_n$, such that $q_0 \in Q_0$ and for each $i < n$, $q_{i+1} \in \delta(q_i, a_{i+1})$. The run is accepting if $q_n \in F$. The input $a_1 a_2 \ldots a_n$ is accepted by the automaton if there is a run on it which is accepting. The set of words accepted by the automaton is called the language of the automaton. The terminologies word and string are used interchangeably.

Without further constraints, the automaton is called a *nondeterministic finite automaton* (NFA). An NFA accepts a word iff it has an accepting run for this word, that is, a word is accepted iff there is at least one run ending in an accepting state and a word is rejected iff there is either no run for the word at all or all runs for the word end in a rejecting state.

An NFA which has, for each word in its language L, exactly one accepting run and for each word outside L no accepting run is called an *unambiguous finite automaton* (UFA). An NFA which has exactly one start state and for which $\delta(q, a)$ has cardinality exactly one, for all $q \in Q$ and $a \in \Sigma$, is called a *deterministic finite automaton* (DFA). Note that every DFA has for each word exactly one run and this run is accepting iff the word is in the given language L. A language L is called *regular* iff L is the language of some finite automaton. The concept does not change if one requires this automaton to be a UFA or to be a DFA.

In the case of a unary alphabet one can bring an NFA or UFA into Chrobak Normal Form [2]. For the unary alphabet, a Chrobak Normal Form of the NFA is defined as follows: The NFA consists of states $q_1, q_2, \ldots, q_s$, $p_0^i, p_2^i, \ldots, p_{n_i-1}^i$, for i with $1 \le i \le r$, for some s, r and n_i, such that the following holds:

Table 3 Space Bounds; Result for Universality assumes input in Chrobak Normal Form and lower bound for NFA comparison assumes ETH and holds only infinitely often.

Operation	Lower Bound	Source	Upper Bound	Source
UFA Universality	–	–	$O(\log n)$	Theorem 5
NFA Comparison	ETH $\Rightarrow \Omega'(n^{1/3})$	Fernau and Krebs [5]	$O((n \log n)^{1/3})$	Theorem 2

(a) If $s > 0$ then q_1 is the starting state else $\{p_0^i : 1 \leq i \leq r\}$ is the set of starting states.
(b) $\delta(q_i, a) = \{q_{i+1}\}$, for i with $1 \leq i < s$ and $a \in \Sigma$.
(c) If $s > 0$ and $a \in \Sigma$, $\delta(q_s, a) = \{p_0^i : 1 \leq i \leq r\}$.
(d) $\delta(p_j^i, a) = \{p_{j+1 \mod n_i}^i\}$, for $a \in \Sigma$, $i \leq r$ and $j < n_i$.

The states $q_1, q_2, \ldots, q_s$ and the transitions within them is called the *stem* of the NFA, with q_s the last state of the stem. For $1 \leq i \leq r$, the states $(p_j^i)_{j<n_i}$ and the transitions among them, are called the *cycles* of the NFA, where n_i is the length of the cycle and p_0^i is the *entry state* of the cycle. Which states of an NFA are accepting depends on its language.

The Exponential Time Hypothesis (ETH)

Impagliazzo, Paturi and Zane [10, 11] observed that for many concepts similar to 3SAT the running time is exponential in the number n of variables. They investigated this topic and formulated what is now known as the Exponential Time Hypothesis: There is a constant $c > 0$ such that, every algorithm which solves 3SAT, has for infinitely many values of n the worst case time complexity of at least 2^{cn}. One defines $f \in \Omega'(g)$ iff there is a constant $c > 0$ such that for infinitely many n it holds that $f(n) > c \cdot g(n)$. Then one can obtain for many problems a conditional lower bound, that is, a lower bound implied by the Exponential Time Hypothesis where an absolute lower bound proof is unavailable. For example, for the unary NFA universality problem one gets the conditional lower bound of $2^{\Omega'(n^{1/3})}$.

In the present work, several lower bounds for the runtime to solve problems related to unary UFAs or NFAs are formulated under the assumption of the Exponential Time Hypothesis using Ω'-expressions. Furthermore, for constructing lower bounds, the following additional result of Impaglazzio, Paturi and Zane [10, 11] is important: Assuming the Exponential Time Hypothesis, there are constants $c, d > 1$ such that, for every algorithm solving 3SAT, there are infinitely many n such that for each of these n there is an instance with n variables, each occurring in the instance at most d times, such that the run time of the algorithm on this instance is at least c^n. Furthermore, one can show that one can choose $d = 3$ (at the expense of a perhaps smaller c, but still with $c > 1$).

The Prime Number Theorem

The prime number theorem says that ratio of the n-th prime number and $n \cdot \log n \cdot \log 2.71828\ldots$ converges to 1 where $2.71828\ldots$ is the Euler's number. A direct consequence of the prime number theorem is that there is a constant c such that, for each $n \geq 2$, there are $n/\log n$ prime numbers between n and cn.

In this paper, variations of these consequences are used in constructions to code 3SAT instances into NFAs and UFAs in order to show the hardness of decision problems. Often in proofs, the constructed NFA is in Chrobak Normal Form and consists of a stem of s states, where $s = 0$ is possible, and disjoint cycles $C_1, C_2, \ldots$ of lengths $p_1, p_2, \ldots$, with $n = s + p_1 + p_2 + \ldots$ being the overall number of states. In these cases, the states in cycle C_i will be considered to be ordered as 0-th, 1-st, … states in the cycle, where the unary alphabet takes the NFA from the current to the next state modulo p_i in the cycle C_i. If $s > 0$ the 0-th state in the cycle is the entry point into the cycle from the last state of the stem else each cycle has a start state which is its 0-th state. Assume that the stem has s states and an input of length t has been processed; if $s > t$ then the NFA is in the t-th state of the stem else the NFA is in the $(t - s \mod p_i)$-th state of the cycle C_i for some i.

Often (though not always) the p_i above would be either distinct prime numbers or distinct prime numbers times a common factor, the latter is in particular used for UFAs. This allows us to use Chinese Remainder Theorem to get that some possible combination of states is reachable in the different cycles for the same input word; in the case that all lengths are pairwise coprime, every combination can be reached.

3 The Nondeterministic Finite Automata Comparison Algorithm

The upper bound of the next result matches the lower bound $2^{\Omega'(n^{1/3})}$ of the timebound from the universality problem of unary NFAs [5] up to a factor $O((\log n)^{1/3})$ when comparing the logarithms of the corresponding run time bounds. A sketch of Tan's alternative proof [21] is in the appendix, as later results build on this method.

▶ **Proposition 1** (Fernau and Krebs [5]). *Given an m-variable 3-occur 3SAT instance, one can construct an $n = \Theta(m^3)$ sized NFA such that this NFA accepts all words over the unary alphabet iff the given instance is unsolvable. Thus unary NFA universality requires $2^{\Omega'(n^{1/3})}$ computation time provided that the Exponential Time Hypothesis holds.*

The upper bound $2^{((n \log n)^{1/3})}$ in the next result is only slightly larger than the lower bound $2^{\Omega'(\log n)^{1/3})}$. As the time used by an algorithm is at most $2^{O(\text{space used})}$, any further improvement in the space bound would also result in improvement in the time bound.

▶ **Theorem 2.** *Given two nondeterministic finite automata over the unary alphabet and letting n denote the maximum number of their states, one can decide whether the first NFA computes a subset of the second NFA in deterministic time $O(c^{(n \log n)^{1/3}})$ for a suitable constant $c > 1$. This timebound also applies directly to the comparison algorithm for equality and the algorithm for checking universality (all strings in the language). Furthermore, the algorithm can be adjusted such that the space used is $O((n \log n)^{1/3})$.*

Proof. Let n denote the maximum number of states of the two automata. Without loss of generality assume n is large enough so that the prime number theorem and other bounds needed below apply. One can transform (within polynomial time in the number of states) the nondeterministic finite automata into Chrobak Normal Form [2], where it consists of a stem of up to n^2 states, followed by parallel cycles which, together, use up to n states. Note that one can assume that the two stems have the same length. To see this, note that in Chrobak Normal Form, a stem can be made one longer by adding one state at the end of the stem and shifting the entry point into each cycle by one state. The new state in the stem would be accepting iff one of the prior entry points in the cycles was accepting. This is done repeatedly (at most $O(n^2)$ times) until the stems have the same length. The comparison of the behaviour on the stems of equal length before entering the cycles can be done by just comparing if the corresponding states at the same distance from the start are accepting/rejecting.

The comparison of the cycle part is therefore the difficult part. Thus, for the following, one assumes without loss of generality that the input NFAs are in Chrobak Normal Form, and do not have any stem. The NFAs thus consist only of disjoint cycles, each having one start state and the only nondeterminism is the choice of the start state, that is, the cycle to be used.

Note that a cycle C of length m in an NFA can be converted into a cycle C' of length w, where m divides w, by having the s-th state of C' as accepting if ($s \mod m$)-th state of C is accepting. This way, for appropriate value of w, one can combine several cycles whose lengths divide w into one cycle of length w. If one can get a set X of small number of w's (which pairwise have the same greatest common divisor (gcd) r), such that the lengths of all the cycles of the two NFAs divide at least one $w \in X$, then converting the two NFAs to have cycles only of lengths $w \in X$, allows for easier comparison of the two NFAs (these NFAs are called comparison normal form NFAs below).

Intuitively, lengths of only few (at most $(n/\log^2 n)^{1/3}$) cycles of the NFAs can have two large (at least $(n \log n)^{1/3}$) prime factors. The aim of P defined below is to collect all such prime factors along with the small primes. Set Q defined below collects the large primes which are not in P. Thus, P and Q together provide all the prime factors of the lengths of the cycles of the two NFAs.

Let $P = \{$prime number $p : p < (n \log n)^{1/3}$ or there exist a prime $q \geq (n \log n)^{1/3}$ such that one of the NFAs given in the input has a cycle with length divisible by $p \cdot q\}$.

Let $Q = \{$prime $p \leq n : p \notin P\}$.

Note that the number of primes smaller than $(n \log n)^{1/3}$ is at most $O((n/\log^2 n)^{1/3})$. Furthermore, the number of primes $p \geq (n \log n)^{1/3}$ such that the length of some cycle in one of the input NFAs is divisible by $p \cdot q$ for some prime $q \geq (n \log n)^{1/3}$ is at most $O((n/\log^2 n)^{1/3})$, as the cycles within each NFA are disjoint. Thus, the cardinality of P is at most $O((n/\log^2 n)^{1/3})$.

For each $p \in P$, let k_p be maximum number such that $p^{k_p} \leq n$. Note that p^{k_p} is the highest power of $p \in P$ which could divide the length of some cycle in the input NFAs. Similarly, q^2 is the highest power of $q \in Q$ which could divide length of any cycle in the input NFAs. Let r be the product of all p^{k_p}, $p \in P$. Note that r is in $O(n^{c' \cdot (n/\log^2 n)^{1/3}}) = O(2^{c' \cdot (n \log n)^{1/3}})$, for some constant c'. Thus, $r \leq c^{(n \log n)^{1/3}}$ for some constant $c > 1$.

Let $X = \{r \cdot q^2 : q \in Q\}$.

Note that lengths of all cycles in the two NFAs divide some $w \in X$. Moreover, the gcd of any two numbers in X is r. Now, for ease of comparing the two NFAs, one transforms each of these NFA into an equivalent "comparison normal form" NFA of size at most $r \cdot n^3$ as follows. For each $q \in Q$, the comparison normal form NFA has a cycle of length $r \cdot q^2$ such that the s-th state in this cycle is accepting iff there is a cycle of length p in the original automaton, where p divides $r \cdot q^2$ and $s \mod p$-th state in that cycle is accepting. Note that the comparison normal form NFA accepts the same language as the original NFA. The comparison normal form can be constructed in time $r \cdot Poly(n)$ by constructing each cycle separately, and comparing it with all cycles of length p in the original automaton, where p divides $r \cdot q^2$.

As the comparison normal form constructed is equivalent (for accepting language) to the original NFAs, it suffices to compare the two input NFAs in the comparison normal form, which is assumed below.

Now the first automaton recognises a subset of the set recognised by the second automaton iff for all $s < r$ one of the following two options holds:

(A) There is a $q \in Q$ such that in the second automaton, for all $t < q^2$, $(s + t \cdot r)$-th state in the cycle of length $r \cdot q^2$ are accepting;

(B) For every $q \in Q$ and for all $t < q^2$, if the $(s + t \cdot r)$-th state, in the cycle of length $r \cdot q^2$ is accepting in the first automaton then it is also accepting in the corresponding cycle of the second automaton.

This condition can be checked in time $r \cdot Poly(n)$: There are r possible values of s and for each such s, one has to check only $O(n^3)$ states, namely for each $q \in Q$, the $(s+t\cdot r)$-th state, where $t \in \{0,1,\ldots,q^2-1\}$; note that $q^2 \leq n^2$.

For correctness, it is first shown that (A) and (B) are sufficient conditions. Let $s < r$ be given. If (A) is satisfied, then the second automaton, for all t, accepts strings of length $s+t\cdot r$, as for the given q, all these strings are accepted by the cycle of length $r \cdot q^2$.

If (B) is satisfied and the first NFA accepts a string of length $s+t\cdot r$, then there is a $q \in Q$ such that, in the cycle of length $r\cdot q^2$, the $(s+t\cdot r) \mod (r\cdot q^2)$-th state is accepting. From the condition it follows that in the second automaton, in the corresponding cycle, $(s+t\cdot r) \mod (r\cdot q^2)$-th state is accepting, and therefore it also accepts the corresponding string.

For the converse, assume that the following condition (C) holds: For every $q \in Q$ there exists a t_q such that the $(s+t_q\cdot r)$-th state is rejecting in the cycle of length $r\cdot q^2$ in the second automaton and furthermore, for one q, the $(s+t_q\cdot r)$-th state is accepting in the cycle of length $r\cdot q^2$ in the first automaton. So (C) is true iff both (A) and (B) are false. Now there is an s' such that, after processing a string of length s', the first automaton can be at $(s+t_q\cdot r)$-th state for the cycle of length $r\cdot q^2$, for each $q \in Q$. It follows that the first automaton accepts a string of length s' while the second automaton rejects it.

For the space-bounded variant of the algorithm, the algorithm cannot bring the automaton into a normal form, as that cannot be stored within the space allowed. The comparison algorithm therefore has the translation into the above used normal form more implicit. One does the following:

First one computes r. Then one needs constantly many variables bounded by $2n^4\cdot r$, these variables can be stored in $O(\log r) = O((n\log n)^{1/3})$ space.

Second, for all cycles in the first automaton and all numbers $s \leq n^4\cdot r$ such that a string of length s is accepted by the current cycle of some length p in the first automaton, one does the following: If p divides r then let $q=1$ else let q be the unique prime such that p divides rq^2. Note that $q \leq n$. Now one checks if there is a number q' such that either $q'=1$ or q' is a prime $\leq n$ which does not divide r and for each $\ell = 0,1,\ldots,n^2$ there is a cycle of a length dividing rq'^2 in the second automaton which accepts the string of length $s+\ell rq^2$.

The language recognised by the first automaton is now a subset of the language recognised by the second automaton iff all the above tests in the algorithm have a positive answer. Note that r is chosen such that $\log c^{(n\log n)^{1/3}}$ and $\log r$ have the same order of magnitude and thus $O(\log c^{(n\log n)^{1/3}}) = O(\log(r\cdot n^4))$. Therefore, a real improvement of the space usage would also give an improvement of the computation time. ◀

4 Unambiguous Finite Automata and their Algorithmic Properties

Recall that an unambiguous automaton (UFA) satisfies that for every input word, there is either exactly one accepting run or none. On one hand, these are more complicated to handle than nondeterministic finite automata so that the union of n n-state automata cannot be done with n^2 states. On the other hand, they still, at least for unary alphabets, have good algorithmic properties with respect to regular operations (union, intersection, complementation, formation of Kleene star) and comparison (subset and equality).

▶ **Theorem 3.** *A UFA with up to n states has a complement with $n^{\log(n)+O(1)}$ states which can be computed in quasipolynomial time from the original automaton.*

Proof. Assume without loss of generality that the automaton is in Chrobak Normal Form. Furthermore, as inverting the states on the stem is trivial, it can be assumed without loss of generality, for an easier notation, that the given UFA consists just of m disjoint cycles $C_0, C_1, \ldots, C_{m-1}$ for some m, each having exactly one start state (denoted to be the 0-th state of the cycle).

Intuitively, the idea is to output a UFA using a recursive algorithm. At the start of the recursion, the aim is to output a UFA (without any stem) which accepts exactly the strings in the complement which have lengths $k \mod d$, with $k = 0, d = 1$. At each step of the recursion, with parameters k, d, either the algorithm

(a) returns a UFA (without any stem) which accepts exactly the strings in the complement which have lengths $k \mod d$ or

(b) makes recursive calls to obtain UFAs (without any stem) for accepting exactly the strings in the complement with lengths $(k + d \cdot s) \mod (d \cdot \ell)$, for some value of ℓ and s being $0, 1, \ldots, \ell - 1$. As the above UFAs would be without any stem accepting disjoint languages, the union of these UFAs will give a UFA for accepting exactly the strings in the complement which have lengths $k \mod d$.

Though, the algorithm is presented as a recursive algorithm, one can also view the solution as a tree, where the root of the tree has parameters $(k = 0, d = 1)$. Any node of the tree is either a leaf (i.e., it gives a UFA, without any stem, for accepting exactly the strings in the complement which have lengths $k \mod d$), or has ℓ children, for some ℓ, with parameters $(k + sd, d \cdot \ell)$, for s being $0, 1, \ldots, \ell - 1$ respectively in the ℓ children. The UFA for the complement thus will be the union of the UFAs at the leaves. Note that for any two leaves with parameters k', d' and k'', d'' there is no length m with m being same as both $k' \mod d'$ and $k'' \mod d''$. Thus, the above tree is also called a tree of different modulo residua.

Now the formal recursive algorithm is presented. Initially the algorithm is called with parameters $k = 0$ and $d = 1$.

Function UFAcomplement(k, d)

1. If there is a cycle C_i in the input UFA such that all strings of length $k + sd$ with $s < |C_i|$ are accepted by this cycle, then return to the calling instance of the recursion a UFA for emptyset (as the input UFA accepts all strings of length $k \mod d$, UFA for the complement needs to reject all strings of length $k \mod d$).
2. If there is no cycle C_j accepting any string of length $k + sd$ with $s < |C_j|$, then return one cycle of length d for which the k-th state is accepting and all other states are rejecting (as the given input UFA rejects all strings of length $k \mod d$, the UFA for complement needs to accept all such strings).
3. Otherwise there is a cycle C_h which accepts some but not all strings of the length $k \mod d$. The algorithm computes now the least common multiple $d' = lcm(d, |C_h|)$ and makes, for $s = 0, 1, \ldots, d'/d - 1$ a recursive call with the parameters $(k + ds, d')$. Return the union of the answers obtained from the recursive calls.

End of function UFAcomplement(k, d)

The algorithm clearly terminates, as when d is the multiple of all the cycle lengths and k is a number between 0 and $d - 1$, then every cycle C_i has the property that it either accepts all strings of length $k + sd$ or rejects all strings of length $k + sd$. However, the following claim shows that the value of d is much smaller at the termination step.

▷ Claim 4. Value of d at the termination step is at most $n \cdot (n/2) \cdot (n/2^2) \ldots \leq n^{0.5 \log n + c}$, for some constant c.

To see the claim, consider any branch of the recursive descent, with the values of (k, d) in the recursive calls being (k_0, d_0) (at the root), (k_1, d_1), …, where when the values were (k_i, d_i), then cycle C_{e_i} is chosen in step 3 (except at the last level which terminates in step 1 or step 2). For i not being the last level of the recursive descent, following properties hold:

(i) $d_{i+1} = lcm(d_i, |C_{e_i}|)$. In particular, d_i divides d_{i+1}.
(ii) $k_{i+1} = k_i + s_i d_i$, for some s_i.
(iii) C_{e_i} accepts some but not all strings of length $k_i \mod d_i$.
(iv) $|C_{e_i}|$ does not divide d_i but divides d_{i+1}.

Thus, for any levels g, h not being the last level of the recursive descent with $g < h$, using (i) and (ii) repeatedly, $k_h = k_g + s_{g+1}d_{g+1} + \ldots + s_{h-1}d_{h-1} = k_g + s'_h d_g$, for some s'_h, and d_g divides d_h. Using (iii) both C_{e_g} and C_{e_h} accept some words of length $k_g \mod d_g$. Thus, by UFA property of the input NFA, there is a common factor $b > 1$ of $|C_{e_g}|$ and $|C_{e_h}|$ which does not divide d_g. Note that b divides d_{g+1} as $|C_{e_g}|$ divides d_{g+1} (by (iv)). Thus, there is an extra common factor greater than 1 between d_{g+1} and $|C_{e_h}|$ compared to d_g and $|C_{e_h}|$, for each $g < h$. Thus common factor between d_g and $|C_{e_h}|$ is at least 2^g. It follows that d_{h+1}/d_h is at most $n/2^h$.

Thus, the number of levels is at most $\log n$ and the value of d is at most $n \cdot (n/2) \cdot (n/2^2) \ldots \leq n^{0.5 \log n + c}$, for some constant c, where one can safely assume $c \leq 5$. This proves the claim.

Also, it is easy to see by induction that the automaton generated by the algorithm is a UFA, as UFAcomplement(k, d) either returns a UFA in steps 1 or 2, or combines the UFAs generated by recursive calls in step 3, sets accepted by which are disjoint as they only accept strings of length $k + ds \mod d'$, for different values of s.

The automaton is the union of cycles up to length $n^{\log n/2+c}$ and in some kind of post-processing, one can unify distinct cycle of the same length d, with k_1-th, … k_s-th states as accepting into a single cycle of length d which has the k_1-th, …, k_s-th states as accepting states. After this post-processing, there are at most $n^{(\log n)/2+c}$ cycles and thus the overall size is at most $n^{\log n+2c}$. ◀

▶ **Theorem 5.**

(a) *One can decide in **LOGSPACE** whether an UFA in Chrobak Normal Form accepts all words. Without the **LOGSPACE** constraint, the running time is quasilinear, that is, of the form $O(n \log^2(n))$.*

(b) *Furthermore, one can decide in **LOGSPACE** whether an UFA U_1 in Chrobak Normal Form accepts a subset of the language accepted by another UFA U_2 in Chrobak Normal Form.*

Proof.

(a) First check if the states of the stem are all accepting, which can clearly be done in **LOGSPACE** by automata walk-through - pointers needed $O(\log n)$ memory to track positions in the UFA. Then one walks through each cycle and counts the number i_k of accepting states and the length j_k of the cycle. Now the UFA accepts all words, that is, is universal iff $\sum_k i_k/j_k = 1$. The proof starts by initially showing how to do this without being careful about space, but later it is shown how the computation can be modified to be done in **LOGSPACE**.

As the computation with rational numbers might be prone to rounding, one first normalises to one common denominator, namely $p = \prod_k j_k$ and furthermore computes $s = \sum_k i_k \cdot \prod_{h \neq k} j_k$. Now the above equality $\sum_k i_k/j_k = 1$ holds iff $s = p$.

The values of s and p can be computed iteratively by the following algorithm, note that there are at most n cycles and each time a cycle is processed, the corresponding values i_k and j_k can be established by an automata walk-through. So the loop is as follows:

1. Initialise $s = 0$ and $p = 1$;
2. For each k do find i_k and j_k; update $s = (s \cdot j_k) + i_k \cdot p$; $p = p \cdot j_k$ endfor;
3. if $s = p$ then accept else reject.

In this algorithm, only the variables p and s need more space than $O(\log n)$; the other variables are all pointers or numbers between 0 and n which can be stored in $O(\log n)$ space.

The values of s and p are at most $n^{2(n^{1/2})}$. To see this note that in Chrobak Normal Form, the cycle lengths can be assumed to be different (as same length cycles can be combined). Thus, there are at most $2n^{1/2}$ cycles, as the sum of their lengths is at most n implying that at most $n^{1/2}$ cycles have at least length $n^{1/2}$ and, furthermore, there are at most $n^{1/2}$ cycles shorter than $n^{1/2}$ due to different cycles having different length. Thus, instead of doing the above algorithm once with exact numbers, one computes in time $O(n \log n)$ the first $5 \cdot n^{1/2} + 2$ primes out of which 80% are above $n^{1/2}$ so that their product is above the maximum values s and p can take. As their product is larger than the upper bound $n^{1+2(n^{1/2})}$ of s and p, one then accepts iff all computations modulo each such prime q result in $s = p$ modulo q; this condition is, by the Chinese remainder theorem, equivalent to $s = p$ without taking any remainders. So the modified algorithm would be as follows.

1. Let $q = 2$; $\ell = 1$;
2. Initialise $s = 0$ and $p = 1$ (both are kept modulo q)
3. For each k do find i_k and j_k by transversal of the corresponding cycle; update (both computations modulo q) $s = (s \cdot j_k) + (i_k \cdot p)$; $p = p \cdot j_k$ endfor;
4. if $s \neq p$ (modulo q) then reject;
5. Let $\ell = \ell + 1$ and replace q by the next prime using a fast primality test and exhaustive search;
6. If $\ell < 5n^{1/2} + 2$ then goto step 2.

The automata transversal of each of these $O(n^{1/2})$ loops needs at most time $O(n \log n)$ as one transverses each of the cycles of the UFA to determine the corresponding i_k and j_k, the cycles are disjoint and have together at most n states. So the overall running time is $O(n^{3/2} \log n)$. If one would use more space, about $O(n^{1/2} \log n)$, then one can avoid the repeated computation of i_k, j_k by automata walk-throughs and gets the upper bound on the computation time of $O(n \log^2(n))$.

For the space usage, the computations modulo q need only $O(\log q)$ space which is then $O(\log n)$ space, as the first $5n^{1/2} + 2$ primes q and the storage of variables modulo q is all of $O(\log n)$. Primality tests can be done in $O(\log n)$ space for the usual way of doing it – checking all divisors up to the square root of the number.

(b) For this, note that basically the same idea as in (a) can be used to check if a UFA accepts all unary strings in sets vw^* for some v, w of length up to $2n$ with a slight modification of the UFA walk-throughs.

One partitions the words accepted by U_1 into two groups:

(i) A finite set X_1 of strings of length at most n (where n is the size of the UFA) and

(ii) A set X_2 consisting of subsets of the form vw^*, where v, w are unary strings with $n < |v| \leq 2n$ and $|w| \leq n$.

The strings in group (ii) above are from the cycles in U_1, by considering each accepting state in a cycle separately, and taking $|v|$ as the length of the smallest string leading to the state and $|w|$ as the length of the corresponding cycle.

Strings in group (i) can easily be checked using a walk-through of U_2, where if there is a branching into the cycles, one can do a depth first search.

For strings in group (ii), each set of form vw^*, where $n < |v| \leq 2n$ and $|w| \leq n$, is checked separately. As $|v| >$ the length of the stem part of U_2, one can first modify the cycle part of U_2 to always start in a state which is reached after $|v|$ steps, and ignore the stem part. This would basically mean that one needs to check if w^* is accepted in the modified U_2 (denote this modified U_2 as U_2'). For space constraints, note that one does not need to write down U_2', but just need to know the length by which the starting state of each cycle is shifted (which is the difference between $|v|$ and the length of the stem part of U_2). Now, for checking whether w^* is accepted by U_2', consider a further modified U_2'' formed as follows: for each cycle C in U_2' with length r and states $s_0, s_1, \ldots, s_{r-1}$ (s_0 being starting state, and transitions on unary input being from s_i to s_{i+1}, where $i+1$ is taken mod r) consider a cycle C' in U_2'' with states $s_0', s_1', \ldots, s_{r-1}'$ (s_0' being starting state, and transitions on unary input being from s_i' to s_{i+1}', where $i+1$ is taken mod r) where s_i' is an accepting state iff $s_{|v|+i\cdot|w| \mod r}$ was an accepting state in C. This new UFA U_2'' also has at most n states, the new length of each cycle is still the same as the lengths of the old cycles and the number of cycles do not increase. Now, similar to part (a), one just needs to check if U_2'' is accepting all unary strings. Here again note that one doesn't need to write down U_2'' fully, but just needs to check, for each cycle, its length and the number of accepting states, which can be done in **LOGSPACE**. ◀

Converting an UFA into Chrobak Normal Form is in **P** and **LOGSPACE** $\subseteq$ **P**, thus the following corollary holds.

▶ **Corollary 6** (Stearns and Hunt [20]). *One can decide the universality problem and the inclusion problem for two n-state UFAs in polynomial time.*

▶ Remark 7. One can improve Corollary 6 to computations in **NLOGSPACE** and, by Savitch's Theorem [19], in **DSPACE**$((\log n)^2)$. Details are in the appendix.

▶ Remark 8. For an UFA (of size n) for a language L over the unary alphabet, the language L^* can be recognised even by a DFA of quadratic size in n [1].

Thus if one allows in regular languages Kleene star, Kleene plus and the Boolean set-theoretic operations (but no concatenation), then the output of constant-size expressions, with parameters being given by languages accepted by n-state UFAs, can be recognised by UFAs of quasipolynomial size. Furthermore, Boolean-valued constant-sized quantifier-free formula with same type of parameters and comparing such subexpressions by $=$, $\subseteq$ and $\neq$ can be evaluated in quasipolynomial time.

Furthermore, one can show, as **NLOGSPACE** $\subseteq$ **POLYLOGSPACE**, that the above mentioned operations can all be computed by **POLYLOGSPACE** algorithms. This is not true for the concatenation, as the next Theorem 9 shows that there is an exponential-type blow-up – therefore **POLYLOGSPACE** is not enough to count the number of output symbols. Here some more details for the regular operations different from concatenation.

For the complementation, the handling of the stem is standard. The algorithm in Theorem 3 above has mainly two running variables for the recursive descent: d and k. Both take at most the value $n^{\log n/2+c}$ for some constant c and therefore can be written with $O(\log^2 n)$ bits. Furthermore, during recursion, one has to archive the old values before branching off, thus the algorithm archives $(d_{h'}, k_{h'}, e_{h'}, s_{h'})$ from the algorithm in Theorem 3 for each level h' and this can be done with $O(\log^3 n)$ space. Furthermore, note that the transformation into Chrobak Normal Form also takes just $O(\log^2 n)$ space due to Savitch's Theorem: $O(\log n)$ nondeterministic space is contained in $O(\log^2 n)$ deterministic space. In particular the algorithm is a **POLYLOGSPACE** algorithm. The above space bound can be improved somewhat by noting the following: instead of archiving the full tuples $(d_{h'}, k_{h'}, e_{h'}, s_{h'})$, one could archive the index information $e_{h'}, s_{h'}$ when going from level h' to $h'+1$, along with the quotient $d_{h'+1}/d_{h'}$. When returning one level down, one computes $d_{h-1} = d_h/(d_h/d_{h-1})$ where d_h/d_{h-1} was archived for $h-1$ and furthermore, one computes $k_{h-1} = k_h - s_{h-1} \cdot d_{h-1}$. With these modifications, the algorithm runs in $O(\log^2 n)$ space.

The intersection of two automata can just be done by computing the product automaton. This is known to be doable in **LOGSPACE**.

The union is the complement of the intersection of the complements and, by the above algorithms, can be done in **POLYLOGSPACE**.

Yu, Zhuang and Salomaa [22] showed that the Kleene star of a unary language given by an n-state NFA can be computed by a DFA of size $(n-1)^2+1$; thus one can search using an **NLOGSPACE** algorithm for the first $m \geq 1$ such that, for $k = 0, 1, \ldots, (n+1)^4$, every word of length n^2+k are accepted iff the word of length n^2+m+k is accepted and this length is then the period which starts latest at length n^2. This allows to output a DFA of size $O(n^2)$ which consists of a stem of length n^2 and a period of length m, where the **NLOGSPACE** algorithm allows for each of the states involved to check whether it is accepting.

▶ **Theorem 9.** *There is an exponential-type blow-up for UFA sizes when recognising the concatenation of unary languages; the concatenation of two languages given by n-state UFAs requires, in the worst case, an UFA with $2^{\Omega(n^{1/6})}$ states.*

Proof. Let m be a numeric parameter and let $p_0, \ldots, p_{k-1}$ be the first $k = m/\log m$ primes of size at least m; note that $m > k+4$ for all sufficiently large m. These primes are within $\Theta(m)$ by the prime number theorem. Now the UFA U to be constructed contains k cycles C_ℓ of length $p_\ell \cdot (k+3)$ for $\ell = 0, 1, \ldots, k-1$. The cycle C_ℓ has, for $h = 0, 1, \ldots, p_\ell - 2$, $(\ell + 2 + h \cdot (k+3))$-th state as accepting. There is one further cycle C' of length $k+3$ which has 0-th, 1-st and $k+2$-th states as accepting. Let L denote the language recognised by this UFA. The lengths of k consecutive unary strings not being accepted by the above UFA are exactly at lengths $r \cdot (k+3) + 2, \ldots, r \cdot (k+3) + k + 1$ where r is $p_\ell - 1$ modulo p_ℓ for $\ell = 0, 1, \ldots, k-1$, and this does not happen at any other lengths. Let H be the finite language which contains the words of length $0, 1, \ldots, k-1$ and no other words. Now $L \cdot H$ contains all words whose length does, for at least one ℓ, not have the remainder $k + 1 + (k+3) \cdot (p_\ell - 1)$ modulo $(k+3) \cdot p_\ell$. The complement of $L \cdot H$ is a periodic language which contains exactly those words which have, for all ℓ, the remainder $k+1+(k+3) \cdot (p_\ell - 1)$ modulo $(k+3) \cdot p_\ell$. So every NFA or UFA recognising this language needs cycles of length at least $(k+3) \cdot p_0 \cdot p_1 \cdot \ldots \cdot p_{k-1}$. The length of this cycle is at least $\Theta(m^k \cdot (k+3))$ and any UFA recognising it needs at least the same number of states. Furthermore, the UFA U has n states with $n \in \Theta(m) \cdot \Theta(k^2) = \Theta(m^3/\log^2 m)$. It follows that $n \cdot \Theta(\log^2(m)) = \Theta(m^3)$ and, using $\Theta(\log n) = \Theta(\log m)$ as n, m are polynomially related, that $\Theta(n \cdot \log^2(n)) = \Theta(m^3)$. So $m \in \Theta((n \cdot \log^2 n)^{1/3})$. Now the concatenation of $L \cdot H$ has an UFA of size o to be determined and its complement, by the above, has an UFA of at least size $(m)^{m/\log m}$. Using that the complement of an UFA of size h can be done in size $h^{\log h + O(1)}$, one has that the logarithm of the corresponding sizes satisfies this: $\log^2 o \geq \Theta((\log m + O(1)) \cdot m/\log m) \geq \Theta(m)$ and $\log o \geq \Theta(m^{1/2})$. Now the input UFA is of size n with $m \in \Theta((n \log^2 n)^{1/3})$ and thus one has that $\log o \in \Omega(n^{1/6})$. So concatenating two languages given by n-state UFAs can require an UFA with $2^{\Omega(n^{1/6})}$ states. ◀

▶ Remark 10. The above result stands in contrast to the situation of NFAs. It is well-known that the concatenation of two n-state NFAs needs only $2n$ states. However, the above construction shows that forming the complement in the unary NFAs then blows up from $2n$ states to $2^{\Omega(n^{1/6})}$ states, giving an exponential-type lower bound for this operation; a direct construction leading to the bound $2^{\Omega((n \log n)^{1/2})}$ is known [15]. Furthermore, Pighizzini [16, 17] showed that the concatenation of two unary DFAs of size n can be realised by an DFA of size $O(n^2)$; actually, he gives an explicit formula on the size of the stem and the cycle which depends only on the size of the stems and the cycles in the two input automata. Pighizzini's result allows for an implementation of the following concatenation algorithm [16, 17]: Convert the two UFAs into DFAs and then apply the algorithm to make the concatenation of DFAs; this gives the upper bound of $2^{O((n \log^2 n)^{1/3})}$ for the size of the concatenation UFA.

▶ Remark 11. The following facts are known about the complexity of operations with UFAs.

Holzer and Kutrib [7] showed that the intersection of two UFAs is just given by the n^2 state sized product automaton of the two n-state UFAs which preserves the property of being UFAs.

The bound $2n$ for the disjoint union by Jirásková and Okhotin [14] are obtained by the simple union of the two UFAs (this might give multiple start states, which is allowed for UFAs); note that as the languages are disjoint, on each word in the union, one can reach as accepting state only in one of the UFAs and there, by assumption, the way into the accepting state is unique. The lower bound $2n - 4$ for the disjoint union can be obtained by two even length cycles differing by length 2; one cycle has the accepting states at some of the odd numbered states and the other one at some of the even numbered states.

For the general union of two languages L, H given by n-state UFAs, consider the formula $L \cup (H - L)$ where $H - L$ is equal to the intersection of H and the complement of L which gives the upper bound $n \cdot c(n) + n$ with $c(n)$ being the bound for complementation.

The symmetric difference of two languages L, H is the disjoint union of $L - H$ and $H - L$, thus an upper bound is $2n \cdot c(n)$. For its lower bound, note that if L is given and H is the set of all strings, then the symmetric difference $(L - H) \cup (H - L)$ is just the complement of L.

Finite formulas refers to a formula with several input automata combining the input n-state UFAs with regular operations to a new UFA. Subsequent results provide the lower bound of $O(2^{\Omega'(n^{1/3})})$ for the time complexity to evaluate formulas. This almost matches the upper bound provided by transforming the UFAs into DFAs and exploiting the polynomial bounds on size increase of DFAs over the unary alphabet for regular operations.

In summary: All standard regular operations except concatenation have polynomial or quasipolynomial size-increase but concatenation has exponential-type size-increase.

Next one considers the evaluation of constant-sized formulas using UFAs as input. In the following, H_1, H_2, L are sets of words given by n-state UFAs and K is a finite language.

▶ **Theorem 12.** *Assuming the Exponential Time Hypothesis, it needs time $2^{\Omega'((n \log n)^{1/3})}$ to evaluate the truth of the formula $(H_1 \cap H_2) \cdot K = L$ where H_1, H_2, K are given by at most n-state UFAs and L is the set of all words.*

Proof. Consider a 3SAT formula with clauses $c'_1, c'_2, \ldots, c'_m$, where each variable appears in at most 3 clauses. Divide the clauses into $r = \lceil m/\log m \rceil$ disjoint groups of $\log m$ clauses each (where the rounded down value of $\log m$ is used). Group G_i has the clauses $c'_{(i-1)\cdot(\log m)+1}, \ldots, c'_{i\cdot(\log m)}$ (the last group G_{r-1} may have lesser number of clauses due to m not being divisible by $\log m$).

If a variable, say x, appears in different groups of clauses, then one can rename x in different groups to $x', x'', \ldots$ and add equality clauses $(x' = x''), \ldots$. Thus, by adding some additional equality clauses, it can be assumed that no variable appears in two different group of clauses. Note that there can be at most $O(m)$ equality clauses.

So, now the SAT formula has the clauses $c_1, c_2, \ldots, c_m$ (which have at most 3 literals each) and the equality clauses $c_{m+1}, c_{m+2}, \ldots, c_{m'}$, where the clauses $c_1, c_2, \ldots, c_m$ are divided into groups $G_0, G_1, \ldots, G_{r-1}$ containing at most $\log m$ clauses each, and any variable appears in clauses from at most one group, and perhaps in equality clauses.

As each group contains at most $\log m$ clauses, and thus at most $3 \log m$ variables, there are at most $8m$ possible truth assignments to variables appearing in clauses of any group. Below, k-th truth assignment (starting from $k = 0$) to variables assigned to p_i assumes some

Table 4 Accepting and Rejecting States.

	$k \cdot (2m' + 2) + 2j$-th state in C_i / C'_i	$k \cdot (2m' + 2) + 2j + 1$-th state in C_i / C'_i
$j = 0$, $i = 0$	accepting	accepting
$0 < j \leq m$ and all variables of c_j belong to G_i	accepting iff c_j not satisfied by the k-th truth assignment to variables assigned to p_i	accepting iff c_j not satisfied by the k-th truth assignment to variables assigned to p_i
This row is applicable only for C_i where $m < j \leq m'$ and c_j is $(x = y)$ and x is assigned to p_i	accepting iff truth value assigned to x is true in the k-th truth assignment to the variables assigned to p_i	accepting iff truth value assigned to x is false in the k-th truth assignment to the variables assigned to p_i
This row is applicable only for C'_i where $m < j \leq m'$ and c_j is $(x = y)$ and y is assigned to p_i	accepting iff truth value assigned to y is false in the k-th truth assignment to the variables assigned to p_i	accepting iff truth value assigned to y is true in the k-th truth assignment to the variables assigned to p_i
All other cases	not accepting	not accepting

ordering among the truth assignments where if the number of truth assignments is less than k, then k-th truth assignment is assumed to be the 0-th truth assignment (the latter part is just for ease of writing the proof).

Consider r distinct primes $p_0, p_1, \ldots p_{r-1}$, each greater than $8m$ but below some constant c times m. Note that, for large enough constant c, there exist such distinct primes by the prime number theorem.

Assign variables / clauses appearing in group G_i to prime p_i. Now UFA for H_1 consists of the r cycles $C_0, C_1, \ldots, C_{r-1}$ and UFA for H_2 consists of the r cycles $C'_0, C'_1, \ldots, C'_{r-1}$. The cycles C_i and C'_i are of length $2(m' + 1) \cdot p_i$. For $k < p_i$, and $j < m'$, the $k \cdot (2m' + 2) + 2j$-th and $k \cdot (2m' + 2) + 2j + 1$-th states in cycle C_i and C'_i are accepting or non-accepting as given in Table 4. Intuitively, for c_j assigned to p_i, C_i and C'_i will test for satisfaction (3rd row in the table). If $c_j = (x = y)$, and x is assigned to p_i and y to $p_{i'}$, then 4th and 5th rows in the table for C_i and $C'_{i'}$ respectively will check for consistency in the assignment to the variables x and y. Note that in Table 4 for the entries of the cycles C_i and C'_i, the value i can be considered as constant and the value j is the running variable going over all clauses.

Note that the above automatons are unambiguous, as for $j = 0$, only cycle C_0, C'_0 could accept; for $1 \leq j \leq m$, only C_i, C'_i such that c_j is assigned to p_i can accept; for $m < j \leq m'$, if c_j is $(x = y)$ with x and y assigned to p_i and $p_{i'}$ respectively, only C_i and $C'_{i'}$ respectively can accept. All strings with length being 0 or 1 modulo $(2m' + 2)$ are in $H_1 \cap H_2$ as defined in the table.

Now, if the 3SAT formula is satisfiable, then consider some satisfying truth assignment to the variables, say it is the k_i-th truth assignment to variables assigned to p_i. Then, for any s such that for $i < r$, $k_i = s \mod p_i$, $H_1 \cap H_2$ will not contain strings of length $s(2m' + 2) + 2j$ and $s(2m' + 2) + 2j + 1$ for $1 \leq j \leq m'$: (a) if $1 \leq j \leq m$, clause c_j is satisfied and thus these strings are not accepted by H_1 and H_2 (see 3rd row in the table for definition of C_i and C'_i, for c_j being assigned to p_i); (b) if $m < j \leq m'$ and $c_j = (x = y)$, where x is assigned to p_i and y to $p_{i'}$, then as the variable assignments are consistent again these strings are accepted by only one of H_1 and H_2 (see 4th and 5th rows in the table for definition of C_i and $C'_{i'}$ respectively). Thus $H_1 \cap H_2$ misses $2m'$ consecutive strings.

On the other hand if $H_1 \cap H_2$ misses $2m'$ consecutive strings, then it must be strings of length $s(2m'+2)+j$ for $2 \leq j \leq 2m'+1$, for some value of s (as all strings with lengths being 0 or 1 modulo $2m'+2$ are in $H_1 \cap H_2$). Then let $k_i = s \mod p_i$. Then, for the k_i-th assignment of truth values to the variables assigned to p_i, it must be the case that all the clauses c_j assigned to p_i are satisfied (otherwise by 3rd row in the table for definition of C_i, C'_i, H_1 and H_2 will contain $s(2m'+2)+2j$ and $s(2m'+2)+2j+1$). Furthermore, the equality clauses are satisfied, as if some equality clause $c_j = (x = y)$ is not satisfied then, if x is assigned to p_i and y is assigned to $p_{i'}$, then by the 4th and 5th rows in the table for definitions of C_i and $C_{i'}$ respectively, one of $s(2m'+2)+2j$ and $s(2m'+2)+2j+1$ length strings is in both $H_1 \cap H_2$ depending on the truth assignment to x in the k_i-th and y in the $k_{i'}$-th truth assignments to the variables assigned to p_i and $p_{i'}$ respectively.

Thus, $H_1 \cap H_2$ misses out on $2m'$ consecutive strings iff the given 3SAT formula is not satisfiable. Taking K to be set of strings of length $1, 2, \ldots, 2m'-1$, gives us that $(H_1 \cap H_2) \cdot K$ is universal iff 3SAT formula is not satisfiable.

The size n of the UFAs for H_1, H_2, K is bounded by $(2m'+2) \cdot (\sum_i p_i) = \Theta(m^3/\log m)$. Thus, $n \log n = \Theta(m^3)$ or $m = \Theta((n \log n)^{1/3})$. The theorem now holds assuming ETH. ◀

Note that Okhotin [15] provides an upper bound by converting the UFAs into DFAs and then carrying out the operations with DFAs. These operations run in time polynomial in the size of the DFAs constructed. While the size lower bound for concatenation of two n-state UFAs is just $2^{\Omega(n^{1/6})}$, the following conditional bound on the computational complexity of finding an UFA for the concatenation is $2^{\Omega'(n^{1/4})}$ when using the Exponential Time Hypothesis – for details see the appendix.

▶ **Theorem 13.** *Under the assumption of the Exponential Time Hypothesis, one needs at least $2^{\Omega'(n^{1/4})}$ time to compute an UFA for the language of the concatenation of the languages of two given n-state UFAs in worst case.*

▶ Remark 14. The above result also proves that for deciding whether the concatenation of two languages given by n-state UFAs is universal, the Exponential Time Hypothesis implies a $2^{\Omega'(n^{1/4})}$ lower bound. The upper bound of this is slightly better than the DFA-conversion bound $2^{O((n \log^2 n)^{1/3})}$ of Okhotin [15]: Theorem 2 proves that the universality of an NFA can be checked in time $2^{O((n \log n)^{1/3})}$ and as the concatenation of two n-state UFAs can be written as an $2n$-state NFA whose universality can be checked with the same upper bound.

5 Conclusion

The main results of the present work are the following ones: (1) There is an $2^{O((n \log n)^{1/3})}$ time algorithm for comparing NFAs with respect to $=$ and $\subseteq$; (2) given n-state UFAs, the size of UFAs for their union and complement is at most $O(n^{\log n + O(1)})$ states; (3) the concatenation of two n-state UFAs needs, in the worst case, at least $2^{\Omega(n^{1/6})}$ states. Furthermore, conditional lower bounds and upper bounds for the time and space complexities of operations and decision problems on UFAs are provided. The main question remaining unresolved is to close the gap between the lower bound $2^{\Omega(n^{1/6}))}$ and Okhotin's upper bound $2^{O((n \log^2 n)^{1/3})}$ for concatenating two n-state UFAs. Several related results can be found in the longer technical report version of this paper on `https://arxiv.org/abs/2302.06435`.

References

1 Kristína Čevorová. Kleene star on unary regular languages. In *Descriptional Complexity of Formal Systems: 15th International Workshop, DCFS 2013, London, ON, Canada, July 22-25, 2013. Proceedings 15*, volume 8031 of *LNCS*, pages 277–288. Springer, 2013.

2 Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.

3 Thomas Colcombet. Unambiguity in automata theory. In *Seventeenth International Workshop on Descriptional Complexity of Formal Systems*, volume 9118 of *LNCS*, pages 3–18. Springer, 2015.

4 Maciej Dębski. State complexity of complementing unambiguous automata, Master's thesis, University of Warsaw, 2017.

5 Henning Fernau and Andreas Krebs. Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24, 2017.

6 Mika Göös, Stefan Kiefer, and Weiqiang Yuan. Lower bounds for unambiguous automata via communication complexity. In *Forty Ninth International Colloquium on Automata, Languages and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 126:1–13, 2022. See also the technical report on `arXiv:2109.09155`.

7 Markus Holzer and Martin Kutrib. Unary language operations and their nondeterministic state complexity. In *International Conference on Developments in Language Theory*, volume 2540 of *LNCS*, pages 162–172. Springer, 2002.

8 Markus Holzer and Martin Kutrib. Descriptional and computational complexity of finite automata – A survey. *Information and Computation*, 209(3):456–470, 2011.

9 John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 3rd edition, 2007.

10 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

11 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

12 Emil Indzhev and Stefan Kiefer. On complementing unambiguous automata and graphs with many cliques and cocliques. *Information Processing Letters*, 177:106270:1–5, 2022.

13 Jozef Jirásek Jr, Galina Jirásková, and Juraj Šebej. Operations on unambiguous finite automata. *International Journal of Foundations of Computer Science*, 29(05):861–876, 2018.

14 Galina Jirásková and Alexander Okhotin. State complexity of unambiguous operations on deterministic finite automata. In *Twentieth International Conference on Descriptional Complexity of Formal Systems*, volume 10952 of *LNCS*, pages 188–199. Springer, 2018.

15 Alexander Okhotin. Unambiguous finite automata over a unary alphabet. *Information and Computation*, 212:15–36, 2012.

16 Giovanni Pighizzini. Unary language concatenation and its state complexity. In *Implementation and Application of Automata: 5th International Conference, CIAA 2000 London, Ontario, Canada, July 24–25, 2000 Revised Papers 5*, volume 2088 of *LNCS*, pages 252–262. Springer, 2001.

17 Giovanni Pighizzini and Jeffrey Shallit. Unary language operations, state complexity and Jacobsthal's function. *International Journal of Foundations of Computer Science*, 13(01):145–159, 2002.

18 Michael Raskin. A superpolynomial lower bound for the size of non-deterministic complement of an unambiguous automaton. In *Fortyfifth International Colloquium on Automata, Languages and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 138:1–11, 2018.

19 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

20 Richard Edwin Stearns and Harry B Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, 1985.

21 Christopher Tan. Characteristics and computation of minimal automata, UROP, National University of Singapore, 2022.

22 Sheng Yu, Qingyu Zhuang, and Kai Salomaa. The state complexities of some basic operations on regular languages. *Theoretical Computer Science*, 125(2):315–328, 1994.

A Appendix of Outcommented Proofs

▶ **Proposition 1** (Fernau and Krebs [5], Tan [21]). *Given an m-variable 3-occur 3SAT instance, one can construct an $n = \Theta(m^3)$ sized NFA such that this NFA accepts all words over the unary alphabet iff the given instance is unsolvable. Thus unary NFA universality requires $2^{\Omega'(n^{1/3})}$ computation time provided that the Exponential Time Hypothesis holds.*

Proof. Suppose an m-variable 3SAT instance with at most $3m$ clauses, where each variable occurs at most three times, is given. Without loss of generality assume $m \geq 8$ and $\log m$ is a whole number. Let the clauses be $u_1, u_2, \ldots, u_{m'}$ (where $m' \leq 3m$) and the variables be $x_1, \ldots, x_m$.

Let $r = \lfloor \frac{\log m}{3} \rfloor$, and $s = \lceil m'/r \rceil$. Consider the primes $p_0, p_1, \ldots, p_{s-1}$, where $8m \leq p_i \leq c'm$, for some constant c'. Note that by the prime number theorem there exists such a constant c'.

Assign to each prime p_i, the clauses $u_{(i-1)\cdot r+1}, \ldots, u_{i \cdot r}$. Intuitively, for each $i < s$, there will be a cycle of length p_i which will explore all possible truth assignments to variables in the clauses assigned to p_i, and check whether they satisfy the corresponding clauses assigned. Consistency of truth assignments to variables across clauses assigned to different primes p_i and p_j will be checked using a cycle of size $p_i \cdot p_j$. Further details can be found below.

Note that r clauses have at most $3r$ literals, and thus the number of possible truth assignments to these literals is at most 2^{3r} with $2^{3r} \leq m \leq p_i$. Order these assignments in some way so that one can say k-th truth assignment etc.

(1) For each $i < s$, form a cycle of length p_i. The k-th state (for $k < 2^{3r}$) in this cycle is rejecting iff the k-th truth assignment to the $3r$ literals in the clauses assigned to p_i satisfy all the clauses assigned to p_i. Note that if $k \geq 2^{3r}$, then the k-th state is accepting.

(2) For each pair i, j such that clauses assigned to p_i and p_j have a common variable, form a cycle of length $p_i \cdot p_j$. The k-th state in this cycle is accepting iff the $(k \mod i)$-th truth assignment to the literals in the clauses assigned to p_i and the $(k \mod j)$-th truth assignment to the literals in the clauses assigned to p_j are inconsistent within or with each other.

Now note that the above NFA rejects a unary string of length ℓ iff the following are satisfied:

(A) for each $i < s$, $(\ell \mod p_i)$-th truth assignment to the literals in clauses assigned to p_i satisfy the clauses assigned to p_i.

(B) for each $i, j < s$, if clauses assigned to p_i and p_j have a common variable, then $(\ell \mod p_i)$-th and $(\ell \mod p_j)$-th truth assignment to the literals in clauses assigned to p_i and p_j respectively are consistent.

Thus, the language accepted by the above NFA is universal iff the 3SAT formula is not satisfiable. The number of states in the above NFA is bounded by $(3m/r) \cdot c'm$ (for cycles in (1)), plus $(c'm)^2 \cdot 3m$ (for cycles in (2), as there are m variables each appearing at most thrice, so one needs to check at most 3m pairs). So the number of states is proportional to m^3. It follows from the Exponential Time Hypothesis that the complexity of testing universality for n-state NFA is at least $2^{\Omega'(n^{1/3})}$. ◀

▶ Remark 7. One can improve Corollary 6 to computations in **NLOGSPACE** and, by Savitch's Theorem [19], in $\mathbf{DSPACE}((\log n)^2)$. The reason is that **NLOGSPACE** is closed under complementation, allows to store constantly many states (given by their address in the input) and allows to check whether there is a path from one state to another in a given number of steps (as long as that is bounded by a polynomial). For that reason, one can also check whether, for a number $m \leq n$, one can reach an accepting state in exactly m steps from the start state. This allows to produce a stem of length n from the input where acceptance / rejection is labelled correctly for all of these states; let q be the last state of this stem. Then for each number $m = n+1, \ldots, 2n$, one does the following:

- Check whether there exist numbers h, k such that
  1. $h < k$ and $n+1 \leq m \leq n+k-h$ and m modulo $k-h = k$ modulo $k-h$;
  2. There is an accepting state r reachable from start state in exactly h, k, m steps – note that there is exactly one such accepting run for each of these numbers due to the UFA property;
  3. The run to the state r of length h has no repetition of states;
  4. The run to the state r of length k has some state q' repeated twice and all states visited twice and all states visited before reaching q' or from the second visit of q' onwards are also visited by the run accepting the word of length h;
  5. No state of the run to state r of length k is visited three or more times;
  6. The accepting runs for the words of length k and length m visit the same states.
- If so, then retain h, k, m and continue else go to the next m.
- In the case that h, k, m are found and retained, output a cycle connected by one edge from state q of length $k-h$ which has exactly one accepting state which is visited on reading a word of length m.

This so output automaton is in Chrobak Normal Form and it is also unambiguous due to the input automaton being unambiguous; furthermore, its size is at most quadratic in the size of the input automaton and thus, the complexity class of this algorithm combined with that of Theorem 5 is **NLOGSPACE**.

▶ **Theorem 13.** *Under the assumption of the Exponential Time Hypothesis, one needs at least $2^{\Omega'(n^{1/4})}$ time to compute an UFA for the language of the concatenation of the languages of two given n-state UFAs in worst case.*

Proof. Recall the proof of Theorem 12 and the values of m, m' there. The cycles of H_1 and H_2 in the proof of Theorem 12 have length $2(m'+1) \cdot p$ for some number p of size $O(m)$ which is either a prime or the constant 1. Furthermore for each number $\ell = 0, 1, \ldots, 2m'+1$, there is at most one cycle A_ℓ in the UFA for H_1 and at most one cycle B_ℓ in the UFA for H_2 which accepts a string of length ℓ modulo $2(m'+1)$ (if any).

Thus one can construct a new intersection automaton of H_1 and H_2 such that it consists of $2(m'+1)$ cycles E_ℓ, where E_ℓ has the length $\mathrm{lcm}(|A_\ell|, |B_\ell|)$ and the t-th state of E_ℓ is accepting iff t has the remainder ℓ when divided by $2(m'+1)$ and both cycles A_ℓ and B_ℓ, after t steps from the start state of those cycles, are in an accepting state in the corresponding automata H_1 and H_2 – here $\mathrm{lcm}(a, b)$ denotes least common multiple of a and b. Thus the so constructed product automaton consists of all cycles E_ℓ and accepts a word iff both H_1 and H_2 accept this word. Furthermore, the automaton, for each $\ell = 0, 1, \ldots, 2m'+1$, for any h, accepts a string of length $\ell + 2(m'+1)h$, only in the cycle E_ℓ, if at all. Thus the constructed automaton is an UFA.

As the size of each cycle E_ℓ is $O(m^3)$ and as there are $2(m'+1)$ cycles and $m' \in \Theta(m)$, the overall size of this automaton is $O(m^4)$. Thus, when n is the size of the product automaton, as the new automaton recognizes $H_1 \cap H_2$, then using Theorem 12, the universality problem for the concatenation of two n-state UFAs is in $2^{\Omega'(n^{1/4})}$ under the assumption of the Exponential Time Hypothesis. ◀