Notes on Counting with Finite Machines

Dmitry Chistikov

Max Planck Institute for Software Systems (MPI-SWS)
Kaiserslautern and Saarbrücken, Germany
dch@mpi-sws.org

Abstract

We determine the descriptional complexity (smallest number of states, up to constant factors) of recognizing languages $\{1^n\}$ and $\{1^{tn}\colon t = 0, 1, 2, \ldots\}$ with state-based finite machines of various kinds. This task is understood as *counting* to n and modulo n, respectively, and was previously studied for classes of finite-state automata by Kupferman, Ta-Shma, and Vardi (2001). We show that for Turing machines it requires $\log n/ \log\log n$ states in the worst case, and individual values are related to Kolmogorov complexity of the binary encoding of n. For deterministic pushdown and counter automata, the complexity is $\log n$ and $\sqrt{n}$, respectively; for alternating counter automata, we show an upper bound of $\log n$. For visibly pushdown automata, i. e., if the stack movements are determined by input symbols, we consider languages $\{a^n b^n\}$ and $\{a^{tn} b^{tn}\colon t = 0, 1, 2, \ldots\}$ and determine their complexity, of $\sqrt{n}$ and $\min(n_1 + n_2)$, respectively, with minimum over all factorizations $n = n_1 n_2$.

1998 ACM Subject Classification F1.1 Models of Computation, F4.3 Formal Languages

Keywords and phrases State complexity, Unary languages, Counting

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.339

1 Introduction

The task of counting is one of the most basic tasks that can be entrusted to computing devices. In the framework of formal language theory, it is natural to associate counting with finite machines that recognize singleton languages $\{1^n\}$, where $n = 0, 1, 2, \ldots$ Since even most primitive devices, namely finite automata, are expressive enough to perform this task, questions from the realm of descriptional complexity arise.

A standard problem setting here can be traced back to a classic 1971 paper by Meyer and Fischer [16] and can be stated as follows: given a specific class $\mathcal{C}$ of computational machines (such as nondeterministic finite automata), estimate the function $f(n)$, whose value on an arbitrary non-negative integer n is defined as the smallest descriptional complexity of a machine from this class that recognizes the language $\{1^n\}$. One then says that machines from $\mathcal{C}$ *count to* n with complexity $f(n)$. All classes are usually expressive enough to contain at least one appropriate machine for each n, and so it is always the case that $f(n) < \infty$.

To the best of our knowledge, the most thorough account of the problems of counting with finite machines can be found in a technical report from 2001 by Kupferman, Ta-Shma, and Vardi [12], which coins the term *counting to* n to refer to the problem of recognizing the language $\{1^n\}$. Kupferman et al. study this problem for standard classes of finite machines: deterministic, nondeterministic, universal, and alternating automata, and characterize the smallest number of states sufficient to count to n within each of these classes. Disregarding constant factors, this number is n for deterministic and nondeterministic automata, $\sqrt{n}$ for universal and $\log n$ for alternating automata.

Our contribution

Our primary focus is on the problems of counting with more powerful devices, capable of recognizing non-regular languages. As far as we know, these problems have never been addressed systematically. Observe that although one of the goals of the study of descriptional complexity is to compare the expressive power of machines from different classes, the notion of complexity in the definition above is tied to a specific class of machines. As a consequence, for different classes one has to use "matching" complexity measures. For all classes of machines considered in this paper, we use measures that generalize state complexity of finite automata.

Our results are as follows. First, for Turing machines with tape alphabet $\{0, 1\}$, we show in Section 3 that the complexity of counting is at most $\log n / \log\log n$. As a corollary, for an arbitrary string s (not necessarily over the one-letter alphabet), the state complexity of generating it with a Turing machine is asymptotically equal to $K(s)/(|\Delta|-1)\log K(s)$, where $K(\cdot)$ denotes the standard Kolmogorov complexity and Δ is the tape alphabet. (For $|\Delta| \geq 3$, this follows from a construction by Chaitin [4, sections 1.5–1.6], and we fill in the gap for $|\Delta| = 2$.) In other words, whenever there exists a Turing machine that produces s when run on empty tape and has binary encoding of length k, there always exists another Turing machine with the same properties that has at most $k/(|\Delta| - 1)\log k$ states, and the latter bound is asymptotically tight.

Second, we study the problems of counting for classes of pushdown automata (PDA). For deterministic PDA, we show in Section 4 that matching upper and lower bounds of $O(\log n)$ and $\Omega(\log n)$ for all n follow from earlier results due to Pighizzini [19] and Chistikov and Majumdar [7]. Subclasses of PDA, however, require separate consideration. For deterministic counter automata, i. e., when the stack alphabet contains only one symbol apart from the bottom-of-stack, we obtain an upper bound of $O(\sqrt{n})$ and show a matching lower bound $\Omega(\sqrt{n})$ by reducing to counting with deterministic finite automata (DFA). For alternating counter automata, we prove an upper bound of $O(\log n)$ (here we use a complexity measure that is more refined than just the number of states, so this result is not subsumed by the fact, due to Kupferman et al., that alternating finite automata count to n with $\lceil \log n \rceil$ states).

Third, we consider another well-known subclass of PDA called visibly pushdown automata (VPA, also known as nested word automata and input-driven PDA) in Section 5. While counting (and, indeed, recognizing any unary language) for VPA is easily shown to be as hard as for finite automata, we prove that deterministic VPA recognize languages $\{a^n b^n\}$—another problem interpreted as counting—with complexity $\Theta(\sqrt{n})$. We also show that the complexity of recognizing languages $\{a^{tn} b^{tn} \colon t = 0, 1, 2, \ldots\}$ is $\Theta(\min(n_1 + n_2))$, where the minimum is taken over all factorizations $n = n_1 n_2$; this function ranges, for different values of n, between $\Theta(\sqrt{n})$ and $\Theta(n)$. By showing sequences for which its value is $o(n)$, we refute a conjecture of Salomaa [22] on the state complexity, with respect to VPA, of a related language.

Acknowledgements. This paper would be very different without many people's help. I would like to thank Rupak Majumdar for support and for sharing a reference to an earlier version, from 1999, of the technical report by Kupferman, Ta-Shma, and Vardi [12]. I am grateful to Alexander Shen, who pointed me to Chaitin's work [4] upon hearing the result of Corollary 2. Abuzer Yakaryılmaz asked a question on StackExchange that let me come up with a construction of an alternating counter automaton recognizing a non-regular language $\{1^n \colon n = 2^k,\ k = 0, 1, 2, \ldots\}$ [3] and lead to Theorem 5. I would also like to thank Rose Hoberman for useful discussions.

2 Related work

As explained above, the starting point for our work is the technical report of Kupferman, Ta-Shma, and Vardi [12], who build upon the results of Leiss [13], Birget [2], and Chrobak [8]. From the general language-theoretic perspective, the problems of counting are closely tied to numerous phenomena of unary languages, that is, languages over a one-letter alphabet (also known as tally languages). Classes of unary languages possess many properties that cannot be observed in classes of languages over larger alphabets. Perhaps the most widely known is the theorem saying that every unary context-free language is regular, first proved by Ginsburg and Rice in 1962 [11].

Although the languages $\{1^n\}$ studied in the present paper are finite and therefore regular, we are interested in their descriptional complexity with respect to classes of machines that recognize, in general, non-regular languages. Hence, related are not only descriptional complexity questions for machines specifying non-regular sets, but also questions of so-called succinctness of representations, or economy of description. This term is associated with the following question, first asked by Meyer and Fischer in 1971 [16]: suppose that some language L belongs to a certain class $\mathcal{C}$; how short a description can this language L have with respect to $\mathcal{C}$, compared to the shortest description within some specific subclass $\mathcal{C}' \subsetneq \mathcal{C}$ such that $L \in \mathcal{C}'$? Or, in other words, what is the largest blowup that can be observed when translating a description from the class $\mathcal{C}$ into the "terms" of $\mathcal{C}'$?

For general context-free and regular languages (with sizes of context-free grammars—CFG—and deterministic finite automata—DFA—as complexity measures), the answer to this question is given by Meyer and Fischer: the description of a DFA cannot be bounded by any recursive function in the size of a CFG. For unary languages, translations from CFG (in Chomsky normal form) into NFA and DFA are shown to be at most exponential [20].

One can easily see that using PDA instead of CFG also leaves the translation exponential. Tight bounds on the size of the blowup in the case of unary nondeterministic PDA are also given in [20], and the unary deterministic case is studied in [19]. A general connection between unary deterministic PDA and grammar compression of binary words is established in [7]. In the present paper, we use these results to obtain bounds on the size of PDA.

In general, descriptional complexity problems for general context-free languages per se have so far been mostly associated with grammars and not with automata. As pointed out above, however, PDA have been intensively studied in connection with questions of economy of description. In this area, we wish to highlight the paper [10], which not only continues the line of research started by Meyer and Fischer, but also proves exponential lower bounds for descriptional complexity of PDA recognizing several natural finite languages.

For context-free grammars, we refer the reader to a recent paper [9], which obtains strong lower bounds on the size of CFG for several specific finite languages. Counting problems for context-free grammars are tightly related to the well-studied concept of an addition chain (see, e. g., [6, Section V-B]), which is a restriction, to the unary alphabet, of a context-free grammar that generates a single word. (A more general topic is, of course, grammar compression of non-unary words, see also [21, 6, 15].) As an illustrative example, one can easily show that the language $\{1^n\}$ can be generated by a context-free grammar with at most $O(\log n)$ symbols in right-hand sides of productions, and prove that this bound is tight.

For the subclass of context-free languages recognized by visibly pushdown automata, descriptional complexity questions have been studied somewhat more extensively; see, e. g., [17, 18], where these machines are called "input-driven pushdown automata".

As for descriptional complexity for Turing machines, the number of states as a complexity measure was studied by Chaitin [4]; his paper initiated the study of what is now known as Kolmogorov complexity [14], but then the focus very quickly shifted towards a different measure, the length of a binary string that encodes the description of a machine. In our work, we fill in the remaining gap for the original measure, i. e., for the number of states.

3 Turing machines

A *Turing machine* (TM) with input alphabet Σ and tape alphabet $\Delta \supseteq \Sigma$ is a tuple $\mathcal{M} = (\Sigma, \Delta, \Box, Q, q_0, H, \delta)$, where $\Box \in \Delta \setminus \Sigma$ is the blank symbol, Q the set of states, $q_0 \in Q$ the initial state, H the set of halting states, and $\delta \subseteq (Q \times \Delta) \times (Q \times \Delta \times \{-1, 0, +1\})$ the transition relation.

Configurations of $\mathcal{M}$ have the form (q, μ, n), where $q \in Q$, $\mu \colon \mathbb{Z} \to \Delta$, and $n \in \mathbb{Z}$; the interpretation is that cells of the infinite tape, indexed by $\mathbb{Z}$, contain symbols from Δ specified by μ, with $|\mu^{-1}(\Delta \setminus \{\Box\})| < \infty$, the control state of the machine is q, and the head of the machine is at the nth cell of the tape. If $\mathcal{M}$ is run on input $w \in \Sigma^*$, then the initial configuration is $(q_0, \mu_w, 0)$ where $\mu_w(i)$ is $w[i]$ for $0 \le i < |w|$ and $\Box$ otherwise; halting configurations are those with $q \in H$. The transition relation imparts a step-reachability on configurations: if $(q, \sigma, q', \tau, d) \in \delta$ and $q \notin H$, then at a configuration (q, μ, n), if $\mu(n) = \sigma$, the machine can overwrite this σ with τ, change its control state to q', and change the head's position by d. A machine is *deterministic* if from each configuration at most one configuration is reachable in one step (i. e., δ defines a mapping from $Q \times \Delta$ to $Q \times \Delta \times \{-1, 0, +1\}$).

Instead of the problem of *recognizing* the language $\{1^n\}$ with a Turing machine, we consider the problem of *generating* this language; our results can be easily extended to language recognition as well. A machine $\mathcal{M}$ *outputs* a word $u \in (\Delta \setminus \{\Box\})^*$ when run on $w \in \Sigma^*$ if from $(q_0, \mu_w, 0)$ it reaches some configuration (q, μ, n) with $q \in H$ such that $\mu^{-1}(\Delta \setminus \{\Box\}) = [a, a + |u| - 1]$ for some $a \in \mathbb{Z}$ and $\mu(a + i) = u[i]$ for $0 \le i < |u|$.

Let $Q_\Delta(s)$, $s \in \Sigma^* \subsetneq \Delta^*$, be the smallest number m such that there exists a deterministic TM with m states and tape alphabet Δ that outputs s when run on empty tape. The interpretation is that TMs with tape alphabet Δ count to n with complexity $Q_\Delta(1^n)$.

▶ **Theorem 1.** *Fix $\Sigma = \{1\}$, $\Delta = \{0, 1\}$, and $\Box = 0$. Then $Q_\Delta(1^n) \lesssim \log n / \log\log n$.*[1]

Proof. We construct, for each n, a Turing machine $\mathcal{M}_n$ with tape alphabet $\{0, 1\}$ that has $\log n / \log\log n\,(1 - o(1))$ states and outputs 1^n when run on empty tape. Let us show the workings of $\mathcal{M}_n$. Let $n = \sum_{i=0}^{l-1} b_i k^{l-1-i}$ be the k-ary representation of n, with $b_i \in \{0, 1, \ldots, k-1\}$. The value of k will be fixed later. The TM will keep two counters on the tape, denoted i and v, and written out in unary, as 1^i and 1^v respectively. Initially $v = 0$ and $i = 0$. Next, the machine will get into the main loop and do the following, as long as $i < l$: multiply v by k, add b_i to v, and increment i. For multiplication, two auxiliary counters u and j are needed: at first $j = k$ and $u = 0$, then the machine will add v to u and decrement j as long as $j > 0$; when j becomes 0, the value of u takes the place of v. In the main loop, when i becomes l, the TM will terminate with output v.

It is easy to see that this procedure is correct: $v = n$ at the end of the computation. Let us count the number of states needed to implement it. For initial setup, $O(1)$ states suffice. In the main loop, all operations are fixed and take up $O(1)$ states, with two exceptions: one of setting j to k, and another of choosing (and generating) b_i according to i (we also include

[1] We write $f(n) \lesssim g(n)$ iff $f(n) \le g(n) \cdot (1 + o(1))$.

termination on $i = l$ into this exception). For the former, $k + O(1)$ states suffice; we focus on the latter next. (We omit the description of how to keep all counters on the tape, because this can be done with standard techniques.)

The choice of b_i is implemented as follows: suppose on the tape there are only blanks (i. e., 0s) to the right of the counter i. This counter is kept as 1^i. Create a set of $l + k$ states of $\mathcal{M}_n$, denoted $q_0, q_1, \ldots, q_{l-1}$ and $p_0, \ldots, p_{k-1}$ (here q_0 is *not* the initial state of $\mathcal{M}_n$). Let the machine transition to q_0 and step on the leftmost 1 in 1^i. At each of the states q_j, $j < l - 1$, if the observed cell tape contains a 1, the machine moves its head one cell to the right and goes to q_{j+1}. If the observed cell is 0, then the value of the counter is j. In this case, $\mathcal{M}_n$ also moves its head one cell to the right and goes to p_s for $s = b_j \in \{0, 1, \ldots, k-1\}$. At q_{l-1}, if the machine observes 1, it invokes the termination procedure as $i = l$; otherwise, it moves its head one cell to the right and goes to p_s for $s = b_{l-1}$. All these transitions are hardwired in $\mathcal{M}_n$ and encode, when taken together, the number n. At each of the states p_s, $0 < s \le k - 1$, the machine writes a 1, moves its head one cell to the right, and goes to p_{s-1}; there is no need to write anything at the state p_0. As a result, if put in q_0, the machine will arrive at the state p_0 with the unchanged counter i and with 1^{b_i} written to the right of 1^i and separated from it by a 0. This is indeed the desired result; the number of states used in this gadget is $l + k + O(1)$.

In total, $\mathcal{M}_n$ has $|Q| \le l + 2k + O(1)$ states. Notice that $l = \lceil \log_k(n+1) \rceil = \log n / \log k + O(1)$ and take $k = \log n / (\log\log n)^2$, then $2k = o(\log n / \log\log n)$ as $n \to \infty$, and so $|Q| \le \log n / \log\log n \,(1 - o(1))$, which completes the proof. ◀

Corollary 2 below shows that the upper bound given by Theorem 1 is tight, i. e., that $\limsup Q_{\{0,1\}}(n) / (\log n / \log\log n) = 1$. However, the lower bound of $\log n_k / \log\log n_k \,(1 + o(1))$ only holds for certain sequences of naturals, and not for all $n \ge 0$.

For individual values of n, we relate the complexity of counting to Kolmogorov complexity; the following definitions are standard [14]. If some Turing machine $\mathcal{M}$, when given input y, halts and outputs x, then the string y is called a *description* of x with respect to $\mathcal{M}$. Let us also fix some *universal* Turing machine $\mathcal{U}$; given input (z, y), the TM $\mathcal{U}$ runs the machine described by z on input y. Here we fix, in advance, some descriptional system for Turing machines in which strings $z \in \{0,1\}^*$ encode machines. Define the *Kolmogorov complexity* of a string $x \in \{0,1\}^*$ as the smallest number k such that x has some description of size k, $y \in \{0,1\}^k$, with respect to $\mathcal{U}$.

It follows from results of Chaitin [4, sections 1.5–1.6] that for any alphabets $\Sigma \subsetneq \Delta$, $|\Delta| \ge 3$, and all strings $s \in \Sigma^*$ it holds that $Q_\Delta(s) \sim K(s)/(|\Delta| - 1)\log K(s)$ as $K(s) \to \infty$; Theorem 1 allows us to extend this result to a quite different case $|\Delta| = 2$ (where, unlike Chaitin, we have to use unary-encoded numbers). By $\mathrm{bin}(n)$ we denote the binary encoding of a non-negative integer n; that is, a word over the alphabet $\{0,1\}$.

▶ **Corollary 2.** *Under conditions of Theorem 1, for all $n \ge 0$ it holds that $Q_{\{0,1\}}(1^n) \sim K(1^n)/\log K(1^n) \sim K(\mathrm{bin}(n))/\log K(\mathrm{bin}(n))$ as $K(\mathrm{bin}(n)) \to \infty$.*

Another way of stating the last result is as follows: whenever there exists a Turing machine, with tape alphabet Δ, $|\Delta| \ge 2$, that produces a string s when run on empty tape and has an encoding of length k, there always exists another Turing machine with the same properties that has at most $k/(|\Delta| - 1)\log k \cdot (1 + o(1))$ states; in other words, encodings of Turing machines can always be fit into the smallest possible number of states.

▶ Remark. The lower bound of $K(\mathrm{bin}(n))/\log K(\mathrm{bin}(n))$ holds for all classes of machines considered in this paper (including nondeterministic and alternating ones), provided that the number of transitions leaving each control state is bounded by a constant.

4 Pushdown and counter automata

A *pushdown automaton* (PDA) over an input alphabet Σ is a tuple $\mathcal{A} = (\Sigma, Q, q_0, P, \perp, F, \delta)$, where Q is the set of (control) states, $q_0 \in Q$ the initial state, P the set of stack symbols, $\perp \in P$ the bottom-of-the-stack symbol, $F \subseteq Q$ the set of accepting states, and $\delta \subseteq (Q \times P \times (\Sigma \cup \{\varepsilon\})) \times (Q \times P^{\le 2})$ the transition relation.

Configurations of a PDA $\mathcal{A}$ are tuples of the form (q, s, u), where $q \in Q$ is the current state, $s \in (P \setminus \{\perp\})^* \perp$ the contents of the stack, and $u \in \Sigma^*$ the remaining input tape. For an input word $w \in \Sigma^*$, the initial configuration is $(q_0, \perp, w)$. For every transition $(q, p, \sigma, q', t) \in \delta$, the PDA can, consuming σ from the input, move from a configuration (q, s, u) to a configuration (q', s', u') such that $u = \sigma u'$, $s = pv$ and $s' = tv$ for some $v \in P^*$. If $p = \perp$, then $t \in (P \setminus \{\perp\})^{\le 1}\perp$, otherwise $t \in (P \setminus \{\perp\})^{\le 2}$.

The PDA *accepts* a word $w \in \Sigma^*$ if from $(q_0, \perp, w)$ it can reach a configuration (q, s, ε) with $q \in F$. The PDA is *deterministic* (DPDA) if from each configuration at most one configuration is reachable in one step; it suffices that, first, for each $q \in Q$ transitions $(q, p, \sigma, q', t) \in \delta$ either all have $\sigma = \varepsilon$ or $\sigma \ne \varepsilon$, and, second, that δ defines a partial mapping from $Q \times P \times (\Sigma \cup \{\varepsilon\})$ to $Q \times P^{\le 2}$.

The product $|Q| \cdot |P|$ shall be called the *complexity* of a deterministic PDA. We recall that machines from a certain class are said to *count to n* with complexity $f(n)$ if $f(n)$ is the smallest size of a machine from the class that recognizes the language $\{1^n\}$; for a related language $\{1^{tn} \colon t = 0, 1, 2, \dots\}$, we use the term *counting modulo n*.

▶ **Theorem 3.** *Deterministic pushdown automata count to n and modulo n with complexity* $\Theta(\log n)$.

This result can be obtained as an application of Theorem 1 in Chistikov and Majumdar [7]. The theorem states that the size of the smallest DPDA recognizing a language $R \subseteq \{1\}^*$ is within a constant factor of the smallest size of a pair of context-free grammars $\mathcal{P}, \mathcal{L}$ generating single words $\mathrm{eval}(\mathcal{P})$, $\mathrm{eval}(\mathcal{L})$ such that the ith element of the sequence $c = \mathrm{eval}(\mathcal{P}) \cdot (\mathrm{eval}(\mathcal{L}))^\omega$ is 1 if $1^i \in R$ and 0 if $1^i \notin R$. In our case $c = 0^n \cdot 1 \cdot 0^\omega$ for $R = \{1^n\}$ and $c = (10^{n-1})^\omega$ for $R = \{1^{tn} \colon t = 0, 1, 2, \dots\}$; in both cases, logarithmic upper and lower bounds on the size of $(\mathcal{P}, \mathcal{L})$ follow, because $\Theta(\log n)$ constant-size productions are necessary and sufficient for a CFG to generate a word of length n. The lower bound can also be deduced, under minor structural assumptions about DPDA, from an earlier result of Pighizzini [19, Theorem 12].

A *counter automaton* (CA) is a pushdown automaton with $|P| = 2$, that is, with just one stack symbol apart from $\perp$. It is convenient to think of a CA as of a finite automaton with a non-negative integer counter. Available operations on the counter are increment, decrement, and zero test; the CA cannot directly distinguish between different non-zero counter values.

▶ **Theorem 4.** *Deterministic counter automata count to n and modulo n with complexity* $\Theta(\sqrt{n})$.

Proof. We first prove the upper bound. Denote $r = \lfloor \sqrt{n} \rfloor$. Let a deterministic counter automaton $\mathcal{A}_n$ first consume $n - r^2 \le 2r$ letters from the input and then increase the counter value to r; these operations require at most $3r + 1$ states, the last of which we denote by q. At this state q, the automaton $\mathcal{A}_n$ performs a zero test on the counter: if its value is zero, $\mathcal{A}_n$ accepts and terminates; otherwise it decrements the counter, consumes r letters from the input, and returns to q. It is easy to see that $\mathcal{A}_n$ has $4r + O(1)$ states and recognizes $\{1^n\}$; if instead of termination it goes to the initial state, it will recognize $\{1^{tn} \colon t = 0, 1, 2, \dots\}$.

To prove the lower bound, consider any deterministic counter automaton $\mathcal{A}$. Suppose its set of states is Q, with $|Q| = m$. Construct an auxiliary device: a deterministic finite automaton $\mathcal{D}$ with states of the form (q, k), for $q \in Q$ and $0 \leq k \leq m + 1$. Direct the transitions of $\mathcal{D}$ in such a way that $\mathcal{D}$ simulates $\mathcal{A}$, keeping the value of the counter in the component of the control state denoted k; do not add transitions from states $(q, m + 1)$ that need to increase the counter.

If $\mathcal{D}$ recognizes the same language as $\mathcal{A}$, then the desired bound holds, because DFA—even with ε-transitions—need n states to count both to and modulo n. Now consider the only alternative: suppose that a computation of the CA $\mathcal{A}$ on some input word 1^s gets to a state q with counter value $m + 1$ and then increases the counter. For each $i = 1, \ldots, m + 1$ consider, in this computation, the last configuration before this point when the counter value is i. Denote these configurations by (q_i, i); they occur in the computation in the order with $i = 1$ first and $i = m + 1$ last. Note that between $(q_1, 1)$ and $(q_{m+1}, m + 1)$ the counter value is always positive, and recall that the automaton cannot distinguish between different positive values. By pigeonhole principle, there exist indices $j < k$ such that $q_j = q_k$, so $\mathcal{A}$ essentially gets into a loop: if the input tape provides it with infinitely many 1s, then it will be following the transitions of this loop forever. In $\mathcal{D}$, rerouting to $(q_k, j) = (q_j, j)$ the transitions with destination (q_k, k) will make it simulate $\mathcal{A}$ faithfully, i. e., recognize the same language.

Let us now consider each counting task separately. First suppose that $\mathcal{A}$ recognizes the language $\{1^n\}$, then either the loop we have found does not contain any accepting state, or it contains an accepting state but does not consume any letters from the input. (Indeed, if neither of these two conditions held, then $\mathcal{A}$ would accept infinitely many words.) In both cases it is possible to replace this loop with at most one state in the DFA. But then the obtained DFA, possibly with ε-transitions, will accept $\{1^n\}$, so it should contain at least $n + 1$ states. As a result, $m(m + 2) \geq n + 1$, and the desired bound follows.

Now suppose that $\mathcal{A}$ recognizes $\{1^{tn} : t = 0, 1, 2, \ldots\}$. Since the constructed DFA, like $\mathcal{A}$, is trapped forever in the loop, the states in the loop should enable it to accept input words 1^{tn} and reject all other input words. This means that the loop should consume at least n letters from the input; since it contains at most $m^2 + 1$ transitions, we conclude that $n \leq m^2 + 1$. This completes the proof. ◀

We next consider the *alternating* version of counter automata; we use the standard definition of alternation, which is a little different from the one that appeared originally in Chandra, Kozen, and Stockmeyer [5]. Informally, the machine is extended with the ability to make guesses and, dually, assertions during the computation.

More formally, suppose each state $q \in Q$ is labeled with either $\exists$ or $\forall$; consider the computational tree imparted on configurations by step-reachability. A node in this tree is *accepting* if, first, its state q belongs to F, or, second, its state is labeled by $\exists$ and has an accepting successor, or, third, its state is labeled by $\forall$ and all its successors are accepting (we use the least fixpoint here; i. e., only finite branches can count towards acceptance). An input word is *accepted* if the root in the computational tree is accepting.

Note that whenever the syntax of a state-based machine allows an unbounded number of transitions from a particular state (as sometimes needed by, e. g., nondeterministic and alternating machines), the number of control states is no longer a good complexity measure. For instance, alternating finite-state automata counting to n described by Kupferman et al. [12] have $O(\log n)$ states but $\Omega(\log^2 n)$ transitions. To avoid this issue, we use the number of transitions in δ as the complexity measure for alternating counter automata.

▶ **Theorem 5.** *Alternating counter automata count to n with complexity $O(\log n)$.*

Proof. Suppose $n = \sum_{i=0}^{l-1} b_i 2^i$, with $l = \lceil \log_2(n + 1) \rceil$ and all $b_i \in \{0, 1\}$, and create control states $q_0, q_1, \ldots, q_l$. The state q_0 will be the initial state of the automaton $\mathcal{A}$. Starting at a state q_i, $i < l$, with a zero counter value, $\mathcal{A}$ will, if $b_i = 1$, consume a letter from the input tape, and then, regardless of the value of b_i, get into a loop where it simultaneously reads from the input and increases the value of the counter. It can nondeterministically choose to exit the loop; the interpretation is that it guesses some value m and, with m iterations, reads 1^m from the input and increases the value of the counter to m. Upon exiting the loop, $\mathcal{A}$ branches universally: the first branch verifies that the value of the counter is equal to the number of remaining letters on the input tape, and the second branch goes to q_{i+1}. At the state q_l, $\mathcal{A}$ accepts. It is easy to see that $\mathcal{A}$ indeed recognizes the language $\{1^n\}$; $O(1)$-bounded branching ensures that $\mathcal{A}$ has $O(\log n)$ transitions. ◀

5 Visibly pushdown automata

A *visibly pushdown automaton* [1], or a VPA, $\mathcal{A}$ is a pushdown automaton that has the following property: there exists a partition of the input alphabet Σ into three parts, $\Sigma = \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}} \cup \Sigma_{\mathsf{int}}$, such that $\mathcal{A}$ always pushes a symbol upon reading a letter from Σ_{call}, pops a symbol upon reading a letter from Σ_{ret}, and does not use the stack on letters from Σ_{int}. It is implied that, in a VPA, destinations of transitions driven by letters from $\Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{int}}$ cannot depend on the top symbol of the stack. Furthermore, a VPA is not allowed to have ε-transitions—i. e., all transitions are required to consume at least one letter from the input.

It is easy to see that in the problem of counting, defined as recognizing languages $\{1^n\}$ and $\{1^{tn} : t = 0, 1, 2, \ldots\}$, visibly pushdown automata cannot do better than standard finite automata. So it is natural to consider different, although closely related languages $\{a^n b^n\}$ and $\{a^{tn} b^{tn} : t = 0, 1, 2, \ldots\}$, where $n = 0, 1, 2, \ldots$, with the restriction that $a \in \Sigma_{\mathsf{call}}$ and $b \in \Sigma_{\mathsf{ret}}$. Basically, we convert "linear" input words over Σ into nested words, as in [1], which enables our automata to use the stack. Note that in the framework of nested words, our languages $\{a^n b^n\}$ and $\{a^{tn} b^{tn} : t = 0, 1, 2, \ldots\}$ translate to languages over a one-letter alphabet, that is, to a unary language of nested words.

Salomaa [22] proved that every deterministic visibly pushdown automaton over the alphabet $\Sigma = \Sigma_{\mathsf{call}} \cup \Sigma_{\mathsf{ret}}$, with $\Sigma_{\mathsf{call}} = \{a_1, a_2\}$ and $\Sigma_{\mathsf{ret}} = \{b_1, b_2\}$, recognizing a related language

$$L_n = \{a_{i_1} \ldots a_{i_k} b_{i_k} \ldots b_{i_1} : i_1, \ldots, i_k \in \{1, 2\},\ k = tn,\ t = 0, 1, 2, \ldots\}, \tag{1}$$

should have at least $\sqrt{n}$ states. The proof does not essentially use different kinds of push and pop symbols, so the lower bound extends to $\{a^{tn} b^{tn} : t = 0, 1, 2, \ldots\}$. Moreover, the crucial point is just that the word $a^n b^n$ is accepted, and all words $a^m b^m$, $0 < m < n$, rejected; as a result, the lower bound of $\sqrt{n}$ holds for the language $\{a^n b^n\}$ as well.

Salomaa also constructed a VPA with $n + 1$ states and 3 stack symbols for L_n, and conjectured that this VPA is optimal in terms of $|Q| + |P|$. We refute this conjecture by constructing a VPA with $O(\sqrt{n})$ states and $O(1)$ stack symbols that recognizes L_n.

▶ **Theorem 6.** *For every $n \geq 1$, there exists a deterministic visibly pushdown automaton $\mathcal{A}$ that recognizes the language $\{a^n b^n\}$ and satisfies $|Q| \leq O(\sqrt{n})$ and $|P| \leq O(1)$. In contrast, every (possibly nondeterministic) visibly pushdown automaton that recognizes this language satisfies the inequality $|Q| \geq \sqrt{n}$.*

▶ **Theorem 7.** *If $n = n_1 n_2 \geq 1$, then there exists a deterministic visibly pushdown automaton $\mathcal{A}$ that recognizes the language $\{a^{tn} b^{tn} : t = 0, 1, 2, \ldots\}$ and satisfies $|Q| \leq O(n_1 + n_2)$ and $|P| \leq O(1)$. In contrast, every deterministic visibly pushdown automaton that recognizes this language satisfies the inequality $|Q| \geq \Omega(\min(n_1 + n_2))$, where the minimum is over all factorizations $n = n_1 n_2$ with natural $n_1, n_2 \geq 1$.*

Theorem 7 shows that the smallest VPA counting modulo n have $\Theta(f(n))$ control states, with $f(n) = \min(n_1 + n_2)$ over all factorizations $n = n_1 n_2$; here $\sqrt{n} \le f(n) \le n + 1$ for all n, the left-hand inequality is tight for perfect squares, $n = k^2$, and the right-hand inequality for prime numbers, $n = p$.

▶ **Proposition 8.** *The results of Theorem 7 also hold for the language L_n, as defined by* (1)*. For $n = k^2$, $k = 1, 2, \dots$, the automaton witnessing the upper bound satisfies $|Q| + |P| = O(k) = O(\sqrt{n})$, which disproves the conjecture of Salomaa* [22]*. For $n = p$, p prime, the lower bound shows that the construction of VPA for L_n given in* [22] *is optimal up to a constant factor.*

The rest of the section is devoted to the proofs of Theorems 6 and 7 and Proposition 8.

5.1 Main ingredients for upper bounds

The high-level idea for proving the upper bounds is simple and common for all three languages: let the automaton first count blocks of a, each of size n_1, and mark their starting positions by pushing a special symbol on the stack. Then, as the automaton reads bs and pops from the stack, it counts the number of blocks. Here it can count either to n_2 or modulo n_2, and the bound on the total number of states is $O(n_1 + n_2)$. So we will need an auxiliary construction, that of a VPA recognizing the language

$$L_{\mathsf{cycles}} = \{a^s b^{t+d} \colon n_1 \mid s,\ n_1 n_2 \mid t,\ s \ge t + d,\ 0 \le d < n_1\}. \tag{2}$$

Let us define such a VPA, denoted $\mathcal{V}(n_1, n_2)$, as follows.

First take n_1 control states and connect them in a single cycle such that reading an a from the input moves the device one step along the cycle. Make one of these states, q_1, the initial state and let the device push 1 when it reads a at q_1, and push 0 when it reads a at any other state of the cycle. Now take new n_2 control states and connect them into a different cycle. The automaton will only use these states when reading bs from the input; more specifically, demand that popping 0 from the stack not change the control state, and popping 1 moves the device one state further along the cycle.

Now connect these two disjoint cycles in the following manner. Let the device, upon reading b and popping 0 at q_1, move to some specific state of the second cycle, q_2. We shall call this state q_2 the entry point of the second cycle. Make q_1 and q_2 the only final states, and turn all missing transitions into some (new) sink state.

▶ **Lemma 9.** *The VPA $\mathcal{V}(n_1, n_2)$ has $n_1 + n_2 + O(1)$ control states and 3 stack symbols and recognizes L_{cycles}.*

5.2 Proof of Theorem 6

As mentioned above, the lower bound is essentially shown in Salomaa [22, Theorem 4.1]; and it is instructive to see that it holds even for nondeterministic VPA: it suffices to carry out the reasoning for an accepting branch of the computation. So, in what follows we only need to focus on the upper bound.

Given n, choose $n_1 = n_2 = \lfloor \sqrt{n-1} \rfloor$ and $r = n - n_1^2 > 0$, construct the VPA $\mathcal{V}(n_1, n_2)$ described above, and modify it in the following way. First, add a new stack symbol 2, make a simple path of r transitions reading a and pushing 2, and attach it to the VPA so that the last of these transitions leads to q_1, the “entry point” of the first cycle in $\mathcal{V}(n_1, n_2)$. Mark the source of the attached path, q_0, as the initial state of the VPA. Next, consider the second cycle of the VPA and transform it into a path by changing the destination of the transition that pops 1 and leads from a state q to q_2, the "entry point" of the second cycle. Make this transition lead from q to a new state q' instead, and then add to q' a path of r transitions that read b from the input and pop 2 from the stack. Make the last state on this path the only final state. (As previously, all missing transitions are sent to the sink state.) Now it is easy to see that the obtained VPA accepts exactly the words $a^s b^{s'}$ with $s = r + kn_1$, $s' = n_1^2 + r$, and $s = s'$—or, to put it differently, recognizes the language $\{a^n b^n\}$. Since it has $n_1 + n_2 + 2r + O(1) \leq 4\sqrt{n} + O(1)$ control states and 4 stack symbols, this completes the proof of Theorem 6.

5.3 Main ingredients for lower bounds

We now turn our attention to the lower bound of Theorem 7, for the language $\{a^{tn} b^{tn} : t = 0, 1, 2, \ldots\}$. Our goal is to show that every deterministic VPA recognizing this language has at least $\Omega(\min(n_1 + n_2))$ states, where min is taken over all factorizations $n = n_1 n_2$ with integers $n_1, n_2 \geq 1$.

The key insight is the decomposition of a VPA for this language into two finite-state transducers. The first transducer transforms the first part of the input into a sequence of stack symbols, and the second transducer transforms the reversal of this sequence into a sequence of accepting and non-accepting states. Periodic behaviour of these transducers enables us to obtain the desired lower bound.

To use this idea, we need to go through some auxiliary constructions first. Take a (deterministic) VPA $\mathcal{A}$ and denote by Q the set of its control states. First consider the behaviour of $\mathcal{A}$ on words of the form a^s, $s = 0, 1, 2, \ldots,$—one can regard them all as prefixes of the infinite sequence a^ω. Restricted to these words, the VPA $\mathcal{A}$ behaves as a deterministic letter-to-letter finite-state transducer, with symbols pushed on the stack interpreted as output letters. Since the input to this *push-transducer* is a prefix of the infinite sequence a^ω, it is easy to see that the sequence of the states $\mathcal{A}$ gets into is eventually periodic with some period $k \leq |Q|$, and the word pushed on the stack is of the form $uw^m w'$, where $|w| = k$ and $w = w'w''$ for some w''. That is, the push-transducer transforms a^ω into uw^ω; words u, w, and w' are uniquely determined by q, the state in which $\mathcal{A}$ arrives upon reading a^s. Denote by Q' the set of states that $\mathcal{A}$ visits infinitely often; $|Q'| = k$.

Now let $\mathcal{A}$ stop in some state $q \in Q'$ after an input a^s. Suppose this state q is fixed. If, from this point on, the input tape supplies $\mathcal{A}$ with symbols b only, then $\mathcal{A}$ will operate as another finite-state transducer, the *pop-transducer*: the symbols popped from the stack are interpreted as input letters of the transducer, and the states from Q the device visits are the output letters. Observe that the input to this transducer is $(\bar{w})^m \bar{u}$, for $\bar{w} = (w')^R (w'')^R$ and $\bar{u} = (w')^R u^R$, where by v^R we denote the reversal of a word v. This input is a prefix of the infinite periodic sequence $\alpha_q = (\bar{w})^\omega$ augmented with the finite word $\bar{u}$. Note that the words $\bar{w}$ and $\bar{u}$ are uniquely determined by q.

▶ **Lemma 10.** *For each $q \in Q'$, the pop-transducer transforms the sequence α_q into an eventually periodic sequence with a period r_q such that $r_q \mid t_q k$ for some $t_q \leq |Q|$.*

▶ **Lemma 11.** *For each i, $0 \leq i < k$, the behaviour of $\mathcal{A}$ on words $a^s b^s$ with $s \bmod k = i$ is eventually periodic with period r_q for some fixed $q \in Q'$: more precisely, for each i, $0 \leq i < k$, there exists some $q \in Q'$ and an integer $s^{(i)}$ such that the VPA $\mathcal{A}$ either accepts or rejects both words $a^{s_1} b^{s_1}$ and $a^{s_2} b^{s_2}$, provided that $s_1, s_2 \geq s^{(i)}$, the period r_q divides $|s_1 - s_2|$, and k divides $|s_1 - s_2|$.*

▶ **Lemma 12.** *If $\mathcal{A}$ recognizes $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\}$, then $n \mid g_1 g_2$ for some integers $g_1, g_2 \le |Q|$.*

As the proof shows, the conclusion of Lemma 12 holds for any VPA $\mathcal{A}$ that recognizes a language L with $L \cap \{a^s b^s : s \ge s_0\} = \{a^{tn}b^{tn} : t = t_0, t_0 + 1, \ldots\}$ for some fixed $s_0, t_0 \in \mathbb{N}$.

5.4 Proofs of Theorem 7 and Proposition 8

Proof of Theorem 7. The lower bound can be obtained as a corollary of Lemma 12, by the following argument. Since $n | g_1 g_2$, there exists some f such that $nf = g_1 g_2$. Therefore, we can assume that $n = n_1 n_2$ for some n_1, n_2 with $n_1 \,|\, g_1$ and $n_2 \,|\, g_2$. Furthermore, as $g_1, g_2 \le |Q|$, it also holds that $n_1, n_2 \le |Q|$, that is, $|Q| \ge \max\{n_1, n_2\}$. But $\max\{n_1, n_2\} \ge (n_1 + n_2)/2$, so it follows that $|Q| \ge (n_1 + n_2)/2$, and so $|Q| \ge \min(n_1' + n_2')/2$, with min over all factorizations $n = n_1' n_2'$. This concludes the proof of the lower bound.

As mentioned above, we use the same idea as in Theorem 6 to prove the upper bound. Given a factorization $n = n_1 n_2$, we show how to construct a VPA that accepts this language and satisfies the stated bounds. First note that $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\} = L_{\mathsf{balanced}} \cap L_{\mathsf{cycles}}$, where $L_{\mathsf{balanced}} = \{a^s b^s : s \ge 0\}$ and L_{cycles} is given by the definition in (2) on p. 347. It is straightforward to construct a VPA for L_{balanced} with 5 control states and 2 stack symbols, and a VPA for the intersection of languages is easily obtained by taking the products of control states and stack alphabets. Therefore, it suffices to show that L_{cycles} can be recognized with a VPA with $|Q| = n_1 + n_2 + O(1)$ and $|P| \le 3$, but we already know how to do this with Lemma 9. This completes the proof of Theorem 7. ◀

Proof of Proposition 8. The lower bound holds, because the intersection $L_n \cap \{a_1, b_1\}^*$ is essentially $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\}$, just with a_1, b_1 in place of a, b. As a result, the smallest VPA recognizing L_n can be at most a constant factor smaller than the smallest VPA recognizing $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\}$. As for the upper bound, it suffices to replace, in the VPA $\mathcal{V}(n_1, n_2)$ for L_{cycles}, transitions reading a and b with pairs of transitions reading a_i and b_j, and replace the VPA for L_{balanced} with a VPA for $\{a_{i_1} \ldots a_{i_k} b_{i_k} \ldots b_{i_1} : i_1, \ldots, i_k \in \{1, 2\}\}$. ◀

6 Open questions

Unanswered questions of a related nature abound. Probably the most natural is the following one: for which classes of machines are other modes of operation (nondeterministic, universal, and alternating ones) of help for the task of recognizing the language $\{1^n\}$? Kupferman et al. [12] provide a comprehensive answer for the classes of finite-state machines. For other classes, a short list of some specific problems follows:

1. Does nondeterminism or alternation make it possible for pushdown automata to count to n with $o(\log n)$ states for all n? The arguments that lead to the lower bound of $\Omega(\log n)$ in Theorem 3 only work for deterministic PDA and seem tricky or impossible to generalize. For other modes of operation, the only known lower bound is at most $\log n / \log\log n\,(1 + o(1))$, given by the Kolmogorov argument of Corollary 2 and Remark 3.
2. Can nondeterministic or universal counter automata count to n with $o(\sqrt{n})$ states? As seen from the upper bound of Theorem 5, counter automata with unbounded alternation can, but is it achievable with bounded alternation depth?
3. What is the state complexity of recognizing the language $\{a^{tn}b^{tn} : t = 0, 1, 2, \ldots\}$ with nondeterministic visibly pushdown automata?

Needless to say, many other related problem settings exist.

References

1 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–43, 2009. Revised version available at `http://robotics.upenn.edu/~alur/Jacm09.pdf`.

2 Jean-Camille Birget. Two-way automata and length-preserving homomorphisms. *Mathematical Systems Theory*, 29(3):191–226, 1996.

3 Can one-way alternating automata with one-counter recognize some unary non-regular languages? `http://cstheory.stackexchange.com/q/19046/13649`, 2013–2014.

4 Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *J. ACM*, 13(4):547–569, 1966.

5 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

6 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and abhi shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

7 Dmitry Chistikov and Rupak Majumdar. Unary pushdown automata and straight-line programs. In *ICALP'14, Part II*, volume 8573 of *LNCS*, pages 146–157, 2014.

8 Marek Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986.

9 Yuval Filmus. Lower bounds for context-free grammars. *Inf. Process. Lett.*, 111(18):895–898, 2011.

10 Matthew M. Geller, Harry B. Hunt III, Thomas G. Szymanski, and Jeffrey D. Ullman. Economy of description by parsers, DPDA's, and PDA's. *Theor. Comput. Sci.*, 4(2):143–153, 1977.

11 Seymour Ginsburg and H. Gordon Rice. Two families of languages related to ALGOL. *J. ACM*, 9(3):350–371, 1962.

12 Orna Kupferman, Amnon Ta-Shma, and Moshe Y. Vardi. Concurrency counts. Technical report, available at `http://www.cs.tau.ac.il/~amnon/Papers/KTV.submitted.cjtcs.ps`, 2001.

13 Ernst L. Leiss. Succinct representation of regular languages by boolean automata. *Theor. Comput. Sci.*, 13(3):323–330, 1981.

14 Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications.* Texts and monographs in computer science. Springer, 1993.

15 Markus Lohrey. Algorithmics on SLP-compressed strings: a survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

16 Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. In *SWAT (FOCS) 1971*, pages 188–191, 1971.

17 Alexander Okhotin, Xiaoxue Piao, and Kai Salomaa. Descriptional complexity of input-driven pushdown automata. In *Dassow Festschrift 2012*, volume 7300 of *LNCS*, pages 186–206, 2012.

18 Alexander Okhotin and Kai Salomaa. Complexity of input-driven pushdown automata. *SIGACT News*, 45(2):47–67, 2014.

19 Giovanni Pighizzini. Deterministic pushdown automata and unary languages. *Int. J. Found. Comput. Sci.*, 20(4):629–645, 2009.

20 Giovanni Pighizzini, Jeffrey Shallit, and Ming-wei Wang. Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. *J. Comput. Syst. Sci.*, 65(2):393–414, 2002.

21 Wojciech Rytter. Grammar compression, LZ-encodings, and string algorithms with implicit input. In *ICALP'04*, volume 3142 of *LNCS*, pages 15–27, 2004.

22 Kai Salomaa. Limitations of lower bound methods for deterministic nested word automata. *Inf. Comput.*, 209(3):580–589, 2011.