Concurrency Counts

Orna Kupferman
Hebrew University*

Amnon Ta-Shma
Tel Aviv University†

Moshe Y. Vardi
Rice University‡

November 21, 2001

Abstract

One of the natural examples to the power of concurrency is the ability to count to n with $\log n$ concurrent processes. In this work we study the cost of counting within weaker types of concurrency, where no cooperation takes place between the concurrent processes, except when time comes to decide whether the input should be accepted. We first study the number of states required for deterministic, nondeterministic, universal, and alternating automata to count to n, namely to accept the unary language $\{1^n\}$. We then study settings in which the task of counting should be accomplished before the whole input is read (e.g., in the language $\Sigma^n \cdot 0 \cdot \Sigma^*$). We show that then, the weaker types of concurrency are not helpful. Our results explain the strength of synchronized concurrent systems, where synchronization between the underlying components is allowed, and imply that semi-extended regular expressions, which can be efficiently translated to partially input-synchronized alternating automata, cannot be efficiently translated to alternating automata.

1 Introduction

Classical models of computations, such as Turing machines and automata, have been enriched with existential and universal branching to capture concurrency. Unlike the models used in the study of real concurrent systems, in these types of branching no cooperation takes place between the spawned processes, except when time comes to decide whether the input should be accepted. It turned out that this weak cooperation is sufficient to make nondeterministic and universal automata exponentially more succinct than deterministic automata, and to make their combination, namely alternating automata, doubly exponentially more succinct than deterministic automata [CKS81]. Nevertheless, as studied in [DH94], enriching automata with real concurrency, where the spawned processes can cooperate all along the computation, results in even more succinct automata. In particular, concurrent alternating automata are triply exponentially more succinct than deterministic automata.

*Address: School of Engineering and Computer Science, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il
†Address: Department of Computer Science, Tel Aviv 69978, Israel. Email: amnon@post.tau.ac.il
‡Address: Department of Computer Science, Houston, TX 77251-1892, U.S.A. Email: vardi@cs.rice.edu.

One of the natural examples to the power of concurrency is the ability to count to n with $\log n$ concurrent processes, with the i'th processor being responsible for the value of the i'th bit in the boolean representation of n. In this work we study the cost of counting within the other types of concurrency.

We first study the number of states required for an automaton to count to n, namely to accept the unary language $\{1^n\}$. It is well known and it is easy to see that nondeterministic automata need $O(n)$ states to count to n. For other types of automata, the problem was stated in [**?**], and was solved in [Bir93, Bir96] for alternating and two-way automata. Here we complete the picture with results about universal automata. Our result is based on known succinctness relations between the various types of automata with respect to unary languages [**?**, Cho86, **?**]. Together, these results show that nondeterminism, universality, and alternation induce three different counting costs: like deterministic automata, nondeterministic automata need $\Theta(n)$ states to count to n. Then, universal automata need only $\Theta(\sqrt{n})$ states, and finally, alternating automata can count to n with $\Theta(\log n)$ states.

In many applications, the only task of concurrency is efficient counting. The above results show that weaker types of concurrency are sufficient for efficient counting, thus they question the necessity of full concurrency in such applications[1]. This led us to the study of "purposeful counting", where the task of counting should be accomplished before the whole input is read. Formally, we say that a given type of automata *purposefully counts* to n with $f(n)$ states iff for every language L that is accepted by an automaton of this type with k states, the language $(0+1)^n \cdot 0 \cdot L$ is accepted by an automaton of this type with $k + f(n)$ states. We show that all types of bounded concurrency cannot purposefully count efficiently, thus even alternating automata purposefully count to n with $\Theta(n)$ (rather than $\Theta(\log n)$) states. In other words, the disability of the spawned processes to cooperate all along the computation is a real weakness, even when the only task of concurrency is efficient counting.

Our results on purposeful counting give a better understanding of the power of synchronized concurrent machines [**?**, DHK$^+$89] and lead to a solution of an open problem raised in [**?**]. Yamamoto considers *semi-extended regular expressions*, in which conjunction is allowed, and studies the problem of deciding whether a given word belongs to the language of a given semi-extended regular expression. It seems that the universal branches of alternating automata can be used in order to handle the conjunctions allowed in semi-extended regular expressions, making semi-extended regular expressions efficiently translatable to alternating automata. Nevertheless, as Yamamoto writes, "it is difficult to use alternating finite automata for our aim." The difficulty lies in expressions like $(r_1 \wedge r_2) \cdot r_3$, where the two processes that check membership in r_1 and r_2 should be synchronized in order to guess the position in the word in which a suffix in r_3 starts. Indeed, $(r_1 \wedge r_2) \cdot r_3$ is not equal to $(r_1 \cdot r_3) \wedge (r_2 \cdot r_3)$. In contrast, $(r_1 \vee r_2) \cdot r_3$ is equal to $(r_1 \cdot r_3) \vee (r_2 \cdot r_3)$, and two processes that check membership in r_1 and r_2 do not have to be synchronized when guessing the suffix. This is why regular expressions (with no $\wedge$) can be efficiently translated to alternating automata, yet no translation is

[1]In particular, our interest in the cost of counting arose when we tried to improve the construction in [**?**], where we had to blow-up the state space of an alternating automaton by n just for the sake of counting. We were hoping that the ability of alternating automata to count to n with $O(\log n)$ states would lead to a smaller blow-up.

known for semi-extended regular expressions. Consequently, Yamamoto introduces *partially input-synchronized alternating automata*, where synchronization between processes, as required above, can take place. Yamamoto efficiently translates semi-extended regular expressions to such automata, and leaves open the problem of whether synchronization is necessary, namely whether semi-extended regular expressions can be efficiently translated to alternating automata. We show that semi-extended regular expressions can purposefully count modulo n, implying that no efficient translation exists. It also follows that in some cases input synchronization can compensate on the lack of cooperation between the spawned processes along the computation.

2 Automata on finite words

Given an alphabet Σ, a *word* over Σ is a finite sequence $w = \sigma_1 \cdots \sigma_n$ of letters in Σ. We denote by w^l the suffix $\sigma_l \cdot \sigma_{l+1} \cdot \sigma_{l+2} \cdots$ of w. A *nondeterministic automaton on finite words* is $\mathcal{A} = \langle \Sigma, Q, q_{in}, M, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, $M : Q \times \Sigma \to 2^Q$ is a transition function, $q_{in} \in Q$ is an initial state, and $F \subseteq Q$ is an acceptance condition. Intuitively, $M(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Since $\mathcal{A}$ may have several initial states and its transition function may specify many possible transitions for each state and letter, $\mathcal{A}$ is not *deterministic*. A *run* of $\mathcal{A}$ on $w = \sigma_1 \cdots \sigma_n$ is a function $r : \{0, \ldots n\} \to Q$ where $r(0) = q_{in}$ (i.e., the run starts in the initial state) and for every $l \geq 1$, we have $r(l) \in M(r(l-1), \sigma_l)$ (i.e., the run obeys the transition function). The run is *accepting* iff $r(n) \in F$ (i.e., the run terminates in one of the accepting states). The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of words $w \in \Sigma^*$ for which there exists an accepting run of $\mathcal{A}$ on w. The automaton $\mathcal{A}$ is *unary* if $|\Sigma| = 1$.

Since $\mathcal{A}$ is not deterministic, it may have many runs on w. There are two dual ways in which we can refer to the many runs. When $\mathcal{A}$ is a nondeterministic automaton as above, it accepts an input word w iff there exists an accepting run of $\mathcal{A}$ on w. Dually, when $\mathcal{A}$ is a *universal* automaton, it accepts an input word w iff all the runs of $\mathcal{A}$ on w are accepting. *Alternating* automata [BL80, CKS81] combine both existential and universal branching choices.

An alternating automaton is $\mathcal{A} = \langle \Sigma, Q, q_{in}, M, F, \rho \rangle$, where Σ, Q, q_{in}, M, and F are as in nondeterministic automata, and $\rho : Q \to \{\exists, \forall\}$ is a *classification function* that classifies each state q as either existential (in case $\rho(q) = \exists$), or universal (in case $\rho(q) = \forall$). The intuition is that the automaton accepts a suffix w^l of w from an existential state q if it accepts w^{l+1} from some state in $M(q, \sigma_l)$, and it accepts w^l from a universal state q if it accepts w^{l+1} from all states in $M(q, \sigma_l)$.

Formally, a run of the alternating automaton $\mathcal{A}$ on w is a function $r : \{0, \ldots n\} \to 2^Q$ where $r(0) = \{q_{in}\}$ and for every $l \geq 1$ and $q \in r(l)$, either $\rho(q) = \exists$, in which case $M(q, \sigma_l) \cap r(l+1) \neq \emptyset$, or $\rho(q) = \forall$, in which case $M(q, \sigma_l) \subseteq r(l+1)$. Thus, the run chooses a successor of existential states and proceeds with all successors of universal states. The run is accepting iff $r(n) \subseteq F$ (i.e., all the "processes" that the run involve terminates in accepting states).

Deterministic, nondeterministic, and universal automata can be viewed as special cases of alternating automata. Formally, an alternating automaton is deterministic if for all q and σ, the set $M(q, \sigma)$ is either empty or a singleton. An automaton is nondeterministic if all states are existential, and it is universal if all states are universal. The classification of states to existential and universal makes the *complementation* problem for alternating automata as easy as the problem of complementing deterministic automaton. For a classification function ρ, let $\tilde{\rho}$ be dual to ρ, thus, for all $q \in Q$, we have that $\tilde{\rho}(q) = \forall$ iff $\rho(q) = \exists$ and $\tilde{\rho}(q) = \forall$ iff $\rho(q) = \exists$. Given $\mathcal{A} = \langle \Sigma, Q, q_{in}, M, F, \rho \rangle$, it is easy to see that the automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, q_{in}, M, Q \setminus F, \tilde{\rho} \rangle$ accept exactly all words rejected by $\mathcal{A}$.

3 Counting with Automata

For $n \in \mathbb{N}$, we define the language $C_n = \{1^n\}$ over the alphabet $\{1\}$. We say that an automaton $\mathcal{A}$ *counts to* n iff $\mathcal{A}$ recognizes C_n. We say that a type of automata *counts to* n *with* $f(n)$ *states* if there is an automaton of this type that accepts C_n with $f(n)$ states. Since we deal with unary languages, then if $\mathcal{A}$ accepts 1^k for some $k \geq 0$, we say that $\mathcal{A}$ accepts k and we write $k \in \mathcal{A}$.

3.1 Deterministic and nondeterministic automata

Theorem 3.1 below is easy and well known, we give it here for the sake of completeness.

Theorem 3.1 *For every $n \geq 0$, deterministic and nondeterministic automata count to n with $n + 1$ states.*

Proof: For the upper bound, consider the deterministic automaton $\mathcal{A} = \langle \{1\}, \{0, \ldots n\}, 0, M, \{n\} \rangle$ where $M(i, 1) = \{i + 1\}$ for $i \leq n$ and $M(n, 1) = \emptyset$. It is easy to see that $\mathcal{A}$ recognizes C_n. For the lower bound, assume by way of contradiction that there exists a nondeterministic automaton $\mathcal{A}_n$ with less than $n + 1$ states that recognizes C_n. Since an accepting run of $\mathcal{A}_n$ on 1^n must visit at least one state more than once, the transition function of $\mathcal{A}_n$ contains a cycle that enables the acceptance of more words, not in C_n, and we reach a contradiction. □

3.2 Alternating automata

It is proved in [?] that every unary deterministic automaton with n states has an equivalent alternating automaton with $\lceil \log n \rceil$ states. Hence, Theorem 3.1 immediately implies the following theorem.

Theorem 3.2 *For every $n \geq 0$, alternating automata count to n with $\lceil \log n \rceil$ states.*

We say that an alternating automaton is a *one-switch* alternating automaton if it cannot move from existential states to universal states. Thus, a one-switch alternating automaton can change its “branching mode” at most once, from a universal mode to an existential mode. The following theorem is proven in [Bir96] and we give it here with its proof, which we would use later.

Theorem 3.3 [Bir96] *For every $n \geq 0$, one-switch alternating automata count to n with $O(\log^2 n \log\log n)$ states.*

Proof: Given n, let $p_1, \ldots, p_k$ be the first k prime numbers such that $n \leq p_1 \cdot p_2 \cdots p_k$. By the Chinese Remainder Theorem, there is a tuple $\langle a_1, \ldots, a_k \rangle$ such that for all $1 \leq i \leq k$, we have $n = a_i \bmod p_i$, and n is the unique number smaller than $p_1 \cdot p_2 \cdots p_k$ with the above property. That is, for all $n \neq m \leq p_1 \cdot p_2 \cdots p_k$, there exists $1 \leq i \leq k$ such that $m \neq a_i \bmod p_i$. An alternating automaton $\mathcal{A}$ that recognizes C_n has, for each $1 \leq i \leq k$, two cycles each of size p_i. The first, i-good, cycle, accepts all words of length $l = a_i \bmod p_i$. The second, i-bad, cycle, accepts all words of length $l \neq a_i \bmod p_i$. Given an input word w, the alternating automaton checks that for all $1 \leq i \leq k$, we have $|w| = a_i \bmod p_i$, and that no proper suffix of w has this property. For that, $\mathcal{A}$ uses its universality to make sure that w is accepted by all the i-good cycles, and to make sure that all the proper suffixes of w are accepted by some, nondeterministically chosen, i-bad cycle. Since $p_i = O(i \log i)$ [?] and $k \leq \log n$, we have that $p_i \leq \log n \log\log n$, and we are done. $\square$

The automaton $\mathcal{A}$ described in the proof is "almost universal". Indeed, using the Chinese Reminder Theorem, we can easily define a universal automaton with *polylog*(n) states that accepts exactly all numbers k for which $k = a_i \bmod p_i$ for all $1 \leq i \leq k$. Then, nondeterminism is used only to "mask" all k's except the smallest one that satisfy the above. Can we do define such a mask without nondeterminism? In the next section we show that we can, yet while such a nondeterministic mask requires only *polylog*(n) states, a universal one requires $\Theta(\sqrt{n})$ states.

3.3 Universal automata

The *greatest common divisor* of $a_1, \ldots, a_k \in \mathbb{N}$, denoted $gcd(a_1, \ldots, a_k)$, is the largest integer d dividing all a_i's. Two integers x and y are *relatively prime* if $gcd(x, y) = 1$. For two integers x and y, with $x \leq y$, we define the nondeterministic automaton $\mathcal{B}_{x,y} = \langle \{1\}, \{0, \ldots, y-1\}, 0, M, \{0\} \rangle$, where $M(0) = \{1, y-x+1\}$, and for all $i > 0$, we have $M(i) = i+1 \bmod y$. Thus, $\mathcal{B}_{x,y}$ has two loops, one of length x and the other of length y, both connected at the state 0. Therefore, $\mathcal{B}_{x,y}$ accepts exactly these numbers of the form $ax + by$ for some $a, b \in \mathbb{N}$. It is known (see [Cho86]) that if x and y are relatively prime numbers, then the greatest c such that $ax + by = c$ has no solution in natural numbers is $(x-1)(y-1)-1$. This immediately implies the following property of $\mathcal{B}_{x,y}$.

Lemma 3.4 *Let x and y be two relatively prime numbers, and let $n' = (x-1)(y-1)-1$. The automaton $\mathcal{B}_{x,y}$ rejects n' and accepts all integers $k > n'$.*

For example, if $x = 10$ and $y = 11$, then $n' = 89$. The nondeterministic automaton $\mathcal{B}_{10,11}$ has 11 states. It rejects 89 and accepts all integers $i > 90$. Lemma 3.4 provides the mask we were looking for. We can now use $\mathcal{B}_{x,y}$ and Lemma 3.4 to count with universal automata.

Theorem 3.5 *For every $n \geq 0$, universal automata count to n with $\Theta(\sqrt{n})$ states.*

Proof: We start with the upper bound. We describe a dual construction, namely a nondeterministic automaton $\mathcal{A}$ such that $\mathcal{A}$ accepts exactly all integers $k \neq n$. Then, the automaton $\tilde{\mathcal{A}}$ counts to n. Given n, we define $\mathcal{A}$ as the union of two nondeterministic automata, $\mathcal{A}_n$ and $\mathcal{B}_n$. The automaton $\mathcal{A}_n$ rejects n, accepts all $k < n$, and we do not care about its behavior with respect to integers $k > n$. The automaton $\mathcal{B}_n$ rejects n, accepts all $k > n$, and we do not care about its behavior with respect to integers $k < n$. Since $\mathcal{A}$ accepts iff either $\mathcal{A}_n$ or $\mathcal{B}_n$ accept, $\mathcal{A}$ accepts all integers $k \neq n$. The automaton $\mathcal{A}_n$ has $O(\log^2 n \cdot \log \log n)$ states and the automaton $\mathcal{B}_n$ has $O(\sqrt{n})$ states. Then, $\tilde{\mathcal{A}}$ is a universal automaton that counts to n with $O(\sqrt{n})$ states.

The automaton $\mathcal{A}_n$ is similar to the automaton described in the proof of Theorem 3.2. Let $p_1, \ldots, p_k$ be the first k prime numbers such that $p_1 \cdot p_2 \cdots p_k \geq n$. For each $1 \leq j \leq k$, the automaton $\mathcal{A}_n$ contains a cycle that accepts all integers of length $l \neq a_j \bmod p_j$. Thus, $\mathcal{A}_n$ accepts $i \geq 0$ iff there is $1 \leq j \leq k$ such that $i \neq a_j \bmod p_j$. By the Chinese reminder theorem, $\mathcal{A}_n$ rejects n, accepts all $i < n$, and its behavior with respect to integers $i > n$ is arbitrary. Since $p_i = O(i \log i)$ and $k \leq \log n$, we have that $p_i \leq \log n \log \log n$, and the automaton $\mathcal{A}_n$ has $O(\log^2 n \log \log n)$ states.

The automaton $\mathcal{B}_n$ is based on an automaton $\mathcal{B}_{x,y}$ for appropriate x and y. Given $n \in \mathbb{N}$, let $x = \lfloor \sqrt{n} \rfloor$ and $y = x + 1$. Since no prime can divide both x and $x + 1$, the numbers x and y are relatively prime. Therefore, by Lemma 3.4, the nondeterministic automaton $\mathcal{B}_{x,y}$ rejects $n' = (x-1)(y-1) - 1$ and accepts all $i > n'$ with $y \leq \sqrt{n} + 1$ states. The automaton $\mathcal{B}_n$ is now obtained by preceding the initial state of $\mathcal{B}_{x,y}$ with a chain of $n - n'$ states. Thus, B_n reads the first $n - n'$ letters in the input and then runs $\mathcal{B}_{x,y}$. As $\sqrt{n} - 1 < x \leq \sqrt{n}$, a simple calculation shows that $0 \leq n - n' < 3\sqrt{n}$, implying that $\mathcal{B}_n$ has at most $4\sqrt{n}$ states.

It is left to prove a lower bound. In [Bir96], it is shown that two-way deterministic automata (automata in which the reading head can move both forward or backward over the input) need $n + 2$ states in order to count to n. Since two-way deterministic automata are easy to complement, this implies that two-way deterministic automata need $n + 2$ states in order to accept exactly all integers $k \neq n$. The lower bound then follows from fact that every n-state nondeterministic automaton over a unary language can be translated to an $O(n^2)$-state two-way deterministic automaton [Cho86]. $\square$

Remark 3.6 While for general unary languages, universal two-way automata are more succinct than one-way automata [Cho86], the ability to go back and forth the input is not helpful for universal automata in the context of counting. To see that, we prove that a two-way nondeterministic automaton needs $O(\sqrt{n})$ states to recognize $\{1^k : k \neq n\}$. In [**?**], it is proved that if a unary language is accepted by a two-way nondeterministic automaton $\mathcal{A}$ with x states, then there is an integer $l < O(e^{\sqrt{x \ln x}})$ such that for every integer $k > 8x^2$, we have that $1^k \in \mathcal{L}(\mathcal{A})$ iff $1^{k+l} \in \mathcal{L}(\mathcal{A})$. So if there is an automaton with $x < \sqrt{\frac{n}{8}}$ states for $\{1^k : k \neq n\}$, its behavior on 1^n coincides with its behavior on numbers larger than n, which is wrong. $\square$

4 Purposeful Counting with Automata

For $n \in \mathbb{N}$ and a regular language $L \subseteq \Sigma^*$, for some alphabet Σ, we define the language $L_n = (0+1)^n \cdot 0 \cdot L$. Thus, L_n contains all the words obtained by adding a prefix of length $n+1$ to a word in L, where the prefix is an arbitrary combination of 0 and 1, ending with 0. So, all the words in L_n have 0 in their $n+1$'th position. We say that a type of automata *purposefully counts to n with $f(n)$ states* iff for every language L that is accepted by an automaton of this type with k states, the language L_n is accepted by an automaton of this type with $k + f(n)$ states. Intuitively, unlike an automaton that merely counts to n, an automaton that purposefully counts to n is able to use its counting information in order to take decisions other than acceptance during the run.

Theorem 4.1 *For every n, deterministic, nondeterministic, universal, and alternating automata purposefully count to n with $\Theta(n)$ states.*

Proof: For the upper bound, given n and L, let $\mathcal{B} = \langle \Sigma, Q, q_{in}, M, F, \rho \rangle$ be the automaton for L. Consider the automaton $\mathcal{A} = \langle \Sigma \cup \{0, 1\}, Q \cup \{0, \dots n + 1\}, 0, M', F, \rho' \rangle$, where for $q \in Q$ and $\sigma \in \Sigma$ we have $M'(q, \sigma) = M(q, \sigma)$, and for the new states $\{0, \dots n+1\}$, we have $M'(i, \sigma) = i+1$ for $i \leq n$ and $\sigma \in \{0, 1\}$, $M(n + 1, 0) = q_{in}$, and $M(n + 1, 1) = \emptyset$. Also, ρ' agrees with ρ about the classification of states in Q and maps all states in $\{0, \dots n + 1\}$ to $\exists$. It is easy to see that $\mathcal{B}$ recognizes L_n.

For the lower bound, we fix L to $(0+1)^*$. Thus, for all $n \geq 1$, we have $L_n = (0+1)^n \cdot 0 \cdot (0+1)^*$. We prove that an alternating automaton that recognizes L_n must have $n + 1$ states. Consider the reverse languages $L_n^{rev} = (0+1)^* \cdot 0 \cdot (0+1)^n$. It is well known that a deterministic automaton that recognizes L_n^{rev} must have 2^{n+1} states [?]. We describe the proof here, as we would later need a revised version of it for the proof of Theorem 5.2. Assume by way of contradiction that there is a deterministic automaton $\mathcal{A}$ with less than 2^{n+1} states that recognizes L_n^{rev}. Then, there are two words w_1 and w_2 of length $n + 1$ such that $w_1 \neq w_2$ yet $\mathcal{A}$ reaches the same state q after reading each of the words. Let $h \cdot 0 \cdot t_1$ and $h \cdot 1 \cdot t_2$ be partitions of w_1 and w_2, respectively, such that h is the maximal common prefix of w_1 and w_2. Consider now the words $w_1' = w_1 h = h \cdot 0 \cdot t_1 \cdot h$ and $w_2' = w_2 h = h \cdot 1 \cdot t_2 \cdot h$. Since $|t_1| + |h| = |t_2| + |h| = n$ we have that $w_1' \in L_n^{rev}$ and $w_2' \notin L_n^{rev}$. On the other hand, since $\mathcal{A}$ reaches the state q after reading both the prefix w_1 of w_1' and the prefix w_2 of w_2', the automaton $\mathcal{A}$ either rejects both words or accepts both words. Now, by [BL80], for every regular language R, we have that R can be recognized by a deterministic automaton with n states iff R^{rev} can be recognized by an alternating automaton with $\log n$ states. Since $(R^{rev})^{rev} = R$, it follows that R cannot be recognized by an alternating automaton with less than $n = \log 2^n$ states.

□

Theorem 4.1 shows that the disability of the spawned processes of the automaton to collaborate during the processing of the input word is crucial.

Remark 4.2 In our definition of purposeful counting, the termination of the task of counting may not coincide with the end of the input, which disables a collaboration of the spawned processes of the automaton. On the other hand, the start of the task of counting does coincide with the beginning of the input. Consequently, it is not hard to see that for every n, two-way alternating automata can purposefully count to n with $\Theta(\log n)$ states. Indeed, in order to recognize L_n, a two-way automaton can guess the 0 in position $n+1$ and then universally branches into two processes. One counts backwards to n with $\log n$ states (as in Theorem 3.2), and one follows the automaton for L. Nevertheless, a definition of purposeful counting in which both the start and the termination of the task of counting do not necessarily coincide with the beginning and end of the input (say by defining $L_n = L \cdot 0 \cdot (0+1)^n \cdot 0 \cdot L$) would prevent alternating two-way automata from efficient purposeful counting. $\square$

5 Counting with Synchronized Automata

In Section 4, we saw that while alternating automata can efficiently count to n, the disability of the spawned processes of the automaton to communicate during the run prevents alternating automata from efficiently purposefully count to n. It is clear that the full power of concurrency leads to an efficient purposeful counting. Weaker types of concurrency studied in the literature involve *synchronization* between the spawned processes [**?**, DHK+89]. Essentially, in synchronous alternating automata, some of the states are designated as synchronization states, and the spawned processes of the automaton have to agree on positions in the input word in which they visit synchronization states. In particular, in *partially input-synchronized alternating automata* [**?**], the spawned processes have to agree on the input letter they read in synchronization states.

It turns out that synchronization is helpful for handling regular expressions containing conjunctions. Formally, for an alphabet Σ, a semi-extended regular expression (SERE) over Σ is one of the following.

- $\emptyset$ (the empty set), ϵ (the empty word), or σ, for $\sigma \in \Sigma$.
- $r_1 \vee r_2$, $r_1 \cdot r_2$, r_1^*, or $r_1 \wedge r_2$, for SEREs r_1 and r_2.

Thus, SERE extend regular expressions with the $\wedge$ operator. For an SERE r, we denote by $L(r)$ the set of words generated by r. While it is easy to efficiently translate a regular expression to an alternating (and even nondeterministic) automaton recognizing its language, the question whether such a translation exists for SERE is open, and the translation of SERE to automata in [**?**] uses partially input-synchronized alternating automata. We use purposeful counting in order to answer the question to the negative.

What we would like to do is to claim that SERE can count to n efficiently. Since SERE can be concatenated, it would follow that SERE can also purposefully count to n efficiently. Indeed, if r_n and r are SERE such that $L(r_n) = (0+1)^n$, then $L(r_n \cdot 0 \cdot r)$ contains exactly all words $(0+1)^n \cdot 0 \cdot w$, for $w \in L(r)$. Since, on the other hand, we know that automata cannot purposefully count to n efficiently, a lower bound for the translation of SERE to automata would follow. It turns out that this plan cannot be followed: as we show in Theorem 5.7, SERE cannot count to n efficiently, and our plan should be modified and consider counting *modulo n*.

For an alphabet Σ and $n \in \mathbb{N}$, let $C_n^{mod} = \{\Sigma^i : i = 0 \text{ mod } n\}$. We say that an SERE r *counts mod* n iff $L(r) = C_n^{mod}$. We now examine purposeful counting modulo n. For $n \in \mathbb{N}$ and a regular language $L \subseteq \Sigma^*$, for some alphabet Σ, we define the language $L_n^{mod} = \{(0+1)^i \cdot 0 \cdot L : i = 0 \text{ mod } n\}$. Thus, L_n^{mod} contains all the words obtained by adding a prefix of length $cn+1$ to a word in L, for $c \geq 0$, where the prefix is an arbitrary combination of 0 and 1, ending with 0. We say that a type of automata *purposefully counts mod* n *with* $f(n)$ *states* iff for every language L that is accepted by an automaton of this type with k states, the language L_n^{mod} is accepted by an automaton of this type with $k + f(n)$ states. Similarly, we say that SERE *purposefully counts mod* n *with expressions of length* $f(n)$ iff for every language L that is definable by an SERE of length k, the language L_n^{mod} is definable by an SERE of length $k+f(n)$ states. Finally, for $m \geq 1$, let $n_m = p_1 \cdot p_2 \cdots p_m$ be the product of the first m prime numbers, and let $\mathcal{P}$ denote the set of integers of this form. That is, $\mathcal{P} = \{1, 2, 6, 30, 210, 2310, \ldots\}$.

Theorem 5.1 *For $n \in \mathcal{P}$, SERE purposefully count mod n with expressions of length polylog(n).*

Proof: Let m be such that $n = p_1 \cdot p_2 \cdots p_m$, where $p_1, \ldots, p_m$ are the first m prime numbers. Define $r_n = \bigwedge_{1 \leq j \leq m} ((0 \vee 1)^{p_j})^*$. Thus, a word $(0 \vee 1)^i$ is in $L(r_n)$ iff for all $1 \leq j \leq m$, we have that $i = 0 \text{ mod } p_j$. This happens iff $i = 0 \text{ mod } n$. Since $p_j = O(j \log j)$ and $m \leq \log n$, we have that $p_j \leq \log n \log \log n$, and the expression r_n is of length $O(\log^2 n \log \log n)$.

Now, since SERE can be concatenated, it follows that SERE can also purposefully count mod n with expressions of length *polylog*(n). Indeed, if r_n is such that $L(r_n) = C_n^{mod}$, then $r_n \cdot 0 \cdot r_L$ defines $L(r)_n^{mod}$. □

Theorem 5.2 *There is no efficient translation of semi-extended regular expressions to alternating automata.*

Proof: By Theorem 5.1, SERE can purposefully count mod $n \in \mathcal{P}$ with expressions of length *polylog*(n). On the other hand, we now show that alternating automata need $\Omega(n)$ states in order to purposefully count mod n, for $n \in \mathcal{P}$. We follow the same considerations as in the proof of Theorem 4.1.

Let $L = (0+1)^*$. Thus, for all $n \in \mathcal{P}$, we have $L_n^{mod} = \{(0+1)^i \cdot 0 \cdot (0+1)^* : i = 0 \text{ mod } n\}$. We prove that an alternating automaton that recognizes L_n^{mod} must have $n+1$ states. Consider the languages $L' = \{(0+1)^* \cdot 0 \cdot (0+1)^i : i = 0 \text{ mod } n\}$. Note that $L' = (L_n^{mod})^{rev}$. We prove that a deterministic automaton that recognizes L' must have 2^{n+1} states. The proof is very similar to the lower-bound proof in Theorem 4.1, only that here, the argument that $w_2' = h \cdot 1 \cdot t_2 \cdot h \notin L'$ is less straightforward, and it follows from the fact that $|w_2'| = n + 1 + |h| \leq 2n$, thus the only candidate for $i = 0 \text{ mod } n$ is $i = n$. Now, by [BL80], for every regular language R, we have that R can be recognized by a deterministic automaton with n states iff R^{rev} can be recognized by an alternating automaton with $\log n$ states. Since $L' = (L_n^{mod})^{rev}$, it follows that L_n^{mod} cannot be recognized by an alternating automaton with less than $n = \log 2^n$ states. □

Note that Theorem 5.2 refers only to the lower bound of translating SERE to alternating automata, and we do not discuss here the upper bound. In [**?**], Yamamoto describes a linear translation of SERE to partially input-synchronized alternating finite automata. It follows that partially input-synchronized alternating finite automata can purposefully count mod n, for $n \in \mathcal{P}$, with $polylog(n)$ states. Hence the following corollary.

Corollary 5.3 *Partially input-synchronized alternating automata are super-polynomially more succinct than alternating automata*[2].

Recall that while in Section 4 we consider purposeful counting to all $n \in \mathbb{N}$, here we consider purposeful counting mod n, for $n \in \mathcal{P}$. Thus, we do it only for integers in $\mathcal{P}$. While this is sufficient for getting the super-polynomial lower bound on a translation of SERE to alternating automata, the question whether SERE can efficiently count to n mod n, for all $n \in \mathbb{N}$, is still open. We now answer this question to the negative; thus our restriction to counting modulo n cannot be avoided. (Clearly, if SERE cannot count to n efficiently, they also cannot purposefully count to n efficiently.) We first need some notations. Intuitively, the weakness of SERE is that while alternating automata can directly refer to strict suffixes of the input word, which is the key to exact (as opposed to modulo) counting in Theorem 3.3, SERE have no direct way to refer to strict suffixes. Recall we are interested in unary languages, and we assume there is only one letter 1 in the alphabet. For simplicity, we refer to a unary language as a set of integers and associates the word 1^n with the number n.

We say that a set S of integers is *periodical* if it contains exactly all elements of the form $ax + b$ for some fixed integers a and b. We say that an SERE r *accepts* n *with a period*, if there is a periodical set S such that $n \in S$ and $S \subseteq L(r)$. Note that $L(r)$ does not have to be periodical, just to contain a periodical set that contains n. We say that an SERE r *accepts* n *without a period*, if it accepts n, yet it does not accept n with a period.

Example 5.4 Let r be the expression $(111 \cdot (11)^*)^*$. Then, $L(r)$ contains all multiples of $3, 5, 7, 9, 11, \ldots$. I.e., r accepts all integers except powers of two. Clearly, $L(r)$ itself is not periodical. On the other hand, every number in $L(r)$ is accepted with a period.

Example 5.4 is no coincidence. We will show that every small SERE that accepts a large integer must accept it with a period. We first need the following properties of SERE.

Claim 5.5

(1) *If* r_1 *and* r_2 *accept* k *with a period, then so does* $r_1 \wedge r_2$.

(2) *If* $k_1 \in L(r_1)$, $k_2 \in L(r_2)$, *and either* r_1 *accepts* k_1 *with a period or* r_2 *accepts* k_2 *with a period, then* $r_1 \cdot r_2$ *accepts* $k_1 + k_2$ *with a period.*

Proof: We start with the first claim. Let $S_1 = a_1x + b_1$ be such that $k \in S_1 \subseteq L(r_1)$ and let $S_2 = a_2x + b_2$ be such that $k \in S_2 \subseteq L(r_2)$. Then, $S = (a_1a_2)x + k$ satisfies $k \in S$ and $S \subseteq S_1 \cap S_2$. Hence, $r_1 \wedge r_2$ accepts k with a period.

[2] We use the term "super-polynomial" to denote the gap between $polylog(n)$ and n, which is more than polynomial but less than exponential. For example, if $m_1 = \Theta(\log^2 n)$ and $m_2 = \Theta(n)$, then $m_2 = \Theta(2^{\sqrt{m_1}})$.

For the second claim, assume without loss of generality that r_1 accepts k_1 with a period. let $S_1 = a_1x + b_1$ be such that $k_1 \in S_1 \subseteq L(r_1)$. Then, $S = a_1x + (b_1 + k_2)$ is such that $k_1 + k_2 \in S$ and $S \subseteq L(r_1 \cdot r_2)$. Hence, $r_1 \cdot r_2$ accepts $k_1 + k_2$ with a period. $\square$

Lemma 5.6 *If r accepts n without a period, then r has length at least n.*

Proof: The proof proceeds by induction on n. The base case $n = 1$ is clear. Assume that the claim holds for every $n = 1, \ldots, k - 1$. Let r be the smallest expression that accepts k without a period. Let us look at the outermost operation in r. It can be $\vee, \wedge, \cdot$, or $*$. We consider each of these cases.

- If $r = r_1 \vee r_2$, then either r_1 or r_2 accept k. Assume without loss of generality that r_1 accepts k. Since r accepts k without a period, so does r_1, contradicting the fact that r is the smallest SERE that accepts k without a period.
- If $r = r_1 \wedge r_2$, then both r_1 and r_2 accept k. If both accept k with a period, then by Claim 5.5 (1), so does r, contradicting the fact that r accepts k without a period. Hence, at least one of r_1 and r_2 accepts k without a period, contradicting the fact that r is the smallest SERE that accepts k without a period.
- If $r = (r_1)^*$, then r accepts k with a period, contradicting the fact r accepts k without a period.
- If $r = r_1 \cdot r_2$, then there are k_1 and k_2 such that r_1 accepts k_1, r_2 accepts k_2, and $k_1 + k_2 = k$. Since r accepts k without a period, then, by Claim 5.5 (2), both expressions r_i accept k_i without a period. Hence, by the induction hypothesis, the length of r_i is at least k_i, implying that r has length at least $k_1 + k_2 = k$.

$\square$

We now use Lemma 5.6 in order to conclude that SERE cannot count efficiently.

Theorem 5.7 *For every n, SERE count to n with expressions of length $\Theta(n)$.*

Proof: The upper bound is trivial. For the lower bound, let r_n be an SERE with $L(r_n) = \{1^n\}$. Since r_n accepts n and rejects k for all $k > n$, it must accept n without a period. Hence, by Lemma 5.6, r_n has length at least n. $\square$

We note that, unlike SERE, partially input-synchronized alternating automata can count to n with *polylog*(n) states [**?**]. Hence, partially input-synchronized alternating automata are super-polynomially more succinct not only than alternating automata, but also than SERE.

Acknowledgment We thank Jean-Camille Birjet for helpful discussions.

References

[Bir93] J.C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical Systems Theory*, 26(3):237–269, 1993.

[Bir96] J.C. Birget. Two-way automata and length-preserving homomorphisms. *Mathematical Systems Theory*, 29(3):191–226, 1996.

[BL80] J.A. Brzozowski and E. Leiss. Finite automata and sequential networks. *Theoretical Computer Science*, 10:19–35, 1980.

[Cho86] M. Chorbak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[DH94] D. Drusinsky and D. Harel. On the power of bounded concurrency I: Finite automata. *Journal of the ACM*, 41(3):517–539, 1994.

[DHK+89] J. Dassow, J. Haromkovic, J. Karhuaki, B. Rovan, and A. Slobodova. On the power of synchronization in parallel computing. In *Proc. 14th International Symp. on Mathematical Foundations of Computer Science*, volume 379 of *Lecture Notes in Computer Science*, pages 196–206. Springer-Verlag, 1989.