Finding the Growth Rate of a Regular of Context-Free Language in Polynomial Time

Paweł Gawrychowski[1,⋆], Dalia Krieger[2], Narad Rampersad[2], and Jeffrey Shallit[2]

[1] Institute of Computer Science, University of Wrocław
ul. Joliot-Curie 15, PL-50-383 Wrocław, Poland
gawry1@gmail.com
[2] School of Computer Science, University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
{d2kriege@cs,nrampersad@math,shallit@graceland}.uwaterloo.ca

Abstract. We give an $O(n + t)$ time algorithm to determine whether an NFA with n states and t transitions accepts a language of polynomial or exponential growth. Given a NFA accepting a language of polynomial growth, we can also determine the order of polynomial growth in $O(n+t)$ time. We also give polynomial time algorithms to solve these problems for context-free grammars.

1 Introduction

Let $L \subseteq \Sigma^*$ be a language. If there exists a polynomial $p(x)$ such that $|L \cap \Sigma^m| \leq p(m)$ for all $m \geq 0$, then L has *polynomial growth*. Languages of polynomial growth are also called *sparse* or *poly-slender*. If there exists a real number $r > 1$ such that $|L \cap \Sigma^m| \geq r^m$ for infinitely many $m \geq 0$, then L has *exponential growth*. Languages of exponential growth are also called *dense*. If there exist words $w_1, w_2, \ldots, w_k \in \Sigma^*$ such that $L \subseteq w_1^* w_2^* \cdots w_k^*$, then L is called a *bounded language*.

Ginsburg and Spanier (see [6, Chapter 5], [7]) proved many deep results concerning the structure of bounded context-free languages. One significant result [6, Theorem 5.5.2] is that determining if a context-free grammar generates a bounded language is decidable. However, although it is a relatively straightforward consequence of their work, they did not make the following connection between the bounded context-free languages and those of polynomial growth: a context-free language is bounded if and only if it has polynomial growth. Curiously, this result has been independently discovered at least six times: namely, by Trofimov [23], Latteux and Thierrin [14], Ibarra and Ravikumar [9], Raz [19], Incitti [11], and Bridson and Gilman [2]. A consequence of all of these proofs is that a context-free language has either polynomial or exponential growth; no intermediate growth is possible.

⋆ Research supported by MNiSW grant number N N206 1723 33, 2007-2010.

M. Ito and M. Toyama (Eds.): DLT 2008, LNCS 5257, pp. 339–358, 2008.

The particular case of the bounded regular languages was also studied by Ginsburg and Spanier [8], and subsequently by Szilard, Yu, Zhang, and Shallit [22] (see also [10]). Shur [20,21] has also studied the growth rate of regular languages. It follows from the more general decidability result of Ginsburg and Spanier that there is an algorithm to determine whether a regular language has polynomial or exponential growth (see also [22, Theorem 5]). Ibarra and Ravikumar [9] observed that the algorithm of Ginsburg and Spanier runs in polynomial time for NFA's, but they gave no detailed analysis of the runtime. Here we give a linear time algorithm to solve this problem. If the growth rate is polynomial we show how to find the order of polynomial growth in linear time.

For the general case of context-free languages, an analysis of the algorithm of Ginsburg and Spanier shows that it requires exponential time. Assuming that we want to explicitly construct words $w_1, w_2, \ldots, w_k \in \Sigma^*$ such that $L \subseteq w_1^* w_2^* \cdots w_k$, this exponential complexity is unavoidable, as there are context-free languages for which an exponentially large value of k is required. Surprisingly, it turns out that using a more complicated algorithm it is possible to check if a given context-free language has polynomial growth in polynomial time. We also give a polynomial time algorithm for finding the exact order of this growth.

Due to space considerations, the proofs of certain lemmas have been removed to the appendix.

2 Regular Languages

2.1 Polynomial vs. Exponential Growth

In this section we give an $O(n+t)$ time algorithm to determine whether an NFA with n states and t transitions accepts a language of polynomial or exponential growth.

Theorem 1. *Given a NFA M, it is possible to test whether $L(M)$ is of polynomial or exponential growth in $O(n+t)$ time, where n and t are the number of states and transitions of M respectively.*

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We assume that every state of M is both accessible and co-accessible, i.e., every state of M can be reached from q_0 and can reach a final state. For each state $q \in Q$, we define a new NFA $M_q = (Q, \Sigma, \delta, q, \{q\})$ and write $L_q = L(M_q)$.

Following Ginsburg and Spanier, we say that a language $L \subseteq \Sigma^*$ is *commutative* if there exists $u \in \Sigma^*$ such that $L \subseteq u^*$. The following lemma is essentially a special case of a stronger result for context-free languages (compare [6, Theorem 5.5.1], or in the case of languages specified by DFA's, [22, Lemmas 2 and 3]).

Lemma 1. *The language $L(M)$ has polynomial growth if and only if for every $q \in Q$, L_q is commutative.*

We now are ready to prove Theorem 1.

Proof. Let n denote the number of states of M. The idea is as follows. For every $q \in Q$, if L_q is commutative, then there exists $u \in \Sigma^*$ such that $L_q \subseteq u^*$. For any $w \in L_q$, we thus have $w \in u^*$. If z is the primitive root of w, then z is also the primitive root of u. If $L_q \subseteq z^*$, then L_q is commutative. On the other hand, if $L_q \not\subseteq z^*$, then L_q contains two words with different primitive roots, and is thus not commutative. This argument leads to the following algorithm.

Let $q \in Q$ be a state of M. We wish to check whether L_q is commutative. Any accepting computation of M_q can only visit states in the strongly connected component of M containing q. We therefore assume that M is indeed strongly connected (if it is not, we run the algorithm on each strongly connected component separately; they can be determined in $O(n+t)$ time [3, Section 22.5]).

We first construct the NFA M_q accepting L_q. This takes $O(n+t)$ time. Then we find a word $w \in L(M_q)$, where $|w| < n$. If $L(M_q)$ is non-empty, such a w exists and can be found in $O(n+t)$ time. Next we find the *primitive root* of w, i.e., the shortest word z such that $w = z^k$ for some $k \geq 1$. This can be done in $O(n)$ time using any linear time pattern matching algorithm. To find the primitive root of $w = w_1 \cdots w_\ell$, find the first occurrence of w in $w_2 \cdots w_\ell w_1 \cdots w_{\ell-1}$. If the first occurrence begins at position i, then $z = w_1 \cdots w_i$ is the primitive root of w.

For $i = 0, 1, \ldots, |z| - 1$, let A_i be the set of all $q' \in Q$ such that there is a path from q to q' labeled by a word from $z^* z_1 z_2 \cdots z_i$. Observe that if some q' belongs to A_i and A_j where $i < j$ then we can find two different paths from q to q: $z^a z_1 \cdots z_i s$ and $z^b z_1 \cdots z_j s$, where a and b are non-negative integers and s is the label of some path from q' to q. For L_q to be commutative, both these words must be powers of z, which is impossible: their lengths are different modulo $|z|$. Thus, the A_i's must be disjoint.

We determine the A_i's as follows. To begin, we know $q \in A_0$. For any i, if $q' \in A_i$, then we know $q'' \in A_{(i+1) \bmod |z|}$ for all $q'' \in \delta(q', z_{1+i \bmod |z|})$. Based on this rule, we proceed to iteratively assign states to the appropriate A_i until all states have been assigned. If some q' appears in two distinct A_i's, we terminate and report that L_q is not commutative. Since we never need to examine a state more than once, it follows that the complexity of computing the A_i's is $O(n+t)$.

Next, for each i we check that for each $q' \in A_i$ all outgoing transitions are labeled by $z_{1+i \bmod |z|}$. If not, L_q cannot be commutative (as we could then find a path from q to q that is not a power of z). If this holds for all i, the automaton has a very simple structure: it is an $|z|$-partite graph (the A_i's forming the partition classes) and edges outgoing from one partition class all have the same label. Thus, every path that starts and ends in q is a power of z. Furthermore, every path that starts and ends in some q' is a power of some cyclic shift of z. Thus, $L_{q'}$ is also commutative, so we do not have to repeat the computation for the remaining states of M (i.e., the states in $Q \setminus \{q\}$).

We have therefore determined whether L_q is commutative for all $q \in Q$, and hence whether $L(M)$ has polynomial or exponential growth, in $O(n+t)$ time. □

2.2 Finding the Exact Order of Polynomial Growth

In this section we show that given an NFA accepting a language of polynomial growth, it is possible to efficiently determine the exact order of polynomial growth.

Let M be an NFA accepting a language of polynomial growth. We will need the following definition:

Definition 1. *We call $x_0y_1^*x_1y_2^*x_2\dots y_k^*x_k$, where each y_i is non-empty, a* star expression of level k. *We say that it is* primitive *when each y_i is primitive.*

We would like to decompose $L(M)$ into languages described by such expressions. Let us look at the exact order of polynomial growth of the language described by a primitive star expression. It is easy to see that $L(x_0y_1^*x_1y_2^*x_2\dots y_k^*x_k)$ has $O(m^{k-1})$ growth; getting a lower bound is slightly more involved. Let $T = x_0y_1^*x_1y_2^*x_2\dots y_k^*x_k$ be a primitive star expression. If there exists $1 \le i < k$ such that $x_i = y_i^l y_i[1..j]$ and $y_{i+1} = y_i^{(j)}$ (y_i cyclically shifted by j) for some $l \ge 0$ and $j < |y_i|$, then we say that i is a *fake index*.

Lemma 2. *Let $T = x_0y_1^*x_1y_2^*x_2\dots y_k^*x_k$ be a primitive star expression. $L(T)$ has $\Theta(m^{k-1})$ growth iff T has no fake indices.*

Extending the above lemma gives us the following result:

Lemma 3. *Let $T = x_0y_1^*x_1y_2^*x_2\dots y_k^*x_k$ be a primitive star expression. $L(T)$ has $\Theta(m^{k-1-d})$ growth iff there are exactly d fake indices in T.*

We thus have an efficient way of calculating the growth order of a language described by a primitive star expression. However, we need a stronger result:

Theorem 2. *Let $T = x_0y_1^*x_1y_2^*x_2\dots y_k^*x_k$ be a star expression. $L(T)$ has $\Theta(m^{k-1-d})$ growth iff there are exactly d indices $1 \le i < k$ such that $x_i = \mathrm{root}(y_i)^l\mathrm{root}(y_i)[1..j]$ and $\mathrm{root}(y_{i+1}) = \mathrm{root}(y_i)^{(j)}$ for some $l \ge 0$ and $j < |\mathrm{root}(y_i)|$, where $\mathrm{root}(w)$ stands for the primitive root of w.*

Proof. Obviously, replacing each y_i by its primitive root cannot remove any word from the described language. Thus $L(T)$ has $O(m^{k-1-d})$ growth. To show that it is $\Omega(m^{k-1-d})$, we use the same method as in the previous two lemmas (see the appendix for the proofs): first we get rid of all d fake indices (here it may happen that we decrease the language in question, as replacing $(y^a)^*(y^b)^*$ by $(y^a)^*$ only might be necessary), then construct the corresponding equation and show that it has at most one solution for each word. □

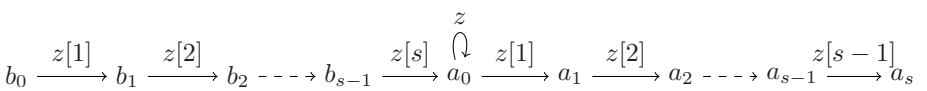
Corollary 1. *Inserting an additional y^* into a star expression T and replacing any y_i by some power of its primitive root or y_i^* by $y_i^a y_i^* y_i^b$ do not change the growth order of $L(T)$.*

Now we return to the original problem. Given M, we will construct a new automaton M' which will be almost acyclic, the only cycles being self-loops. Recall

that as $L(M)$ has polynomial growth, the algorithm of Theorem 1 partitions the vertices of each strongly connected component into sets $A_0, A_1, \ldots, A_{|z|-1}$. Take one such component S and let z be the corresponding primitive word. Now choose any $u, v \in S$ and consider labels of paths from u to v:

Case 1: if $u \in A_i, v \in A_j$ and $i \geq j$, then they are all of the form $z[i+1..|z|]z^p z[1..j]$. Furthermore, for some $q, r > 0$, there are such paths for every $p \in \{q, q+r, q+2r, \ldots\}$.
Case 2: if $u \in A_i, v \in A_j$ and $i < j$, then they are all of the form $z[i+1..j]$ or $z[i+1..|z|]z^p z[1..j]$. As in the previous case, for some $q, r > 0$, there are paths of the second form for every $p \in \{q, q+r, q+2r, \ldots\}$.

This suggests the following construction: for each state v create two copies v_{in} and v_{out}. Each edge connecting u and v belonging to two different strongly connected components gets replaced by an edge from u_{out} to v_{in} with the same label. For each non-singleton strongly connected component S we create the following gadget:



For each $u \in S$ we find i such that $u \in A_i$ and add edges $u_{in} \xrightarrow{\epsilon} b_i$, $b_i \xrightarrow{\epsilon} u_{out}$ and $a_i \xrightarrow{\epsilon} u_{out}$. The starting state of M' is simply q_{in}, where q is the start state of M, and v_{out} is a final state of M' whenever v is a final state of M.

$L(M')$ should be viewed as a finite sum of languages described by star expressions corresponding to labels of simple paths from the start state to some final state (whenever there is a self-loop adjacent to some vertex v and labeled with z, we should think that v is labelled with z^*). Let $\mathcal{T}$ denote the set of all such star expressions, so that $L(M') = \sum_{T \in \mathcal{T}} L(T)$.

Theorem 3. *The orders of polynomial growth of $L(M)$ and $L(M')$ are the same.*

Now we can focus on finding the growth order of $L(M')$. Although M' has a relatively simple structure, $\mathcal{T}$ can be of exponential size, so we cannot afford to construct it directly. This turns out to not be necessary due to the characterization of growth orders of primitive star expressions that we have developed. Take an expression $T = x_0 y_1^* x_1 y_2^* x_2 \ldots y_k^* x_k$ and observe that to calculate the growth order of $T y_{k+1}^* x_{k+1}$ we only need to know the growth order of T, the word y_k and if x_k is of the form $y_k^l y_k[1..j]$, the value of j. This suggests a dynamic programming solution: for each vertex v of M' we calculate: (1) the greatest possible growth order of a star expression $T = x_0 y_1^* x_1 y_2^* x_2 \cdots y_k^* x_k$ such that x_k is not a prefix of y_k^∞ and T is a label of a path from the start state to v; (2) for each y_k being a label of some self-loop and $j < |y_k|$, the greatest possible growth order of a star expression $T = x_0 y_1^* x_1 y_2^* x_2 \cdots y_k^* x_k$ such that x_k is of the form $y_k^l y_k[1..j]$ and T is a label of a path from the start state to v.

We process the vertices of M' in topological order (ignoring the self-loops gives us such order). Assuming that we already have the correct information

for some vertex, we can iterate through the outgoing transitions and update the information for all its successors. For example, knowing that there is a path ending in u whose label is a star expression $\cdots y_k^* y_k^l y_k[1..j]$ having a growth order of d, a transition $u \xrightarrow{c} v$ and a self-loop $v \xrightarrow{z} v$, we can construct a path ending in v whose label is a star expression $\cdots y_k^* y_k^l y_k[1..j] c z^*$ having a growth order of $d+1$ if $c \neq y_k[j+1]$ or $z \neq y_k^{(j+1)}$, and d otherwise. At the very beginning we know only that for any self-loop $v \xrightarrow{z} v$ there is a path ending in v whose label is a star expression $\cdots z^*$ having a growth order of 0.

We have to calculate $O(|Q|)$ information for each vertex. There are at most $O(|Q||\delta|)$ updates and each of them can be done in $O(1)$ time if we can decide in constant time whether $z' = z^{(j)}$ for any j and labels z, z' of some self-loops in M'. As both z, z' are primitive, there can be at most one such j. We can preprocess it for all pairs of labels in time $O(|Q|^2)$, giving a $O(|Q||\delta|)$ algorithm.

It turns out that we can achieve linear complexity by reducing the amount of information kept for each vertex. First observe that whenever we have two labels z, z' of self-loops such that $z' = z^{(i)}$, any star expression $\cdots z'^* z'^l z'[1..j]$ can be treated as an expression $\cdots z^* z^l z[1..1 + (i + j - 1) \mod |z|]$ having the same growth order. After such reductions for each vertex v we can keep information only about those two expressions of the form $\cdots y_k^* y_k[1..j]$ that have the greatest growth order among all possible y_k and j. Indeed, whenever we have an expression $T_1 T_2$ such that $T_1 = \cdots y_k^* y_k[1..j]$ is a label of a path ending in v and there are two different (with respect to the above reduction) expressions T_1', T_1'' that have greater or equal order of growth and are also labels of paths ending in v, at least one of $T_1' T_2$, $T_1'' T_2$ will have growth order as large as $T_1 T_2$. This decreases the amount of information kept for each vertex to a constant. To get linear total complexity we must improve the runtime of the preprocessing phase. Recall that it is possible to find the lexicographically smallest cyclic shift of a word in linear time, for example by using Duval's algorithm. Such a shift is the same for any two conjugate primitive words, so we calculate the smallest cyclic shifts of all labels of self-loops and then group those labels whose shifts are the same. This can be done by inserting them one-by-one into a trie. After such preprocessing we can find the value of j such that $z' = z^{(j)}$ in constant time for any two labels z, z'.

Theorem 4. *Given an NFA M with n states and t transitions such that $L(M)$ is of polynomial growth, there is an algorithm that finds the exact order of polynomial growth of $L(M)$ in $O(n + t)$ time.*

2.3 An Algebraic Approach for DFA's

We now consider an algebraic approach to determining whether the order of growth is polynomial or exponential, and in the polynomial case, the order of polynomial growth. Shur [21] has recently presented a similar algebraic method for this problem. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, where $|Q| = n$, and let $A = A(M) = (a_{ij})_{1 \leq i,j \leq n}$ be the *adjacency matrix* of M, that is, a_{ij} denotes the number of paths of length 1 from q_i to q_j. Then $(A^m)_{i,j}$ counts the number of

paths of length m from q_i to q_j. Since a final state is reachable from every state q_j, the order of growth of $L(M)$ is the order of growth of A^m as $m \to \infty$. This order of growth can be estimated using nonnegative matrix theory.

Theorem 5 (Perron-Frobenius). *Let A be a nonnegative square matrix, and let r be the spectral radius of A, i.e., $r = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } A\}$. Then*

1. *r is an eigenvalue of A;*
2. *there exists a positive integer h such that any eigenvalue λ of A with $|\lambda| = r$ satisfies $\lambda^h = r^h$.*

For more details, see [17, Chapters 1, 3].

Definition 2. The number $r = r(A)$ described in the above theorem is called the *Perron-Frobenius eigenvalue of A*. The *dominating Jordan block* of A is the largest block in the Jordan decomposition of A associated with $r(A)$.

Lemma 4. *Let A be a nonnegative $n \times n$ matrix over the integers. Then either $r(A) = 0$ or $r(A) \geq 1$.*

Proof. Let $r(A) = r, \lambda_1, \ldots, \lambda_\ell$ be the distinct eigenvalues of A, and suppose that $r < 1$. Then $\lim_{m\to\infty} r^m = \lim_{m\to\infty} \lambda_i^m = 0$ for all $i = 1, \ldots, \ell$, and so $\lim_{m\to\infty} A^m = 0$ (the zero matrix). But A^m is an integral matrix for all $m \in \mathbb{N}$, and the above limit can hold if and only if A is nilpotent, i.e., $r = \lambda_i = 0$ for all $i = 1, \ldots, \ell$. □

Lemma 5. *Let A be a nonnegative $n \times n$ matrix over the integers. Let $r(A) = r, \lambda_1, \ldots, \lambda_\ell$ be the distinct eigenvalues of A, and let d be the size of the dominating Jordan block of A. Then $A^m \in \Theta(r^m m^{d-1})$.*

Note: The growth order of A^m supplies an algebraic proof of the fact that regular languages can grow either polynomially or exponentially, but no intermediate growth order is possible. This result can also be derived from a more general matrix-theoretic result of Bell [1].

Lemma 5 implies that to determine the order of growth of $L(M)$, we need to compute the Perron-Frobenius eigenvalue r of $A(M)$: if $r = 0$, then $L(M)$ is finite; if $r = 1$, the order of growth is polynomial; if $r > 1$, the order of growth is exponential. In the polynomial case, if we want to determine the order of polynomial growth, we need to also compute the size of the dominating Jordan block, which is the algebraic multiplicity of r in the minimal polynomial of $A(M)$.

Both computations can be done in polynomial time, though the runtime is more than cubic. The characteristic polynomial, $c_A(x)$, can be computed in $\tilde{O}(n^4 \log \|A\|)$ bit operations (here $\tilde{O}$ stands for soft-O, and $\|A\|$ stands for the L_∞ norm of A). If $c_A(x) = x^n$ then $r = 0$; else, if $c_A(1) \neq 0$, then $r > 1$. In the case of $c_A(1) = 0$, we need to check whether $c_A(x)$ has a real root in the open interval $(1, \infty)$. This can be done using a real root isolation algorithm; it seems the best deterministic one uses $\tilde{O}(n^6 \log^2 \|A\|)$ bit operations [4]. The

minimal polynomial, $m_A(x)$, can be computed through the rational canonical form of A in $\tilde{O}(n^5 \log \|A\|)$ bit operations (see references in [5]). All algorithms mentioned above are deterministic; both $c_A(x)$ and $m_A(x)$ can be computed in $\tilde{O}(n^{2.697} \log \|A\|)$ bit operations using a randomized Monte Carlo algorithm [12].

An interesting problem is the following: given a nonnegative integer matrix A, is it possible to decide whether $r(A) > 1$ in time better than $\tilde{O}(n^6 \log^2 \|A\|)$? Using our combinatorial algorithm, we can do it in time $O(n^2 \log \|A\|)$, as follows. We first construct a graph G from A by creating edges (i, j) in G if the ij entry of A is positive. We can do this in $O(n^2 \log \|A\|)$ time. We then find the strongly connected components of G in $O(n^2)$ time. For each edge (i, j), if i and j are in the same strongly connected component and the ij entry of A is > 1, then we may immediately report that $r(A) > 1$. We thus assume henceforth that if i and j are in the same strongly connected component of G, then the ij entry of A is at most 1.

We now consider the strongly connected components of G separately. For each strongly connected component H of G, we turn H into a DFA M (we don't bother specifying an initial state or final states) over an m-letter alphabet, where m is the number of vertices of H, as follows. For each edge (i, j) in H, we turn the edge (i, j) into a single transition labeled by an alphabet symbol that has not already been used for a transition outgoing from i. This is justified by our previous assumption that the ij entries of A within the same strongly connected component are at most 1. Thus, there are at most m outgoing transitions from a given state, and the DFA M has m states and $O(m^2)$ transitions. We now run the algorithm of Theorem 1 to determine if all of the L_q's of M are commutative in $O(m^2)$ time. If so, then H has polynomial growth; otherwise, it has exponential growth.

If all strongly connected components of G have polynomial growth, then $r(A) \leq 1$; otherwise, $r(A) > 1$. The total running time of the algorithm is $O(n^2 \log \|A\|)$.

3 Context-Free Languages

Given a context-free grammar $G = (V, \Sigma, R, S)$, we are interested in checking whether $L(G)$ has polynomial growth. We assume that G is in Chomsky normal form, each nonterminal can be derived from S and languages generated by nonterminals are nonempty. The following result can be found in [6, Theorem 5.5.1].

Lemma 6. *The language generated by a CFG $G = (V, \Sigma, R, S)$ is bounded if and only if for each nonterminal A both* $\mathrm{left}(A)$ *and* $\mathrm{right}(A)$ *are commutative, where* $\mathrm{left}(A) = \{u : A \stackrel{*}{\Rightarrow} uAw \text{ for some } w \in \Sigma^*\}$ *and* $\mathrm{right}(A) = \{u : A \stackrel{*}{\Rightarrow} wAu \text{ for some } w \in \Sigma^*\}$.

From now on we focus on testing whether each $\mathrm{left}(A)$ is commutative. To test all $\mathrm{right}(A)$ we reverse all productions and repeat the whole procedure. Note that if $L \subseteq \Sigma^*$ is commutative and $w \in L$, then $L \subseteq \mathrm{root}(w)^*$. So, to check if a

nonempty left(A) is commutative we should: (1) take any $u \in \text{left}(A)$ and set w to be its primitive root, and (2) verify that each $u \in \text{left}(A)$ is a power of w.

Before we proceed further we need a convenient description of left(A). Define a graph $H = (V, E)$ where V is the set of all nonterminals and for each production $A \to BC$ we put $A \xrightarrow{\epsilon} B$ and $A \xrightarrow{B} C$ into E. Each left(A) is a sum of languages generated by labels of paths from A to A, where the label of a path is the concatenation of the labels of all of its edges.

In our algorithm we will make heavy use of results concerning *straight-line programs*. A SLP is a context-free grammar in Chomsky normal form such that each nonterminal occurs on the left side of exactly one production and derives exactly one word. Such grammars should be viewed as a convenient way of describing compressed words. Given a text T and a pattern P, both as SLPs, there are polynomial time algorithms for finding the first occurrence of P in T or detecting that there is no such occurrence (see [13] or [15] for a more efficient version). Given a SLP describing some word w we can easily construct a SLP describing any subword of w.

Constructing a SLP describing some $u \in \text{left}(A)$ is quite straightforward. We first define the function $\text{length}(A) := \min\{|w| : A \stackrel{*}{\Rightarrow} w\}$. We need the following:

Lemma 7. *Given a context free grammar, we can calculate for each nonterminal A the value of* length(A) *in polynomial time.*

To construct u, first use the above lemma. For each nonterminal A choose one production: $A \to a$ if there is such an a, and $A \to BC$ for which $\text{length}(A) = \text{length}(B)+\text{length}(C)$ otherwise. It is easy to see that after removing all the other productions any nonterminal A still generates some word. Let $X_1 X_2 \dots X_k$ be a label of one of the simple paths from A to A in H. If $k = 1$ we can take X_1 as the start symbol. Otherwise we add productions $Y_i \to X_i Y_{i+1}$ for $i = 1, \dots, k-2$ and $Y_{k-1} \to X_{k-1} X_k$ and make Y_1 the start symbol. It is easy to verify that in both cases the resulting grammar is a SLP describing some $u \in \text{left}(A)$.

Having a description of some $u \in \text{left}(A)$, we construct the description of $u[2..|u|]u[1..|u|-1]$ and use one of the compressed pattern matching algorithms to find the length of root(u). Having this length, we can easily construct a description of the primitive word w itself.

Now we should verify that $L(X_1 \dots X_k) \subseteq w^*$ for each label $X = X_1 \dots X_k$ of a path from A to A in H. If such containment does not hold we say that we found a *contradiction*. Let S be the strongly connected component of H containing A. The verification can be done in two steps:

Lemma 8. *There is a polynomial time algorithm that detects a contradiction or calculates for any $B \in S$ the value* pathlength(B) *such that the label of any path from A to B derives only words of lengths giving the remainder of* pathlength(B) *when divided by $|w|$.* □

Proof. First we apply Lemma 7. Then we need to verify that for any nonterminal C being a label of some edge connecting two vertices in S, the lengths of all words in $L(C)$ give the same remainder as length(C) when divided by $|w|$. This can

be done by checking that $\text{length}(X) \equiv \text{length}(Y) + \text{length}(Z) \mod |w|$ holds for any production $X \to YZ$ that can appear in a derivation of some word in $L(C)$. This condition is clearly necessary: if it does not hold, we can find $u, v \in L(C)$ such that $|u| \not\equiv |v| \mod |w|$ and a path from A to A having a label of the form $X_1X_2 \dots X_iCX_{i+2} \dots X_k$. This label derives two words whose lengths give different remainders when divided by $|w|$ so they cannot both be a power of w and we found a contradiction. To prove that this condition is also sufficient, we use induction to show that in such a case the lengths of all words in any $L(C)$ give the same remainder as $\text{length}(C)$ when divided by $|w|$. Indeed, it holds for all words having a derivation tree of depth 1. Assume that it holds for all words having a derivation tree of depth less than m and take $u \in L(C)$ having a derivation tree of depth $m > 1$. There must be a production $C \to DE$ such that $D \stackrel{*}{\Rightarrow} u_1$, $E \stackrel{*}{\Rightarrow} u_2$ where $u = u_1u_2$ and the derivation trees of both u_1 and u_2 have depths less than m. Thus from the induction hypothesis $|u_1| \equiv \text{length}(D) \mod |w|$ and $|u_2| \equiv \text{length}(E) \mod |w|$, so $|u| \equiv \text{length}(D) + \text{length}(E) \mod |w|$. We verified that $\text{length}(C) \equiv \text{length}(D) + \text{length}(E) \mod |w|$ so in fact $|u| \equiv \text{length}(C) \mod |w|$ and we are done.

Now we can define the values of $\text{pathlength}(B)$. For each edge in S set its weight to be 0 if its label is ϵ or $\text{length}(B)$ if it is some nonterminal B. Then for each $B \in S$ define $\text{pathlength}(B)$ to be the weight modulo $|w|$ of some path from A to B. Obviously, this value is the only possible candidate for $\text{pathlength}(B)$. We still need to check if it is correct, though. Verify that $\text{pathlength}(B) + \text{length}(s) \equiv \text{pathlength}(C) \mod |w|$ for each edge $B \stackrel{s}{\to} C$ in S where $\text{length}(\epsilon) = 0$. This condition is obviously necessary: otherwise we would have two paths from A to A whose labels derive words having different lengths modulo $|w|$ and a contradiction can be found. To see that it is also sufficient, we use induction to prove that the length of any word that can be derived from a label of any path from A to some B gives the remainder of $\text{pathlength}(B)$ when divided by $|w|$. □

Having the above lemma, we should check whether for each edge in S outgoing from some B and having a nonempty label of C, each word in $L(C)$ is a prefix of $(w[\text{pathlength}(B) + 1..|w|]w[1..\text{pathlength}(B)])^\infty$. This is clearly both necessary and sufficient, as it ensures that any word that can be derived from a label of any path starting in A and ending in B is of the form $w^*w[1..\text{pathlength}(B)]$.

Lemma 9. *There is a polynomial time algorithm that detects a contradiction or verifies that each word in $L(B)$ is a prefix of $(w[i..|w|]w[1..i-1])^\infty$.*

Proof. We describe the algorithm in terms of constraints. The meaning of a constraint $\langle B, i\rangle$ is that each word from $L(B)$ should be a prefix of $(w[i..|w|]w[1..i-1])^\infty$. We begin with only one such constraint, namely $\langle B, i\rangle$, which is initially marked as unprocessed. While there is an unprocessed constraint $\langle B, i\rangle$ for some B from which it is possible to derive a word of length at least $|w|$, we mark it as processed and add new constraints $\langle C, i\rangle$ and $\langle D, 1 + (i + \text{length}(C) - 1) \ mod |w|\rangle$ for each production $B \to CD$. We do not add a new constraint if it has been already processed. To achieve polynomial time we need the following observation: if we

have two processed constraints $\langle B, i\rangle$ and $\langle B, j\rangle$ where $i \neq j$ then $L(B)$ contains a word of length at least $|w|$ that should be a prefix of both $(w[i..|w|]w[1..i-1])^\infty$ and $(w[j..|w|]w[1..j-1])^\infty$. But then $w^{(i)} = w^{(j)}$ and w cannot be primitive. Thus we have found a contradiction as soon as the number of processed constraints is greater than the number of nonterminals and we can terminate. Checking if a given $L(B)$ contains a word of length at least $|w|$ can be done by identifying nonterminals that generate infinite languages first. All the others create an *acyclic* part of the grammar, in the sense that we can order them as $A_1, A_2, \ldots A_m$ in such a way that whenever $A_i \to A_j A_k$ is a production $j, k > i$ holds. Thus we can use a simple dynamic programming to calculate for each A_i the greatest length of a word in $L(A_i)$. This ensures a polynomial running time.

As a result we get a set of unprocessed constraints of the form $\langle A_i, r\rangle$ where all A_i belong to the acyclic part of the grammar. Additionally, we verified earlier that all lengths of words in each $L(A_i)$ give the same remainder when divided by $|w|$. Thus each such constraint can be rewritten as $L(A_i) = \{w'\}$ where w' is described by a SLP of polynomial size. Now we would like to verify this by either using the compressed pattern matching algorithm. Unfortunately, it may happen that the context free grammar describing a given $L(A_i)$ is not a SLP: it may happen that a nonterminal appears on the left side of more than one production. On the other hand, if for each A_i we remove all but one production with A_i on the left side, we get a SLP: each nonterminal generates exactly one word and appears on the left side of exactly one production. If this SLP does not generate w', we found a contradiction. If it does, for each removed production $A_i \to A_j A_k$ check whether $A_j A_k$ and A_i generate the same word in the constructed SLP. Similarly, for each removed production $A_i \to a$ check whether A_i generates a in the constructed SLP. This is obviously a necessary condition. To prove that it is also sufficient, we use induction to show that for each $i = m, m-1, \ldots, 1$ $|L(A_i)| = 1$. It is clear for $i = m$. Assume that it holds for all $j > i$ but A_i contains two different words u, v. We can assume that u is generated by A_i in the SLP. If the uppermost productions in the derivation trees of u and v are the same, we can immediately use the induction hypothesis. Otherwise, the condition we verified with the induction hypothesis give us that $u = v$. □

Combining these two lemmas gives:

Theorem 6. *There is a polynomial time algorithm that checks whether the language generated by a given context-free grammar has polynomial growth.*

Having checked that $L(G)$ is bounded, we would like to calculate the exact order of its polynomial growth as we did in the NFA case. The general idea is almost the same: we decompose $L(G)$ into languages described by star expressions and use dynamic programming (slightly more involved than in the NFA case) to calculate the greatest growth order of those expressions.

First for each nonterminal A we find primitive words $\text{left}_A, \text{right}_A$ using the above method such that $\text{left}(A) \subseteq \text{left}_A^*$, $\text{right}(A) \subseteq \text{right}_A^*$, and for some $\alpha, \beta \geq 1$,

$\text{left}_A^\alpha \in \text{left}(A)$ and $\text{right}_A^\beta \in \text{right}(A)$. In the case that one of the languages in question is empty, we take ϵ as the corresponding word.

We would like to construct a set of star expressions such that the maximum of their growth orders is the same as the growth order of $L(G)$, which was relatively simple in the case of regular languages. When we deal with context-free languages, things get more complicated. Consider a grammar $A \to uAv|a$ where u and v are different primitive words. Its growth order is clearly 0 while the obvious way of representing the language it generates as a star expression would give u^*av^* with a growth order of 1. On the other hand, adding a production $A \to u^2Av$ increases the growth order to 1 and in such a case we can represent the language in question as u^*av^*. It turns out that those two extreme situations are in some sense the only possibilities. To formalize this statement, we need

Definition 3. *We call $x_0 y_1^{v_1} x_1 y_2^{v_2} x_3 \dots y_k^{v_k} x_k$, where each y_i is nonempty and all v_i are different variables, a* generalized star expression*. Additionally, we are allowed to add constraints of the form $(v_i, v_j) \in C \subseteq \mathbb{N}^2$ as long as each variable is a part of at most one such constraint.*

The set of words described by such an expression contains words that we get by assigning nonnegative values to variables in a way that is consistent with all the constraints. Thus a star expression is just a generalized star expression with no constraints. This notion allows us to represent $L(G)$ in a convenient way. For any nonterminal A define

$$L_1(A) := \{a : A \to a \text{ is a production}\}$$
$$L_{i+1}(A) := L_i(A) \cup \bigcup_{A \to BC} \text{left}_A^\alpha L_i(B) L_i(C) \text{right}_A^\beta,$$

where $i = 1, 2, \dots, n-1$ and $(\alpha, \beta) \in \text{context}(A) := \left\{(\alpha, \beta) : A \stackrel{*}{\Rightarrow} \text{left}_A^\alpha \, A \, \text{right}_A^\beta\right\}$. Each $L_i(A)$ corresponds in a natural way to a finite sum of languages described by generalized star expressions and it is clear that $L_n(S) = L(G)$. We need to get rid of the constraints without modifying the growth order. This can be done in two phases:

Definition 4. *Given a nonterminal A such that $\text{left}_A, \text{right}_A \neq \epsilon$, we say that it is independent if $(\alpha, \beta_1), (\alpha, \beta_2) \in \text{context}(A)$ for some $\alpha, \beta_1 \neq \beta_2$.*

First we remove constraints concerning independent nonterminals, then we modify constraints concerning dependent nonterminals:

Lemma 10. *Given a generalized star expression, we can remove a constraint $(v_i, v_j) \in \text{context}(A)$, where A is an independent nonterminal, without changing the growth order of the language in question.*

Lemma 11. *Given a generalized star expression, we can replace each constraint $(v_i, v_j) \in context(A)$, where A is a dependent nonterminal, by either $(v_i, v_j) \in \mathbb{N} \times \{1\}$ or $(v_i, v_j) \cup \{1\} \times \mathbb{N}$ without changing the growth order.*

These two lemmas allow us to modify the definition of each $L_{i+1}(A)$ so as to get a set of primitive star expressions. If A is an independent nonterminal, we set

$$L_{i+1}(A) := L_i(A) \cup \bigcup_{A \to BC} \text{left}_A^* L_i(B) L_i(C) \text{right}_A^*$$

and otherwise we take

$$L_{i+1}(A) := L_i(A) \cup \bigcup_{A \to BC} \text{left}_A L_i(B) L_i(C) \text{right}_A^* \cup \bigcup_{A \to BC} \text{left}_A^* L_i(B) L_i(C) \text{right}_A$$

Before we show how to calculate the greatest growth order of an expression in $L_n(S)$, we need to prove that the above definition can be effectively used:

Lemma 12. *Given a context-free grammar, we can check in polynomial time if a given nonterminal A is independent.*

Now we can focus on finding the greatest growth order of a primitive star expression in $L_n(S)$. As in the NFA case, we observe that calculating the growth order of $\text{left}_A^* T_1 T_2 \text{right}_A^*$ requires only a partial knowledge about the structure of T_1 and T_2. Indeed, if $T_1 = x_0 y_1^* x_1 \cdots y_k^* x_k$ and $T_2 = x'_0 y_1'^* x'_1 \cdots y_{k'}'^* x'_{k'}$, where $k, k' \geq 1$, we need only to know the words y_k, y'_1, the growth orders of T_1, T_2, the length of x_k modulo $|y_k|$ if $x_k \in y_k^* y_k[1..j]$, and the length of x'_0 modulo $|y'_1|$ if $x'_0 \in y'_1[j'..|y'_1|]y_1'^*$. So, for each $L_i(A)$ we would like to calculate the greatest possible growth order of a star expression $T = x_0 y_1^* x_1 \ldots y_k^* x_k$ from $L_i(A)$ such that $k \geq 1$, y_1 and y_k are some left_B or right_B, and one of the following cases applies:

1. $x_0 \notin y_1[j'..|y_1|]y_1^*$ for all j' and $x_k \notin y_k^* y_k[1..j]$ for all j,
2. $x_0 \notin y_1[j'..|y_1|]y_1^*$ for all j' and $x_k \in y_k^* y_k[1..j]$,
3. $x_0 \in y_1[j'..|y_1|]y_1^*$ and $x_k \notin y_k^* y_k[1..j]$ for all j,
4. $x_0 \in y_1[j'..|y_1|]y_1^*$ and $x_k \in y_k^* y_k[1..j]$,

where $j = 0, 1, \ldots, |y_k| - 1$ and $j' = 1, 2, \ldots, |y_1|$. There is one problem, though: the lengths $|y_1|$ and $|y_k|$ can be exponential and we cannot afford to store information about an exponential number of states. To overcome this, observe that for a fixed y_1, y_k and j', it makes sense to keep information only about two different values of j for which the corresponding growth orders are greatest. The same applies to j'. This allows us to use dynamic programming: now there is only a polynomial number of different states to consider. Assuming that we have calculated the greatest growth orders of expressions in $L_i(A)$ for any nonterminal A, we can calculate growth order of expressions in all $L_{i+1}(A)$. There is one problem, though: in the above reasoning we assumed that both T_1 and T_2 have orders of at least 1 but three other cases are also possible:

Case 1: both T_1 and T_2 have orders of 0. As we are interested in getting an expression of order at least 1, we may assume that $\text{left}_A \neq \epsilon$ or $\text{left}_B \neq \epsilon$. $T_1 T_2$

is just a word that can be derived from B_nC_n using only the following new productions:

$$A_i \to a \text{ for each original production } A \to a$$
$$A_{i+1} \to B_iC_i \text{ for each original production } A \to BC$$

where $i = 1, 2, \ldots, n-1$. W.l.o.g. assume that $\text{left}_A \neq \epsilon$. To update information about growth orders of expressions of the form $\text{left}_A^* T_1 T_2 \text{right}_A^*$, we could calculate all j such that T_1T_2 can be of the form $\text{left}_A^* \text{left}_A[1..j]$ and check if it is possible that T_1T_2 is not of such form. As we mentioned before, in case there are many such j, we need only two of them. So, for each nonterminal A_i, in a bottom-up order, we calculate the set of remainders modulo $|\text{length}_A|$ of lengths of words that can be derived from A_i, which we call $R(A_i)$. If $|R(A_i)| > 1$, we can forget about all but two values so the complexity is polynomial. We can take $R(B_nC_n)$ as the set of j such that T_1T_2 can be of the form $\text{left}_A^* \text{left}_A[1..j]$. Of course even if $j \in R(B_nC_n)$ we do not know if we can find T_1T_2 being the corresponding prefix of left_A^∞, we only know that we can find T_1T_2 having the corresponding length. Fortunately, if T_1T_2 is not a prefix of left_A^∞, the growth order can only increase. What is more, if $|R(B_nC_n)| > 1$ we can forget about the possibility that T_1T_2 is not a prefix of left_A^* because at least one value of j will give us the same growth order. We still need to consider the case of $R(B_nC_n) = \{r\}$, though. In such a case we should check if each word in $L(B_nC_n)$ is of the form $\text{left}_A^* \text{left}_A[1..r]$. We can use Lemma 9 for that; it may not happen that it finds a contradiction as the language in question would not be bounded then.

Case 2: T_1 has an order of 0 but T_2 has a nonzero order. Now we cannot assume that $\text{left}_A \neq \epsilon$ (which was crucial in the previous case), but knowing that T_2 is of the form $x_0 y_1^* \ldots y_k^* x_k$ and has a specified growth order, we can use the same method as above, replacing left_A by y_1.

Case 3: T_1 has a nonzero order but T_2 has an order of 0. Similar as above.

We have proven the following theorem:

Theorem 7. *Given an CFG G such that $L(G)$ is of polynomial growth, there is a polynomial-time algorithm that finds the exact order of polynomial growth of $L(G)$.*

Acknowledgments

We would like to thank Arne Storjohann for his input regarding algorithms for computing the Perron-Frobenius eigenvalue of a nonnegative integer matrix.

References

1. Bell, J.: A gap result for the norms of semigroups of matrices. Linear Algebra Appl. 402, 101–110 (2005)
2. Bridson, M., Gilman, R.: Context-free languages of sub-exponential growth. J. Comput. System Sci. 64, 308–310 (2002)

3. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
4. Eigenwillig, A., Sharma, V., Yap, C.K.: Almost tight recursion tree bounds for the Descartes method. In: ISSAC 2006, pp. 71–78 (2006)
5. Giesbrecht, M., Storjohann, A.: Computing rational forms of integer matrices. J. Symbolic Comput. 34, 157–172 (2002)
6. Ginsburg, S.: The Mathematical Theory of Context-free Languages. McGraw-Hill, New York (1966)
7. Ginsburg, S., Spanier, E.: Bounded ALGOL-like languages. Trans. Amer. Math. Soc. 113, 333–368 (1964)
8. Ginsburg, S., Spanier, E.: Bounded regular sets. Proc. Amer. Math. Soc. 17, 1043–1049 (1966)
9. Ibarra, O., Ravikumar, B.: On sparseness, ambiguity and other decision problems for acceptors and transducers. In: Monien, B., Vidal-Naquet, G. (eds.) STACS 1986. LNCS, vol. 210, pp. 171–179. Springer, Heidelberg (1985)
10. Ilie, L., Rozenberg, G., Salomaa, A.: A characterization of poly-slender context-free languages. Theoret. Informatics Appl. 34, 77–86 (2000)
11. Incitti, R.: The growth function of context-free languages. Theoret. Comput. Sci. 225, 601–605 (2001)
12. Kaltofen, E., Villard, G.: On the complexity of computing determinants. Comput. Complex. 13, 91–130 (2004)
13. Karpinski, M., Rytter, W., Shinohara, A.: An efficient pattern-matching algorithm for strings with short descriptions. Nordic Journal of Computing 4, 172–186 (1997)
14. Latteux, M., Thierrin, G.: On bounded context-free languages. Elektron. Informationsverarb. Kybernet. 20, 3–8 (1984)
15. Lifshits, Y.: Solving classical string problems on compressed texts. In: CPM 2007, pp. 228–240 (2007)
16. Lyndon, R.C., Schützenberger, M.-P.: The equation $a^M = b^N c^P$ in a free group. Michigan Math. J. 9, 289–298 (1962)
17. Minc, H.: Nonnegative Matrices. Wiley, Chichester (1988)
18. Plandowski, W.: The Complexity of the Morphism Equivalence Problem for Context-Free Languages, PhD thesis
19. Raz, D.: Length considerations in context-free languages. Theoret. Comput. Sci. 183, 21–32 (1997)
20. Shur, A.M.: Combinatorial complexity of rational languages. Discr. Anal. and Oper. Research, Ser. 1 12(2), 78–99 (2005)
21. Shur, A.M.: Combinatorial complexity of regular languages. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) Computer Science – Theory and Applications. LNCS, vol. 5010, pp. 289–301. Springer, Heidelberg (2008)
22. Szilard, A., Yu, S., Zhang, K., Shallit, J.: Characterizing regular languages with polynomial densities. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629, pp. 494–503. Springer, Heidelberg (1992)
23. Trofimov, V.I.: Growth functions of some classes of languages. Cybernetics 6, 9–12 (1981)

A Appendix

In this appendix we give the full proofs of certain lemmas, which were omitted in the main text above due to space considerations.

A.1 Proofs Omitted from Section 2

Lemma 2. *Let $T = x_0y_1^*x_1y_2^*x_2 \dots y_k^*x_k$ be a primitive star expression. $L(T)$ has $\Theta(m^{k-1})$ growth iff T has no fake indices.*

Proof. Assume there is a fake index i. Then $y_i^*x_iy_{i+1}^*$ can be rewritten as $y_i^*y_i^l y_i[1..j](y_i[j+1..|y_i|]y_i[1..j])^*$, which is the same as $y_i^*y_i^l y_i^* y_i[1..j] = y_i^*y_i^l y_i[1..j]$. Thus $L(T)$ can be described by a primitive star expression of level $k-1$, so its growth is $O(m^{k-2})$.

Now assume there is no such i. First we modify T in the following way: whenever some x_i is of the form y_iz, replace $y_i^*x_i$ by $y_iy_i^*z$. This does not change $L(T)$. So, we can assume that for each $1 \le i < k$ either $x_i = y_i[1..j]$ and $y_{i+1} \ne y_i^{(j)}$ or $x_i = y_i[1..j]cz$ where $c \ne y_i[j+1]$ for some $j < |y_i|$. Then we observe that $L(T') \subseteq L(T)$ where $T' = x_0y_1'^+x_1y_2'^+x_2 \dots y_k'^+x_k$ and each y_i' is a power of y_i chosen so that all lengths $|y_i'|$ are the same.

Now we can show that for each $w \in \Sigma^*$ the equation

$$w = x_0y_1'^{\alpha_1}x_1y_2'^{\alpha_2}x_2 \dots y_k'^{\alpha_k}x_k, \text{ all } \alpha_i \ge 1,$$

has at most one solution. This will prove that $L(T')$ has $\Omega(m^{k-1})$ growth. We apply induction on k. The cases in which $k = 0, 1$ are obvious. Suppose $k > 1$. Assume that there are $a > b \ge 1$ such that both $x_0y_1'^ax_1y_2'$ and $x_0y_1'^bx_1y_2'$ are prefixes of w. Then x_1y_2' is a prefix of $y_1'^cx_1y_2'$ for some $c \ge 1$. Consider two possibilities:

Case 1: $x_1 = y_1'[1..j]$ for some $j < |y_i|$ (y_i' is a power of y_i). This means that y_2' is the same as $y_1'^{(j)}$. Therefore, the primitive roots of these two words should be the same. The length of the primitive root does not change after rotation, so we get $y_2 = y_1^{(j)}$, a contradiction.

Case 2: $x_1 = y_1'[1..j]cz$ for some $j < |y_1|$ and $c \ne y_1[j+1]$. This means that $y_1'[1..j]c$ is a prefix of y_1', which is impossible. □

Lemma 3. *Let $T = x_0y_1^*x_1y_2^*x_2 \dots y_k^*x_k$ be a primitive star expression. $L(T)$ has $\Theta(m^{k-1-d})$ growth iff there are exactly d fake indices in T.*

Proof. We have already proved this lemma for $d = 0$. Assume that $d > 0$. We can transform T to get an level $k-d$ primitive star expression containing no fake indices using the following operation: take a maximal contiguous segment of fake indices $i, i+1, \dots, i+m-1$ and observe that the whole expression $y_i^*x_iy_{i+1}^*x_{i+1} \dots y_{i+m}^*$ can be replaced by $y_i^*y_i^l y_i[1..j]$ for some $l \ge 0$ and $j < |y_i|$ without changing the language in question. Such a replacement decreases the number of fake indices by exactly m: the only possible place where a new fake index could be created is $y_i^*y_i^l y_i[1..j]x_{i+m}y_{i+m+1}^*$, but as we have chosen a maximal segment, there are only two possibilities (y' stands for $y_i^{(j)}$):

Case 1: x_{i+m} is not of the form $y'^*y'[1..j']$. Then $y_i^l y_i[1..j]x_{i+m}$ cannot be written as $y_i^{l'}y_i[1..j'']$;

Case 2: x_{i+m} is of the form $y'^* y'[1..j']$ but $y_{i+m+1} \neq y'^{(j')}$. Then $y_i^l y_i[1..j] x_{i+m} = y_i^{l'} y_i[1..j'']$ but $y_{i+m+1} \neq y_i^{(j'')}$.

Thus, repeating the above operations leaves us with a level $k-d$ primitive star expression having no fake indices and describing the same language (describing only some subset of $L(T)$ would be enough). Thus $L(T)$ has $\Theta(m^{k-d-1})$ growth. □

Theorem 3. *The orders of polynomial growth of $L(M)$ and $L(M')$ are the same.*

Proof. By the construction of M', it is easy to see that for each $w \in L(M)$ we can find a $T \in \mathcal{T}$ such that $w \in L(T)$.

The other direction is more involved: we must show that for each $T \in \mathcal{T}$ we can find a subset of $L(M)$ having the same order of growth. Of course, there is a natural way of doing that: T corresponds to a sequence of vertices of M:

$$u_1 \overset{w_1}{\to} v_1 \overset{c_1}{\to} u_2 \overset{w_2}{\to} v_2 \overset{c_2}{\to} \ldots \overset{c_{m-1}}{\to} u_m \overset{w_m}{\to} v_m, \qquad T = w_1 c_1 w_2 c_2 \ldots c_{m-1} w_m$$

such that u_k and v_k are in the same strongly connected component of M, $v_k \overset{c_k}{\to} u_{k+1}$ are transitions connecting different strongly connected components in M and w_k is either $z_k[i+1..j]$ or $z_k[i+1..|z_k|]z_k^* z_k[1..j]$, where z_k is the primitive root associated with the strongly connected component containing u_k, v_k. We would like to take the labels of all paths in M going through the above sequence of vertices. Unfortunately, two bad things may happen:

Case 1: $w_k = z_k[i+1..j]$ for some k but there is no path from u_k to v_k labeled with such a w_k in M. However, we know that there are paths labeled with $z_k[i+1..|z_k|]z_k^{q+\alpha r} z_k[1..j]$, so we can modify T, setting $w_k = z_k[i+1..j](t^r)^* t^{1+q}$, where $t = z_k^{(j)}$, which does not decrease the growth order. Then we can observe that setting $w_k = z_k[i+1..|z_k|](z_k^r)^* z_k^q z_k[1..j]$ results in the same language.

Case 2: $w_k = z_k[i+1..|z_k|]z_k^* z_k[1..j]$ but paths from u_k to v_k are of the form $z_k[i+1..|z_k|]z_k^{q+\alpha r} z_k[1..j]$. In this case, we know that replacing z_k^* in T by $(z_k^r)^* z_k^q$ does not change its growth order, so we can set $w_k = z_k[i+1..|z_k|](z_k^r)^* z_k^q z_k[1..j]$.

Thus we can modify T without decreasing its growth order so that $L(T)$ is contained in the set of labels of paths in M going through the above sequence of vertices. □

Lemma 5. *Let A be a nonnegative $n \times n$ matrix over the integers. Let $r(A) = r, \lambda_1, \ldots, \lambda_\ell$ be the distinct eigenvalues of A, and let d be the size of the dominating Jordan block of A. Then $A^m \in \Theta(r^m m^{d-1})$.*

Proof. The theorem trivially holds for $r = 0$. Assume $r \geq 1$. Without loss of generality, we can assume that A does not have an eigenvalue λ such that $\lambda \neq r$ and $|\lambda| = r$; if such an eigenvalue exists, replace A by A^h. Let J be the Jordan canonical form of A, i.e., $A = SJS^{-1}$, where S is a nonsingular matrix, and J is a diagonal block matrix of Jordan blocks. We use the following notation: $J_{\lambda,e}$ is a Jordan block of order e corresponding to eigenvalue λ, and O_x is a square

matrix, where all entries are zero, except for x at the top-right corner. Let $J_{r,d}$ be the dominating Jordan block of A. It can be verified by induction that

$$J^m_{r,d} = \begin{pmatrix} r^m & \binom{m}{1}r^{m-1} & \binom{m}{2}r^{m-2} & \cdots & \binom{m}{d-2}r^{m-d+2} & \binom{m}{d-1}r^{m-d+1} \\ 0 & r^m & \binom{m}{1}r^{m-1} & \cdots & \binom{m}{d-3}r^{m-d+3} & \binom{m}{d-2}r^{m-d+2} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & r^m & \binom{m}{1}r^{m-1} \\ 0 & 0 & 0 & \cdots & 0 & r^m \end{pmatrix}.$$

Thus the first row of $J^m_{r,d}$ has the form

$$r^m \left[1 \quad \frac{m}{r} \quad \frac{m(m-1)}{2!r^2} \quad \cdots \quad \frac{m(m-1)\cdots(m-(d-2))}{(d-1)!r^{d-1}}\right],$$

and so

$$\lim_{m\to\infty} \frac{J^m_{r,d}}{r^m m^{d-1}} = O_\alpha, \quad \text{where } \alpha = \frac{1}{(d-1)!r^{d-1}}\,.$$

All Jordan blocks other than the dominating block converge to zero blocks. and

$$\lim_{m\to\infty} \frac{A^m}{r^m m^{d-1}} = S \lim_{m\to\infty} \frac{J^m}{r^m m^{d-1}} S^{-1}.$$

The result follows. □

A.2 Proofs Omitted from Section 3

Lemma 7. *Given a context free grammar, we can calculate for each nonterminal A the value of* $\mathrm{length}(A)$ *in polynomial time.*

Proof. We calculate the values of $\mathrm{length}_i(A)$ defined as follows:

$$\mathrm{length}_1(A) := \begin{cases} 1 & A \to a \text{ is a production for some } a \\ \infty & \text{otherwise} \end{cases}$$

$$\mathrm{length}_{i+1}(A) := \min\{\mathrm{length}_i(B) + \mathrm{length}_i(C) : A \to BC \text{ is a production } \}$$

for every nonterminal A and $i = 1, 2, \ldots, n$, where n is the number of all nonterminals. This obviously takes polynomial time as we have n^2 values to compute and each $\mathrm{length}_i(A)$ is either ∞ or at most 2^i. Observe that $\mathrm{length}_i(A)$ is in fact the smallest possible length of a word that can be derived from A in such a way that the derivation tree is of depth at most i. If we consider a shortest word that can be derived from A, its derivation tree is of depth at most n, so we can take $\mathrm{length}(A) = \mathrm{length}_n(A)$. □

In the following two proofs we show that the order of polynomial counting all words of length at most N does not change. It is equivalent to checking that the growth order does not change.

Lemma 10. *Given a generalized star expression, we can remove a constraint* $(v_i, v_j) \in \text{context}(A)$*, where* A *is an independent nonterminal, without changing the growth order of the language in question.*

Proof. By joining maximum segments of fake indices we can show that the number of words of length at most N is exactly the number of different vectors $\vec{x} = [\vec{a}_1\vec{v}, \vec{a}_2\vec{v}, \ldots, \vec{a}_k\vec{v}]$ where all $\vec{a}_k$ have nonnegative coordinates and the i-th coordinate of $\vec{v}$ is simply the value assigned to the i-th variable, for all possible assignments consistent with constraints such that $\vec{x}\vec{c} \leq N$ where $\vec{c}$ is some fixed vector with strictly positive coordinates. If all $\vec{a}_k^{(i)}$ are zero or all $\vec{a}_k^{(j)}$ are zero we can safely remove the constraint because the value of one variable does not matter. Otherwise there is exactly one k such that $\vec{a}_k^i$ is nonzero and exactly one k such that $\vec{a}_k^{(j)}$ is nonzero. If they are the same we can also safely remove the constraint because only the sum of those variables matters. Thus by reordering the coordinates of $\vec{x}$ we can assume that $\vec{a}_{k-1}^{(i)}, \vec{a}_k^{(j)} > 0$. Now the number of words of length N when we ignore the constraint $(v_i, v_j) \in \text{context}(A)$ is at most N^2 multiplied by the number of different vectors $[\vec{a}_1\vec{v}, \vec{a}_2\vec{v} \ldots \vec{a}_{k-2}\vec{v}]$ such that $\sum_j \vec{a}_j\vec{v}c_j \leq N$. On the other hand, we know that $(\alpha, \beta_1), (\alpha, \beta_2) \in \text{context}(A)$ for some $\alpha, \beta_1 < \beta_2$ and it is possible to show that the number of words of length tN when we keep all constraints is at least N^2 multiplied by the number of different vectors $[\vec{a}_1\vec{v}, \vec{a}_2\vec{v} \ldots \vec{a}_{k-2}\vec{v}]$ such that $\sum_j \vec{a}_j\vec{v}c_j \leq N$, where t is some constant ($t = 2\alpha c_{k-1} + 3\beta_2 c_k$ is big enough). This shows that ignoring the constraint $(v_i, v_j) \in \text{context}(A)$ does not increase the growth order. □

Lemma 11. *Given a generalized star expression, we can replace each constraint* $(v_i, v_j) \in \textit{context}(A)$*, where* A *is a dependent nonterminal, by either* $(v_i, v_j) \in \mathbb{N} \times \{1\}$ *or* $(v_i, v_j) \cup \{1\} \times \mathbb{N}$ *without changing the growth order.*

Proof. As in the previous lemma, we join maximum segments of fake indices. If two constrained variables occur in the same segment, we can treat them as one unconstrained variable. If some segment contains an unconstrained variable, it contributes 1 to the final order. We can delete such segment, removing all constraints concerning variables it contains. Thus w.l.o.g. we can assume that each variable is constrained by some $\text{context}(A)$ where A is a dependent nonterminal and each two constrained variables occur in different segments.

Again, the number of words of length at most N is exactly the number of different vectors $\vec{x} = [\vec{a}_1\vec{v}, \vec{a}_2\vec{v}, \ldots, \vec{a}_k\vec{v}]$ for all possible consistent assignments such that $\vec{x}\vec{c} \leq N$. Imagine a multigraph on k vertices, each of them corresponding to one coordinate of $\vec{x}$, in which we put an edge (x, y) for each constraint (v_i, v_j) such that $\vec{a}_x^{(i)}, \vec{a}_y^{(j)} > 0$. Now observe that coordinates corresponding to vertices from different connected components are completely independent; the resulting order is simply the sum of orders corresponding to different components. Thus it is enough to consider one such component. There are two possibilities:

Case 1: this component is a tree. Then the number of vectors $\vec{x}$ such that $\vec{x}\vec{c} \leq N$ is $\Theta(N^{k-1})$. To see why, choose any vertex to be the root of this tree and then fix coordinates starting from the leaves in a bottom-up fashion. After choosing the

values of all coordinates but the root, the value of the coordinate corresponding to the root is uniquely determined. Thus the number of vectors is $O(N^{k-1})$. To see that it is $\Omega(N^{k-1})$, observe that there are constants $\alpha_v < \beta_v$ such that for each coordinate corresponding to a non-root v we can assign any value from $[\alpha_v N, \beta_v N]$.

Case 2: this component contains a cycle. Then the number of vectors $\vec{x}$ such that $\vec{x}\vec{c} \leq N$ is $\Theta(N^k)$. Obviously, it is $O(N^k)$. Take any spanning tree of this component and choose the root to be a vertex having an incident edge outside this spanning tree. As in the previous case, we can choose any value from some $[\alpha_v N, \beta_v N]$ for each coordinate corresponding to a non-root v. Then we can modify the coordinate corresponding to the root by using the variable corresponding to the additional edge. An appropriate choice of the constants $\alpha_v < \beta_v$ gives us that there are $\Omega(N^k)$ possible choices.

This shows how to replace each constraint by either $\mathbb{N} \times \{1\}$ or $\{1\} \times \mathbb{N}$ without decreasing the growth order. More specifically, we replace $(v_i, v_j) \in \mathit{context}(A)$ by $\mathbb{N} \times \{1\}$ if we want to fix the value of v_i and the value of v_j is not fixed yet.

It can be also checked that no replacement will result in an increased growth order. □

Lemma 12. *Given a context-free grammar, we can check in polynomial time if a given nonterminal A is independent.*

Proof. First we check if for each nonterminal B that can appear in some derivation $A \stackrel{*}{\Rightarrow} u_1 B u_2 A v$ or $A \stackrel{*}{\Rightarrow} u A v_1 B v_2$, it holds that $|\{|w| : B \stackrel{*}{\Rightarrow} w\}| = 1$. This can be done in polynomial time by checking that for any production $C \to DE$ that can appear in some derivation $B \stackrel{*}{\Rightarrow} w$, $\mathrm{length}(C) = \mathrm{length}(D) + \mathrm{length}(E)$. If it does not hold then clearly A is independent. Otherwise we build a graph $H = (V, E)$ where V is the set of all nonterminals and E contains edges $B \to C$ of weight $(0, \mathrm{length}(D))$ and $B \to D$ of weight $(\mathrm{length}(C), 0)$ for any production $B \to CD$ and take the strongly connected component containing A. We define the weight of a path to be the component-wise sum of weights of its edges. It is clear that to check if A is independent, we should check if there are two paths from A to A having different weights (α, β_1) and (α, β_2). Take any path from A to A with a weight of (x, y) where $x, y > 0$ (if there is no such path, we are done) and for each edge replace its weight (α, β) by $y\alpha - x\beta$. Now it is clear that we should check if there is a path from A to A with a nonzero modified weight. This can be done in polynomial time using a similar method to the one from Lemma 8. □