Finding the growth rate of a regular or 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

[2] School of Computer Science, University of Waterloo

September 18, 2008

Given a language $L \subseteq \Sigma^*$ we define its growth function:

$$f_L(m) = |L \cap \Sigma^m|$$

Theorem

If L is a context-free language, there are only two possible cases:

1. *it has polynomial growth: $f_L(m) \leq p(m)$ for all m,*
2. *it has exponential growth: $f_L(m) \geq c^m$ for infinitely many values of m*

This result seems to have been independently discovered quite a few times but not much have been known about the computational complexity of distinguishing between those two cases.

Main result

Given a context-free language L, we can test whether it is of polynomial or exponential growth in polynomial time. Moreover, if it is of polynomial growth we can also find the exact order of this polynomial in polynomial time.

where the exact order is d if $f_L(m) \in O(m^d)$ for all m and $f_L(m) \in \Omega(m^d)$ for infinitely many values of m.

Main result

Given a regular language L, we can test whether it is of polynomial or exponential growth in linear time.
Moreover, if it is of polynomial growth we can also find the exact order of this polynomial in linear time.

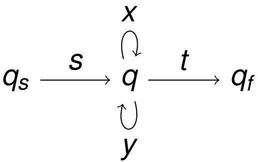
where the exact order is d if $f_L(m) \in O(m^d)$ for all m and $f_L(m) \in \Omega(m^d)$ for infinitely many values of m.

Theorem

Given a regular language L, we can test whether it is of polynomial or exponential growth in linear time.

Choose any state q and consider the set of all paths from q to q. If there are two paths x, y such that $xy \neq yx$, L has exponential growth:

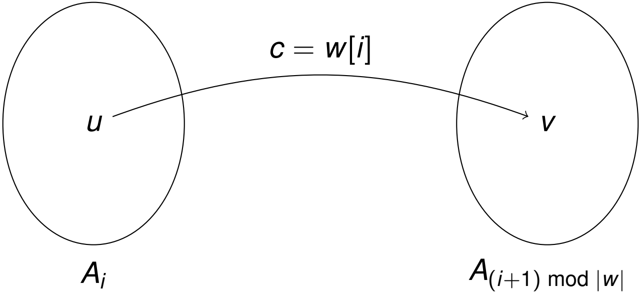
$$s(xy|yx)^*t \subseteq L$$



Otherwise this set must be commutative i.e. contained in w^* for some w.

To prove the theorem, we give an efficient algorithm for verifying that all those sets are indeed commutative.

First find any simple path from q to q and set w to be its primitive root. Then try to partition the strongly connected component containing q into $A_0, A_1, \ldots, A_{|w|-1}$ such that for any transition $u \xrightarrow{c} v$ the following holds:



Theorem

Given a context-free language L, we can test whether it is of polynomial or exponential growth in polynomial time.

Assume that L is specified by a context-free grammar $G = \langle V, \Sigma, R, S \rangle$. We use the following criteria:

Lemma

L is of polynomial growth if and only if all $left(A)$ and $right(A)$ are commutative where

- $left(A) = \{u : A \overset{*}{\Rightarrow} uAw \text{ for some } w \in \Sigma^*\}$
- $right(A) = \{u : A \overset{*}{\Rightarrow} wAu \text{ for some } w \in \Sigma^*\}$

How to check whether a single $\mathit{left}(A)$ is commutative?

1. choose any $s \in \mathit{left}(A)$ and set w to be its primitive root,
2. verify that all words in $\mathit{left}(A)$ are powers of w.

But it is not that simple. For example, it may happen that even the shortest such w is of exponential length.

Definition

A straight-line program is a sequence of assignments of the form $A_i \to a$ or $A_i \to A_j A_k$ where $j, k < i$.

Example

$$\begin{aligned} F_1 &\to 1 \\ F_2 &\to 0 \\ F_3 &\to F_2 F_1 \\ F_4 &\to F_3 F_2 \\ F_5 &\to F_4 F_3 \\ F_6 &\to F_5 F_4 \end{aligned}$$

Checking if two such programs describe the same string is not trivial! But there are efficient algorithms for that. In fact, even pattern matching for strings specified by such programs can be done in polynomial time.

We can easily find a description of some $s \in \mathit{left}(A)$ as a SLP of polynomial size. Then we use existing pattern matching algorithms to construct a SLP describing its primitive root w.

How to verify that $\mathit{left}(A) \subseteq w^*$?

1. look at lengths of words in $\mathit{left}(A)$,
2. construct a set of constraints stating that all words in $\mathit{left}(A)$ are powers of w,
3. transform those constraints in such a way that they can be verified using SLP equality checking.

Determining the exact order of an arbitrary language might be quite complicated. In some cases it is pretty easy, though:

Definition

We call $x_0 y_1^* x_1 y_2^* \dots y_k^* x_k$ a star expression of level k.

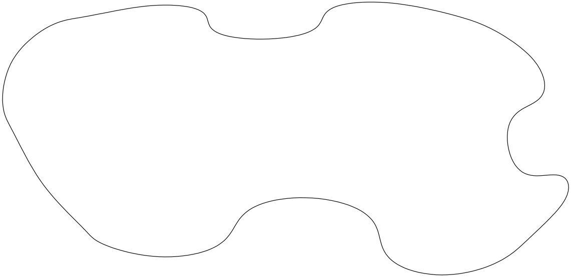
What is the growth order of such language?

$(abc)^* ab(cba)^*$ has $d = 0$
$(abc)^* ab(abc)^*$ has $d = 1$
$(abc)^* ab(bca)^*$ has $d = 1$

There is a simple and efficient formula for the value of d.

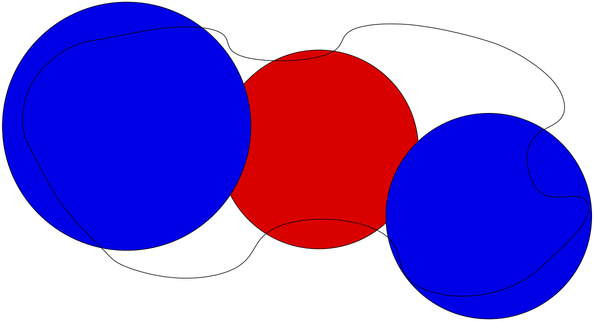
To find the growth order of a language L we try to cover it by those nicely behaving star expressions.

Unfortunately, we may need an exponential number of different star expressions so we cannot afford to explicitly construct this cover.



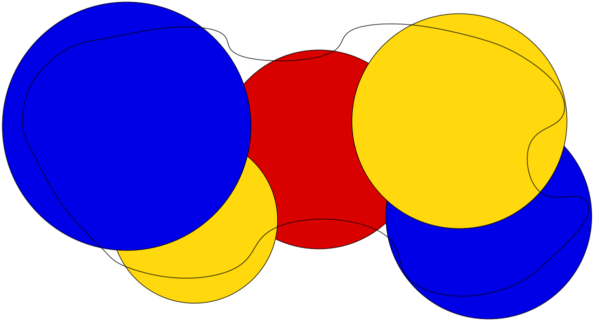
To find the growth order of a language L we try to cover it by those nicely behaving star expressions.

Unfortunately, we may need an exponential number of different star expressions so we cannot afford to explicitly construct this cover.

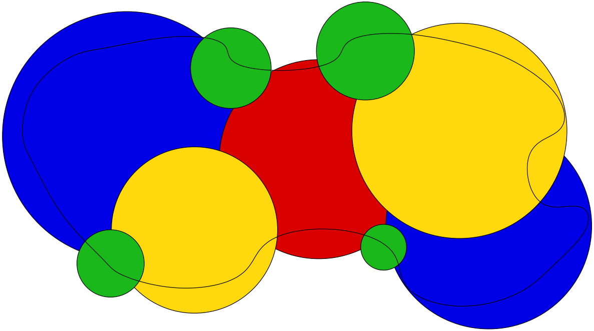


To find the growth order of a language L we try to cover it by those nicely behaving star expressions.

Unfortunately, we may need an exponential number of different star expressions so we cannot afford to explicitly construct this cover.



To find the growth order of a language *L* we try to cover it by those nicely behaving star expressions.
Unfortunately, we may need an exponential number of different star expressions so we cannot afford to explicitly construct this cover.



Observation

To calculate the order of $T' = Ty_{k+1}^* x_{k+1}$ where $T = x_0 y_1^* \dots y_k^* x_k$ we do not need to have the whole T. Knowing y_k, x_k and the order of T is enough.

This observation allows us to apply dynamic programming in which we calculate the maximum order of a star expression in a specific cover of L.

Context-free languages

In this case things get way more complicated. Consider the following two examples:

$$S \to aSb|\epsilon \quad S \to aSb|a^2Sb|\epsilon$$

a natural cover for both of them would be a^*b^* but only one of them has linear growth rate!

Observation

To calculate the order of $T' = Ty_{k+1}^* x_{k+1}$ where $T = x_0 y_1^* \ldots y_k^* x_k$ we do not need to have the whole T. Knowing y_k, x_k and the order of T is enough.

This observation allows us to apply dynamic programming in which we calculate the maximum order of a star expression in a specific cover of L.

Context-free languages

In this case things get way more complicated. Consider the following two examples:

$$S \rightarrow aSb|\epsilon \quad S \rightarrow aSb|a^2Sb|\epsilon$$

a natural cover for both of them would be a^*b^* but only one of them has linear growth rate!

Thank you for your attention!