Deciding whether a regular language is power-closed is PSPACE-complete

Yuval Filmus

September 2, 2012

Abstract

A regular language L is power-closed if whenever $x \in L$, also $x^k \in L$ for all $k \geq 1$. We show that given a deterministic finite automaton A, it is PSPACE-complete to decide whether the language accepted by A is power-closed.

1 Introduction

Let L be a language over some finite alphabet Σ. Calbrix and Nivat [4], while studying prefix and period languages of ω-languages, defined the power language of L:

$$\mathrm{Pow}(L) = \{x^k : x \in L, k \geq 1\}.$$

We say that L is *power-closed* if $L = \mathrm{Pow}(L)$. With each regular ω-language, Calbrix and Nivat associate two regular languages, the prefix language and the period language. The latter language is power-closed.

Calbrix [3] posed the problem of characterizing for which regular languages L, the power language $\mathrm{Pow}(L)$ is also regular. The problem was solved for unary languages by Cachat [2], and partial results of Horváth, Leupold and Lischke [7] were followed by a complete solution by Fazekas [6]. Other related research includes Lischke [9], which considered the complexity of the language consisting of all roots of a given language, and Anderson, Rampersad, Santean and Shallit [1], which (among other results) consider the complexity of determining whether all words in a language are powers.

Calbrix and Nivat showed that a regular language is power-closed if and only if it can be written as a finite union

$$L = \bigcup_{i=1}^{N} L_i^+, \tag{1}$$

where all L_i are regular. Their proof is constructive: L_i are the congruence classes of the syntactic congruence of L. Since L is the union of the L_i, their proof gives an algorithm for deciding whether a regular language is power-closed.

The complexity of the algorithm depends on the number of congruence classes. If the language L is presented by an n-state deterministic finite automaton, then there can be as many as n^n congruence classes, and therefore the algorithm is $\mathsf{EXPTIME}$. This algorithm is explicitly mentioned by Fazekas [6].

We consider the problem of deciding whether a regular language, presented as a deterministic finite automaton, is power-closed. We improve on Calbrix and Nivat's method by giving a PSPACE algorithm. Complementing this result, we show that the problem is PSPACE-hard. This also shows that our algorithm is optimal.

Anderson et al. showed that it is PSPACE-complete to determine, given a deterministic finite automaton A and an integer k, whether the kth power of the language accepted by A is regular. Our result generalizes similarly: it is PSPACE-complete to determine, given a deterministic finite automaton A and an integer k, whether the language accepted by A is closed under taking kth powers. Anderson et al. prove the hardness part of their result using an old result of Kozen [8], whose proof is very similar to our PSPACE-hardness proof.

2 Definitions

A *deterministic finite automaton* (DFA for short) is given by a quadruple $A = \langle Q, q_0, F, \delta \rangle$, where Q is the set of states, q_0 is the initial state, F is the set of accepting states, and δ is the transition function. For simplicity, we assume that the DFA operates over the binary alphabet $\Sigma = \{0, 1\}$. The language accepted by the DFA is $L(A)$.

Fix some standard encoding of DFAs with the property that the encoding of a DFA with n states has length $\mathsf{poly}(n)$. PC is the language consisting of all DFAs A such that $L(A)$ is power-closed.

For $n \geq 1$, $[n] = \{1, \ldots, n\}$. For a function f on a set S, $f^{(k)}$ is the kth composition of f on itself. So $f^{(1)}(x) = f(x)$, $f^{(2)}(x) = f(f(x))$, and so on. The length of a string w is denoted $|w|$. The empty string is ϵ. We say that w is a *power* if $w = z^k$ for some word z and $k > 1$.

For $n \geq 1$ and $0 \leq x \leq 2^n - 1$, $B_n(x)$ is a string of length n which is the binary encoding of x. For a string w and $i \in [|w|]$, $\mathrm{bit}(w, i)$ is the ith bit of w. The first bit of w is $\mathrm{bit}(w, 1)$, and so on.

3 PSPACE algorithm

Our algorithm for deciding whether a regular language is power-closed uses the fact that the syntactic congruence has at most n^n congruence classes, where n is the size of the minimal DFA. This fact implies that if a language is not power-closed, then there is a counterexample of length at most n^n.

Lemma 3.1. *Let $A = \langle Q, q_0, F, \delta \rangle$ be a DFA with n states. If $L(A)$ is not power-closed then there is a word w of length at most n^n and $2 \leq k \leq n$ such that $w \in L(A)$ and $w^k \notin L(A)$.*

Proof. Let Λ be the set of congruence classes of the syntactic congruence of A. Each congruence class $L \in \Lambda$ has a representation

$$L = \{w : \forall q \in Q, \delta(q, w) = \lambda(q)\}, \quad \lambda \colon Q \to Q.$$

Since L is determined by λ, $|\Lambda| \leq n^n$. We can construct a DFA with set of states Λ that upon reading a word w, reaches the unique state $L \in \Lambda$ such that $w \in L$. This shows that each congruence class contains a word of length at most n^n.

It is easy to see that $L(A)$ is power-closed if and only if for all congruence classes L, the following property holds for their representing function λ: if $\lambda(q_0) \in F$ then $\lambda^{(k)}(q_0) \in F$ for all $k \geq 1$. Hence $L(A)$ is not power-closed if for some congruence class L with corresponding function λ it is true that $\lambda(q_0) \in F$ but $\lambda^{(k)}(q_0) \notin F$ for some $k > 1$. Since the domain of λ has size n, we can assume $k \leq n$. □

Theorem 3.2. *The language* PC *is in* PSPACE*.*

Proof. According to Savitch's theorem [10], $\mathsf{NPSPACE} = \mathsf{PSPACE}$. Therefore it is enough to give an $\mathsf{NPSPACE}$ algorithm for $\overline{\mathsf{PC}}$. Given a DFA $A = \langle Q, q_0, F, \delta \rangle$ with n states, the algorithm guesses a word w of length at most n^n, and calculates the function $\lambda \colon Q \to Q$ given by $\lambda(q) = \delta(q, w)$. This requires space $O(n \log n)$. It then verifies that $\lambda(q_0) \in F$ while $\lambda^{(k)}(q_0) \notin F$ for some $2 \leq k \leq n$. □

4 PSPACE hardness

In order to show that PC is PSPACE-hard, we will reduce a variant of TQBF to PC.

Definition 4.1. An instance of TQBF consists of a totally quantified Boolean formula

$$\psi = Q_1 x_1 \cdots Q_n x_n \phi(x_1, \ldots, x_n),$$

where $Q_i \in \{\forall, \exists\}$. The language TQBF consists of all true totally quantified Boolean formulas. The language cTQBF consists of all true totally quantified Boolean formulas in which ϕ is in conjunctive normal form.

Lemma 4.2. *The language* cTQBF *is* PSPACE*-complete.*

Proof sketch. It is well-known that TQBF is PSPACE-complete. Clearly $\mathsf{cTQBF} \in \mathsf{PSPACE}$, and it remains to reduce TQBF to cTQBF. Given a formula ϕ in the variables $x_1, \ldots, x_n$, one can construct a formula σ in conjunctive normal form with extra variables $\vec{y}$ such that $\phi \Leftrightarrow \exists \vec{y} \sigma \leftrightarrow \forall \vec{y} \sigma$. Moreover, σ can be constructed in size which is polynomial in the size of ϕ. Since

$$Q_1 x_1 \cdots Q_n x_n \phi \Leftrightarrow Q_1 x_1 \cdots Q_n x_n \exists \vec{y} \sigma,$$

this reduces TQBF to cTQBF. □

The general idea of the reduction is given by the following lemma, whose proof will occupy most of the section.

Lemma 4.3. *Let $\psi = Q_1 x_1 \cdots Q_n x_n \phi(x_1, \ldots, x_n)$ be an instance of* cTQBF*, where ϕ consists of m clauses. Let $p \geq 3n$ be prime. There is an algorithm running in time* $\mathsf{poly}(n, m, p)$ *which constructs a DFA A with the following properties:*

(a) There is a word $z_\psi \notin L(A)$ such that $z_\psi^p \in L(A)$, and furthermore z_ψ^p is the only power in $L(A)$.

(b) For some k, $\mathrm{bit}(z_\psi, k)$ is the truth value of ψ.

In the rest of the section, whenever we say "polysize", we mean an object whose size is $\mathsf{poly}(n, m, p)$. The main theorem of this section follows directly from the lemma.

Theorem 4.4. *The language* PC *is* PSPACE*-hard.*

Proof. We reduce cTQBF to PC. Let $\psi = Q_1 x_1 \cdots Q_n x_n \phi(x_1, \ldots, x_n)$ be an instance of cTQBF. Bertrand's postulate shows that there is a prime $p \geq 3n$ such that $p \leq 6n$. We find such a prime in time $\mathsf{poly}(n)$. Construct the polysize DFA A of Lemma 4.3. Using A, construct another polysize DFA B such that

$$L(B) = \overline{L(A) \cap \Sigma^{k-1} 0 \Sigma^*}.$$

We claim that $L(B)$ is power-closed if and only if ψ is true. Indeed, if ψ is false then $z_\psi \in L(B)$ while $z_\psi^p \notin L(B)$, and so $L(B)$ is not power-closed. Conversely, if $L(B)$ is not power-closed then there is a power $w = z^k \notin L(B)$ such that $z \in L(B)$. Since $w \in L(A)$, necessarily $w = z_\psi^p$. Since $\mathrm{bit}(w, k) = 0$, we conclude that ψ is false. $\square$

The idea behind the proof of Lemma 4.3 is that while a polysize DFA cannot recognize z_ψ using one pass, it can recognize it using multiple passes. We proceed to define z_ψ.

Definition 4.5. The word z_ψ is a concatenation $z_\psi = M_\psi y_\psi$ of $M_\psi = 10^{2n(m+1)}1$ and

$$y_\psi = B_n(0)^m v_{B_n(0)} \cdots B_n(2^n - 1)^m v_{B_n(2^n-1)},$$

and $v_{x_1 \ldots x_n}$ is defined as follows, for $i \in [n]$:

$$\mathrm{bit}(v_{x_1 \ldots x_n}, i) = \begin{cases} Q_i y_i \cdots Q_n y_n \phi(x_1, \ldots, x_{i-1}, y_i, \ldots, y_n) & \text{if } x_i = \cdots = x_n = 1, \\ 0 & \text{otherwise.} \end{cases}$$

We divide the string y_ψ into blocks of size n and superblocks of size $(m+1)n$.

The word M_ψ serves as a marker, and the actual data appears in y_ψ. We start by showing how to recognize y_ψ using $3n$ passes.

Lemma 4.6. *There are $3n$ efficiently constructible polysize DFAs $A_1, \ldots, A_{3n}$ such that*

$$\bigcap_{i=1}^{3n} L(A_i) = \{y_\psi\}.$$

Proof. For each $i \in [n]$ we will construct three DFAs A_i, A_{n+i}, A_{2n+i} which are in charge of checking the ith bit in each input block.

The DFA A_i accepts the language

$$(((\Sigma^{i-1}0\Sigma^{n-i})^m + (\Sigma^{i-1}1\Sigma^{n-i})^m)\Sigma^n)^*.$$

Together, the DFAs $A_1, \ldots, A_n$ verify that each superblock is of the form $B_n(x)^m v$.

The DFA A_{n+i} checks that the first superblock is of the form $B_n(0)^m v$, that the last superblock is of the form $B_n(2^n - 1)^m v$, and that any two consecutive superblocks conform to the pattern

$$\Sigma^{i-1}0(\Sigma^{n-i} \setminus 1^{n-i})\Sigma^{nm}\Sigma^{i-1}0\Sigma^{n-i}\Sigma^{nm} + \Sigma^{i-1}01^{n-1}\Sigma^{nm}\Sigma^{i-1}1\Sigma^{n-i}\Sigma^{nm}$$
$$+\Sigma^{i-1}1(\Sigma^{n-i} \setminus 1^{n-i})\Sigma^{nm}\Sigma^{i-1}1\Sigma^{n-i}\Sigma^{nm} + \Sigma^{i-1}11^{n-1}\Sigma^{nm}\Sigma^{i-1}0\Sigma^{n-i}\Sigma^{nm}.$$

In words, if two consecutive superblocks are of the form $x_1 \cdots x_n\Sigma^{nm}$ and $x'_1 \ldots x'_n\Sigma^{nm}$, then $x_{i+1} = \cdots x_n = 1$ implies $x'_i = \overline{x_i}$, and otherwise $x'_i = x_i$. Together, the DFAs $A_1, \ldots, A_{2n}$ verify the structure of y_ψ up to the value of $v_{B(0)}, \ldots, v_{B(2^n-1)}$.

Before defining A_{2n+i}, we define a helper function $o_i \colon \{0,1\}^2 \to \{0,1\}$:

$$o_i(b,c) = \begin{cases} b \wedge c & \text{if } Q_i = \forall, \\ b \vee c & \text{if } Q_i = \exists. \end{cases}$$

In order to define A_{2n+i}, consider first the case $i = n$. The automaton has one bit of memory b. It operates one superblock $x[1] \ldots x[m]v$ at a time (here $|x[1]| = \cdots = |x[m]| = |v| = n$). While reading $x[j]$, it computes the truth value c_j of the jth clause of ϕ. After reading $x[m]$, it calculates $c = c_1 \wedge \cdots \wedge c_m$. Now there are two cases: if $x[m]_n = 0$ then the automaton stores c at memory b and verifies that $\operatorname{bit}(v, n) = 0$. If $x[m]_n = 1$, it verifies that $\operatorname{bit}(v, n) = o_n(b, c)$.

The case $i < n$ is similar. Again, the automaton has one bit of memory b, and operates one superblock $x[1] \ldots x[m]v$ at a time. This time there are three cases. If $x[m]_{i+1} \ldots x[m]_n \neq 1^{n-i}$ then the automaton simply verifies that $\operatorname{bit}(v, i) = 0$. If $x[m]_{i+1} \ldots x[m]_n = 1^{n-i}$ then the automaton calculates $c = \operatorname{bit}(v, i+1)$. If $x[m]_i = 0$ then it stores c at memory b and verifies that $\operatorname{bit}(v, i) = 0$. If $x[m]_n = 1$, it verifies that $\operatorname{bit}(v, i) = o_i(b, c)$.

The DFAs $A_1, \ldots, A_{3n}$ together verify the exact structure and contents of y_ψ, and they are all polysize. □

The automata $A_1, \ldots, A_{3n}$ are pasted together using the following lemma.

Lemma 4.7. *Let $B_1, \ldots, B_q$ be DFAs of maximal size S, and M a word. There is an efficiently constructible DFA B whose size is $\mathsf{poly}(S, q, |M|)$, such that*

$$L(B) = \{w_1 M w_2 M \cdots M w_q : w_i \in L(B_i) \cap \overline{\Sigma^* M \Sigma^*}\}.$$

In other words, $L(B)$ consists of M-free words from $L(B_1), \dots, L(B_q)$, separated by M.

Proof. Let C be a DFA such that $L(C) = \{M\}$. For $i \in [q-1]$, construct a DFA C_i with a single accepting state such that

$$L(C_i) = \{wM : w_i \in L(B_i) \cap \overline{\Sigma^* M \Sigma^*}\}.$$

The DFA C_i keeps track of the current state of both B_i and C, as well as the state that B_i were in $|M|$ symbols ago. Also, construct a DFA C_q such that

$$L(C_q) = \{w : w_i \in L(B_q) \cap \overline{\Sigma^* M \Sigma^*}\}.$$

Finally, the required DFA B is constructed by taking the DFAs $C_1, \dots, C_q$, and for $i \in [q-1]$, identifying the accepting state of C_i and the initial state of C_{i+1}. □

We are now ready to prove Lemma 4.3.

Proof of Lemma 4.3. Let A_0 be a DFA accepting the language $\{\epsilon\}$, and let $A_1, \dots, A_{3n}$ be the DFAs constructed by Lemma 4.6. Construct a DFA A using Lemma 4.7 from $A_0, A_1, \dots, A_{3n}, A_1, \dots, A_{p-3n}$ (assuming $p \le 6n$), using $M = M_\psi$. It is easy to check that A is polysize, and it remains to verify the properties of A claimed in the lemma. For the second claim, the truth value of ψ is equal to $\mathrm{bit}(v_{1^n}, 1)$.

For the first claim, it is easy to check that y_ψ does not contain M_ψ, and therefore $z_\psi^p \in L(A)$ while $z_\psi \notin L(A)$. On the other hand, suppose $w = z^k \in L(A)$ is a power. If M_ψ appears l times in z then it appears kl times in w, hence $kl = p$. Since $k > 1$, we conclude that $k = p$ and $l = 1$. Also, z must be of the form $z = M_\psi y$. The definition of $L(A)$ implies that $y \in L(A_1) \cap \cdots \cap L(A_{3n})$, hence $y = y_\psi$ and $z = z_\psi$. □

5 Acknowledgments

This paper answers a question posed on `cstheory.stackexchange.com` by Vincenzo Ciancia. The proof of Theorem 4.4 was inspired by [5].

References

[1] Terry Anderson, Narad Rampersad, Nicolae Santean, and Jeffrey Shallit. Finite automata, palindromes, powers, and patterns. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications*, pages 52–63. 2008.

[2] Thierry Cachat. The power of one-letter rational languages. In *DLT'01*, pages 145–154, 2002.

[3] Hugues Calbrix. *Mots ultimement periodiques des langages rationnels de mots infinis.* PhD thesis, Université Denis Diderot-Paris VII, 1996.

[4] Hugues Calbrix and Maurice Nivat. Prefix and period languages of rational ω-languages. In *Developments in Language Theory 1995*, pages 341–349, 1996.

[5] Keith Ellul, Benny Krawetz, Jeffrey Shallit, and Ming-wei Wang. Regular expressions: new results and open problems. *J. Autom. Lang. Comb.*, 10(4):407–437, 2005.

[6] Szilrd Fazekas. Powers of regular languages. In *Developments in Language Theory*, volume 5583 of *LNCS*, pages 221–227. 2009.

[7] Sándor Horváth, Peter Leupold, and Gerhard Lischke. Roots and powers of regular languages. DLT'02, pages 220–230, 2003.

[8] Dexter Kozen. Lower bounds for natural proof systems. In *FOCS'77*, pages 254–266, 1977.

[9] Gerhard Lischke. The root of a language and its complexity. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa, editors, *Developments in Language Theory*, volume 2295 of *Lecture Notes in Computer Science*, pages 57–64. Springer Berlin / Heidelberg, 2002.

[10] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comp. Syst. Sci.*, 4(2):177–192, 1970.