THE FROBENIUS AND FACTOR UNIVERSALITY PROBLEMS
OF THE FREE MONOID ON A FINITE SET OF WORDS

MAKSYMILIAN MIKA AND MAREK SZYKULA

ABSTRACT. We settle complexity questions of two problems about the free monoid $L^*$ generated by a finite set $L$ of words over an alphabet $\Sigma$. The first one is the Frobenius monoid problem, which is the question whether for a given finite set of words $L$, the language $L^*$ is cofinite. The open question concerning its computational complexity was originally posed by Shallit and Xu in 2009. The second problem is whether $L^*$ is factor universal, which means that every word over $\Sigma$ is a factor of some word from $L^*$. It is related to the longstanding Restivo’s open question about the maximal length of the shortest words which are not factors of any word from $L^*$. We show that both problems are PSPACE-complete, which holds even if the alphabet is binary. Both solutions share a large part of the construction. As auxiliary tools, we introduce three immortality problems concerning some types of rewriting systems. Additionally, we exhibit families of sets $L$ that show exponential (in the sum of the lengths of words in $L$) worst-case lower bounds on the lengths related to both problems: the length of the longest words not in $L^*$ when $L^*$ is cofinite, and the length of the shortest words that are not a factor of any word in $L^*$. The second family essentially settles in the negative the Restivo’s conjecture and its weaker variations. Finally, we note upper bounds on the computation time and the length for both problems, which are exponential only in the length of the longest words in $L$.

KEYWORDS: cofinite language, complete set, factor universality, finite list of words, free monoid, immortality, Frobenius monoid, mortality, regular language, Restivo’s conjecture

1. Introduction

Given a finite set of words $L$ over an alphabet $\Sigma$, the language $L^*$ (Kleene star or free monoid) contains all finite strings built by concatenating any number of words from $L$. Languages of this form are fundamental in formal language theory. The case when $L$ is a finite set has been studied e.g. in the theory of codes [3], for their syntactic parameters [18], and for word membership algorithms [7]. In general, we can think about $L$ as a dictionary and $L^*$ as the language of all available phrases. One of the most basic questions that one could ask is whether $L$ generates all words over the alphabet $\Sigma$ of $L$. The answer is, however, trivial, because this is the case if and only if $L$ contains all single letters $a \in \Sigma$. Thus, more interesting relaxed universality properties are considered. In this paper, we consider and solve two famous problems of this kind.

1.1. Frobenius monoid problem. The classical Frobenius problem is the question, for a given finite set of positive integers $x_1, \ldots, x_k$, what is the largest integer $x$ that is not expressible as a non-negative linear combination of the given integers. An integer $x$ is expressible as a non-negative linear combination if there are integers $c_1, \ldots, c_k \geq 0$ such that $x = c_1x_1 + \ldots + c_kx_k$. In a decision version, we can ask whether the largest integer exists, i.e. whether the set of non-expressible positive integers is finite. It is well known that the answer is “yes” if and only if $\text{gcd}(x_1, \ldots, x_k) = 1$. 

Institute of Computer Science, University of Wroclaw, Wroclaw, Poland

E-mail addresses: mika.maksymilian@gmail.com, msz@cs.uni.wroc.pl
The Frobenius problem was extensively studied and found applications across many fields, e.g. to primitive sets of matrices [8], to the Shellsort algorithm [10], and to counting points in polytopes [1]. The problem of computing the largest non-expressible integer is NP-hard [14] when the integers are given in binary, and it can be solved polynomially if the number \( k \) of given integers is fixed [11].

A generalization of the Frobenius problem to the setting of free monoids was introduced by Kao, Shallit, and Xu [12]. Instead of a finite set of integers, we are given a finite set of words over some finite alphabet \( \Sigma \), and instead of multiplication, we have the usual word concatenation. The original question becomes whether all except a finite number of words can be expressed as a concatenation of the words from the given set. If \( L \) is our given finite language, then the problem is equivalent to deciding whether \( L^* \) is cofinite (i.e. the complement of \( L^* \) is finite).

**Problem 1.1** (Frobenius Problem for a Finite Set of Words). *Given a finite set of words \( L \) over an alphabet \( \Sigma \), is \( L^* \) cofinite?*

It is a simple observation that, if \( \Sigma \) is a unary alphabet, then Problem 1.1 is equivalent to the original Frobenius problem on integers.

**Example 1.1.** The language \( L = \{ 000, 0000 \} \) generates the cofinite language \( L^* \), since \( \gcd(3, 5) = 1 \) so it includes all words longer than \( 3 \cdot 5 - 3 - 5 = 7 \).

**Example 1.2.** For the language \( L = \{ 0, 01, 10, 11 \} \), the words in \( L^* \) are:

\[
0, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, \ldots .
\]

We can see that word \( 111 \notin L^* \), and actually every word of the form \( 111(111111)^* \) does not belong to \( L^* \). However, if we add \( 111 \) to \( L \), the answer becomes that \( L^* \) is cofinite. Since we can build all words of length 2 and 3 over the alphabet \( \{ 0, 1 \} \), and \( \gcd(2, 3) = 1 \), we know that \( L^* \) must contain all long enough words.

Kao, Shallit, and Xu [12, 21] showed that, in particular, if \( L^* \) is cofinite, then the longest non-expressible words can be exponentially long in the length of the longest words from \( L \). This is in contrast with the classical Frobenius problem, where the largest non-expressible integer is bounded quadratically in the largest given integer [5]. In 2009, Shallit and Xu posed the open question about the computational complexity of determining whether \( L^* \) is cofinite [21]. They proved that it is NP-hard and in \( \text{PSPACE} \) when \( L \) is given as a regular expression. The question also appears on the Shallit’s list of open problems [19].

The problem can be seen as *almost universality* of the language \( L^* \). It models a situation where we consider whether a given dictionary is sufficient to generate all long enough sequences. For example, consider sound synthesis. A common method there is unit selection, which is generating the sound by concatenating various recorded sequences [20]. In a simple setting, if we do not care about short sequences (as for them we anyway require all one-sound samples), testing whether a given sound bank is strong enough to generate everything is equivalent to the Frobenius monoid problem.

### 1.2. Factor universality problem.

A word \( u \in \Sigma^* \) is a *factor* of a word \( w \in \Sigma^* \) if \( vu = w \) for some words \( v, v' \in \Sigma^* \).

**Problem 1.2** (Factor Universality for a Finite Set of Words). *Given a finite set of words \( L \) over an alphabet \( \Sigma \), is every word over \( \Sigma \) a factor of a word from \( L^* \)?*

Sets \( L \) such that the language of all factors of the words in \( L^* \) is universal are one of the basic concepts in the theory of codes [3] Section 1.5]. They are called complete sets of words, and words that are factors of some word in \( L^* \) are called completable.
Example 1.3. The set \( L = \{01, 10, 11, 000\} \) is not complete, since word 10010001 is not completable. If we want to create 1 in the middle, we have to use either 10 or 01. In each case, one of the adjacent 0s is also consumed, so we cannot use word 000.

Example 1.4. The set \( L = \{00, 01, 10, 11\} \) is complete, because every binary sequence of even length is in \( L^* \). We can construct every odd length binary sequence by removing the first letter of a suitable even length sequence.

The question about the length of the shortest incompletable words was posed in 1981 by Restivo [16], who conjectured that if a set \( L \) is not complete, then the shortest incompletable words have length at most \( 2k^2 \), where \( k \) is the length of the longest words in \( L \). The conjecture in this form turned out to be false [9] (\( 5k^2 - O(k) \) is a lower bound), but the relaxed question whether there is a quadratic upper bound remained open and became one of the longstanding unsolved problems in automata theory.

There is a trivial exponential upper bound in the sum of the lengths of words in \( L \). A sophisticated experimental research [17] suggested that the tight upper bound is unlikely quadratic and may be exponential. However, a polynomial upper bound was derived for the subclass of sets \( L \) (codes) that guarantee a unique factorization of any word into the words from \( L \) [12].

The computational complexity was also an open question. The problem of factor universality in a more general setting, where instead of \( L^* \), we are given an arbitrary language specified by an NFA, is PSPACE-complete [15]. Both computational complexity question and the tight upper bound question appears also as one of Berstel and Perrin’s research problems [3, Research problems] and on the Shallit’s list [19].

The problem itself has been connected with a number of different problems, e.g., testing if all bi-infinite words can be generated by the given list of words [15], the famous Černý conjecture [6], and the matrix mortality problem [13] in a restricted setting.

1.3. Contribution. We provide the answers to the questions about the computational complexity of Problem 1.1 and Problem 1.2. Both solutions share a common construction. Therefore, it is probably applicable to some other problems concerning the free monoid on a finite set of words.

Problem 1.1 is PSPACE-complete. The answer can be quite surprising because the problem is equally hard when \( L \) is represented by a popular more succinct representation, i.e. a DFA, a regular expression, or an NFA. In [12], there were shown finite languages \( L \) such that the longest words not present in the generated cofinite language \( L^* \) are of exponential length in the length of the longest words in \( L \). However, the number of words in \( L \) is also exponential in these examples, thus they do not provide an exponential lower bound in terms of the size of the input \( L \). Here, we additionally show stronger examples, where the longest not present words in cofinite \( L^* \) are of exponential length in the sum of the lengths of the words.

To make the reduction feasible, we construct it in several steps through different problems. We introduce three PSPACE-complete problems concerning the immortality of a rewriting system of some type. The immortality question is whether there exists any configuration such that starting from it, we can apply rules infinitely long. This is in contrast with the usual settings where the initial configuration is given. It turns out that the existence of an arbitrary cycle is an essential property for Problem 1.1.

Then we show that Problem 1.2 is also PSPACE-complete. We use a similar construction to that of the previous problem. As a corollary, we exhibit a family of sets \( L \) of binary words whose minimal incompletable words are of exponential length in the length of the longest words in \( L \) or
in the sum of the lengths of the words in $L$. This settles in the negative all weak variations of the Restivo’s conjecture and essentially closes the problem.

In both cases, the complexity remains the same if the alphabet is binary.

Finally, we note that both problems can be solved in exponential time in the length of the longest word in $L$ while polynomial in the sum of the lengths of words in $L$. This means that they can be effectively solved when the given set is dense, so the maximal length of words is e.g. logarithmic in the number of words.

We conclude that for a list $L$ of words over a fixed alphabet, $2^{O(||L||_{\text{max}})}$, where $||L||_{\text{max}}$ is the length of the longest words in $L$, is a tight upper bound on both the length of the longest word not in $L^*$ when $L^*$ is cofinite and the length of the shortest incompletable words when $L^*$ is not factor universal. Furthermore, the length $2^{\Theta(\sqrt[5]{||L||_{\text{sum}}})}$, where $||L||_{\text{sum}}$ is the sum of the lengths of words in $L$, is attainable.

2. Mortality of bounded rewriting systems

We consider three types of rewriting systems and consider their immortality as auxiliary problems.

2.1. Linearly bounded Turing machine. We start from the immortality problem of a deterministic Turing machine where the tape has a linear length. It is the question whether for a given Turing machine there exists any configuration such that the machine does not halt, which, in the bounded tape case, is equivalent to the question whether there exists a configuration that is repeated periodically. The mortality problems for Turing machines were studied before with an unbounded tape [4], where the problem is undecidable, and also for specific machine types [2].

First, we need to define a slight variation of the classical Turing machine. A deterministic semi-Turing machine is $M = (Q_M, \Sigma_M, \delta_M)$, where $Q_M$ is a set of states, $\Sigma_M$ is a tape alphabet, and $\delta: Q_M \times \Sigma_M \rightarrow Q_M \times \Sigma_M \times \{L, R\}$ is a partial function called the transition function. A configuration of length $\ell$ of a semi-Turing machine is a tuple $C \in Q_M \times \{1, \ldots, \ell\} \times \Sigma^\ell$. We assume that the machine stops when it tries to go outside the tape (to the left from the first position or to the right from the $\ell$-th position of the head).

Adding an initial configuration to a semi-Turing machine yields a classical Turing machine. Determining whether a given semi-Turing machine from a given configuration reaches any configuration with a distinguished accepting state is a canonical PSPACE-complete problem. The hardness also holds when the tape alphabet is binary. Furthermore, we can simplify the canonical problem and assume that the initial configuration has the tape filled with zeros and the machine head starts at the first (leftmost) cell.

It is a folklore result that the problem of determining whether, from a given configuration, a given semi-Turing Machine stops or cycles infinitely while staying within the tape boundaries, is also PSPACE-complete. For this, we just add a counter to the machine from the canonical problem, which is incremented after each step, and make it looping infinitely in the accepting state.

Now, we consider the problem where, instead of a given configuration, only the length $\ell$ is given and the question is about the existence of a periodic configuration. We assume $\ell$ is given in unary, so the length of the input is larger than the tape.

**Problem 2.1** (Immortality of a Turing machine with a linearly bounded tape). *Given a deterministic semi-Turing Machine $M$ and the length of the tape $\ell$ written in unary, is there a configuration $C$ of length $\ell$ such that $M$ repeats $C$ infinitely many times?*
This immortality problem is easily seen to be in \textsc{NPSPACE} hence in \textsc{PSPACE} since we can guess \( C \) and simulate \( M \) for at most \(|Q_M| \cdot \ell \cdot |\Sigma|^\ell\) steps, which is the number of all configurations. For \textsc{PSPACE}-hardness, since the configuration \( C \) that is periodic is arbitrary, in the reduction from a canonical problem we cannot add any loop or a simple cycle, and adding any counter implies that it can be arbitrarily initialized in \( C \).

A similar problem was considered in [2, Lemma 3.10] for Turing machines with a linear queue as a tape and with an additional counter register that resets the machine to a fixed configuration. It is also \textsc{PSPACE}-complete. We also use the idea of a counter and resetting the configuration, but since we deal with the classical Turing machines, we need to keep it stored on the regular tape.

**Theorem 2.1.** Problem [2.1] is \textsc{PSPACE}-hard, even if the tape alphabet is binary.

**Proof.** We reduce from the canonical \textsc{PSPACE}-complete problem. Let \( M = (Q_M, \{(0,1)\}, \delta_M, q_0, q_f) \) be a given semi-Turing machine, \( \ell \) be a polynomial (in the size of \( M \)) tape length, \( q_0 \) be an initial state, and \( q_f \in Q_M \) be a distinguished accepting state. In the initial configuration \( I \), the machine is in state \( q_0 \), the head is at the first cell, and the tape is filled with zeros. The question whether \( M \) from the initial configuration reaches a configuration with \( q_f \) is \textsc{PSPACE}-complete.

We construct a semi-Turing machine \( M' \) with a tape length \( \ell' \) such that there exists a configuration \( C \) for \( M' \) which is repeated cyclically by \( M' \) if and only if \( M \) from \( I \) reaches a configuration with \( q_f \). Furthermore, the size of \( M' \) and the tape length \( \ell' \) will be polynomial in the size of \( M \).

The general idea is as follows. We let \( M' \) simulate \( M \). Additionally, \( M' \) has a counter on the tape that is incremented for every step performed by the original machine. When the counter reaches its maximum, \( M' \) stops. It also stops when \( M \) stops, either by not having the next transition or going outside its tape. However, if \( M \) gets into the accepting state \( q_f \), then \( M' \) resets to the configuration \( I' \) that contains the initial configuration \( I \) for \( M \) extended by the counter which is set to zero. Clearly, if \( M \) reaches \( q_f \), then \( M' \) repeats cyclically \( I' \). Otherwise, from every configuration for \( M' \), the machine eventually overflows the counter and hence stops.

Technically, we implement two procedures in \( M' \), one performing one simulation step of \( M \) and the second one resetting the configuration. We always store the current head position in the machine state, which allows safely going from one end of the tape to the other and avoiding confusion between the tape of \( M \) and the counter. If the stored position does not agree with the real position, the machine will try to go outside the tape bounds thus will halt. Hence a potentially periodic configuration must contain equal positions. Additionally, in the second procedure, we store the head position of \( M \) in the state, so we can find its place after a simulation step.

The tape alphabet of \( M' \) is \{0, 1\}, the same as that of \( M \). We set the length of the tape \( \ell' = \ell + \ell_{\text{counter}} \), where \( \ell_{\text{counter}} = \ell + \lceil \log_2 \ell \rceil + \lceil \log_2 |Q_M| \rceil \). There are \( \ell \) cells devoted for the tape of \( M \) and \( \ell_{\text{counter}} \) cells for the counter. The counter allows storing in binary all integers from 0 to an upper bound (minus one) on the number \( 2^\ell \cdot \ell \cdot |Q_M| \) of all configurations of \( M \). We will require that during an infinite computation, our tape will store the tape content of \( M \) on the left part and the counter on the second part.

Now, we implement the two procedures in \( M' \). Between the procedures, \( M \) is assumed to be in a state \( p_{q,i} \), where \( q \in Q_M \) and \( i \in \{1, \ldots, \ell\} \). This state stores the current state of \( M \) together with the position on the tape of \( M \), which must agree with the position on the tape of \( M' \). The configuration \( I' \) contains all zeros on the tape, the head at the leftmost cell, and the state \( p_{q_0,1} \).

The first procedure is constructed for every transition \( \delta_M(q, s) = (q', s', d') \), where \( q \in Q_M \setminus \{q_f\} \) and \( s \in \{0, 1\} \). It contains the following steps: first, go to the rightmost cell on the tape \( (\ell' - i) \) steps); then go to the leftmost cell on the tape and increment the counter by the way \( (\ell' - i) \) steps;
and then go back to the original $i$-the position ($i - 1$ steps). Finally, the original transition is performed. At all times, the current state stores the supposed position $i$ on the tape. Note that the procedure fails if the current state is not of the form of $p_{q,i}$ for some $q$ because then the machine will try to go outside the tape either on the right or the left side.

The second procedure starts when the current state is a state $p_{q,f,i}$. It has a similar but simpler construction. First, it goes to the rightmost cell and then goes to the leftmost cell while zeroing the tape.

The states of $M$ are named by $p$ with some subscripts and an optional fixed superscript. The set of states is derived from the transitions defined as follows. For every $q \in Q_M \setminus \{q_f\}$ and $i \in \{1, \ldots, \ell\}$, we implement the first procedure:

- For $s \in \{0, 1\}$: $\delta(p_{q,i}, s) = (p_{q,i,i+1,1}, s, R)$; start the first procedure and move one step right.
- For $j \in \{i + 1, \ldots, \ell' - 1\}$ and $s \in \{0, 1\}$: $\delta(p_{q,i,j}, s) = (p_{q,i,j+1,1}, s, R)$; go to the right.
- For $s \in \{0, 1\}$: $\delta(p_{q,i,\ell'-1}, s) = (p_{q,i,\ell',1}, s, R)$; the last step when moving right.
- For $j \in \{\ell' - \ell + 2, \ldots, \ell'\}$: $\delta(p_{q,i,j}, 1) = (p_{q,i,j-1,0}, 0, L)$; in the counter, switch all consecutive ones from the right to zeros.
- For $j \in \{\ell' - \ell + 1, \ldots, \ell'\}$: $\delta(p_{q,i,j}, 0) = (p_{q,i,j-1,1}, 1, L)$; in the counter, switch the first found zero to one.
- For $j \in \{3, \ldots, \ell' - 1\}$ and $s \in \{0, 1\}$: $\delta(p_{q,i,j}, s) = (p_{q,i,j-1,1}, s, L)$; move to the left.
- For $s \in \{0, 1\}$: $\delta(p_{q,i,2}, s) = (p_{q,i,1}, s, L)$; the last step when moving left.
- For $j \in \{1, \ldots, i - 1\}$ and $s \in \{0, 1\}$: $\delta(p_{q,i,j}, s) = (p_{q,i,j+1,1}, s, R)$; move right back to the $i$-th position.
- If $\delta_M(q, s) = (q', s', L)$ and $i > 1$, then $\delta(p_{q,i,i}, 1) = (p_{q',i-1,1}, s', L)$; perform the original transition with left direction.
- If $\delta_M(q, s) = (q', s', R)$ and $i < \ell$, then $\delta(p_{q,i,i}, 0) = (p_{q',i+1,1}, s', R)$; perform the original transition with right direction.

For every $i \in \{1, \ldots, \ell\}$, we implement the second procedure:

- For $s \in \{0, 1\}$: $\delta(p_{q,i}, s) = (p_{q,i+1,0}, 0, R)$; start the second procedure and move one step right.
- For $j \in \{i + 1, \ldots, \ell' - 1\}$ and $s \in \{0, 1\}$: $\delta(p_{q,i,j}, s) = (p_{q,i,j+1,0}, 0, R)$; go to the right.
- $\delta(p_{q,i-1}, s) = (p_{q,i}, 0, R)$; the last step when moving right.
- For $j \in \{2, \ldots, \ell'\}$ and $s \in \{0, 1\}$: $\delta(p_{q,i,j}, 0) = (p_{q,i,j-1}, 0, L)$; go to the left and fill the tape with zeros.
- For $s \in \{0, 1\}$: $\delta(p_{q,i}, s) = (p_{q,i}, 0, 0, L)$; the last step of filling, get into the configuration $I'$.

The number of introduced states is polynomial in $|Q_M|$ and $\ell$, thus the size of $M'$ is polynomial in the size of $M$. To observe correctness, note that if the machine is at a state $p_{q,i}$ but the head position is different than $i$, then the machine stops. The machine cannot cycle inside a single call of a procedure since their transitions form an acyclic graph. Since the counter is incremented every time by the first procedure, $M'$ cycles if and only if the second procedure, thus the zero configuration $I'$, is repeated. This is the case when $M$ reaches $q_f$ from $I$.

2.2. Positional rewriting. In the next step, we simplify the immortality problem of a Turing machine, by reducing it to an immortality problem of a simple rewriting system. There, a configuration of the Turing machine is simplified to a binary sequence of a fixed length. A rule in this system just checks some positions for 0 or 1, and it sets some positions to be 0 or 1 in contrast to a Turing machine, where a rule checks and sets only one position, here a rule checks and sets
arbitrary subsets of fixed positions, but we do not need to bother about the head position nor the current state.

**Definition 2.2.** A positional rewriting system is a pair \((\ell, R)\), where \(\ell\) is a positive integer and \(R\) is a finite set of rules. A rule in \((c, s) \in R\) consists of a checking function \(c: \{1, \ldots, \ell\} \to \{0, 1, *\}\) and a setting function \(s: \{1, \ldots, \ell\} \to \{0, 1, *\}\).

Given a positional rewriting system and a binary sequence \(T = t_1 \ldots t_\ell \in \{0, 1\}^\ell\), a rule \((c, s)\) is *legal* if for every position \(i \in \{1, \ldots, \ell\}\) there is either \(c(i) = t_i\) or \(c(i) = *\). The resulted sequence from applying a legal rule is \(T \cdot s = t'_1 \ldots t'_\ell\), where every \(t'_i = s(i)\) if \(s(i) \in \{0, 1\}\) and \(t'_i = t_i\) otherwise.

**Problem 2.3** (Improbability of Positional Rewriting). Given a positional rewriting system \((\ell, R)\) where \(\ell\) is given in unary, does there exist a sequence \(T \in \{0, 1\}^\ell\) and a non-empty sequence of rules \(r_1, \ldots, r_k\) that are legal when applied consecutively and yield \(T\), i.e. \(T \cdot r_1 \cdot \ldots \cdot r_k = T\)?

**Proposition 2.2.** Problem 2.3 is PSPACE-complete.

**Proof.** We can solve the problem in NPSPACE as usual. We guess \(T\) and \(k\) (at most exponential), and then we guess and apply rules, storing only the current binary sequence.

To show PSPACE-hardness, we reduce from Problem 2.1. Let \(M = (Q_M, \{0, 1\}, \delta_M)\) be the machine and \(\ell\) be the length of the tape. We encode a configuration as a binary sequence of length \(\ell' = \lceil \log_2 |Q_M| + \log_2 \ell \rceil + \ell\). The first \(\lfloor \log_2 |Q_M| + \log_2 \ell \rfloor\) positions store the current state and the tape position encoded as a binary number. For every rule of \(M\), we create \(\ell - 1\) rules of the rewriting system, for every tape position where the machine’s rule can be applied. The constructed rule just checks if the encoded state together with the position agrees with the rule, checks the appropriate bit on the encoded tape, sets the new encoded state with the position, and sets the appropriate bit on the tape as well.

Clearly, if there exists a periodic configuration for Problem 2.1 then also its encoded binary sequence works for Problem 2.3. For the other direction, note that the binary sequence may not represent a proper machine’s configuration since \(2^{\lfloor \log_2 |Q_M| \cdot \ell \rfloor}\) may be larger than \(|Q_M| \cdot \ell\). For such sequences, however, we cannot apply any rule. \(\square\)

### 2.3. Set Rewriting

In the next step, we reduce to the immortality problem of another rewriting system, which is more general. Now, we operate on a finite set \(P\) and a configuration is any subset \(S\) of \(P\). Given a subset, a rule transforms each of the elements of \(S\) into predefined subsets of \(P\) by the rule, and the resulted subset is the union of them. A rule may be forbidden for subsets containing specific elements.

**Definition 2.4.** A set rewriting system is a pair \((P, R)\), where \(P\) is a finite non-empty set of items and \(R\) is a finite non-empty set of rules. A rule is a function \(r: P \to 2^P \cup \{\bot\}\).

Given a set rewriting system and a subset \(S \subseteq P\), a rule \(r\) is *legal* if \(\bot \notin r(S)\) (i.e. there is no \(s \in S\) such that \(r(s) = \bot\)). The resulted subset from applying a legal rule \(r\) to \(S\) is \(S \cdot r = \bigcup_{s \in S} r(s)\).

In contrast to the previous rewriting systems, every set rewriting system contains a trivial cycle which is the loop on the empty set. Therefore, we are interested only in non-trivial cycles and add another condition which ensures that the empty set is reachable only from itself; then we can consider only non-empty subsets. A set rewriting system is *non-emptying* if for every element \(p \in P\) and every rule \(r \in R\), we do not have \(r(p) = \emptyset\). Note that it implies that for every non-empty subset \(S\) and a rule \(r\), either \(S \cdot r \neq \emptyset\) or \(r\) is illegal for \(S\).
Problem 2.5 (Immortality of Set Rewriting). Given a non-emptying set rewriting system \((P, R)\), is there a non-empty subset \(S \subseteq P\) and a non-empty sequence of rules \(r_1, \ldots, r_k\) that are legal when applied to \(S\) consecutively and yield \(S\), i.e. \(S \cdot r_1 \cdot \ldots \cdot r_k = S\)？

Theorem 2.3. Problem 2.5 is PSPACE-complete.

Proof. Solving the problem in NPSPACE can be done as in the proof of Proposition 2.2.

To show PSPACE-hardness, we reduce from Problem 2.3. Given a positional rewriting system \((\ell, R)\), we create a set rewriting system \((P, R')\). We define

\[
P = \{p^i_b \mid i \in \{1, \ldots, \ell\} \land b \in \{0, 1\}\} \cup \{c^i_j \mid i \in \{1, \ldots, \ell\} \land j \in \{1, \ldots, \ell\}\}.
\]

The \(p^i_b\) elements will be used for simulating the positional rewriting system. The upper indices are called positions, which correspond to the positions in \(R\). The presence of elements \(p^i_0\) and \(p^i_1\) defines the value(s) at the \(i\)-th position. The \(c^i_j\) elements will form counters, which will enforce the correctness of the simulation. An \(i\)-th counter consists of the elements \(c^i_1, \ldots, c^i_{\ell}\). Applying any rule will add one element to every counter, with the exception of one selected counter by the rule, which is emptied. Simultaneously when an \(i\)-th counter is emptied, a value is set at the \(i\)-th position. Hence, a counter that contains \(c^i_j\) will allow at most \(\ell\) steps before setting a value at the position. Altogether, the counters will make sure that after applying any long enough sequence of rules, at every position there will be at least one value.

For every rule \(r = (c, s) \in R\), we create \(2\ell\) rules of the form \(r'_{i,b}\), for all \(i \in \{1, \ldots, \ell\}\) and \(b \in \{0, 1\}\). They are defined as follows.

Let \(C_1 = \{c^i_j \mid j \in \{1, \ldots, \ell\}\}\); this is the set of the first elements in each counter. For all \(k \in \{1, 2, \ldots, \ell\}\) and \(x \in \{0, 1\}\) we define:

\[
r'_{i,b}(p^k_x) = \begin{cases} 
\bot, & \text{if } c(k) = 1 - x; \\
\{p^k_x\} \cup C_1, & \text{if } (c(k) = x \lor c(k) = *) \land s(k) = *; \\
\{p^k_0\} \cup C_1, & \text{if } (c(k) = x \lor c(k) = *) \land s(k) = 0; \\
\{p^k_1\} \cup C_1, & \text{if } (c(k) = x \lor c(k) = *) \land s(k) = 1.
\end{cases}
\]

This part will simulate the original positional rewriting system. It also initializes all counters so that they have at least one element. Note that a rule is illegal if some position has a wrong value in terms of the checking function \(c\).

For all \(k \in \{1, 2, \ldots, \ell\}\) and \(y \in \{1, 2, \ldots, \ell - 1\}\) we define:

\[
r'_{i,b}(c^k_y) = \begin{cases} 
\{p^k_y\}, & \text{if } i = k; \\
\{p^k_y, c^k_{y+1}\}, & \text{if } i \neq k.
\end{cases}
\]

This part adds an element to every counter, except the \(i\)-th counter that is reset. Simultaneously, the value \(b\) is set at the \(i\)-th position.

Finally, for all \(k \in \{1, 2, \ldots, \ell\}\) we define

\[
r'_{i,b}(c^k_\ell) = \begin{cases} 
\{p^k_\ell\}, & \text{if } i = k \\
\bot, & \text{if } i \neq k.
\end{cases}
\]

This ensures that one cannot overflow any counter. It follows that one must set each position at least once when applying every \(\ell\) consecutive rules.

The construction allows a proper simulation of the positional rewriting system. If there exists a cycle for Problem 2.3 then also its embedding works for Problem 2.5. We just need to repetitively
reset each \(i\)-th counter after exactly \(\ell\) steps, while setting the same value at the \(i\)-th position that is already there in the embedding. For the converse, we observe that after \(\ell + 2\) steps every position will have a value: after applying any rule we have a value in at least one position, then we must have \(C_0\) in the subset, and to apply the next \(\ell\) rules every counter must be reset. Thus our periodic set must have either one or two values at every position. We can find a binary sequence for the positional rewriting system, by picking the single value or any of the values in the case of two, and the periodic sequence of rules \(r\) obtained from the periodic sequence \(r_i^{i,b}\). \(\square\)

By the following observation, for Problem 2.5 it is enough to consider only singleton subsets \(S\), from which we start applying rules to find a cycle (but the singleton does not necessarily occur in a cycle).

**Lemma 2.4.** If a rule \(r\) is legal for a subset \(S \subseteq P\), then it is also legal for every subset \(S' \subseteq S\) and \(S' \cdot r \subseteq S \cdot r\).

A similar property is essential for Problem 1.1, because if a word \(wu \notin L^*\) for a word \(w \in L^*\), then also suffix \(u \notin L^*\).

### 3. The Frobenius monoid problem

Before we go for PSPACE-hardness, we note the known result about PSPACE-membership.

**Proposition 3.1** ([21]). **Problem** 1.1 is in PSPACE.

**Proof.** If \(L^*\) is cofinite, then the longest words not in \(L\) have at most exponential length [12]. Otherwise, the length of such words is unbounded. Thus, we can construct an NFA recognizing \(L^*\) and verify in NPSPACE whether there exists a longer word that is not accepted [21]. \(\square\)

We reduce from Immortality of Set Rewriting (Problem 2.5) to Frobenius Problem for a Finite Set of Words (Problem 1.1).

#### 3.1. The DFA construction

In the first step, we reduce to the case when \(L\) is specified as a DFA instead of a list of words.

We get a non-emptying set rewriting system \((P, R)\). Without loss of generality, we assume that the set \(P = \{p_1, p_2, \ldots, p_\ell\}\) and the rules \(R = \{r_1, r_2, \ldots, r_m\}\).

We construct a DFA \(A = (Q_A, \Sigma, \delta, q_0, F)\) such that \(L^*\) is not cofinite, where \(L\) is the language recognized by \(A\), if and only if there exists a non-empty subset \(S \subseteq P\) and a non-empty sequence of rules \(r_1, \ldots, r_k\) such that \(S \cdot r_1 \cdot \ldots \cdot r_k = S\). Our reduction will be polynomial in \(|P| + |R|\). The number and the lengths of the words in \(L\) will be polynomial, which will allow further polynomial reduction to the case of a list of words.

The alphabet of \(A\) is \(\Sigma = R \cup \{\alpha\}\). The letters from \(R\) are the rule letters. The set of states \(Q_A\) is the disjoint sum of the following sets:

- \(\{q_0\}\): the initial state.
- \(Q_P = P\): the set rewriting elements.
- \(Q_F = \{f_x \mid x \in \{0, 1, \ldots, \ell\}\}\): the forcing states.
- \(\{s_x^i,j \mid i \in \{1, 2, \ldots, \ell\} \land j \in \{1, 2, \ldots, m\} \land x \in \{\ell, \ell - 1, \ldots, 1\} \land r_j(p_i) \neq \perp\}\): the setting states.
- \(\{q_g\}\): the guard state.
- \(\{q_s\}\): the sink state.
The transition function and the final states will be defined later, after explaining the overall construction idea.

We use a standard NFA construction recognizing the Kleene star of a language specified by a DFA. Let $\mathcal{A}^* = (Q_\mathcal{A}^*, \Sigma, \delta_\mathcal{A}^*, q_0, F_\mathcal{A}^*)$ be the NFA obtained from $\mathcal{A}$ as follows. The set of states $Q_\mathcal{A}^*$ is $Q_\mathcal{A} \setminus \{q_0\}$; we remove the sink state since it is represented by the empty subset of states. We construct the extended transition function $\delta_\mathcal{A}^*: 2^{Q_\mathcal{A}^*} \times \Sigma^* \rightarrow 2^{Q_\mathcal{A}^*}$ from $\delta$ by adding $\varepsilon$-transitions from every final state to the initial state $q_0$ and removing transitions to the sink state. We assume that it is closed under $\varepsilon$-transitions, i.e. for $C \subseteq Q_\mathcal{A}^*$ and $w \in \Sigma^*$, $\delta_\mathcal{A}^*(C, w)$ is the set of all states reachable from a state in $C$ through a path labeled by $w$ interleaved with any $\varepsilon$-transitions. We say that $\delta_\mathcal{A}^*(C, w)$ is the set of active states after applying $w$ to $C$. The set of final states $F_\mathcal{A}^*$ is $F \cup \{q_0\}$; we can make $q_0$ final in our NFA construction, since the DFA is non-returning, i.e. there is no non-empty word $w$ such that $\delta(q_0, w) = q_0$ in the DFA. It is well known that the constructed NFA recognizes the language $L^*$ (see e.g. [22]).

A word $w \in \Sigma^*$ is irrevocably accepted if for every $u \in \Sigma^*$, the word $wu$ belongs to $L^*$. A word $w$ is simulating for a subset $S \subseteq Q_P$ if it is of the form $r_1 \alpha^\ell r_2 \alpha^r \ldots r_k \alpha^f$ and the sequence of the rules $r_1, r_2, \ldots, r_k$ in $w$ is legal for $S$.

A word $w \in \Sigma^*$ is $f_0$-omitting for a subset $C$ if there is no prefix $u$ of $w$ such that $f_0 \in \delta_\mathcal{A}^*(C, u)$. It is simply $f_0$-omitting if it is $f_0$-omitting for subset $\{q_0\}$.

Now, we explain the idea of the construction. We will have the property that whenever the word does not follow the simulating pattern, it will not be $f_0$-omitting. When this happens, some forcing state will be always active and the word will be irrevocably accepted, which means that all its extensions are in $L^*$. The forcing states will be responsible for this property of $f_0$. On the other hand, words following the simulating pattern will be $f_0$-omitting and not irrevocably accepted. Thus, if there will be infinitely many such simulating words, which is equivalent to the immortality of the set rewriting system, then infinitely many words will not be in the language.

States $Q_P$, together with the initial state $q_0$, form a chain on the transition of the letter $\alpha$, which is ended by $f_0$, i.e. $q_0 \xrightarrow{\alpha} p_1 \xrightarrow{\alpha} \ldots \xrightarrow{\alpha} p_\ell \xrightarrow{\alpha} f_0$. A subset of active states $S \subseteq Q_P$ corresponds to the current subset of elements in our set rewriting system. At this point, by applying a rule letter $r_j$, which corresponds to applying the rule $r_j$, the states from $Q_P$ are mapped into the setting states. If the rule is not legal for some element, that state in $S$ is mapped directly to $f_0$ instead. The setting states form chains $s_1^j, \ldots, s_{q_j}^j$ on letter $\alpha$, for every rule $r_j$ and every element $p_i \in Q_P$. Each such a chain has its final states defined according to the action of the rule $r_j$ for the element $p_i$. When $s_1^j$ becomes active, one must apply the word $\alpha^f$ in order to avoid $f_0$. The setting states that are final in the chain activate $q_0$ at some point, which is then mapped by the action of the remaining $\alpha$ letters to the right state of $Q_P$. One cannot apply more than $\ell$ letters $\alpha$ in such a simulation step because of the guard state $q_0$, which is at the end of every setting chain. The guard state becomes active after $\alpha^f$ applied for any non-empty $S$; it allows transitions only on rule letters, which map it to the empty subset (to the sink state in the DFA).

Therefore, if in the set rewriting system one has $S \subseteq Q_P$ and applies a sequence of rules that results in $S' = S \cdot r_{i_1} \cdot \ldots \cdot r_{i_k}$, then this corresponds to applying the word $r_{i_1} \alpha^\ell \ldots r_{i_k} \alpha^\ell$, which is a simulating word for $S$.

A special case occurs at the beginning when the subset of active states is $\{q_0\}$. Since no other states (in particular, the guard state) are active, we can use an arbitrary sequence $\alpha^i$, for $1 \leq i \leq \ell$, before the first rule letter. This determines the first singleton subset from which we start applying rules.

The construction is presented in Fig. 1. The transition function $\delta$ is formally defined as follows:
Figure 1. The scheme of the DFA $A$ for a set rewriting system. All omitted transitions go to $f_0$.

- $\delta(q_0, \alpha) = p_1$.
- $\delta(p_i, \alpha) = p_{i+1}$ for all $i \in \{0, 1, \ldots, \ell - 1\}$.
- $\delta(p_\ell, \alpha) = f_0$; this is required for the irrevocably accepting property of $f_0$.
- $\delta(p_i, r_j) = \begin{cases} s^{i,j}_\ell, & \text{if } r_j(p_i) \neq \bot \\ f_0, & \text{otherwise} \end{cases}$ for all $i \in \{1, 2, \ldots, \ell\}$ and $j \in \{1, 2, \ldots, m\}$; when a rule is used, these transitions map a state from $Q_P$ to the beginning of the setting chain or to $f_0$ if the rule is not legal when $p_i$ is in the subset.
- $\delta(q_0, r_j) = f_0$ for all $j \in \{1, 2, \ldots, m\}$; this forbids applying rule letters when $q_0$ is active.
- $\delta(s^{i,j}_x, \alpha) = s^{i,j}_{x-1}$ for all $i \in \{1, 2, \ldots, \ell\}$, $j \in \{1, 2, \ldots, m\}$, and $x \in \{\ell, \ell - 1, \ldots, 2\}$; these are the setting chains on $\alpha$.
- $\delta(s^{i,j}_1, \alpha) = q_\alpha$ for all $i \in \{1, 2, \ldots, \ell\}$ and $j \in \{1, 2, \ldots, m\}$; the setting chains end with the guard state.
\[ \delta(s^{i,j}_k, r_y) = f_0 \] for all \( i, x \in \{1, 2, \ldots, \ell \} \) and \( j, y \in \{1, 2, \ldots, m\} \); when the simulation pattern is not yet complete (less than \( \alpha \) letters were applied, so there are some active states in the setting chains), this forbids using rule letters.

- \( \delta(q_s, \alpha) = f_0 \); this forbids applying \( \alpha \) when the guard state is active.
- \( \delta(q_s, r_j) = q_s \); rule letters are allowed when the guard state is active and they deactivate it.
- \( \delta(f_i, \alpha) = f_i+1 \) for all \( i \in \{0, 1, \ldots, \ell - 1\} \); this chain of forcing states provides the property that whenever \( f_0 \) becomes active, the word is irrevocably accepted.
- \( \delta(f_i, r_j) = q_s \) for all \( i \in \{0, 1, \ldots, \ell \} \) and \( j \in \{1, \ldots, \ell\} \); rule letters clean the forcing states.
- \( \delta(f_i, \alpha) = q_s \); the chain of the forcing states ends with the sink state.

The set of final states \( F \) is the union of:

- \( Q_F \); all forcing states are final.
- \( \{s^{i,j}_k \mid i, k \in \{1, 2, \ldots, \ell\} \land j \in \{1, 2, \ldots, m\} \land p_k \in r_j(i)\} \); states in a setting chain are final according to the rule of that chain.

Whenever a final state becomes active, \( q_0 \) becomes active through an \( \varepsilon \)-transition. Note that the indices in the setting chains are decreasing. This keeps the correspondence that if a state \( s^{i,j}_k \) is final and \( p_i \) is in a subset \( C \), then \( p_k \) will be active after applying \( r_j \alpha^i \) to \( C \).

**Correctness.** The correctness is observed through the following lemmas.

The first lemma states that whenever \( f_0 \) becomes active, all subsequent words will be accepted, thus it must be avoided when constructing a non-accepted word.

**Lemma 3.2.** If a word \( w \in \Sigma^* \) is not \( f_0 \)-omitting, then it is irrevocably accepted.

**Proof.** There is a prefix \( u \) of \( w \) such that \( f_0 \in \delta_A(q_0, u) \). It is enough to observe that for every word \( v \), \( \delta_A(\{f_0\}, v) \) contains a forcing state. All forcing states are final, thus \( uv \) and, in particular, all words containing \( w \) as a prefix will be accepted. Suppose this is not the case, and let \( v \) be a shortest word such that \( \delta_A(\{f_0\}, v) \) does not contain a forcing state. Then for every non-empty proper prefix \( v' \) of \( v \), \( \delta_A(\{f_0\}, v') \) does not contain \( f_0 \), which would contradict that \( u \) is a shortest word. Thus the only possibility for \( v \) is to start with \( \alpha^{\ell+1} \); otherwise, active state \( q_0 \) would be mapped to \( f_0 \) by the transition of a rule letter after \( \alpha^i \) for \( i \leq \ell \). However, the transition of \( \alpha^{\ell+1} \) through the chain on \( Q_F \) also maps \( q_0 \) to \( f_0 \), which yields a contradiction. \( \square \)

**Lemma 3.3.** Let \( C \subseteq Q_{F} \cup \{q_s\} \), let \( S = C \cap Q_{F} \) be non-empty, and let \( w = r_1, \alpha^\ell \ldots r_i, \alpha^\ell \) be a simulating word for \( S \). Then \( C' = (S \cdot r_1 \ldots r_i) \cup \{q_s\} \).

**Proof.** Let \( C \) and \( S \) be as in the lemma, and let \( r_j \) be a rule. The transitions of \( r_j \) map each state \( p_i \in S \) to \( s^{i,j}_k \). Then the transitions of \( \alpha^\ell \) map these active states along the setting chains, maybe activating state \( q_0 \) when the setting state is final. Eventually, they are mapped to \( q_s \). A state \( s^{i,j}_k \) is final if and only if \( p_h \in r_j(p_i) \). From the construction, if \( s^{i,j}_k \) is final, then \( q_0 \) becomes active after \( \alpha^{\ell-h} \), which is then mapped to \( p_h \) by the transition of the remaining \( \alpha^h \). After the last \( \alpha \) letter, the setting states are mapped to guard state \( q_s \). Hence, we have \( C' = (S \cdot r_j) \cup \{q_s\} \).

Since the set rewriting system is non-emptying, the set \( S' = S \cdot r_j \) is non-empty, and we can apply the argument iteratively. Hence, the lemma follows by induction on \( k \). \( \square \)

We show that, unless \( f_0 \) is activated, a word applied to a subset \( C \subseteq Q_{A} \) must be a prefix simulating word for \( C \cap Q_{F} \). The required condition is that the guard state is also in \( C \), so one cannot shift the states on \( Q_{F} \).

**Lemma 3.4.** Let \( S \subseteq Q_{F} \) be non-empty, and let \( C = S \cup \{q_s\} \). If \( w \) is \( f_0 \)-omitting for \( C \), then \( w \) is a prefix of a simulating word for \( S \).
Proof. First, we observe that every word \( w \) which does not activate \( f_0 \), unless it is the empty word, must start with a rule letter \( r_j \), since using \( \alpha \) maps \( q_0 \) to \( f_0 \) and we have assumed \( q_0 \in C \). Additionally, \( r_j \) must be legal for \( S \), as otherwise \( f_0 \) would be activated. Afterwards, some of the first setting states must be active, because \( S \neq \emptyset \). Hence \( \alpha^j \) must be used, unless \( w \) ends. By Lemma 3.3 for \( C \) and \( r_j \alpha^j \), we know that the set of active states is \( C' = (S \cdot r_j) \cup \{ q_\emptyset \} \). By iterating this argument, we observe that between each rule letter there must be exactly \( \ell \) letters \( \alpha \), and at the end, there are at most \( \ell \) letters \( \alpha \). Furthermore, each of the rules applied must be legal. Therefore, we know that word \( w \) has to be a prefix of some simulating word for \( S \).

In the beginning, before we can apply a simulating word, we can choose an arbitrary singleton \( \{ p_i \} \) as the initial word. Then a simulating word must be applied, as otherwise \( f_0 \) is activated.

Lemma 3.5. If a word \( w \) is \( f_0 \)-omitting, then \( w \) is a prefix of \( \alpha^i w' \) for \( 1 \leq i \leq \ell \) and some \( w' \) that is a simulating word for \( \{ p_i \} \).

Proof. Let \( w \) be a \( f_0 \)-omitting word. Since we start from \( \{ q_0 \} \), we know that \( w \) must start with \( \alpha^i \) for some \( 1 \leq i \leq \ell \), unless it is empty. Then, unless \( w \) ends, there is some rule letter \( r_j \), which must be legal for \( \{ p_i \} \), followed by \( \alpha^j \).

Hence \( w = \alpha^i r_j \alpha^j w'' \) for some suffix \( w'' \) of \( w \). By Lemma 3.3, we have \( C = \delta_{A^i}(\{ q_0 \}, \alpha^i r_j \alpha^j) = S \cup q_\emptyset \), for \( S = \{ p_i \} \cdot r_j \). Since, the set rewriting is non-emptying, \( S \neq \emptyset \). By Lemma 3.4 applied to \( C \), since \( f_0 \) cannot be activated, we know that \( w'' \) must be a prefix of a simulating word for \( S \). We let \( w' = r_j \alpha^j w'' \), which is a prefix of a simulating word for \( \{ p_i \} \).

Finally, we show the equivalence between the immortality of the set rewriting system and not cofiniteness of the language of \( A^* \).

Lemma 3.6. The set rewriting system \( (P, R) \) is immortal if and only if there are infinitely many words not accepted by \( A^* \).

Proof. Suppose that the set rewriting system is immortal. For every \( k > 0 \), we will construct a non-accepted word \( w \) of length at least \( \ell \cdot (\ell + 1) \). Since the system is immortal and by Lemma 2.4 there exists a singleton \( \{ p_i \} \) and a sequence of \( k \) legally applied rules \( r_i, \ldots, r_k \) to \( \{ p_i \} \). Hence, \( w = \alpha^i r_i \alpha^f \ldots r_k \alpha^f \) is a simulating word for \( S = \{ p_i \} \). By Lemma 3.3, we know that \( \delta_{A^i}(\{ q_0 \}, w) \subseteq Q_P \cup \{ q_\emptyset \} \), which does not contain any final states, thus \( w \) is not accepted.

Conversely, assume that \( L^* \) is not cofinite. Thus there are infinitely many words that are not accepted, which, in particular, by Lemma 3.2 are \( f_0 \)-omitting.

Let \( w \) be a \( f_0 \)-omitting word longer than \( \ell + (\ell + 1)2^{Q_P} \). By Lemma 3.5 we know that \( w \) has the form of \( \alpha^i w' \), where \( i \leq \ell \) and \( w' \) is a prefix of a simulating word for \( \{ p_i \} \).

This simulating word must have length at least \( (\ell + 1)2^{Q_P} \), hence it contains a sequence of \( k \geq 2^{Q_P} \) rule letters. We conclude that this sequence \( r_{i_1} \ldots r_{i_k} \) is legal for \( \{ p_i \} \), and it does not lead to the empty set as it is unreachable from a non-empty subset. If we look at the sequence of the sets \( S_j = \{ p_i \} \cdot r_{i_1} \ldots r_{i_j} \), for \( j = 0, \ldots, 2^{Q_P} \), then there must be some distinct indices \( x \) and \( y \) such that \( x < y \) and \( S_x = S_y \). Hence, the rewriting system is immortal because of \( S_x \) and the sequence \( r_{i_{x+1}} \cdot r_{i_{x+2}} \ldots, r_{i_y} \).

We conclude this part with

Theorem 3.7. Problem \ref{Frobenius} is PSPACE-hard if \( L \) is specified by a DFA.
3.2. Binarization and a list of words. To show that the PSPACE-hardness remains when the alphabet is binary, we apply a variation of a standard binarization of a language.

We modify the construction of $\mathcal{A}$ from Subsubsection 3.1 to obtain a binary $\mathcal{B} = (Q_B, \{0, 1\}, \delta_B, q_0, F)$, where $Q_B$ is $Q_A$ with some states added, and $q_0$ and $F$ are from the original $\mathcal{A}$.

The letter $\alpha$ is encoded by 0, and every letter $r_i$ is encoded by 1$^0$ for $i \leq m - 1$ and $r_m$ is encoded by 1$^m$. Note that this binary encoding is a complete prefix code, thus the encoding of a word $w \in \Sigma^*$ is unambiguous and every binary word $w'$, after removing at most $m - 1$ symbols from the end, encodes some word $w$.

The construction of $\mathcal{B}$ is as follows. We introduce $m - 1$ new states for each state of $Q_F$ in the way that a word encoding $r_i$ acts as $r_i$ in the original automaton; these new states are not final. The transitions labeled by $\alpha$ are now labeled by 0. The transitions from state $q_0$, from state $q_b$, and from states $f_i$ labeled by a rule letter are simply replaced with one transition labeled by 1. The sink state $q_s$ remains a fixed point on both 0 and 1.

Correctness.

**Lemma 3.8.** If a word $w$ is $f_0$-omitting for a subset $C \subseteq Q_{A'} \setminus Q_F$, then its binary encoding $w'$ is $f_0$-omitting for $C$ and such that $\delta_{A'}(C, w) = \delta_B(C, w')$.

**Proof.** This can be observed by analyzing the transitions from each state in $Q_{A'} \setminus Q_F$ in both automata.

**Lemma 3.9.** If a word $w$ is not $f_0$-omitting for a subset $C \subseteq Q_{A'} \setminus Q_F$ for $A^*$, then its binary encoding $w'$ is not $f_0$-omitting for $C$ in $B^*$.

**Proof.** Suppose that a prefix of $w$ activates $f_0$; let $ua$ be a shortest such prefix for $u \in \Sigma^*$ and $a \in \Sigma$. From (1), we know that $\delta_{A'}(C, u) = \delta_{B'}(C, u')$, where $u'$ is the binary encoding of $u$. If $a = \alpha$, then $u'0$ activates $f_0$ in $B^*$. If $a \in R$, then active $q_0$, an active state $s_k^{j_2}$, or an active state $p_t$ is mapped to $f_0$ by the transition of $a$. In the first two cases, $a'1$ activates $f_0$, and in the third case, $a'0$ activates $f_0$, where $a'$ is the binary encoding of $a$.

**Lemma 3.10.** The language of $B^*$ is cofinite if and only the language of $A^*$ is cofinite.

**Proof.** From Lemma 3.8 and by the fact that all not $f_0$-omitting words for $\{q_0\}$ are accepted, we know that if a word $w \in \Sigma^*$ is not accepted by $A^*$, then its binary encoding $w' \in \{0, 1\}^*$ is not accepted by $B^*$. Thus, we get that if infinitely many words are not accepted by $A^*$, then the language of $B^*$ is also not cofinite.

Assume now that the language of $B^*$ is not cofinite. For any $t \geq m$, let $w'$ be a binary word not accepted by $B^*$ and of length at least $t$. Let $u'$ be the maximal prefix of $w'$ that properly encodes a word $u \in \Sigma^*$; then $u'$ is shorter by at most $m - 1$ than $w'$. We observe that Lemma 3.2 holds for $B^*$. Hence, since $w'$ is not accepted, $u'$ must be $f_0$-omitting. From Lemma 3.9, we know that $u$ also must be $f_0$-omitting. By applying the same argument as in the proof of Lemma 3.10 to $u$ for $t \geq (\ell + 1)2|Q^u| + (m - 1) + (\ell + 1)$, we conclude that the set rewriting system is immortal, thus the language of $A^*$ is not cofinite.

We conclude with

**Theorem 3.11.** Problem [L] is PSPACE-hard if $L$ is specified by a DFA over a binary alphabet.

Finally, we show that the problem remains PSPACE-hard if the language is specified as a list of binary words. It follows from the fact the lengths and the number of words accepted by our binary DFA is polynomial.
Corollary 3.12. Problem (4.7) is PSPACE-hard if $L$ is given as a list of binary words.

Proof. The maximum length of words accepted by our binary DFA $B$ is equal to $3\ell + m + 2$, which
the length of the longest path from $q_0$ to a final state: $q_0 \xrightarrow{0^\ell} p_\ell \xrightarrow{1^m} s_{\ell,m} \xrightarrow{0^\ell} s_1 \xrightarrow{0} q_0 \xrightarrow{1} f_0 \xrightarrow{0^\ell} f_1$.

For the number of words in the recognized language, we consider all final states. The first type
of final states is the setting states. Each such state is reachable from $q_0$ by a unique path, thus
each of them induces one word in the language, which gives at most $m\ell^2$ words. The second type
is forcing states. A state $f_i$ may be reached through different paths, but all such paths consist of a
path to $f_0$, whose number is bounded by the number of states, and a unique path from $f_0$ to $f_i$. In
this case, we have at most $(1 + \ell m(1 + \ell) + 1)(1 + \ell)$ words. □

4. THE FACTOR UNIVERSALITY PROBLEM

It is known that the problem whether a given language specified by an NFA is factor universal
is PSPACE-complete. In contrast, it is solvable in linear time when the language is specified by a
DFA [15].

Let $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ be an NFA. For a subset $S \subseteq Q_N$, a word $w \in \Sigma^*$ is called $S$-
emptying if $\delta_N(S, w) = \emptyset$. If every state in $N$ is reachable from the initial state $q_0$ and from every
state a final state can be reached, then we have the following criterion: the language of $N$ is factor
universal if and only if there exists a $Q_N$-emptying word [15].

Clearly, the slightly more general problem, whether for a given arbitrary semi-NFA, there exists
a $Q_N$-emptying word is also PSPACE-complete. A semi-NFA is an NFA without specified initial
and final states, which are irrelevant to the problem. We can restate this problem in terms of a set
rewriting system from Subsection 2.3. If we resign from forbidding rules by $\bot$, then such a system
$(P, R)$ is equivalent to a semi-NFA whose set of states is $P$ and the alphabet is $R$. We call this kind
of a set rewriting system permissive. For a subset $C \subseteq P$, a sequence of rules $r_{i_1}, \ldots, r_{i_k}$ such that
$C \cdot r_{i_1} \cdot \ldots \cdot r_{i_k} = \emptyset$ is called $C$-emptying.

We obtain the following PSPACE-complete problem:

**Problem 4.1.** For a given permissive set rewriting system $(P, R)$, does there exist a $P$-emptying
sequence of rules?

We will reduce from Problem (4.1) to Problem (1.2) when $L$ is specified as a DFA $A$ instead of a list of words. We slightly modify the DFA construction from Subsection 3 as follows.
We remove the last state $f_1$ and end the chain of the forcing states with $f_{\ell - 1}$. Thus, the set $Q_F$
becomes $\{f_x \mid x \in \{0, 1, \ldots, \ell - 1\}\}$, and we redefine the transition $\delta(f_{\ell - 1}, \alpha) = q_s$. As before,
using a standard construction, we build the NFA $A^*$ recognizing the language $L^*$, where $L$ is the
language of $A$.

The idea is as follows. In the NFA $A^*$, all states are reachable from the initial state $q_0$. Since we
also remove the sink state $q_0$, the NFA meets the mentioned criterion for factor universality. Thus,
the language of $A^*$ is factor universal if and only if there is a $Q_A$-emptying word.

Words in our NFA simulate the set rewriting system in the same way as in Subsection 3.1. The
construction ensures that to map the whole $Q_F$ to the empty set, there must exist a $P$-emptying
sequence of rules in the set rewriting system. The forcing states have the property that whenever
$f_0$ is activated, the only way to get rid of all forcing states is to make the whole $Q_F$ active again.
When $f_0$ is active, which is the case at least at the beginning, this is done by applying the word $\alpha^\ell$.

**Correctness.**
Lemma 4.1. We have:

1. \( \delta_{A^*}(Q_{A^*}, r_1^2) = \{ f_0, q_0 \} \), and
2. \( \delta_{A^*}(f_0, \alpha^\ell) = Q_P \).

We show that when \( f_0 \) is activated, the only way to get rid of all forcing states is to activate the whole \( Q_P \) at some point.

Lemma 4.2. Let \( C \subseteq Q_{A^*} \), let \( f_0 \in C \), and let \( w \) be a word such that \( \delta_{A^*}(C, w) \cap Q_F = \emptyset \). There exists a prefix \( u \) of \( w \) such that \( Q_P \subseteq \delta_{A^*}(C, u) \).

Proof. It is enough to prove the lemma for \( C = \{ f_0 \} \). Let \( w \) be a shortest word with the property. Hence, there is no non-empty prefix \( u \) of \( w \) such that \( f_0 \in \delta_{A^*}(\{ f_0 \}, u) \). Consider a prefix \( \alpha^i \) of \( w \) for an \( i < \ell \). Then \( \delta_{A^*}(\{ f_0 \}, \alpha^i) = \{ f_1, q_0, p_1, \ldots, p_i \} \). Thus \( w \) must have length at least \( \ell \). If \( w \) would start with \( \alpha^i r_j \) for an \( i < \ell \) and some rule letter \( r_j \), then active state \( q_0 \) would be mapped to \( f_0 \) by the transition of \( r_j \). Thus, \( w \) must start with the prefix \( u = \alpha^\ell \), which is that \( \delta_{A^*}(\{ f_0 \}, u) = Q_P \).

We show the properties of a simulating word.

Lemma 4.3. Let \( C \subseteq Q_P \cup \{ q_g \} \), let \( S = C \cap Q_P \) be non-empty, and let \( w = r_i \alpha^\ell \ldots r_k \alpha^\ell \) be a simulating word for \( S \). Then:

\[
C' = \begin{cases} 
(S \cdot r_i \ldots r_k) \cup \{ q_g \}, & \text{if } S \cdot r_i \ldots r_k \neq \emptyset \\
\emptyset = (S \cdot r_i \ldots r_k), & \text{otherwise}.
\end{cases}
\]

Proof. In the case of \( S \cdot r_i \ldots r_k \neq \emptyset \), the proof is the same as that of Lemma 3.3 since for all \( 0 \leq j \leq k-1 \), we have \( S \cdot r_i \ldots r_j \neq \emptyset \), thus all preconditions apply.

Otherwise, let \( j < k \) be the smallest index such that the set \( S \cdot r_i \ldots r_j \) is empty. By the argument for the first case, we know that \( \delta_{A^*}(S, r_i \alpha^\ell \ldots r_j \alpha^\ell) = \{ q_g \} \). Applying the next letter \( r_{j+1} \) removes this single state, yielding the empty set.

Lemma 4.4. Let \( S \subseteq Q_P \) be non-empty, and let \( C = S \cup \{ q_g \} \). If \( w \) is \( f_0 \)-omitting for \( C \), then either:

1. \( w \) is a prefix of a simulating word for \( S \), or
2. \( w \) is a simulating word for \( S \) whose sequence of rules is \( S \)-emptying.

Proof. Following the proof of Lemma 3.4, we observe that a word \( w \) must start with \( r_j \alpha^\ell \), unless it ends prematurely. Then, by Lemma 4.3, we have \( C' = \delta_{A^*}(C, r_j \alpha^\ell) = (S \cdot r_j) \cup \{ q_g \} \). We apply this argument iteratively, until either \( w \) ends, in which case (1) holds, or \( C' \) becomes \( \{ q_g \} \), in which case (2) holds.

Lemma 4.5. Let \( w \) be a word such that \( \delta_{A^*}(Q_P, w) = \emptyset \). Then \( w \) contains a factor \( v \) which is a simulating word for \( Q_P \) whose sequence of rules is \( P \)-emptying.

Proof. It is enough to prove the lemma for words \( w \) that do not have a non-empty prefix \( u \) such that \( \delta_{A^*}(Q_P, u) = Q_P \); otherwise, we can search for a factor \( v \) in \( w \) with \( u \) removed. Hence, by Lemma 4.2, \( w \) must be \( f_0 \)-omitting. By Lemma 4.4, we have two possibilities (1) and (2). In case (2), we immediately know that \( w \) contains a prefix that is a simulating word for \( Q_P \) whose sequence of rules is \( P \)-emptying. In case (1), \( w \) is a prefix of a simulating word for \( Q_P \). If \( w \) itself is not a simulating word, write \( w = w r_{k+1} \alpha^\ell \) for a simulating word \( v = r_1 \alpha^\ell \ldots r_k \alpha^\ell \) for \( Q_P \) and some \( 0 \leq i < \ell \); otherwise let \( v = w \). Let \( C' = \delta(Q_P, v) \). By Lemma 4.3, \( C' \subseteq Q_P \cup \{ q_g \} \) and
For Lemma 4.7.

Assume that there is a

The following conditions are equivalent:

Lemma 4.6.

both constructions differ only on the set

Q

as in Subsection 3.2. We observe that Lemma 3.8 and Lemma 3.9 also hold in this case. It is because

δ

(3) ⇒

Q

has to be also

□

Finally, we show the equivalence between the problems.

Lemma 4.6. The following conditions are equivalent:

(1) The permissive set rewriting system (P, R) admits a P-emptying sequence of rules.

(2) There exists a Q_P-emptying and f_0-omitting for Q_P word for A^*.

(3) There exists a Q_A^* -emptying word for A^*.

Proof. (1) ⇒ (2): Suppose that for the set rewriting system there is a sequence of rules r_{i_1}, ..., r_{i_k}

that is P-emptying. We take the word w = r_{i_1}α^1 ... r_{i_k}α^l, which is a simulating word for Q_P. By Lemma 4.3 we conclude that δ_{A^*}(Q_P, w) ⊆ \{q_g\}. Thus, w_{r_1} is Q_P-emptying and f_0-omitting for Q_P.

(2) ⇒ (3): If w is a Q_P-emptying word, then, by Lemma 4.1, δ_{A^*}(Q_{A^*}, r_{i_1}\alpha^l w) = ∅.

(3) ⇒ (1): If there exists a Q_{A^*} -emptying word w ∈ Σ^*, then, in particular, δ_{A^*}(Q_P, w) = ∅. By Lemma 4.2, w contains a factor v which is a simulating word for Q_P whose sequence of rules is P-emptying.

□

4.2. Binarization and list of words. We reduce to a binary DFA B using the same construction as in Subsection 3.2. We observe that Lemma 3.8 and Lemma 3.9 also hold in this case. It is because both constructions differ only on the set Q_P, whose transitions are irrelevant for the observations.

Lemma 4.7. For B^*, there is a Q_{B^*}-emptying and f_0-omitting for Q_P word if and only if there is a Q_{B^*}-emptying word.

Proof. Assume that there is a Q_P-emptying and f_0-omitting for Q_P word w. We have δ_{B^*}(Q_{B^*}, 101) = \{f_0, q_0\} and δ_{B^*}(f_0, 0^f) = Q_P. Thus, δ_{B^*}(Q_{B^*}, 1010^f w) = ∅.

Conversely, let w be a Q_{B^*}-emptying word. Let u be the longest prefix of w such that Q_P ⊆ δ_{B^*}(Q_{B^*}, u), and let w = u\cdot v. Observe that Lemma 4.2 holds for B^*: for this, it is enough to change in its proof α to 0 and r_j to 1. By this lemma, v has to be f_0-omitting for Q_P, as otherwise u could be longer. Hence, w is Q_P-emptying and f_0-omitting for Q_P.

□

Lemma 4.8. There is a Q_P-emptying and f_0-omitting for Q_P word for A^* if and only if there is such a word for B^*.

Proof. Let w be a Q_P-emptying and f_0-omitting for Q_P word for A^*. From Lemma 5.8 we know that its binary encoding w' is f_0-omitting for Q_P and such that δ_{B^*}(Q_P, w') = δ_{A^*}(Q_P, w) = ∅.

Conversely, assume that there is a Q_P-emptying and f_0-omitting for Q_P binary word w' for B^*. We know that w'0 has the same properties, and it must be an encoding of some word w ∈ Σ_{A^*}. Then, from Lemma 5.9 w must be also f_0-omitting for Q_P. From Lemma 5.8 we conclude that w has to be also Q_P-emptying.

We conclude this part with

Theorem 4.9. Problem 1.2 is PSPACE-hard if L is specified by a binary DFA.

Finally, we conclude that the problem remains PSPACE-hard if the language is specified as a list of binary words.

Corollary 4.10. Problem 1.2 is PSPACE-hard if L is specified by a list of binary words.
Proof. As in the proof of Corollary 3.4.12 we count all the words that are accepted by our binary DFA. In this case, the maximum length of the accepted words is $3\ell + m + 1$. The number of accepted words is bounded by $m\ell^2 + (1 + \ell m(1 + \ell) + 1)\ell$.

5. Lower bounds

By $\|L\|_{\text{max}}$ we denote the length of the longest words in $L$ and by $\|L\|_{\text{sum}}$ we denote the sum of the lengths of the words in $L$. Thus, $\|L\|_{\text{sum}}$ can be treated as the size of the input $L$.

5.1. The longest omitted words. In [12] it is mentioned that for each odd integer $n \geq 5$, there exists a set of binary words $L$ of length at most $n$ such that $L^*$ is cofinite and the longest words not in $L^*$ are of length $\Omega(n^22^{2n})$. However, the constructed $L$ contains exponentially many words, thus an exponential bound in terms of the size of $L$ could not be inferred.

We show an exponential in $\|L\|_{\text{sum}}$ lower bound on the length of the longest words not in $L^*$ when $L^*$ is cofinite. The idea is to construct a set rewriting system with an integer counter that counts from zero up to an exponential number. Then we apply our reduction to obtain a list of binary words.

**Theorem 5.1.** There exists an infinite family whose elements $L$ are finite sets of binary words such that $L^*$ is cofinite and the longest words not in $L^*$ are of length at least $2^{\|L\|_{\text{sum}} - 1/4}$ and this length is $2^{\Omega(\sqrt[4]{\|L\|_{\text{sum}}})}$.

**Proof.** For every $n > 1$, we construct a set rewriting system $(P, R)$ which represents a binary counter of length $n$.

The set $P$ contains elements representing bits of an integer. Let $P = \{p_i : i \in \{0, 1, \ldots, n-1\}\}$ For a subset $S \subseteq P$, we define $\text{val}(S, i) = 2^i$ if $p_i \in S$ and $\text{val}(S, i) = 0$ otherwise, and $\text{val}(S) = \sum_{0 \leq i < n-1} \text{val}(S, i)$.

For each $i \in \{0, 1, \ldots, n-1\}$, we define a rule that will be legal only if there is no $p_i$. Then the rule will clear all the indices smaller than $i$. We will also make sure that using any rule makes the value of the counter larger, which prevents cycling. The set of rules $R$ consists of rules $r_j$ for $j \in \{0, 1, \ldots, n-1\}$, where $r_j$ is defined as follows:

- $r_j(p_j) = \perp$;
- $r_j(p_i) = \{p_j\}$ for $i \in \{0, 1, 2, \ldots, j-1\}$;
- $r_j(p_i) = \{p_j, p_i\}$ for $i \in \{j+1, j+2, \ldots, n-1\}$.

This set rewriting system does not contain a non-trivial cycle, because if its value is non-zero, then using any legal rule makes the value larger. We observe that for a non-empty subset $S \subseteq P$ and a legal a rule $r_j$, we have $\text{val}(S) < \text{val}(S \cdot r_j)$. It is because we know that $\text{val}(S, i) = 0$ and

$$\text{val}(S \cdot r_j) = \sum_{j \leq i < n} \text{val}(S \cdot r_j, i) + 2^j = \sum_{j \leq i < n} \text{val}(S, i) + 2^j > \sum_{j \leq i < n} \text{val}(S, i) + \sum_{0 \leq i < j} 2^i \geq \sum_{0 \leq i < n} \text{val}(S, i) = \text{val}(S).$$

Observe that for $S = \{p_1\}$, the longest possible legal sequence of rules has length $2^n - 2$. This sequence corresponds to incrementing the value $\text{val}(S)$ by 1 each time we apply a rule. For a non-empty subset $S \subseteq P$, we apply the $r_j$ for $j$ being the smallest index such that $p_j \notin S$, and we have $\text{val}(S \cdot r_j) + 1 = \text{val}(S)$. Since our set rewriting system models a binary counter of length $n$ and can start with 1, we know that the longest possible sequence of rules that is legal has length $2^n - 2$. 

Now, we use the construction from Subsection 3.2 to create a list of binary words $L$ for this set rewriting system with the reversed order of elements in $P$. Since the set rewriting system is mortal, $L^*$ is cofinite. The length of the longest words in this list is equal to $||L||_{\max} = 4n + 2$ and there are at most $2 + n + n^2 + 2n^3 + n^4$ words (see the proof of Corollary 3.12), thus $||L||_{\sum} \leq (2 + n + n^2 + 2n^3 + n^4)(4n + 2)$. 

We take the binary simulating word $w'$ for the set $S = \{p_1\}$ that contains the longest possible legal sequence of rules in this set rewriting system. From Lemma 3.3 and Lemma 3.8, we know that the word $\alpha^nw' \notin L^*$. If we bound the length of the encoding of each rule letter by 2, then the length of this word is at least $(2^n - 2) \cdot (n + 2) + n$, since one rule application corresponds to at least $n + 2$ letters (the encoding of a rule letter and $0^n$).

Since $n = \frac{||L||_{\max} - 2}{4}$ and $n = \Omega(\sqrt[4]{||L||_{\sum}})$, the length of this word is at least $2^{\frac{||L||_{\max} - 2}{4}} \cdot \frac{||L||_{\max} - 2}{4}$, when written in terms of $||L||_{\max}$, and it is $2^{\Omega(\sqrt[4]{||L||_{\sum}})}$ in terms of $||L||_{\sum}$. $\square$

5.2. The shortest incompletable words. We show that when $L^*$ is not factor universal, the length of the shortest words that are not completable can be exponential in either $||L||_{\max}$ or $||L||_{\sum}$.

The idea is similar to that for the lower bound on the length of the longest omitted words. We construct a permissive set rewriting system with an integer counter that will count from an exponential number to zero. Whenever a wrong rule is used, the counter will be reset to its maximum value. Then we apply our reduction to obtain a list of binary words.

**Theorem 5.2.** There exists an infinite family whose elements $L$ are finite sets of binary words such that the shortest incompletable binary words are of length at least $2^{\frac{||L||_{\max} - 2}{4}} \cdot ||L||_{\max} - 1$ and this length is $2^{\Omega(\sqrt[4]{||L||_{\sum}})}$.

**Proof.** For every $n > 1$, we construct a permissive set rewriting system $(P, R)$, which represents a binary counter of length $n$. The set $P$ contains elements representing bits of an integer; let $P = \{p_i \mid i \in \{0, 1, \ldots, n - 1\}\}$ For a subset $S \subseteq P$, we define $\text{val}(S, i) = 2^i$ if $p_i \in S$ and $\text{val}(S, i) = 0$ otherwise, and $\text{val}(S) = \sum_{0 \leq i \leq n - 1} \text{val}(S, i)$.

We define the rules that allow the value of the counter to decrease by 1. If a wrong rule is used, the counter is reset to its maximal value. The set of rules $R$ consists of rules $r_j$ for $j \in \{0, 1, \ldots, n - 1\}$, where $r_j$ is defined as follows:

1. $r_j(p_i) = \{p_i \mid i \in \{0, 1, \ldots, j - 1\}\}$;
2. $r_j(p_i) = P$ for $i \in \{0, 1, \ldots, j - i\}$;
3. $r_j(p_i) = \{p_i\}$ for $i \in \{j + 1, j + 2, \ldots, n - 1\}$.

We observe that the emptying of this set rewriting system corresponds to setting the counter to 0. For a subset $S$, let $i$ be the smallest index such that $p_i \in S$. Then for all the smaller positions $j < i$, $p_j \notin S$. Notice that for all rules $r_k$ for $k \in \{1, 2, \ldots, n - 1\} \setminus \{i\}$, $\text{val}(S \cdot r_k) \geq \text{val}(S)$. This is because if $k < i$, then $S \cdot r_k = S$ and if $k > i$, then $S \cdot r_k = P$. Thus, the only rule that decreases the counter is $r_i$ and $\text{val}(S \cdot r_i) = \text{val}(S) - 1$. Hence, the shortest sequence of rules that is $P$-emptying has length $2^n - 1$.

Now, we use the construction from Subsection 4.2. In this way, we create a list of binary words $L$ which has the property that the shortest words not being a factor of some word from $L^*$ have length at least $(2^n - 1) \cdot (n + 2)$, since one rule application corresponds to at least $n + 2$ letters.
Similarly as before, setting \( n = \frac{\|L\|_{\text{max}} - 1}{4} \) and \( n = \Omega(\sqrt[4]{\|L\|_{\text{sum}}}) \), the length of this word is at least \( 2^{\frac{\|L\|_{\text{max}} - 1}{4}} \cdot \frac{\|L\|_{\text{max}} - 1}{4} \), when written in terms of \( \|L\|_{\text{max}} \), and it is \( 2^{\Omega(\sqrt[4]{\|L\|_{\text{sum}}})} \) in terms of \( \|L\|_{\text{sum}} \).

\[ \square \]

6. Upper bounds

We show algorithms and upper bounds on the related length for both problems, which are exponential only in \( \|L\|_{\text{max}} \) while remains polynomial in \( \|L\|_{\text{sum}} \).

For the Frobenius monoid problem, there was shown upper bound \( \frac{2^k}{2|\Sigma| - 1}(2^{\|L\|_{\text{max}}|\Sigma|\|L\|_{\text{max}} - 1}) \) on the length of the longest words not in \( L^* \) when \( L^* \) is cofinite \([12]\). We show an upper bound that involves both \( \|L\|_{\text{max}} \) and \( \|L\|_{\text{sum}} \).

**Theorem 6.1.** Problem \([L, L] \) can be solved in time exponential only in \( \|L\|_{\text{max}} \) while polynomial in \( \|L\|_{\text{sum}} \). If \( L^* \) is cofinite, then the longest words not in \( L^* \) have length at most \( 1 + (\|L\|_{\text{sum}} + 1)2^{|L|_{\text{max}}} \).

**Proof.** We construct a DFA \( A \) recognizing \( L \) in the way that it forms a radix trie. Then every distinct word \( w \) maps the initial state \( q_0 \) to a different state, unless it is the unique non-final sink state \( q_b \).

By a standard construction for the Kleene star, we construct an NFA \( A^* = (Q_A^*, \Sigma, \delta_A^*, q_0, F_A^*) \) recognizing \( L^* \). We can assume that \( L \) does not contain the empty word, so \( A^* \) contains an \( \varepsilon \)-transition from every final state to the initial state \( q_0 \). The final states \( F_A^* \) is the set of final states of \( A \) with \( q_0 \) added. We can remove the sink state from \( A^* \), hence from every state, a final state is reachable in \( A^* \).

We observe that in \( A^* \), after reading any word \( w \), there are no more than \( |Q_A^*| \cdot 2^{|L|_{\text{max}}} + 1 \) active states. We define the level of a state \( q \in Q_A^* \setminus \{q_b\} \) to be the length of the (unique) shortest word mapping \( q_0 \) to \( q \). Every state by the action of every letter is mapped to at most one state, which has the level larger by 1, and possibly to \( q_0 \) by following \( \varepsilon \)-transition. Hence, for a subset with at most one state at each level, the action of every letter preserves this property. Since the initial subset is \( \{q_0\} \), after reading any word, for every level at most one state can be active. Moreover, if \( q \) is the active state with the largest level \( i \), the set of possible active states with smaller levels is determined, because if \( w \) is the unique shortest word of length \( i \) such that \( q \subseteq \delta(q_0, w) \), then the only possible active state at a level \( j < i \) is that in \( \delta(q_0, w') \) (if it contains a state of level \( j \)), where \( w' \) is the suffix of \( w \) of length \( j \). The largest possible level is \( \ell \). State \( q_0 \) is active if and only if a final state is \( q_0 \) active, with the exception of the initial active subset \( \{q_0\} \). Hence, we can choose one of the \( |Q_A^*| \) states to be that with the largest level, and then any subset of the \( \ell \) states that are determined by the chosen state.

Having the number of reachable active subsets of states bounded, we can determinize \( A^* \) to a minimal DFA \( D_A^* \) with at most \( |Q_A^*| \cdot 2^{|L|_{\text{max}}} + 1 \) states. Finally, the problem of whether a minimal DFA recognizes a cofinite language is equivalent to whether there exists a cycle containing a non-final state.

Since \( |Q_A^*| \leq \|L\|_{\text{sum}} + 1 \), the upper bound on the length follows. \[ \square \]

For the factor universality problem, only trivial upper bound \( 2^{|L|_{\text{sum}} - |L|_{\text{max}}} + 1 \) was known \([9]\).

**Theorem 6.2.** Problem \([L, \Sigma] \) can be solved in time exponential only in \( \|L\|_{\text{max}} \) while polynomial in \( \|L\|_{\text{sum}} \). If the set is not complete, then the shortest incompletable words have length at most \( \|L\|_{\text{max}} + 1 + (\|L\|_{\text{sum}} + 1)2^{|L|_{\text{max}}} \).
Proof. We construct an NFA $A^*$ for $L^*$ as in the proof of Theorem 6.1. We remove its sink state and make all states initial and final, hence it recognizes the language of all factors of $L^*$. The language is universal if and only if there exists a word $w$ such that $\delta(Q_{A^*}, w) = \emptyset$.[15]

Similarly as before, we observe that in $A^*$, after reading any word $w$ of length at least $|L|_{\text{max}}^\sum$, there are no more than $|Q_{A^*}| \cdot 2^{|L|_{\text{max}}^\sum}$ active states. Since we start with the set of all states $Q_{A^*}$, at the beginning there could be more reachable subsets.

If there exists a word $w$ such that $\delta_A^*(Q_{A^*}, w) = \emptyset$, then for every word $u$ we also have $\delta_A^*(Q_{A^*}, uw) = \emptyset$. Hence, we can start from an arbitrary word $u$ of length $|L|_{\text{max}}^\sum$, and then check the reachability of $\emptyset$ visiting at most $|Q_{A^*}| \cdot 2^{|L|_{\text{max}}^\sum}$ states. \hfill $\Box$

Under a fixed-sized alphabet (as otherwise $|L|_{\text{sum}}$ can be arbitrarily large with respect to $|L|_{\text{max}}^\sum$), we have $|L|_{\text{sum}} \leq |\Sigma||L|_{\text{max}}^\sum$. We conclude that $2^{O(|L|_{\text{max}}^\sum)}$ is a tight upper bound on the lengths related to both problems.

References

[1] M. Beck, R. Diaz, and S. Robins. The Frobenius problem, rational polytopes, and fourier–dedekind sums. *Journal of Number Theory*, 96:1–21, 2002.
[2] A. M. Ben-Amram. Mortality of iterated piecewise affine functions over the integers: Decidability and complexity. *Computability*, 4(1):19–56, 2015.
[3] J. Berstel, D. Perrin, and C. Reutenauer. *Codes and Automata*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009.
[4] V. D. Blondel, J. Cassaigne, and C. Nichitiu. On the presence of periodic configurations in Turing machines and in counter machines. *Theoretical Computer Science*, 289(1):573–590, 2002.
[5] A. Brauer. On a problem of partitions. *Amer. J. Math.*, 64(1):299–312, 1942.
[6] A. Carpi and F. D’Alessandro. On incomplete and synchronizing finite sets. *Theoretical Computer Science*, 664:67–77, 2017.
[7] J. Clément, J.-P. Duval, G. Guaiana, D. Perrin, and G. Rindone. Parsing with a finite dictionary. *Theoretical Computer Science*, 340(2):432–442, 2005.
[8] A. L. Dulmage and N. S. Mendelsohn. Gaps in the exponent set of primitive matrices. *Illinois J. Math.*, 8(4):642–656, 1964.
[9] V. V. Gusev and E. V. Privalov. On Non-complete Sets and Restivo’s Conjecture. In Giancarlo Mauri and Alberto Leporati, editors, *DLT*, pages 239–250. Springer, 2011.
[10] J. Incerpi and R. Sedgewick. Improved upper bounds on shellsort. *Journal of Computer and System Sciences*, 31(2):210–224, 1985.
[11] R. Kannan. Lattice translates of a polytope and the Frobenius problem. *Combinatorica*, 12:161–177, 1992.
[12] J.-Y. Kao, J. Shallit, and Z. Xu. The Frobenius Problem in a Free Monoid. In Susanne Albers and Pascal Weil, editors, *STACS*, volume 1 of *LIPIcs*, pages 421–432, 2008.
[13] S. Kiefer and M. Corto. On Finite Monoids over Nonnegative Integer Matrices and Short Killing Words. In *STACS*, LIPIcs, 2019.
[14] J. L. Ramírez-Alfonsín. Complexity of the Frobenius problem. *Combinatorica*, 16:143–147, 1996.
[15] N. Rampersad, J. Shallit, and Z. Xu. The Computational Complexity of Universality Problems for Prefixes, Suffixes, Factors, and Subwords of Regular Languages. *Fundamenta Informaticae*, 116(1–4):223–236, 2012.
[16] A. Restivo. Some remarks on complete subsets of a free monoid. In Giampiero Mauri and Alberto Leporati, editors, *DLT*, pages 239–250. Springer, 2011.
[17] J. Sandrine, A. Malapert, and J. Provillard. A Synergic Approach to the Minimal Uncompletable Words Problem. *Journal of Automata, Languages and Combinatorics*, 22(4):271–286, 2017.
[18] M.-P. Schützenberger. A property of finitely generated submonoids of free monoids. In G. Pollak, editor, *Algebraic Theory of Semigroups, Proc. Sixth Algebraic Conf.*, Szeged, 1976, pages 545–576. North-Holland, Amsterdam, 1979.
[19] J. Shallit. Open problems in automata theory: an idiosyncratic view, LMS Keynote Address in Discrete Mathematics, *BCTCS* 2014, April 10 2014, Loughborough, England. https://cs.uwaterloo.ca/~shallit/Talks/bc4.pdf
[20] S. H. Smita. A Review of Concatenative text To Speech Synthesis. *International Journal of Latest Engineering Research*, 3, 2014.

[21] Z. Xu. *The Frobenius Problem in a Free Monoid*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 2009.

[22] S. Yu, Q. Zhuang, and K. Salomaa. The state complexities of some basic operations on regular languages. *Theoretical Computer Science*, 125(2):315–328, 1994.